UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Comparação do desempenho de ambientes virtuais na computação em nuvem privada usando a análise estatística e o *benchmark* Hadoop.

Ricardo Soares Bôaventura

Uberlândia
Março de 2015

Ricardo Soares Bôaventura

Comparação do desempenho de ambientes virtuais na computação em nuvem privada usando a análise estatística e o *benchmark* Hadoop.

Tese de doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Uberlândia como requisito para a obtenção do título de Doutor em Ciências.

Área de concentração: Processamento da Informação

Orientador: Professor Dr. Keiji Yamanaka

Banca Examinadora:

Prof. Dr. Keiji Yamanaka – FEELT/UFU
Prof. Dr. Fábio Luciano Verdi – UFSCar
Prof. Dra. Mônica Rocha Ferreira de Oliveira – UEMG
Prof. Dr. Lásaro Jonas Carmargos – FACOM/UFU
Prof. Dr. Edmilson Rodrigues Pinto – FAMAT/UFU

Uberlândia, 06 de Março de 2015

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

B662c 2015

Bôaventura, Ricardo Soares, 1980-

Comparação do desempenho de ambientes virtuais na computação em nuvem privada usando a análise estatística e o benchmark Hadoop / Ricardo Soares Bôaventura. - 2015.

209 f. : il.

Orientador: Keiji Yamanaka.

Tese (doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.

Inclui bibliografia.

1. Engenharia elétrica - Teses. 2. Computação em nuvem - Teses. 3. Planejamento experimental - Teses. 4. Algoritmos de computador - Teses. I. Yamanaka, Keiji. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU: 621.3

Ricardo Soares Bôaventura

Comparação do desempenho de ambientes virtuais na computação em nuvem privada usando a análise estatística e o *benchmark* Hadoop.

Tese de doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Uberlândia como requisito para a obtenção do título de Doutor em Ciências.

Área de concentração: Processamento da Informação

Orientador: Professor Dr. Keiji Yamanaka

Uberlândia, 06 de março de 2015

_____

Prof. Dr. Keiji Yamanaka
Orientador

_____

Prof. Dr. Edgard Afonso Lamounier Junior
Coordenador do Programa de Pós-graduação

Aos meus pais Osmar Cid e Maria Lúcia, pelo amor, compreensão e carinho.

Ao meu irmão Lucas, por ter me dado atenção em todos os momentos.

À pessoa mais importante na minha vida que esteve ao meu lado em todos os momentos que compartilhamos. Eu te amo.

Agradecimentos

Agradeço primeiramente a Deus, por minha vida.

Aos meus queridos pais e meu irmão pela dedicação, apoio, amor e carinho em todos os momentos que necessitei de atenção.

Ao meu avô, as minhas tias, tios, primos e primas pela alegria que me proporciona na minha vida. Em especial as minhas avós Rosa e Ruth, que ainda torcem pelo meu sucesso pessoal e profissional.

Ao meu amor que esteve praticamente ao meu lado em todas as etapas desse trabalho, me apoiando nos momentos difíceis e alegres.

À todos os meus grandes amigos que me ajudaram com palavras amigas.

Aos meus amigos de laboratório que diretamente ou indiretamente, me ajudaram no desenvolvimento do projeto, com seus conhecimentos.

Principalmente ao meu orientador professor Keiji Yamanaka, pela grande amizade, paciência e orientação em todos os momentos da realização deste trabalho.

“Para se ter sucesso, é necessário amar de verdade o que se faz. Caso contrário, levando em conta apenas o lado racional, você simplesmente desiste. É o que acontece com a maioria das pessoas.”
(STEVE JOBS)

Resumo

A Computação em Nuvem surge como um novo paradigma dominante em sistemas distribuídos, sendo um modelo que permite usuários acessarem, sob demanda, um conjunto compartilhado de recursos computacionais que podem ser configuráveis, como: redes, servidores, armazenamento, aplicativos e serviços. Esses recursos podem ser rapidamente fornecidos com o mínimo de esforço de gestão ou interação de um fornecedor. Na Computação em Nuvem, a infraestrutura pode ser disponibilizada como serviço através da virtualização com o uso de hipervisores. A virtualização é um mecanismo que abstrai os recursos de *hardware* e de sistema de um dado sistema operacional. Esse tipo de tecnologia é utilizada em ambientes em nuvens através de um grande conjunto de servidores, usando monitores de máquinas virtuais que estão localizadas entre o *hardware* e o sistema operacional. No entanto, existe uma grande disseminação de hipervisores, cada um com a suas próprias vantagens e desvantagens. As características específicas de cada máquina virtual permitem existir desempenhos diferentes. O objetivo do trabalho é propor uma metodologia que busca-se descobrir como, quando e quanto o aumento do desempenho dos algoritmos em ambientes virtuais é determinado pela configuração do ambiente e como os parâmetros de configuração podem influenciar-se mutuamente, e por fim, descobrir através de métodos estatísticos qual configuração de ambiente virtual obteve os melhores resultados em média. Os algoritmos testados (*sudoku, pi, wordcount, testDFSIO read e testDFSIO write*) pertencem ao *benchmark* do *Apache Hadoop*. Esses experimentos foram planejados e executados tendo como base a teoria de planejamento experimental. O planejamento experimental representa um conjunto de ensaios pré-estabelecidos usando critérios científicos e principalmente estatísticos, com o objetivo de determinar a influência de diversos fatores nos resultados (métricas) de um sistema ou processo, identificando e observando as razões que ocasionaram alteração do valor esperado. O planejamento utilizado foi o planejamento fatorial 3^4, onde cada fator (núcleo, memória, sistema operacional e máquina virtual) foram variados em três níveis. Os sistemas operacionais testados foram o Ubuntu 14.04 64bits, CentOS 7.0 64bits e Windows 8.0 64bits; e as máquinas virtuais testadas foram o KVM, Xen e VMware. Os resultados foram coletados e analisados utilizando análise de variância. Os resultados mostram que os fatores principais analisados alteram o desempenho de um algoritmo, porém eles não podem ser analisados separadamente pois existem interações que também são significativas, as quais, esses fatores pertencem. A um

nível de significância de 5%, a análise de variância mostrou que as interações núcleo:memória, memória:SO, memória:VM e SO:VM juntas, impactaram o tempo de execução dos algoritmos analisados. Segundo o método estatístico de comparação de médias, foi possível então fazer uma comparação entre as médias dos tempos da interação significativa SO:VM e com base nos resultados encontrados foi aplicado uma adaptação da teoria de dominância de Pareto denominada "dominância estatística de Pareto". E também, a um nível de 5% de significância foi possível descobrir as fronteiras de Pareto. Levando em consideração o tempo de execução do algoritmo, a dominância de Pareto apresentou o ambiente virtual Xen:CentOS na primeira fronteira como o ambiente virtual que em média obteve os melhores desempenhos computacionais para os algoritmos analisados. Os ambientes virtuais que ocuparam a segunda fronteira foram os ambientes Xen:Ubuntu e VMware:CentOS, ou seja eles tiveram em média tempos inferiores à primeira fronteira e entre si eles foram considerados equivalentes. Os ambientes pertencentes à terceira fronteira foram KVM:Ubuntu, VMware:Ubuntu e VMware: Windows. Os ambientes pertencentes à quarta fronteira foram Xen:Windows, KVM:CentOS e o ambiente que obteve em média tempos inferiores aos demais foi o KVM:Windows. Pode-se concluir que a máquina virtual Xen e o sistema operacional CentOS, em média, obtiveram os melhores desempenhos. Porém, se o usuário quiser utilizar o sistema operacional Ubuntu, aconselha-se instalá-lo na máquina virtual Xen. E caso o usuário deseje usar o sistema operacional Windows, aconselha ser instalado sobre a máquina virtual VMware.

Palavras-chave

Virtualização, Computação em Nuvem, Nuvem Privada, Planejamento Experimental, Experimentos com Algoritmos, Dominância de Pareto, Análise de Variância

Abstract

Cloud computing emerges as a new dominant paradigm in distributed systems, with a model that allows users to access, over demand, to a shared pool of computing configurable resources, such as networks, servers, storage, applications and services. These resources can be rapidly provided with minimal management effort or interaction from a supplier. In cloud computing, the infrastructure can be made available as a service through virtualization using hypervisors. Virtualization is a mechanism that presents the hardware and system resources of a given operating system. This technology is used in environments clouds through a large set of server using virtual machine monitors that are located between the hardware and the operating system. However, there is a wide spread of hypervisors, each with its own advantages and disadvantages. The specific characteristics of each virtual machine generates different performances. The aim of this work is to propose a methodology that seeks to discover how, when and as the increased performance of the algorithms in virtual environments is determined by the environment configuration and how the configuration parameters can influence each other, and finally, discover using statistical methods which settings of virtual environment achieve the best results on average. The tested algorithms (sudoku, pi, wordcount, testDFSIO read and write testDFSIO) belong to the benchmark Apache Hadoop. These experiments were planned and executed based on the experimental design theory. The experimental design is a pre-established set of tests using scientific and statistical criteria mainly, in order to determine the influence of various factors on the results (metric) of a system or process, identifying and observing the reasons that led to change in the expected value. The planning that was used is factorial planning 34, where each factor (core, memory, operating system and virtual machine) were varied in three levels. Tested operating systems were Ubuntu 14.04 64bit, CentOS 7.0 64bit and Windows 8.0 64bit; and virtual machines were tested KVM, Xen and VMware. Data were collected and analyzed using analysis of variance. The results show that the major analyzed factors changes the algorithm performance , but they can not be analyzed separately because there are also significant interactions belonging to these factors . At a 5% significance level, analysis of variance showed that the core interactions: memory, memory with OS, memory with VM and OS with VM, all these factors impact the runtime of the analyzed algorithms. According to the statistical method mean comparison was possible then make a comparison between the mean times of significant interaction between OS and VM, and based on results has been applied an adaptation of Pareto dominance theory called Pareto dominance. Also, with 5% significance level was possible to discover Pareto's borders. Considering the runtime algorithm, the Pareto Dominance introduced the virtual environment Xen with CentOS in the first border as the virtual environment that on average achieved the best performance for the analyzed computational algorithms. Virtual environments that occupied the second border were the environments Xen with Ubuntu and VMware with CentOS, ie they had on average

lower times the first border and between them they were considered equivalent. The environments belonging to third border were KVM with Ubuntu, VMware with VMware and Ubuntu with Windows. The environments belonging to fourth border were Xen with Windows, KVM with CentOS and the environment that got on average lower than the other times was the KVM with Windows. It can be concluded that virtual machine Xen and CentOS operating system on average got the best performance. But if the user wants to use the Ubuntu operating system it is advisable to install it in Xen virtual machine. And if you want to use the Windows operating system recommends be installed on the VMware virtual machine.

Keywords

Virtualization, Cloud Computing, Private Cloud, Experimental Planning, Experiments with Algorithms, Pareto Dominance, Analysis of Variance

Conteúdo

Lista de Abreviaturas

ANOVA	- *Analysis of variance*
API	- *Application Programming Interface*
BSD	- *Berkeley Software Distribution*
CMA-ES	- *Covariance Matrix Adaptation Evolution Strategy*
CPU	- *Central Processing Unit*
CMV	- *C Virtual Machine*
EC2	- *Elastic Compute Cloud*
GPL	- *General Public License*
IaaS	- *Infrastructure as a Service*
I/O	- *Input/Output*
LXC	- *LinuX Containers*
KVM	- *Kernel Virtual Machine*
MPIS	- *Milhões de Instruções Por Segundo*
PaaS	- *Platform as a Service*
RAM	- *Random Access Memory*
SaaS	- *Software as a Service*
SAPS	- *Scaling And Probabilistic Smoothing*
SLA	- *Service Level Agreements*
SMP	- *MultiProcessamento Simétrico*
SO	- *Sistema Operacional*

SPEC - *Standard Performance Evaluation Corporation*

S3 - *Simple Storage Service*

TI - Tecnologia da Informação

vCPU - *Virtual CPU*

VMM - *Virtual Machine Monitor*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A evolução da Internet vem provocando uma mudança permanente na forma em que os sistemas computacionais estão sendo estruturados. Sabe-se que, atualmente, que o acesso à Internet é possível através de uma variedade de dispositivos, em qualquer localização geográfica, com um nível de segurança e velocidade cada vez mais elevado. Junto com essa grande evolução, vem sendo estudada a Computação em Nuvem, que tem como principal característica a transformação dos modelos tradicionais de como empresas utilizam e adquirem os recursos da Tecnologia da Informação (FERNANDEZ, ET AL., 2011).

Computação em Nuvem emergiu como um novo paradigma, onde *hardware* e *software* são entregues como serviços de utilidade geral e disponibilizados para os usuários através da Internet. Graças à camada de virtualização, existente sobre os recursos computacionais físicos, é possível otimizar o uso da infraestrutura e, principalmente, oferecer diferentes tipos de *hardware* e *software* aos usuários da nuvem. Dessa maneira, a Computação em Nuvem permite que os recursos (serviços) sejam alugados e liberados de acordo com a necessidade dos usuários na forma e serviços tarifados por meio do modelo "pague-pelo-uso" (ZHANG, ET AL., 2010).

O crescimento do mercado de computação na nuvem é decorrente das vantagens em termos de custo e contabilidade na utilização de seus serviços em comparação aos serviços tradicionais, que consistem na aquisição de infraestruturas próprias (ARMBRUST, ET AL., 2010). Vários estudos vêm mostrando o grande aumento do uso da Computação em Nuvem.

Segundo um estudo feito pela AT&T com 500 executivos da área de Tecnologia da Informação (TI) dos Estados Unidos, 76% das empresas entrevistadas utilizavam a Computação em Nuvem ou tinham planos de investir em serviços do setor no ano de 2012. Além disso, 62% já incluíram serviços de nuvem em sua infraestrutura, o que representava um crescimento de 11% em relação ao ano anterior; 66% das empresas utilizam ou consideram o uso de serviços de nuvem para reforçar suas estratégias de continuidade e 49% das companhias pretendem aproveitar a Computação em Nuvem para armazenar dados (AT&T, 2012). Já no Brasil, pesquisa realizada pela *Kelton Research* mostrou que 74% das empresas brasileiras disseram que já utilizam algum tipo de serviço em *cloud computing* (BOUÇAS, 2012).

Numa pesquisa encomendada pelo Grupo Educacional Impacta Tecnologia à empresa *MBI Mayer & Bunge Informática*, realizada entre os anos de 2010 a 2013 foram apresentados resultados interessantes como: um crescimento de 50% no uso de armazenamento como serviço (*storage*) pelas empresas; infraestrutura como serviço apresenta crescimento de 44,6% para 48,3% e *software* como serviço apresenta aumento de 53,2% para 58,3%; teste de *software* como serviço de 21,6% para 32,5% e integração de processos como serviço de 15,1% para 23,2% no período. Plataforma como serviço e segurança como serviço permaneceram praticamente estagnados, com aumentos percentuais de 26,6% para 27,2% e de 13,7% para 13,9%, respectivamente (MBI MAYER&BUNGE, 2011).

Esses dados têm justificado o desenvolvimento de pesquisas que envolvem *softwares*, plataformas e infraestruturas em Computação em Nuvem. Os 3 cenários principais de gerenciamento de serviços de Computação em Nuvem: (i) *software* como serviço, que oferecem programas de interesse de uma ampla variedade de usuários; (ii) plataforma como serviço, onde são desenvolvido os serviços em que o dimensionamento de recursos de *hardware* é feito de forma transparente; e (iii) infraestrutura como serviço, que possibilita o gerenciamento de um conjunto de recursos computacionais como armazenamento e capacidade de processamento que, através da virtualização, são capazes de serem redimensionados (VAQUERO, ET AL., 2009).

Os três cenários apresentam trabalhos de pesquisa em diferentes áreas: *software* como serviço (YOUSEFF, ET AL., 2008; KELLER, ET AL., 2010; ERCAN, 2010; CALHEIROS, ET AL., 2011), plataforma como serviço (BUYYA, ET AL., 2008; PENG, ET AL., 2009; VECCHIOLA, ET AL., 2009; KELLER, ET AL., 2010; BONIFACE, ET AL., 2010; TIAN, ET AL., 2010), infraestrutura como serviço (NURMI, ET AL., 2009; SOTOMAYOR, ET AL.,

2009; BRENDEL, 2010; BHARDWAJ, ET AL., 2010; RODERO-MERINOA, ET AL., 2010; KHAJEH-HOSSEINI, ET AL., 2010; GHOSH, ET AL., 2010; LONGO, ET AL., 2011).

1.1 Histórico

Aliada ao surgimento da Internet e a redução dos custos de interligação das redes de computadores, o avanço da padronização de protocolos de comunicação possibilitaram interligar redes de diferentes organizações (VERAS, 2011). Ao mesmo tempo, a capacidade de processamento dos computadores pessoais aumentou substancialmente, devido ao desenvolvimento de novos *hardwares* e às novas técnicas de programação. Entretanto, toda a capacidade adquirida não tem sido muito bem aproveitada. Com isso, em boa parte do tempo, ocorre um grande desperdício de recursos computacionais por não serem utilizados de forma correta e desejada (MERGEN, ET AL., 2006).

Baseado nesse problema, a virtualização surgiu com um novo conceito de utilização de máquinas de grande porte, definindo uma técnica que possibilita a divisão de um mesmo *hardware* em diversas máquinas virtuais (POPEK, ET AL., 1974; SOUZA, 2006). A virtualização foi utilizada nos *mainframes* e, atualmente, com a evolução de recursos computacionais (memória e armazenamento), permitiu também que os servidores de pequeno porte e computadores pessoais tivessem condições de utilizar dessa tecnologia (SOUZA, 2006).

Com a virtualização pode-se adequar o *hardware* à carga de trabalho originada pela aplicação. Ou seja, se uma determinada aplicação demandar um grande recurso computacional pode-se, então, alterar o recurso computacional do *datacenter* dinamicamente. Em outro momento, pode-se novamente restabelecer a configuração original ou até expandir o uso do recurso. Com isso, a virtualização passa a ser a base para a infraestrutura, pois permite alterá-la rapidamente utilizando instrumentos lógicos e não físicos e, ao mesmo tempo, tornando as aplicações independentes dos *hardwares* (VERAS, 2011).

Dentro desse cenário surge a Computação em Nuvem como um novo paradigma para o fornecimento de infraestrutura de computação disponibilizada na rede, reduzindo os custos associados com a gestão de recursos de *hardware* e de *software* (HAYES, 2008; VAQUERO, ET AL., 2009; ARMBRUST, ET AL., 2010). Porém, segundo (ZHANG, ET AL., 2009), não existe um padrão para definição e especificação de uma Computação em Nuvem, mas esse paradigma se baseia em décadas de pesquisa em virtualização, computação distribuída,

computação utilitária, serviços, Web, redes de computadores e *softwares* (FOSTER, ET AL., 2008; MICROSYSTEMS, 2009; VAQUERO, ET AL., 2009; VOUK, 2008).

1.2 Justificativa e problema

Como não existe um padrão para a criação de ambientes virtuais em Computação em Nuvem devido a uma grande disseminação de sistemas operacionais e máquina virtuais. E cada máquina virtual e sistema operacional possui suas respectivas vantagens e desvantagens, levam a desempenho diferente para cada ambiente computacional. O desempenho de um algoritmo em um ambiente virtual ou físico varia devido a diversos fatores, em que, um fator pode ser considerado uma variável independente, ou seja, uma variável que pode ser manipulada em um determinado experimento cuja sua presença ou nível pode determinar a mudança do resultado final (PAIS, 2014).

Segundo Ziviani, (2008) o que afeta o tempo de processamento de um algoritmo é o *hardware,* compilador e a quantidade de memória. Já para Barr et a. (1995), as variáveis que afetam o desempenho de um algoritmo são o tipo de processador, tamanho de memória, frequência do *clock,* sistema operacional, linguagem de programação, compilador, programas em execução em segundo plano e a habilidade do programador.

Caso então exista o interesse em estudar a influência desses fatores no desempenho de um determinado algoritmo, sejam eles executados em ambientes físicos ou virtuais, é necessário então executar experimentos que em algumas mudanças propositais são feitas nesses fatores, de modo a observar e identificar quais foram as razões que ocasionaram a alteração no desempenho do algoritmo. Existem diversas metodologias que podem ser adotadas para guiar toda essa experimentação, porém, para a realização dos experimentos e análise dos resultados qual seria a melhor metodologia a ser adotada? Existe alguma metodologia que seja padrão? Como provar então que um algoritmo é melhor que outro? Será que comparar resultados médios é o suficiente? (PAIS, 2014).

A estatística pode ajudar a responder todas as questões que foram elencadas anteriormente e outras de forma lógica e racional. Com a estatística, o experimentador pode planejar todo o experimento para que possa extrair conclusões lógicas e válidas e que a análise possa ser feita de forma direta (NETO ET AL. 2010).

O planejamento experimental representa um conjunto de ensaios estabelecidos com critérios científicos e estatísticos, com o objetivo de determinar a influência de diversas variáveis nos resultados de um sistema ou processo (Montgomery, 2009).

1.3 Objetivo Geral

O objetivo do trabalho é propor uma metodologia que busca descobrir como, quando e quanto o aumento do desempenho dos algoritmos em ambientes virtuais é determinado pela configuração do ambiente e como os parâmetros de configuração podem influenciar-se mutuamente, e por fim, descobrir através de métodos estatísticos qual configuração de ambiente virtual obteve os melhores resultados em média. Essa metodologia foi elaborada baseada no planejamento experimental estatístico, teste de comparação de médias e dominância de pareto visando estudar a influência da quantidade de núcleo, quantidade de memória, tipo de sistema operacional e tipo de máquina virtual na computação em nuvem.

1.4 Contribuições do trabalho

As principais contribuições do trabalho estão descritas a seguir:

- Uma revisão bibliográfica de trabalhos publicados sobre metodologias de experimentação em algoritmos;
- Uma metodologia para a avaliação da influência de fatores no desempenho de um algoritmo em um ambiente virtual, onde o planejamento fatorial 3^k é aplicado para a estimação dos efeitos e de suas interações;
- Realização de um estudo de caso utilizando o *Apache Hadoop*, para a aplicação do método proposto no estudo para estimar a influência dos parâmetros do tipo de máquina virtual em um ambiente virtualizado para a resolução de problemas;

1.5 Metodologia

O desenvolvimento deste trabalho está dividido nas seguintes etapas:

- Revisão bibliográfica abrangendo os seguintes tópicos: virtualização, complexidade de algoritmos, planejamento experimental, experimentos com algoritmos, Computação em Nuvem;

- Definição da metodologia de planejamento experimental na análise de algoritmos em ambientes virtuais na Computação em Nuvem;
- Implementação de uma nuvem privada utilizando o *OpenStack* para ser utilizada para criação dos ambientes virtuais;
- Estudo de casos: aplicação da metodologia proposta em diferentes algoritmos do *Apache Hadoop*;
- Análise dos dados obtidos;
- Conclusão da metodologia;

1.6 Organização do Trabalho

A tese está organizada da seguinte maneira. O Capítulo 1 apresenta uma introdução sobre a área de estudo, objetivos do trabalho, contribuições e as etapas da metodologia. No Capítulo 2 são apresentados a definição de Computação em Nuvem, os modelos de Computação em Nuvem e as suas implementações; plataformas de IaaS, comparação entre essas plataformas. E por fim, definições de virtualização; as formas de virtualização; os tipos de virtualização; e os ambientes para a construção de máquinas virtuais.

No Capítulo 3 são apresentados a definição de planejamento de experimentos; as técnicas existentes para a definição das sequências dos ensaios; o processo para a condução de um experimento; e a técnica de planejamento fatorial.

O Capítulo 4 apresenta especificamente o referencial teórico sobre a experimentação com algoritmos. O Capítulo 5 apresenta a metodologia proposta para a comparação de ambientes virtuais na computação em nuvem. O Capítulo 6 apresenta a configuração do estudo de caso baseado na metodologia proposta. O Capítulo 7 apresenta as análises dos experimentos e apresentação de resultados. E por fim, o Capítulo 8 apresenta as considerações finais e trabalhos futuros.

Capítulo 2

Computação em Nuvem

Desde o seu surgimento, existe pouco consenso sobre a definição de Computação em Nuvem.

Para Buyya (1999), a Computação em Nuvem é um sistema distribuído e paralelo, constituído por uma coleção de computadores, virtuais ou físicos, interconectados, que fornecem um ou mais recursos computacionais, baseados em acordos de níveis de serviços pré-estabelecidos por meio de uma negociação entre o consumidor e o prestador de serviços.

Segundo Foster et al. (2008), Computação em Nuvem é um paradigma de computação em larga escala, que possui o foco de economia de escala, em que um conjunto abstrato, virtualizado, é dinamicamente escalável em relação ao poder de processamento, armazenamento, plataformas e serviços disponibilizados por demanda para os clientes externos via Internet.

Bhardwaj et al. (2010) define a Computação em Nuvem como uma capacidade de computação que fornece uma abstração entre os recursos computacionais e sua arquitetura subjacente (por exemplo, servidores, armazenamento, redes), permitindo o acesso sob-demanda a um *pool* compartilhado de recursos computacionais configuráveis, que podem ser rapidamente provisionados e lançados com o mínimo esforço de gestão ou interação do prestador de serviços.

Já para Vaquero et al. (2009), Computação em Nuvem é um conjunto de recursos virtuais facilmente utilizáveis e acessíveis tais como *hardware*, *software*, plataformas de desenvolvimento e serviços. Esses serviços podem ser dinamicamente reconfiguráveis para se ajustarem a uma carga de trabalho variável, permitindo a otimização de uso de recursos. Esses recursos são tipicamente explorados através de um modelo pague-pelo-uso com garantias oferecidas pelo provedor por meio de acordos de níveis de serviços.

A definição de Computação em Nuvem, utilizada nesse trabalho, é aquela apresentada por Foster et al. (2008), em combinação com Buyya (1999), Vaquero et al. (2009) e Bhardwaj et al. (2010), isto é, a Computação em Nuvem é um paradigma computacional distribuído e/ou paralelo, constituído de computadores virtuais ou físicos, com o foco em economizar escala, permitindo o acesso à rede *on-demand* através de um *pool* compartilhado de recursos computacionais (servidores, redes, armazenamentos, etc) configuráveis, que podem ser rapidamente provisionados e lançados com o mínimo esforço de gestão ou com a interação do prestador de serviços. Esses recursos são tipicamente explorados através de um modelo pague-pelo-uso com garantias oferecidas pelo provedor por meio de acordos de níveis de serviços.

2.1 Tecnologias relacionadas

Para Zhang et al. (2010), cada uma das tecnologias abaixo apresenta um aspecto da Computação em Nuvem.

2.1.1 Computação em grade

Computação em grade é um paradigma de computação distribuída que coordena os recursos da rede para atingir um objetivo computacional comum. O desenvolvimento da computação em grade foi originalmente impulsionado por aplicações científicas que são, geralmente, computação intensiva. A Computação em Nuvem é semelhante à computação em grade na medida em que também emprega recursos distribuídos para atingir os objetivos em relação à camada de aplicativo. No entanto, a Computação em Nuvem aproveita as tecnologias de virtualização em vários níveis (*hardware* e plataforma de aplicativos) para realizar o compartilhamento de recursos e provisionamento de recursos dinâmicos.

2.1.2 Computação utilitária

A computação utilitária representa o modelo de fornecimento de recursos sob demanda, na qual a cobrança é feita aos clientes com base no uso, em vez de uma taxa fixa. A Computação

em Nuvem pode ser percebida como uma realização da computação utilitária. A Computação em Nuvem adota um regime de preços baseado em utilidade inteiramente por razões econômicas. Com provisionamento de recursos sob demanda e preços baseados pelo uso, prestadores de serviços podem realmente maximizar recursos utilizados e minimizar os seus custos operacionais.

2.1.3 Virtualização

A virtualização é uma tecnologia que abstrai os detalhes de *hardware* físico e fornece recursos virtualizados para aplicações de alto nível. Um servidor virtualizado é normalmente chamado de máquina virtual. Virtualização constitui a base da computação, uma vez que fornece a capacidade de compartilhar recursos computacionais a partir de *clusters* de servidores e atribui dinamicamente recursos virtuais de aplicativos sob demanda.

A tecnologia de virtualização será apresentada em detalhes na Seção 2.9.

2.1.4 Computação autônoma

A computação autônoma visa à construção de sistemas de computação capazes de autogestão, ou seja, capazes de reagir às observações internas e externas, sem intervenção humana. O objetivo da computação autônoma é superar a complexidade dos sistemas atuais de computador de gestão. Embora a Computação em Nuvem apresente certas características autônomas, tais como provisionamento de recursos automáticos, seu objetivo é reduzir o custo dos recursos ao invés de reduzir a complexidade do sistema.

2.2 Características da Computação em Nuvem

A Computação em Nuvem deve apresentar características essenciais como (VAQUERO et Al., 2009):

- *Serviço sob demanda*: funcionalidades computacionais são providas automaticamente sem a interação humana;

- *Amplo acesso aos serviços via rede*: serviços disponíveis através da Internet acessados via mecanismos padronizados para serem utilizados por quaisquer dispositivos (por exemplo, clientes leves, telefones celulares, laptops, e PDAs) ;
- *Pooling de recursos*: os recursos são utilizados para servir múltiplos usuários, sendo alocados e realocados dinamicamente conforme a demanda do usuário;
- *Elasticidade rápida*: as funcionalidades computacionais devem ser rapidamente liberadas. Os usuários devem pensar que a nuvem possui recursos ilimitados adquiridos em qualquer quantidade e a qualquer momento;
- *Medição de serviços*: monitoramento automático de recursos para cada serviço (por exemplo: armazenamento, processamento, largura de banda ou contas de usuário ativas);
- *Virtualização*: tecnologia que ajuda na elasticidade rápida, criando várias instâncias dos recursos solicitados usando um único recurso real;
- *Service Level Agreements – SLA*: contrato que fornece informações sobre os níveis de disponibilidade, funcionalidade, desempenho e outros atributos como: faturamento e até mesmo penalidades em caso de violação.

2.3 Arquitetura

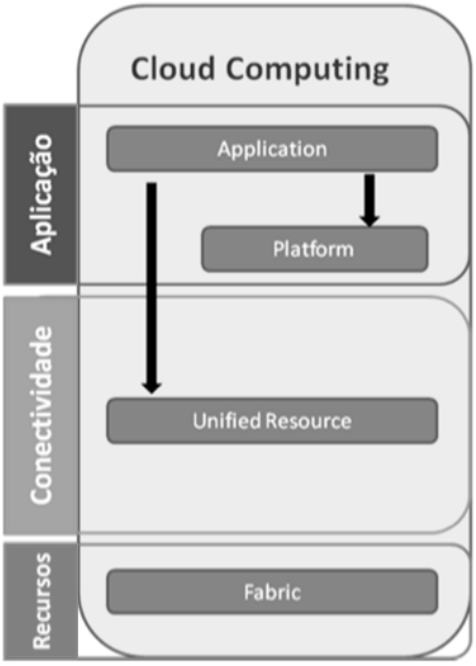
Conforme Foster et al. (2008) e Sun Microsystems (2009), existem também várias versões de definição de uma arquitetura para a Computação em Nuvem; portanto, pode-se definir uma arquitetura de quatro camadas, é representada na (Figura 1), em comparação com a arquitetura da computação em grade, composto de camada de substrato (*Fabric*), de recurso unificado (*Unified Resource*); da plataforma (*Platform*) e aplicação (*Application*).

A camada denominada substrato contém os recursos de *hardware*, como processadores, armazenamento e rede. A camada de recurso unificado contém recursos que foram abstraídos ou encapsulados (geralmente por virtualização), de modo que eles podem estar expostos à camada superior e para os usuários finais como recursos integrados como, por exemplo, um computador e/ou *cluster* virtual, um sistema de arquivos lógico, um sistema de banco de dados, etc.

A camada de plataforma acrescenta uma coleção de ferramentas especializadas, um *middleware* e serviços na camada dos recursos unificados, para fornecer um ambiente de

desenvolvimento e / ou plataforma de implementação. Finalmente, a camada de aplicação contém os aplicativos que são executados em nuvem.

Figura 1: Arquitetura geral de uma Computação em Nuvem.



Fonte: (Foster, et al., 2008)

2.4 Modelo de Computação em Nuvem

A Computação em Nuvem disponibiliza os recursos computacionais sob a forma de serviços, podendo estes serem divididos em três modelos:

2.4.1 SaaS - Software como serviço

Software as a Service (SaaS) oferece, para fins especiais, um *software* que é acessível remotamente pelos consumidores através da Internet, com um modelo na forma de pagamento baseada no uso (FOSTER, ET AL., 2008).

Para Bharowaj et al. (2010), o *software* como serviço é a possiblidade de oferecer um conjunto organizado de programas (rodando em uma plataforma e infraestrutura) em que o usuário não o possui, mas pode pagar por algum elemento de utilização. Neste modelo o usuário não precisa fazer qualquer desenvolvimento ou programação, mas poderá configurar e/ou

personalizar de forma flexível o *software*. Não é necessário, também, comprar *software*, mas pagar somente pelo que é utilizado.

O provedor de SaaS normalmente hospeda e gerencia uma determinada aplicação em seu próprio servidor tornando-a disponível para vários clientes e usuários pela Web.

2.4.2 PaaS - Plataforma como serviço

O modelo de plataforma como serviço (PaaS) fornece sistema operacional, linguagens de programação e ambientes de desenvolvimento para o desenvolvedor de aplicações, auxiliando a implementação de sistemas de *software* que serão disponibilizadas na nuvem (VERAS, 2009). Neste caso, o dimensionamento dos recursos de *hardware*, exigidos pela execução dos serviços, é feito de forma transparente (ARMBRUST, ET AL., 2010).

Segundo Foster et al. (2008), a plataforma como serviço oferece um ambiente integrado de alto nível para construção, testes e implantação de aplicações. Baseados nisso, os programadores terão de aceitar algumas restrições sobre o tipo de *software* que podem desenvolver em troca da escalabilidade da aplicação.

2.4.3 IaaS - Infraestrutura como serviço

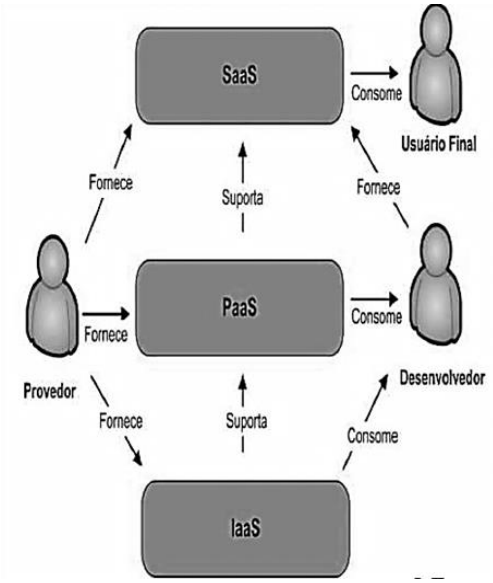
O modelo de infraestrutura como serviço IaaS é a disponibilização, em forma de serviço, por parte de um fornecedor, de uma infraestrutura de processamento e armazenamento de forma transparente. Nesse modelo, o cliente não tem o controle da infraestrutura física, porém, através de virtualização, o usuário possui o controle sobre os sistemas operacionais, armazenamento, aplicações instaladas e, possivelmente, um controle limitado dos recursos de rede (VERAS, 2009; VAQUERO, ET AL., 2009)

A infraestrutura como serviço torna mais fácil e mais acessível o fornecimento de recursos computacionais necessários como *hardware* (servidores, armazenamento e rede) associados com *software* (tecnologia de virtualização de sistemas operacionais, sistema de arquivos) necessários para construir um ambiente de aplicação sob demanda (BHARDWAJ, ET AL., 2010).

2.5 Funcionamento dos modelos

Do ponto de vista de interação entre as três categorias, a IaaS fornece recursos computacionais, seja de *hardware* ou *software*, para a PaaS que, por sua vez, fornece recursos, tecnologias e ferramentas para o desenvolvimento e execução dos serviços implementados, a serem disponibilizados na visão de SaaS (SOUSA ET AL., 2009). Essa relação é mostrada na Figura 2.

Figura 2: Relacionamento entre os modelos de *cloud computing*.



Fonte: (Sousa et al., 2009)

2.6 Tipos de Implementação de Computação em Nuvem

Existem três modelos de implementação de uma nuvem (VERAS, 2009; JAMSA, 2013):

- *Nuvem privada*: infraestrutura que é composta e quase sempre gerenciada pela organização cliente. Os serviços são oferecidos para serem utilizados internamente pela própria organização, não estando publicamente disponíveis para o uso geral. Uma nuvem privada oferece maior controle a um custo maior.
- *Nuvem pública*: tipo de nuvem disponibilizada publicamente através do modelo pague-pelo-uso. São oferecidas por organizações públicas ou por grandes grupos industriais

que possuem grande capacidade de processamento e armazenamento. Uma nuvem pública é geralmente uma solução menos dispendiosa.

- *Nuvem híbrida*: é uma infraestrutura composta de duas ou mais nuvens (privadas ou públicas) que ainda são entidades únicas e que possuem tecnologias proprietárias ou padronizadas, propiciando a portabilidade de dados e aplicações.

2.7 Plataformas de gerenciamento de Computação em Nuvem

Para o desenvolvimento de um sistema em nuvem no modelo de infraestrutura como serviço existem ferramentas que permitem o desenvolvimento e o gerenciamento de uma nuvem privada. Essas ferramentas também facilitam a manutenção, a administração e o gerenciamento de servidores em um ambiente. A seguir são descritas algumas ferramentas de gerenciamento de nuvens:

2.7.1 Eucalyptus

O Eucalyptus é uma plataforma disponível sob GPL feito para criação e gestão de nuvem privada, podendo até mesmo fazer parte de uma nuvem híbrida. Essa plataforma fornece ferramentas necessárias para implementar computação na nuvem em *cluster* ou em um conjunto de estações de trabalho. Eucalyptus foi projetado desde o início para ser fácil de instalar. Sua estrutura de *software* é altamente modular.

Eucalyptus é a única ferramenta que proporciona uma sobreposição de tráfego de rede isolando as redes virtuais de usuários diferentes, e ainda permite que dois ou mais *clusters* pareçam pertencer à mesma rede local. O Eucalyptus fornece suporte à utilização de dois tipos de *hipervisor* como: KVM, XEN e VMware (NURMI, ET AL., 2009).

Segundo Endo et al. (2010), o projeto Eucalyptus apresenta quatro características que o diferenciam de outras soluções de Computação em Nuvem:

- Foi projetado para ser simples, sem a necessidade de recursos dedicados;
- Foi destinado a incentivar extensões de terceiros através da estrutura e *software* modular, mecanismos de comunicação de linguagem agnóstica;
- Interface externa baseada na API Amazon (Amazon EC2); e
- Fornece uma rede virtual de sobreposição em que o tráfego de rede entre grupos isolados apresenta ser parte da mesma rede local.

2.7.2 OpenNebula

O OpenNebula é um conjunto de ferramentas de código aberto usado para construir nuvens privadas, públicas e híbridas. Foi desenvolvido para ser integrado com uma rede juntamente com soluções de armazenamento datacenters. A estrutura do OpenNebula é composta por três tecnologias básicas, que permitem a disponibilização de serviços em uma infraestrutura distribuída composta por virtualização, armazenamento e rede (NURMI, ET AL., 2009).

OpenNebula oferece uma plataforma de nuvem escalável e segura para a entrega rápida e com elasticidade de recursos virtuais. As aplicações podem ser desenvolvidas em multicamadas, usando dispositivos virtuais já pré-configurados chamados de catálogos. Os catálogos disponíveis permitem: (i) armazenamento de imagens que podem ser usadas nas máquinas virtuais criadas; (ii) criação de uma rede para poder interligar as máquinas virtuais; (iii) criação de um catálogo de *templates* que poderão ser usados posteriormente para instanciação de novas máquinas virtuais; e (iv) criação de um catálogo de operações que podem ser controladas e migradas durante o ciclo de vida da máquina virtual (OPENNEBULA.ORG, 2013).

2.7.3 OpenStack

O OpenStack é um *software* de código aberto, projetado para fornecimento e gerenciamento em grande escala de redes de máquinas virtuais, criando uma plataforma de Computação em Nuvem redundante e escalável. O *software* fornece ferramentas de controle e APIs necessárias para gerenciar uma nuvem, incluindo instâncias em execução, gerenciamento de redes e controle de acesso através de usuários e projetos.

O OpenStack, atualmente, possui três principais projetos de *software*: (i) *OpenStack Compute* (Nova), que através de uma interface on-line ou via outros aplicações usando API, controla os recursos de computação; (ii) *OpenStack Object Storage* (Swift) que cria um armazenamento confiável, usando o *hardware* padrão do datacenter; e (iii) *OpenStack Image Service* (Glance) que é responsável por gerenciar e catalogar grandes bibliotecas de imagem de servidores (OPENSTACK.ORG, 2013).

Segundo OpenStack.org (2013), o sistema: (i) age de acordo com as políticas definidas pela Apache 2.0; (ii) suporta diversos hipervisores; (iii) implementa REST API e formato de imagens padrão; e (iv) se compromete conduzir e adotar padrões abertos.

2.7.4 OpenQRM

O OpenQRM é uma plataforma de gestão para o gerenciamento heterogêneo de infraestrutura de *datacenter*. Essa plataforma permite o desenvolvimento de nuvens públicas, privadas ou híbridas. O OpenQRM controla uma multiplicidade de tecnologias de armazenamento, de rede, de virtualização, de monitoramento e de implementações de segurança, para implantar serviços multicamadas, como máquinas virtuais, em infraestruturas distribuídas. Essa ferramenta combina recursos de *datacenter* e recursos de Computação em Nuvem remota, de acordo com as políticas de alocação.

A arquitetura do OpenQRM é totalmente automatizada. As operações do datacenter são baseadas em aplicações de implementação e acompanhamento (GOERKE, ET AL., 2010).

Segundo OpenQRM (2013), as características do OpenQRM são:

- Plataforma de Computação em Nuvem privada / híbrida;
- Gerenciamento dos sistemas de servidores físicos e virtualizados;
- Integração com todas as principais tecnologias de armazenamento aberto e comercial;
- Suporte de gerenciamento de sistemas Windows, Linux, OpenSolaris ou BSD;
- Suporta *hypervisores*: KVM, XEN, Citrix XenServer, VMware ESX (i), LXC, OpenVZ e VirtualBox;
- Suporta migração em P2V, P2P, V2P, V2V e alta disponibilidade;
- Integração com ferramentas de gestão de código aberto - como fantoche, nagios / Icinga ou collected;
- Mais de 50 *plugins* de recursos e integração com sua infraestrutura;
- Portal de autoatendimento para usuários finais.

2.7.5 ConVirt

O *ConVirt Open Source* é o principal produto de código aberto para gerenciamento de Xen e KVM, o que permite padronizar e gerenciar um ambiente virtualizado de forma centralizada. Com o ConVirt, o usuário pode criar e provisionar imagens, diagnosticar

problemas de desempenho e balanceamento de carga em todo o datacenter. Essas funcionalidades são acessadas via interface baseada na *Web* altamente interativa e com característica definida em todas as plataformas de virtualização de código aberto (CONVIRT, 2013).

O *ConVirt Enterprise Cloud* fornece virtualização juntamente com serviços em nuvem, transformando-se em uma infraestrutura em nuvens privadas e datacenters virtuais. Funcionalidades como: gerenciamento unificado e pools de servidores virtuais podem ser gerenciados a partir de um único console de gerenciamento. O *ConVirt Enterprise Cloud* também oferece capacidade de auto provisionamento, permitindo que os clientes gerenciem suas próprias nuvens privadas de forma independente.

2.8 Comparação entre as plataformas de Computação em Nuvem

A Tabela 1 apresenta um quadro comparativo entre as funcionalidades de plataformas como: hipervisores suportados, monitoramento, segurança, suporte, SLA, entre outras características. Para essa comparação não serão consideradas plataformas que oferecem *softwares* e plataformas como serviços, pois o objetivo do trabalho é desenvolver uma infraestrutura em Computação em Nuvem.

Pode-se verificar que todas as plataformas comparadas disponibilizam a infraestrutura como serviço (IaaS) e possuem o código aberto. Importante ressaltar que todas as plataformas apresentam uma interface *Web* para facilitar a interação com os clientes e disponibilizam um repositório para o gerenciamento das máquinas virtuais armazenadas. A plataforma OpenQRM é a única que apresenta um sistema de monitoramento, que permite uma alta escalabilidade e que suporta a construção de sistemas físicos. E, por fim, o OpenNebula das plataformas analisadas é a única que não apresenta suporte para o sistema operacional Windows.

2.9 Virtualização

A virtualização teve origem em meados das décadas de 60 e 70, período em que se estabeleceu um novo conceito de utilização de máquinas de grande porte, definindo uma técnica que possibilitava a divisão de um mesmo *hardware* em diversas máquinas virtuais (Popek, et al., 1974; Creasy, 1981). Com a evolução de recursos computacionais (memória,

armazenamento, entre outros), permitiu-se também que os servidores de pequeno porte e computadores pessoais tivessem condições de utilizar essa tecnologia (SOUZA, 2006).

Tabela 1: Comparação entre plataformas de Computação em Nuvem IaaS

Características	**OpenQRM**	**Eucalyptus**	**ConVirt**	**OpenStack**	**OpenNebula**
API	✓	✓	✓	✓	✓
Gerenciamento de configuração	✓	☒	?	✓	☒
Interface Web	✓	✓	✓	✓	✓
Flexibilidade de recursos	✓	?	☒	✓	?
Gerenciamento de armazenamento	✓	✓	✓	✓	✓
Gerenciamento de rede	Cloud-plugin	✓	✓	✓	✓
Monitoramento	✓	Nagios	✓	☒	
Recursos externos	✓	AWS API		AWS API	✓
Hipervisores suportados	Xen, KVM, VMware, vBox, ESX(i), openVz, LXC e xenServer	ESX(i), KVM e Xen	Xen, KVM	Hyper-V, Xen, xenServer, KVM, QEMU, LXC	Xen, KVM, VMware
Live migration	✓	☒	✓	✓	✓
Gerenciamento de usuários	✓	✓	✓	✓	✓
Suporta sistema físico	✓	☒	☒	☒	✓
Suporta GNU/Linux	✓	✓	✓	✓	✓
Suporte Windows	✓	✓	✓	✓	☒

✓: Sim ☒: Não ?: Informação indisponível

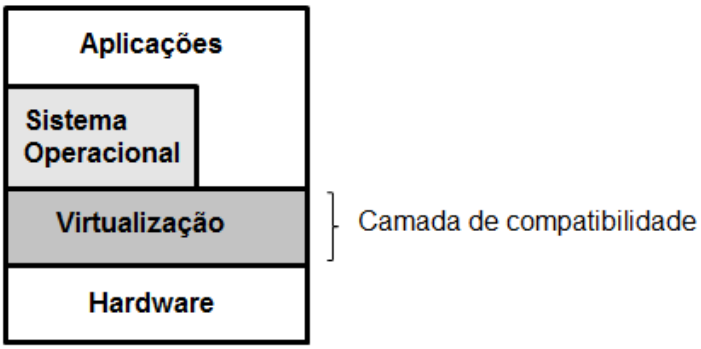
Fonte: adaptação de (Jagannathan, 2012; Machado, et al., 2011; Peng, et al., 2009)

A virtualização pode ser entendida como uma técnica que divide os recursos do computador em múltiplos ambientes de execução com o objetivo de implantar novas tecnologias, criando uma camada de abstração entre dispositivos físicos e aplicações à máquina na qual estão hospedados (FIGUEIREDO, R., ET AL., 2005; ROSE, 2004; NANDA, ET AL., 2005; MENASCÉ, 2005).

A camada de virtualização (Figura 3), também chamada de *hipervisor* ou monitor (VMM - *Virtual Machine Monitor*), molda todas as interfaces virtuais a partir da configuração

da plataforma real, proporcionando maior flexibilidade operacional e aumentando a taxa de utilização do *hardware* físico (IBM, 2007).

Figura 3: Localização da camada de virtualização.



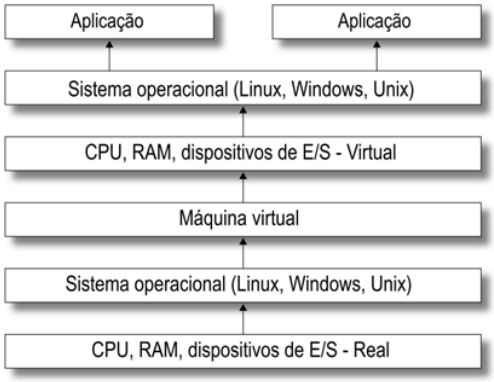
Fonte: (Laureano, 2006)

2.9.1 Categorias de Virtualização

Segundo (Rosenblum, et al., 1998), os *softwares* de virtualização podem ser divididos em três categorias:

- *nível de hardware*: a camada de virtualização está diretamente sobre a máquina física, apresentando-se às camadas superiores como um *hardware* virtual, igual ao original (Figura 4);

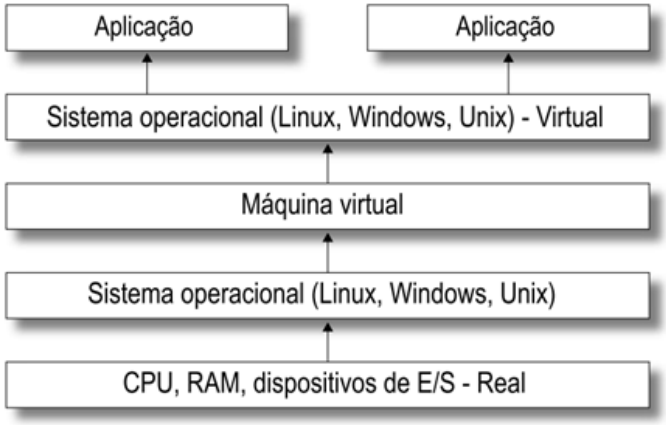
Figura 4 : Virtualização no nível de *hardware*



Fonte: (Laureano, 2006)

- *nível de sistema operacional*: a camada de virtualização está inserida entre o sistema operacional e a aplicação, permitindo que sejam criadas partições lógicas, de forma que cada partição é isolada e compartilha o mesmo sistema operacional (Figura 5);

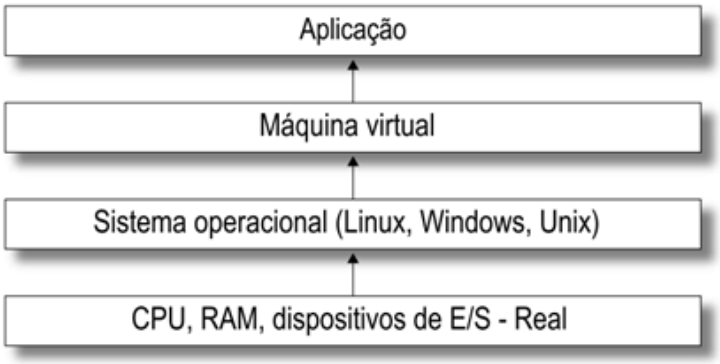
Figura 5: Virtualização no nível de sistema operacional



Fonte: (Laureano, 2006)

- *nível de linguagem de programação*: a camada de virtualização é um programa de aplicação do sistema operacional (Figura 6);

Figura 6: Virtualização no nível de linguagem de programação



Fonte: (Laureano, 2006)

2.9.2 Ferramentas de virtualização

Atualmente existem diversas ferramentas que permitem a criação de ambientes virtuais, porém cada uma possui as suas características e é desenvolvida sobre uma categoria de virtualização.

As Tabelas 2 e 3 apresentam uma comparação das características das ferramentas de virtualização, como criador, CPU suportada e emulada, SO hospedeiro e cliente, modo de virtualização e o tipo de licença.

Tabela 2: Características das ferramentas de virtualização (Parte 01)

Nome	**Criador**	**Arquitetura anfitriã**	**Arquitetura hóspede**	**Licença**
VMware (VMware, 2013)	VMware	i686, x86-64	x86, x86-64	Proprietária
Xen (Xen, 2013)	XenSource	x86-64, IA-64	Mesmo do anfitrião	GPL
KVM (KVM, 2013)	Red Hat	x86-64	Mesmo do anfitrião	GPL
OpenVZ (OpenVZ, 2013)	Parallels Inc.	x86-64	Mesmo do anfitrião	GPL
LXC (LXC, 2013)	lxc.sourceforge.net	x86-64	Mesmo do anfitrião	GPL
VirtualBox (VirtualBox, 2012)	Oracle Corporation	x86-64, IA-64, PowerPC 64	x86-64	GPL Proprietária
XenServer (XenServer, 2013)	Citrix	x86-64, IA-64	Mesmo do anfitrião	Proprietária Edição livre

Tabela 3: Características das ferramentas de virtualização (Parte 02)

Nome	**SO anfitrião**	**SO hospedeiro**	**Método de virtualização**
VMware	Windows, Linux, Mac OS X	Windows Family, Linux Family, Solaris, FreeBSD e NetWare	Virtualização total
Xen	RedHat, Novel, Debian, Fedora, FreeBSD, Ubuntu, OpenSUSE, CentOS	FreeBSD, NetBSD, OpenSolaris, CentOS, Debian, Ubuntu, OpenSUSE, Fedora, NetWare, Plan 9, Windows	Virtualização total Paravirtualização
KVM	FreeBSD, Linux, Illumos	FreeBSD, Linux Family, Solaris, Windows Family, Plan 9	Virtualização total
OpenVZ	Linux: CentOS, Debian, Fedora, OpenSUSE e Ubuntu	Linux: CentOS, Debian, Fedora, OpenSUSE e Ubuntu	Virtualização de software
LXC	Linux: ArchLinux, Debian, Fedora, Gentoo, OpenSUSE e Ubuntu	Linux: ArchLinux, Debian, Fedora, Gentoo, OpenSUSE e Ubuntu	Virtualização de software
VirtualBox	Windows, Linux, Mac OSX x86, Solaris	DOS, Linux (2.4 e 2.6), Windows (todas versões), Solaris 10 e 11, FreeeBSD, OpenBSD, OS/2, Mac OS X	Virtualização total Virtualização assistida por hardware
XenServer	Windows, Linux	Windows 2003, 2008, 7 (64-bit), Windows XP, 2000, Vista (32-bit), Windows 2003, 2008, 7 (32-bit), Linux (32-bit e 64-bit)	Virtualização total Paravirtualização

Todas as ferramentas de virtualização suportam a arquitetura do sistema anfitrião x86. As ferramentas OpenVZ e LXC são consideradas como *contaniers* e permitem somente a virtualização de *software* . As ferramentas XenServer, KVM e Xen apresentam o mesmo

modelo de virtualização (virtualização total e paravirtualização). Já as ferramentas VMware e VirtualBox permitem somente a técnica de virtualização total.

2.10 Trabalhos Relacionados

Esta seção relata os trabalhos relacionados com infraestrutura como serviço em Computação em Nuvem e virtualização.

Coutinho, et al., (2012) realizaram uma análise de desempenho sobre um ambiente público de Computação em Nuvem, o *HP Cloud Services*. Foram analisados consumo de CPU, memória, I/O e rede, através de *benchmarks*. Segundo os autores, os experimentos obtiveram um comportamento bastante variável nos resultados, fugindo um pouco do que seria uma razão de proporcionalidade em relação ao aumento da capacidade de VCPU e memória das instâncias. No experimento de CPU, a variação de tempo foi considerável entre as instâncias, variando mais de 1000 segundos entre algumas instâncias de capacidades diferentes para o mesmo experimento. O tempo dos experimentos com memória teve um aumento à medida que mais arquivos de testes eram utilizados, pois uma das características do *benchmark* utilizado é a de aproveitar toda a memória disponível. Para I/O, o tempo foi relativamente estável. Para rede, o tempo foi praticamente o mesmo para todas as instâncias. Os autores não analisaram os dados estatisticamente para verificar se o aumento do tempo nos experimentos de CPU e de memória foram significativos.

Sridharan, (2012) comparou qualitativamente e quantitativamente o desempenho dos hipervisores VMware ESXi 4.1, Citrix Xen 5.6 e KVM usando padrão de referência SPECvirt_sc2010v1.01 formulado pela *Standard Performance Evaluation Corporation* (SPEC), sob várias cargas de trabalho simulando situações reais. Para cada ambiente foram comparados o ponto de saturação, desempenho geral e qualidade de serviço em diferentes cargas de trabalho, desempenho do servidor *Web* e do servidor de aplicação. Os dados coletados foram comparados e em todos dos casos abordados, os p-valores foram maiores que 0,05.

Pfitscher, et al., (2013) usam monitoração para permitir que um cliente de nuvem determine se os recursos disponíveis para suas máquinas virtuais estão provisionados corretamente, ou se estão sub-provisionados ou super-provisionados. O foco está nos recursos de processador e rede, que podem ser facilmente reservados nos ambientes de virtualização atuais. A eficácia da abordagem proposta é demonstrada por resultados experimentais no hipervisor Xen. As métricas de desempenho são todas coletadas dentro das máquinas virtuais,

sendo, portanto, diretamente acessíveis ao usuário, sem a necessidade de intermediação do provedor. Segundo os autores, os resultados experimentais com Linux e Xen demonstram que a abordagem proposta é capaz de diagnosticar corretamente o provisionamento em diferentes cenários de carga.

O uso de técnicas de aprendizado de máquina para provisionamento de recursos em nuvens é proposto por (Rao, et al., 2011; Kundu, et al., 2012; Vasic, et al., 2012). Esses trabalhos associam a utilização de recursos com medidas de desempenho das aplicações (como tempo de resposta) e usam classificadores estatísticos para determinar a classe à qual pertence uma aplicação e, consequentemente, os recursos que devem ser alocados para sua execução. Além do foco em aplicações, e não em máquinas virtuais, os resultados demonstram que essa abordagem funciona para cargas de trabalho estáticas, mas tem problemas com cargas variáveis e/ou desconhecidas.

Heo, et al., (2009) propõem uma ferramenta para gerenciar a alocação de processador e memória de máquinas virtuais sobre a hipervisor Xen, possibilitando o *overbooking*. A utilização de ambos os recursos é observada periodicamente; a partir dessas observações e de um valor objetivo, calcula-se a quantidade de recursos necessária.

Southern, G., et al., (2009), apresentam um trabalho que permite que os sistemas operacionais hóspedes utilizem multiprocessamento simétrico (SMP) em uma máquina virtual. Por fim, existe uma análise de desempenho de multiprocessamento simétrico nos hipervisores VMware ESX 3.5 e Xen 3.2. Foram realizadas análises de carga *single-threaded* e *multi-threaded* para medir a forma como a virtualização realiza a escala durante a execução SMP.

Campos, et al., (2009) abordam questões relativas à virtualização de *desktops* e visa avaliar a aplicabilidade dessa técnica. Para isso, usa-se teste de carga de programas que realiza um conjunto adequado de instruções que geram cargas no sistema, usando o Benchmark SPEC_virt. Foi utilizado o hipervisor VMware Esxi para dois ambientes virtualizados (virtualização total e paravirtualização), com os sistemas operacionais Windows 7 e GNU/Linux com a distribuição Ubuntu 9.10. Foram realizados testes de carga com os algoritmos: cálculo de primos, alocação de memória, escrita em disco, leitura em disco, para verificar se existe alguma diferença de desempenho entre as máquinas virtuais. Por fim, os autores concluíram existir diferença entre a virtualização total e a paravirtualização.

Beserra, et al., (2012) apresentam um trabalho com o objetivo de descobrir qual virtualizador implementa *clusters* com maior desempenho, qual virtualizador gerencia melhor os recursos de rede em um ambiente de *cluster* e qual virtualizador obtém melhor desempenho

em situações onde é exigido uso intensivo do sistema de arquivos do *cluster*. Foram testadas as principais ferramentas de virtualização para a plataforma Windows, o VirtualPC, o VirtualBox e o VMware Workstation. Os algoritmos testados foram o NetPIPE que efetua simples testes de ping-pong, enviando mensagens entre dois processadores. O MPI-I/O Test emprega a biblioteca MPI-I/O, que permite que vários processos que estejam sendo executados em vários nós abram e compartilhem arquivos de maneira consistente. Para cada algoritmo de teste foram realizadas medidas de desempenho de processamento, rede, operações de I/O. Por fim, os autores concluem que nos testes de desempenho de processamento sustentado e capacidade de comunicação, o VirtualBox obteve o melhor desempenho. Já nos testes com operações I/O VMware Workstation obteve desempenho superior. Os autores concluem que a melhor opção é o VirtualBox.

Li, et al., (2013) realizaram experimentos para medições de vários *benchmarks* usando Hadoop MapReduce para avaliar e comparar o impacto no desempenho de três hipervisores populares: CVM, Xen, e KVM. Foram descobertas diferenças de desempenho entre os hipervisores em relação ao tipo de carga de trabalho (CPU ou I/O intensivo), o tamanho da carga de trabalho e colocação de máquina virtual. Os autores concluem que as diferenças de desempenho mostradas entre os hipervisores deverão ser analisadas mais profundamente no futuro usando aplicações para ambientes de nuvem.

Hwang, et al., (2013) discutem que, apesar dos avanços recentes nas arquiteturas da CPU e das novas técnicas de virtualização terem reduzido o custo do uso de virtualização de desempenho, os custos ainda existem, particularmente, quando várias máquinas virtuais estão competindo por recursos. Então os autores propõem uma comparação de desempenho em configurações de virtualização assistida por *hardware*, considerando quatro plataformas populares de virtualização Hyper-V, KVM, vSphere e Xen. Para a realização dos testes, foram criados ambientes virtuais com o sistema operacional Ubuntu com um núcleo de processamento virtual e 2GB de memória RAM. Foram realizados testes nos *benchmarks*: *Bytemark, Ramspeed, Bonnie++ & FileBench, Netperf.*

Existem outros trabalhos na literatura (Deshane, et al., 2008; Gent, et al., 2011; Jungels, 2012) que realizam comparações de desempenho entre as máquinas virtuais. Porém, o interessante seria analisar os dados encontrados nos experimentos, usando algum modelo estatístico (ver Capítulo 3).

Os trabalhos apresentados não utilizam nenhuma ferramenta estatística para análise dos dados exceto o trabalho apresentado por Sridharan, (2012) que utiliza o teste-t para comparação

de médias dos grupos. Todos os trabalhos apresentados utilizam somente a análise focando somente um tipo de sistema operacional ou tipo de máquina virtual. O trabalho apresentado por Coutinho, et Al. (2012) é o único que utiliza a análise de algoritmo em ambientes em Computação em Nuvens.

2.11 Considerações Finais do Capítulo

Atualmente existe uma quantidade enorme de aplicações que estão exigindo um maior poder de computação para o processamento de dados, tais como: aplicações de mapeamento genético, computação gráfica, previsões meteorológicas e até mesmo programas que exigem um grande número de variáveis de entrada. Existiam, outrora, os supercomputadores para atender a essa demanda, porém, eram sistemas fortemente acoplados e que possuíam um custo muito elevado.

Outra alternativa viável para essa demanda é a utilização de *cluster*, grades computacionais e/ou Computação em Nuvem, já que estes disponibilizam recursos computacionais que são, aproximadamente, uma ordem de grandeza superior aos recursos normalmente disponíveis em simples *desktops* ou estações de trabalho, sendo que essas tecnologias não englobam apenas computadores, mas também outras tecnologias como: rede, os algoritmos e os ambientes necessários para permitir a sua utilização.

No estudo realizado no Capítulo 4, a nuvem implementada no modelo de infraestrutura como serviço foi desenvolvida usando a plataforma OpenStack apresentada na Seção 2.7.3, por contemplar praticamente todos os pontos positivos apresentados na Tabela 1.

Capítulo 3

Planejamento Experimental

A otimização de processos visa diminuir os custos e os tempos de análise, maximizando os rendimentos, a produtividade e a qualidade de produtos. Isso permite que profissionais de diferentes formações busquem novas técnicas que fornecem informações seguras e consistentes fundamentadas em métodos estatísticos.

O planejamento experimental é uma técnica de caráter exploratório que permite representar um conjunto de ensaios estabelecidos com critérios estatísticos e científicos com o objetivo de determinar as influências de vários fatores nos resultados de um determinado sistema ou processo (ROLZ, 1996).

Segundo Box et al. (1978) citado por Rodrigues et al. (2009), essa técnica associada à análise de superfícies de respostas é uma ferramenta fundamentada na teoria estatística que fornece informações seguras sobre o processo, minimizando o empirismo que envolve técnicas de tentativa e erro.

Segundo Montgomery et al. (2009), o planejamento experimental tem como objetivos específicos:

- Otimizar o número de ensaios a ser realizado;
- Mostrar quais fatores são mais influentes no resultado;
- Atribuir níveis aos fatores (qualitativos ou quantitativos) para aperfeiçoar o resultado (produzir a melhor resposta possível);

- Atribuir níveis aos fatores influentes de modo a minimizar a variabilidade dos resultados;
- Atribuir níveis aos fatores influentes de modo a minimizar a influência de variáveis incontroláveis.

Contudo, para que o planejamento experimental consiga atingir os objetivos propostos, é muito importante existir uma integração entre as partes (processo e estatística) e, além disso, muito bom senso, tanto dos responsáveis por montar os experimentos, como também, dos responsáveis pela análise estatística e construção dos resultados finais.

3.1 Conceitos Gerais

Esta Seção apresenta alguns conceitos e termos relacionados a esses conceitos necessários para o entendimento da aplicação das técnicas de planejamento de experimentos (RODRIGUES, ET AL., 2009; PAIS, ET AL., 2014; DEAN, ET AL., 1999):

- *Variáreis*: são relações que afetam as características de qualidade de um produto. Podem ser definidas como quantitativas ou qualitativas. As variáveis quantitativas são aquelas que descrevem quantidades e seus possíveis valores são números. Já as variáveis qualitativas são aquelas que descrevem qualidades, as quais não se usa números para descrevê-las;
- *Fatores*: são variáveis controláveis (independentes) do experimento que possuem os seus valores alterados para avaliar o efeito produzido na resposta;
- *Resposta*: são as variáveis dependentes que podem sofrer algum efeito quando alterações são impostas às variáveis de entrada;
- *Níveis dos fatores*: são valores atribuídos aos fatores no experimento;
- *Efeito principal*: é a diferença média na resposta entre combinações de níveis de fatores ou condições do experimento;
- *Efeito de interação*: é a metade da diferença entre os efeitos principais de um fator nos níveis de outro fator;
- *Tratamento*: é uma condição imposta ou objeto que se deseja medir ou avaliar em um experimento, como por exemplo: equipamentos de diferentes marcas, diferentes tamanhos de peças, doses de um nutriente em um meio de cultura, quantidade de lubrificante em uma máquina, temperatura de armazenamento de um alimento;

- *Região experimental*: é o espaço constituído de todas as combinações de todos os níveis de fatores possíveis;
- *Repetição*: é o número de vezes que um tratamento aparece no experimento;
- *Matriz experimental*: é o plano formal construído para conduzir os experimentos. Nessa matriz estão inclusos os fatores, os níveis e tratamentos do experimento.

3.2 Técnicas para definição de sequência dos ensaios

A análise dos resultados encontrados na execução no planejamento experimental deverá ser baseada em técnicas estatísticas para avaliar os erros experimentais que possam afetar o resultado final do experimento.

Para Montgomery, et al. (2009) existem três princípios básicos para a definição dos ensaios num planejamento experimental: o uso de repetição, o uso da aleatorização e o uso de blocagem.

A repetição consiste em repetir um ensaio, sob condições pré-estabelecidas, para cada uma das combinações de fatores da matriz experimental. Esta técnica permite obter uma estimativa de como o erro experimental afeta os resultados dos ensaios e se esses resultados são estatisticamente diferentes. Ela também permite verificar qual a influência de um determinado fator sobre o comportamento de um processo. Além do mais, a repetição permite a obtenção de uma estimativa mais precisa dos efeitos dos fatores.

A aleatorização é uma técnica puramente estatística em que a sequência dos ensaios é feita de forma aleatória e a escolha dos materiais que serão utilizados nesses ensaios também é aleatória. Dessa forma, os efeitos de fatores não controláveis são minimizados.

A técnica de blocagem possui como objetivo eliminar o efeito de um ou mais fatores no resultado do experimento. Com isso, permite realizar a experimentação com uma maior precisão, reduzindo a influência de fatores incontroláveis. O uso de blocos envolve comparações entre as condições de interesse na experimentação dentro de cada bloco.

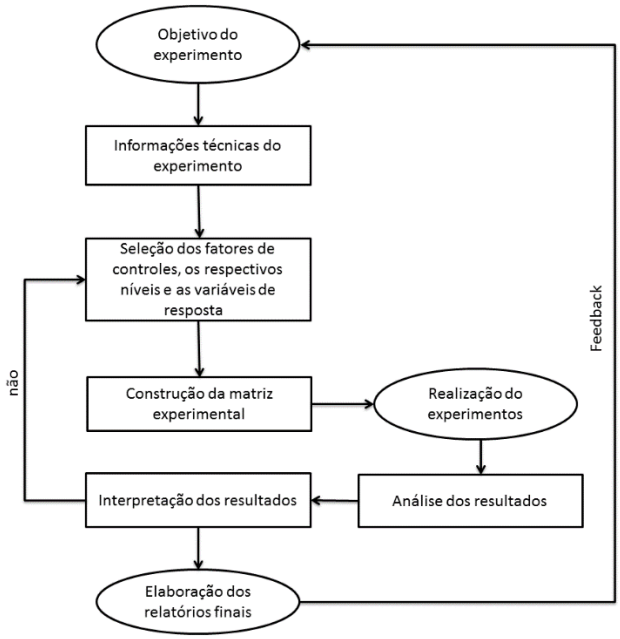
3.3 Processo para conduzir os experimentos

Segundo Antony et al. (1998) e Montgomery (2009) recomenda-se que, em um processo de experimentos, seja feito um plano de trabalho para fornecer apoio às atividades do processo de planejamento experimental, como:

- *Definir os objetivos do experimento*: fase de levantamento de ideias ou informações coletadas inicialmente, que devem ser examinadas para identificar os principais problemas;
- *Definir os fatores do experimento*: fase que coleta as informações técnicas como: fatores de controle, fatores de ruído, os níveis de ajuste e as variáveis de resposta. Nessa fase, as informações técnicas podem resultar numa combinação de conhecimentos práticos e teóricos;
- *Selecionar os fatores de controle e as variáveis de resposta:* fase que seleciona os fatores e as faixas de variação dos níveis desses fatores. Deve-se definir o método de medição dos fatores e a escala numérica que será utilizada para se avaliar as respostas dos experimentos;
- *Selecionar a matriz de experimentos:* fase para construir a matriz de experimentos onde devem ser considerados os fatores de controle, número de níveis do fator e os fatores não controláveis (em processos complexos, com diversos fatores influentes, não se deve partir de um conjunto extenso de experimentos, que envolva um grande número de fatores com diversos níveis);
- *Realizar os experimentos:* fase em que, para cada linha da matriz de experimentos, realiza-se o ensaio real. Para cada ensaio devem ser registradas informações como: data, ensaios adicionais, alteração da sequência de execução;
- *Análise dos dados:* fase em que são aplicados *softwares* estatísticos para auxiliar a utilização das técnicas de planejamento e análise de experimentos, verificando o comportamento dos fatores de controle e a relação entre esses fatores, com a finalidade de estimar os efeitos produzidos nas respostas;
- *Interpretação dos resultados:* fase em que serão extraídas as conclusões dos resultados obtidos e recomendar as ações de melhoria no processo;
- *Elaboração de relatórios:* fase que identificará as limitações teóricas e práticas encontradas e indicar recomendações de realizações de novos experimentos e obter conclusões sobre o processo.

Essas oito etapas se interagem formando um roteiro de processos para a conclusão dos experimentos e análise dos dados (Figura 7). O processo é sequencial e só pode prosseguir para a próxima etapa quando a atual estiver concluída. Pode existir um *feedback* no final do roteiro, caso o experimentador queira repetir todas as etapas existentes para validar os resultados.

Figura 7: Processos do planejamento experimental.



Fonte: (Antony, et al., 1998)

3.4 Planejamento Fatorial

O planejamento fatorial é indicado nos casos em que o pesquisador deseja estudar o efeito de um ou mais fatores (variáveis independentes) que podem influenciar a resposta. Segundo Rodrigues et al. (2009), os esquemas fatoriais não são considerados delineamentos experimentais, mas delineamentos de tratamento, onde cada combinação de fatores denomina-se um tratamento.

O planejamento *screening* fatorial é indicado para a fase inicial do procedimento experimental, quando existe a necessidade de definir os fatores mais importantes e estudar os efeitos sobre a variável resposta. Ou seja, um modelo de efeitos fixos, onde as análises dos efeitos provocados pelos valores não podem ser transferidas para outros níveis que não os analisados no planejamento (DEAN, ET AL., 1999).

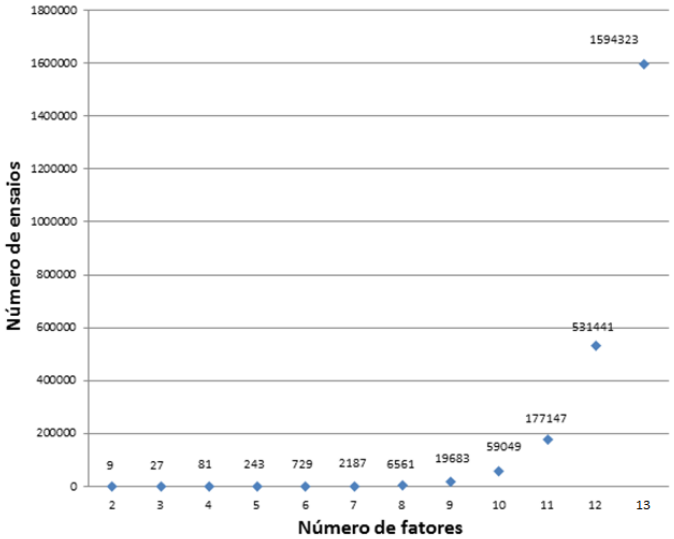
Depois de escolhidos os fatores, o experimentador selecionará os níveis dos fatores. Em cada repetição, todas as possíveis combinações dos níveis de cada fator são investigados. No caso, se o experimentador escolhe dois níveis, o planejamento experimental é nomeado de planejamento fatorial 2^k. Se o experimentador escolher três níveis de fatores o planejamento experimental será denominado planejamento fatorial 3^k.

Segundo Rodrigues et al. (2009), se todas as combinações possíveis, entre todos os níveis de cada fator, estão presentes, o planejamento fatorial é dito completo. E em outros casos é dito como um planejamento fatorial fracionário, que é constituído de frações bem determinadas do planejamento fatorial completo, que são de grande importância na seleção dos níveis ou fatores estudados.

Num planejamento fatorial de dois fatores e cada um com dois níveis, temos 4 combinações. Como por exemplo: se o fator A for representado por números de processadores, com os níveis de '1' e '2' e o fator B é representado por número de memórias, com os níveis '1' e '2', tem-se, então, as possíveis combinações de ensaios: A1B1, A1B2, A2B1 e A2B2. Agora, para o caso de o fator A e o fator B possuir três níveis cada, então, tem-se 8 combinações de ensaios: A1B1, A1B2, A1B3, A2B1, A2B2, A2B3, A3B1, A3B2 e A3B3.

Então, pode-se concluir que: seja 'k' o número de fatores estudados; e cada fator possui 'n' níveis, então se tem um planejamento fatorial completo por n^k. O crescimento das combinações desse tipo de planejamento é muito grande, o que pode deixar inviável a utilização de planejamentos completos para valores grandes como, por exemplo, o planejamento com n = 3 e k = 7, onde tem-se 2187 ensaios (Figura 8).

Figura 8: Total de ensaios de um planejamento fatorial completo (n = 3 e k = 7).



Fonte: (Rodrigues, et al., 2009)

3.5 Planejamento fatorial com dois fatores (k = 2)

O planejamento fatorial 2^2 é muito utilizado em estudos iniciais, quando o experimentador deseja saber quais fatores têm influência sobre a resposta de um determinado

sistema. Nesse tipo de planejamento, são analisados dois fatores influentes, denominados A e B, com as seguintes características:

- O *fator A* possui 2 níveis;
- O *fator B* possui 2 níveis;
- Utiliza-se *n* repetições com a × b combinações;

O planejamento fatorial é iniciado com a definição dos fatores e dos seus respectivos níveis sendo um nível (+) e o outro nível (-). A escolha desses níveis + e – associados aos valores não afeta a análise final dos resultados. Depois de definidos os fatores e os seus níveis, a matriz de planejamento, juntamente com todas as combinações, é gerada para executar os ensaios (TAMHANE, 1999). Após esta etapa, os ensaios são realizados com a possível utilização de réplicas e, no fim, são calculados os dois efeitos dos fatores:

- O efeito principal que é fornecido pela diferença entre a resposta média em cada um dos níveis do fator;
- O efeito de interação que é fornecido pela diferença entre o efeito médio de um fator nos diferentes níveis dos outros fatores.

Para o cálculo dos efeitos, é construída uma tabela de sinais, ou tabela de contrastes onde as colunas correspondem aos fatores principais e suas interações, e uma coluna identidade que corresponde à média. Os sinais da coluna AB são o produto dos sinais das colunas A e B. As linhas são as combinações dos tratamentos. A Tabela 4 mostra uma tabela de sinais para os efeitos no planejamento 2^2.

Tabela 4: Sinais para os efeitos A e B

I	A	B	AB
+	-	-	+
+	+	-	-
+	-	+	-
+	+	+	+

Serão apresentados, a seguir, 2 exemplos que ilustram a utilização da ferramenta de planejamento experimental na prática.

3.5.1 Exemplo 01 - Planejamento fatorial 2^2 (Apresentado por Jain (1991) citado por Pais, (2014))

O estudo do impacto do tamanho da memória (fator A) e da *cache* (fator B) no desempenho de uma *workstation* define dois níveis de cada um dos dois fatores e a resposta é

medida em MIPS (milhões de instruções por segundo). A Tabela 5 apresenta os níveis definidos para os fatores A e B.

Tabela 5: Níveis dos fatores A e B

	Nível -	**Nível +**
Fator A (Memória)	4MB	16MB
Fator B (Cache)	1KB	2 KB

Fonte: (Pais, 2014)

A Tabela 6 mostra a matriz de planejamento com um total de 03 repetições de ensaios e a média dos valores encontrados em MIPS.

Tabela 6: Matriz de planejamento e resultados em 4 ensaios

I	**A**	**B**	**Repetições**			**Médias**
			01	**02**	**03**	
1	4	1	15	18	12	15
2	16	1	45	48	51	48
3	4	2	25	28	19	24
4	16	2	75	75	81	77

Fonte: (Pais, 2014)

Depois dos dados coletados, a matriz de sinais é calculada para verificar os possíveis efeitos (Tabela 7). Os efeitos são calculados pela multiplicação da coluna com a média dos resultados divido por 2^{k-1}. Já o cálculo da primeira coluna corresponde à média e é dividida por 2^{k}.

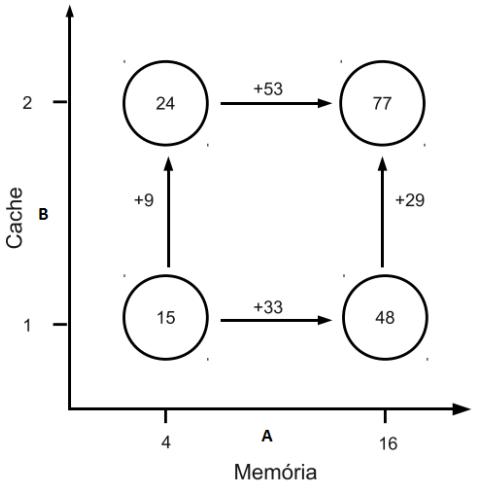
Tabela 7: Cálculo dos efeitos

I	**A**	**B**	**AB**	**Repetições**			**Médias**
				01	**02**	**03**	
1	-1	-1	1	15	18	12	15
1	1	-1	-1	45	48	51	48
1	-1	1	-1	25	28	19	24
1	1	1	1	75	75	81	77
164	86	38	20				Total
164/2	86/2	38/2	20/2				
41	**43**	**19**	**10**				**Efeitos**

Fonte: (Pais, 2014)

A Figura 9 mostra que o efeito de iteração AB indica que a alteração do nível de memória quando a *cache* está no seu nível +, acarreta em um maior aumento do desempenho do que quando a *cache* está no nível -.

Figura 9: Diagrama de interpretação dos resultados do planejamento 2^2.



Fonte: (Pais, 2014)

A análise de variância confirma significância do efeito dos fatores, como é mostrado na Tabela 8. É confirmado que existe interação entre os fatores (p = 0.001271) a um nível de significância de 1% e, portanto, a interpretação dos seus efeitos deve ser feita em conjunto.

Tabela 8: Análise de Variância

Fatores de variação	**Graus de liberdade**	**Soma dos quadrados**	**Quadrados médios**	**F**	**p-valor**	**Hipóteses**
A	1	5547	5547.0	435.059	2.928e-08	*****
B	1	1083	1083.0	84.941	1.556e-05	*****
A:B	1	300	300.0	23.529	0.001271	*****
Resíduo	8	102	12.7	R^2 = 98.55%		
Total	11	7032				

Fonte: (Pais, 2014)

Segundo Kennedy (2008), supõe-se que o coeficiente de determinação represente a proporção da variação da variável dependente que é explicada pela variação da variável independente. O R^2 exercerá esse papel de modo significativo no caso de relações lineares estimadas pelo método dos mínimos quadrados ordinários.

Pode-se considerar então, que R^2 fornece uma medida do quanto o modelo consegue explicar os valores observados. Portanto, 98.55% da variável dependente consegue ser explicada pelos regressores presentes no modelo.

3.5.2 Exemplo 02 - Planejamento fatorial 2^2 (Rodrigues et al. (2009))

Um estudo da atividade enzimática em função do pH (fator A) com dois níveis denominados apenas por baixo (-) e alto (+) e a temperatura (fator B) que possui também dois níveis denominados por baixo (-) e alto (+). Foram realizadas para cada ensaio três repetições como apresentado na Tabela 9.

Tabela 9: Matriz de planejamento e resultados em 4 ensaios.

I	**A**	**B**	**Repetições**			**Médias**
			01	**02**	**03**	
1	-	-	218	212	170	200
2	+	-	67	73	76	72
3	-	+	402	399	411	404
4	+	+	222	258	270	250

Fonte: (Rodrigues, et al., 2009)

Depois dos dados coletados, a matriz de sinais é calculada para verificar os possíveis efeitos (Tabela 10). Os efeitos são calculados pela multiplicação de sua coluna com a média dos resultados divido por 2^{k-1}. Já a primeira coluna corresponde à média e é dividida por 2^k.

Conforme apresentado nos resultados do experimento, a passagem do nível baixo para o nível alto do pH levou a uma diminuição de 141,00 U/ml, na média da atividade enzimática. Pode-se concluir também que a passagem do nível baixo para o nível alto da temperatura levou a um acréscimo de 191,00 U/ml na média da atividade enzimática. Este exemplo possui somente uma interação de primeira ordem e o seu efeito pode ser entendido como a variação causada na resposta, quando varia os níveis de um dos fatores dentro de cada nível do outro.

A análise de variância (Tabela 11) confirma que o efeito de interação pH × temperatura e que os efeitos principais de pH e da temperatura foram altamente significativos ($p < 0,0001$), a um nível de significância bem menor que 5%.

Pode-se concluir que o $R^2 = 98.41\%$, significa que a variável dependente consegue ser explicada pelos regressores presentes no modelo.

Se, e a apenas se, o efeito da interação for não significativo então, pode-se interpretar isoladamente cada efeito principal. Já, se o efeito da interação for significativo, deve-se construir e analisar as hipóteses sobre os níveis de um dos fatores dentro de cada nível do outro (RODRIGUES, ET AL., 2009).

Tabela 10: Cálculo dos efeitos

I	A	B	AB	Repetições			Médias
				01	02	03	
1	-1	-1	1	218	212	170	200
1	1	-1	-1	67	73	76	72
1	-1	1	-1	402	399	411	404
1	1	1	1	222	258	270	250
926	-282	382	-26				Total
926/2	-282/2	382/2	-26/2				
231.5	**-141**	**191**	**-13**				**Efeitos**

Fonte: (Rodrigues, et al., 2009)

Tabela 11: Análise de Variância

Fatores de variação	Graus de liberdade	Soma dos quadrados	Quadrados médios	F	p-valor	Hipóteses
A	1	59643	59643.0	174.39	< 0.0001	*****
B	1	109443	109443.0	320.01	< 0.0001	*****
A:B	1	507	507.0	1.48	< 0.0001	*****
Resíduo	8	2736	342.0	R^2 = 98.41%		
Total	11	172329	-			

Fonte: (Rodrigues, et al., 2009)

2.6 Considerações Finais do Capítulo

Antes que seja definido qual procedimento experimental será mais adequado é sempre necessária uma análise profunda do problema. Portanto, a escolha da melhor estratégia de planejamento experimental é sempre dependente do número de variáveis independentes que se deseja estudar e também do conhecimento inicial do problema.

A teoria descrita será aplicada nos experimentos dos algoritmos e está apresentada no Capítulo 5. A técnica que foi utilizada para gerar os experimentos foi o planejamento fatorial. Ele é indicado em situações em que existem grandes números de fatores analisados, em que tem-se pouco conhecimento do processo e se está longe das condições desejadas e otimizadas.

Capítulo 4

Experimentos com algoritmos

A análise de algoritmos é importante para escolher, entre os vários algoritmos disponíveis na literatura, o mais eficiente para resolver um determinado problema. Além disso, a análise é necessária para prever o desempenho de um algoritmo em um ambiente de execução, escolhendo os melhores parâmetros.

O estudo teórico de algoritmos utiliza métodos dedutivos matemáticos que caracterizam o desempenho do algoritmo em função do tamanho do problema. Os resultados da análise da complexidade são relatados em termos do melhor-pior e caso médio (ZIVIANI, 2008), sem entrar em detalhes do sistema físico (*hardware* do computador) no qual o algoritmo é executado (HOOKER, 1994) citado por (PAIS, 2014). Entretanto, nem sempre os resultados obtidos pela análise teórica são suficientes para explicar o funcionamento de algoritmos na resolução de problemas reais. A análise experimental pode auxiliar nessa tarefa.

Portanto, segundo Bartz-Beielstein et al. (2010), a experimentação de algoritmos em computadores atuais é relevante e necessária para obter informações mais precisas sobre o seu desempenho e robustez.

Para Johnson (2002), no passado, os pesquisadores apresentavam uma falta de rigor científico nos primeiros trabalhos, apresentando, para a análise de algoritmos métodos teóricos e rigorosos (pior caso e caso médio). Portanto, com base nesse rigor, foi difícil usar estudos com experimentos.

Bartz-Beielstein et al. (2010) discutem que o distanciamento entre a análise teórica e a análise experimental que existe na área da computação não ocorre em outras áreas das ciências naturais. Já para Johnson (2002), pode-se perceber um aumento no interesse pelo trabalho experimental na área da ciência da computação, baseado no crescente reconhecimento de que somente os resultados teóricos encontrados nos algoritmos não podem mostrar a verdadeira e completa história sobre o desempenho de algoritmos em mundo real.

Hooker (1994) foi um dos primeiros pesquisadores a apresentar a importância de realizar experimentos com planejamento, com métodos rigorosos para a análise dos dados obtidos e com resultados que possam ser reproduzidos.

4.1 Experimentação

Durante o processo de preparo de um experimento, o experimentador tem um número muito grande de decisões que devem ser tomadas, como: a seleção de problema a ser resolvido pelo algoritmo, o nível de instanciação do algoritmo, o ambiente computacional, medidas de desempenho, parâmetros do problema, parâmetros do algoritmo e relato dos resultados. Cada escolha realizada tem um efeito substancial e significativo nos resultados e na significância do experimento (PAIS, 2014).

O processo para conduzir um planejamento experimental, como apresentado na Seção 3.3, foi adaptado por (BARR, ET AL., 1995) em 5 etapas para conduzir os experimentos com algoritmos: (i) definir os objetivos dos experimentos; (ii) escolher medidas de desempenho e fatores a explorar; (iii) planejar e executar o experimento; (iv) analisar os dados e extrair conclusões; e (v) relatar os resultados.

4.2 Objetivos dos Experimentos

Nesta etapa é sempre importante e imprescindível que o experimentador liste quais são as hipóteses que deverão ser testadas, quais são os resultados a procurar e quais o fatores que serão explorados.

As instâncias do problema podem, de acordo com (PAIS, 2014), serem classificadas em:

- *dados provenientes de situações reais (instâncias reais):* são considerados os melhores conjuntos de informações, porém podem se tornar difíceis de serem conseguidos. Além disso, podem consumir muito tempo e esforço do experimentador;

- *variações de instâncias reais:* permitem como alternativa substituir os valores reais por valores aleatórios. A estrutura da instância real é preservada e alguns valores numéricos são modificados;
- *bibliotecas públicas de referência:* são fontes muito utilizadas e existem muitos repositórios de instâncias para muitos problemas clássicos, inclusive com instâncias reais;
- *instâncias geradas aleatoriamente:* são instâncias geradas de forma aleatória a partir de uma semente e uma lista de parâmetros, possibilitando que um estudo sistemático seja realizado.

4.3 Medidas de Desempenho

Segundo Barr et al. (1995), as medidas de desempenho podem ser divididas em três áreas: qualidade da solução, esforço computacional e robustez.

A medida de desempenho da qualidade da solução apresenta como métodos: cálculo da solução exata para pequenas instâncias; uso de limites inferiores ou superiores; construção de instâncias a partir de valores ótimos conhecidos; estimativa estatística de valores ótimos conhecidos e comparação dos melhores valores encontrados (COSTA, ET AL., 2011).

A medida de desempenho do esforço computacional (velocidade de computação) de um algoritmo pode ser cronometrada totalmente ou em partes, como: o tempo da melhor solução encontrada, o tempo médio total de execução ou tempo por fase (BARR, ET AL., 1995). Segundo Johnson (2002), mesmo que o objetivo principal não seja a análise dos tempos de execução, eles devem ser relatados, pois pode ser de interesse do leitor.

A medida de desempenho da robustez pode ser definida como a capacidade de encontrar soluções sobre uma grande variedade de instâncias de um problema; ou sobre uma grande variedade de instâncias de diferentes problemas (PAIS, 2014).

Para Rardin et al. (2001), as medidas de desempenho são necessárias, principalmente na fase de desenvolvimento da heurística, para, em fase posterior, se obter uma implementação eficaz. Porém, quando nenhuma das medidas de desempenho, existentes na literatura, satisfazem as necessidades do experimento, pode-se criar a sua própria medida de acordo com a necessidade do experimentador.

Nas próximas seções serão apresentadas algumas medidas de desempenho utilizadas para a avaliação de metaheurísticas.

4.3.1 Tempo de execução

O tempo de execução corresponde à medida do tempo de processamento da metaheurística em uma plataforma computacional. Porém, alguns autores (BARR, ET AL., 1995; EIBEN, ET AL., 2001; RARDIN, ET AL., 2001; BARTZ-BEIELSTEIN, ET AL., 2010; COSTA, ET AL., 2011; PAIS, 2014; COFFIN, ET AL., 2000), não recomendam que os experimentadores utilizem o tempo de execução como medida de desempenho, pois essa medida é difícil de reproduzir na mesma plataforma computacional.

Apesar dessa objeção apresentada acima, Johnson (2002) defende que o tempo de execução seja informado aos pesquisadores para terem ideia se ele foi competitivo ou se o algoritmo é claramente mais rápido.

Outras métricas relacionadas ao tempo de execução podem ser encontradas:

- *Quantidade de avaliações:* métrica que corresponde à quantidade de avaliações executadas. Essa medida deverá ser considerada se todas as avaliações gastarem a mesma quantidade e consumirem a maior parte do tempo de execução do algoritmo. Caso a quantidade de avaliação não seja representativa o tempo gasto foi muito pequeno e outros componentes do algoritmo possuem uma importância maior sobre o tempo de execução (EIBEN, ET AL., 2001);
- *Quantidade de gerações*: métrica que é utilizada nos algoritmos evolucionários com o objetivo de quanto menor a quantidade de gerações, melhor o desempenho do algoritmo. Essa métrica não pode ser aplicada para comparação de algoritmos diferentes (PAIS, 2014).

4.3.2 Qualidade da solução

Existem métodos de medição da qualidade da solução, tais como: calcular a solução exata para pequenas instâncias; utilizar limites inferiores ou superiores; construir instâncias a partir de valores ótimos conhecidos; aplicar estimativa estatística de valores ótimos conhecidos e comparar os melhores valores encontrados (RARDIN, ET AL., 2001).

Porém, segundo Costa et al. (2011), ainda não existe um método satisfatório para fazer a análise da qualidade de soluções geradas por heurísticas, pois existem problemas como: (i) comportamento de heurísticas para pequenas instâncias pode ser totalmente diferente para

grandes instâncias, e quanto maior for a instância, pior pode ser o comportamento da heurística; (ii) se os parâmetros estiverem mal configurados ou se decisões na parte de construção da solução forem tomadas erroneamente, a heurística pode ter um desempenho péssimo com instâncias grandes; e (iii) se o limite superior é folgado, não fica claro se o desvio encontrado em relação ao ótimo é devido ao desempenho pobre da heurística ou se o próprio limite está muito longe do valor ótimo.

4.3.3 Outras medidas de desempenho

Existem métricas específicas para verificar o desempenho de algoritmos paralelos. Uma delas é o *speedup* que é uma medida amplamente utilizada para descrever a eficiência conseguida sobre o processamento serial pelo uso de vários processadores. Existem também a *eficiência*, que mede a fração de tempo que um processador é de fato utilizado, a *fração serial*, que mede a proporção do algoritmo que é inerentemente sequencial e a *scaleup* que está relacionada com a medida *speedup* escalável (PAIS, 2014).

Segundo Costa et al. (2011), existem métricas como: (i) *tradeoffs*, uma comparação feita entre dois fatores, onde é verificado qual fator perde ou ganha na execução de um experimento (tenta-se achar um equilíbrio para os dois fatores, com o objetivo de ter um bom desempenho nos testes); (ii) *a precisão numérica*, uma medida de capacidade do algoritmo de computar corretamente a resposta diante da instabilidade numérica; (iii) *a quantidade de interações*, a quantidade de passos que o algoritmo necessita para resolver o problema (independe do computador usado); (iv) *quantidade de chamadas de uma determinada função*, que pode ser chamada da própria função objetivo ou de outra função que tenha relevância no teste; e (v) *operações matemáticas*, que é a quantidade de vezes que uma operação básica é necessária durante a execução do algoritmo.

4.4 Fatores a Explorar

A escolha dos fatores deve ser devidamente documentada e deve-se definir também como eles serão tratados dentro do planejamento. Segundo Barr et al. (1995), os fatores que podem afetar o desempenho de um algoritmo em um planejamento experimental são:

- *Fatores do problema:* características como a dimensão e a estrutura de um problema podem afetar o desempenho de um algoritmo. Por isso, é sempre importante testar

os algoritmos com instâncias de tamanho grande, pois se for realizado testes apenas com instâncias pequenas, essas não poderão produzir predições corretas e realistas;

- *Fatores do algoritmo:* características como a estratégia e os parâmetros de um algoritmo podem afetar o seu desempenho. Fatores como o critério de parada devem ser bem documentados, pois alguns algoritmos podem gastar longo período de tempo para serem executados. Portanto, em geral, os critérios de parada são baseados em esforço computacional ou na qualidade da solução;
- *Fatores do ambiente de teste:* características relacionadas à plataforma onde serão executados os experimentos (tipo de processador, tamanho da memória, frequência do *clock*, sistema operacional, linguagem de programação, compilador, programas executados em segundo plano, habilidades do programador). O ideal seria que alguns desses fatores de ambiente fossem idênticos ou isolados ao realizar a comparação de dois algoritmos.

4.5 Planejamento de Experimentos

Para Montgomery (2009), a realização de um bom experimento tem como objetivo alcançar as metas experimentais traçadas inicialmente, demonstrar claramente o desempenho dos testes, ter justificativas lógicas, gerar boas conclusões e ser passível de reprodução. Portanto, a única forma de conseguir englobar essas características é usando o planejamento experimental.

No planejamento experimental procura-se identificar quais as variáveis que podem influenciar o desempenho; decidir quais medidas de desempenho e avaliar a variância destas medidas, selecionando um conjunto apropriado de instâncias de teste para poder responder às questões que são levantadas nos objetivos do experimento (CROWDER, ET AL., 1978).

4.6 Execução do Experimento

Etapa em que os dados serão coletados assegurando sempre o que foi planejado na etapa anterior (Seção 4.5). Também deve-se considerar que a execução dos testes seja de forma aleatória e que exista uma uniformidade na plataforma computacional.

Johnson (2002) mostra que é sempre inviável realizar testes com instâncias somente uma vez. Isto pode levar a conclusões erradas e até mesmo tornar o experimento irreprodutível. A quantidade de testes necessários varia de acordo com os objetivos delineados inicialmente. Por isso, pode não ser seguro tentar inferir alguma conclusão com um único teste feito sobre uma dada instância. Uma prática experimental que pode ser feita é realizar o teste *k* vezes e selecionar a melhor solução, relatar os tempos de execução e as soluções encontradas.

4.7 Análise de Dados

Etapa em que os dados coletados serão analisados e interpretados, transformando-os em informações. A análise consiste em avaliar os dados que foram obtidos, aplicando técnicas estatísticas e não estatísticas com relação aos objetivos definidos no início do experimento para extrair informações conclusivas (BARR, ET AL., 1995).

Nessa etapa o uso de ferramentas de análise de dados, como bibliotecas estatísticas ou programas de visualização de dados, são muito úteis, pois aplicam testes formais com significância estatística como uma maneira de introduzir mais precisão científica nas investigações dos algoritmos (MGARBAGE COLLECTOREOCH, 2001).

4.8 Apresentação dos resultados

Um dos problemas encontrados nas pesquisas é a apresentação de dados sem interpretação. Não é suficiente realizar os testes, colocar os resultados em tabelas e deixar que o leitor tire suas próprias conclusões (JOHNSON, 2002).

Nessa etapa, espera-se que, no mínimo, devam ser relatados os padrões encontrados nos dados. O relatório do experimento, quando bem estruturado, pode fornecer ao leitor uma orientação padrão, com a descrição dos detalhes necessários para compreender e, se for o caso, reproduzir o experimento.

Conforme Barr, et al., (1995) o relatório deve ser organizado em partes como:

1. Descrição do objetivo geral da pesquisa;
2. Seleção das instâncias de problemas e das medidas de desempenho em execuções preliminares;

3. Concretização das questões com os objetivos do experimento e estabelecimento de afirmações e hipóteses a serem testadas;
4. Especificação das instâncias de problema; do algoritmo e de seus parâmetros; das medidas de desempenho e do ambiente de teste;
5. Apresentação dos dados brutos encontrados durante a execução do algoritmo;
6. Descrição de exceções ou padrões incomuns observados sem avaliações subjetivas; e
7. Conclusão sobre as hipóteses da etapa 3 e fornecimento de interpretações subjetivas necessárias sobre as observações registradas.

4.9 Trabalhos Relacionados

Esta seção é responsável por relatar os trabalhos existentes relacionados com a utilização de planejamento experimental com algoritmos.

Schad, J., et al., 2010 apresentam um estudo da variância do desempenho da infraestrutura em nuvem *Amazon EC2* a partir de diferentes perspectivas. Foi usado *microbenchmarks* estabelecidos para medir a variação de desempenho em CPU, I/O, e rede. Foram coletados dados por um mês inteiro e comparados com os resultados obtidos em um *cluster* local. Os resultados mostram que o desempenho do EC2 varia muito e, na maioria das vezes, cai duas bandas impactando, segundo os autores, numa grande diferença de desempenho. Foram analisados os resultados considerando diferentes zonas de disponibilidade, diferentes momentos e diferentes locais. Os autores concluem que a variância no EC2 é atualmente tão alta que a resposta dos experimentos (*wall clock*) deve ser realizada cuidadosamente.

Passamontes, et al., (2006) utilizam o algoritmo *Simplex* para otimizar o processo de encontrar uma sequência analítica que permite determinar simultaneamente quais características físicas e químicas são semelhantes. Inicialmente, os autores propuseram um planejamento fatorial fracionado 2^{6-2}, a fim de abordar as respostas, reduzir a experimentação e estudar quais fatores são significantes na resposta esperada e como esses fatores se interagem.

Ridge, et al., (2010) realizam uma abordagem de experimentos para ajustes dos parâmetros que afetam o desempenho do algoritmo. O estudo de caso abordado nesse trabalho é a resolução Colônia de Formigas aplicado ao problema de roteamento. Foram analisados 12 fatores nesse problema. O tempo de execução do algoritmo, a qualidade da solução e o erro relativo foram as medidas de desempenho analisadas. Como planejamento de experimentos, foi

realizado o planejamento experimental fatorial fracionado 2^{12-5}. Para a análise dos dados foi utilizado a ANOVA.

Pais, (2014) propõe apresentar uma metodologia baseada no planejamento experimental para obter a configuração ótima dos parâmetros de um algoritmo genético paralelo (AGP) executado em plataforma com processador multicore. Foram analisados 7 fatores relacionados com: número de subpopulações, topologia de migração, frequência de migração, população total, taxa de migrantes, seleção de migrantes, posicionamento de migrantes. Foi aplicado um planejamento fatorial de 2^7. Foram analisadas as funções Rastringin e Rosenbrock. Nos resultados apresentados, os fatores e interações influentes no desempenho do AGP-I foram diferentes para cada uma das funções de teste. Petrovski, et al., (2005) também aplicaram o uso de planejamento experimental na área de algoritmos genéticos.

Hutter, et al., (2010) usam a metaheurística de busca local iterativa no espaço de parâmetros. O método executa a busca em várias instâncias de um problema, variando um conjunto de parâmetros qualitativos e quantitativos. Os algoritmos testados são *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) e *Scaling And Probabilistic Smoothing* (SAPS). Por fim, os autores fazem um comparação do modelo proposto com os modelos existentes na literatura.

Pallister, et al., (2013) apresentam uma técnica para explorar o efeito no consumo de energia de um grande número de otimizações que podem ser executadas em um compilador. Foram avaliados um conjunto de dez benchmarks para 5 diferentes plataformas. Um planejamento fatorial completo e fracionado foi utilizado para explorar o espaço de otimização e estimar a precisão e combinação dos efeitos. Para cada plataforma foi medido o poder de hardware para garantir que o consumo de energia seja capturado. Os autores concluem que a estrutura do benchmark tem um maior efeito do que a arquitetura do hardware se a otimização for efetiva.

Syed, et al., (2010) exploraram as diversas configurações de hardware e carga do processador sobre a existência de falhas intermitentes e do comportamento não determinístico de sistemas de *software*. No estudo utilizaram o sistema Mozilla Firefox. Cada condição que causa falhas foi replicada dez vezes para cada um dos nove hardwares estudados. Os fatores estudados foram: velocidade do processador, memória e disco. O planejamento utilizado foi o planejamento fatorial 3^3. Para a análise dos dados foi utilizado a ANOVA. Os dados foram analisados e os autores concluíram que as configurações de hardware com velocidade menor

de processamento e menos memória tiveram mais falhas que os demais. A carga de processamento pode influenciar a ocorrência de algumas falhas.

Sun, et al., (1999) apresentam uma metodologia de avaliação de sistema avançado de memória. A metodologia é baseada em análise fatorial estatística. O planejamento tem como objetivo determinar o impacto do sistema de memória e de programas para o desempenho do conjunto, identificar o gargalo existente em uma hierarquia de memória e fornecer a comparação custo/desempenho através de análise estatística. A metodologia proposta analisa o ambiente em 4 perspectivas (efeito do código e máquina, classificação do código/máquina, escalabilidade e hierarquia memória) e são baseadas em um planejamento fatorial e método de regressão estatística. Para a análise dos dados utilizou a ANOVA. Foram testados cinco tipos de códigos. Os dados são apresentados e o autores concluem informando que a metodologia proposta é uma ferramenta efetiva para projeto e análise para sistema de memória.

Esses trabalhos mostram que é possivel a uilização de plamejamento fatorial na experimentação de algoritmos o que viabiliza a criação de uma metodologia para também analizar caracteristicas vitais para a criação de um ambiente virtual em Computação em Nuvem (quantidade de núcleo, quantidade de memória, tipo de sistema operacional e máquina virtual).

4.10 Considerações Finais do Capítulo

Segundo Cormen et al. (2009), um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída. A entrada também denominada de instância de um problema satisfaz a quaisquer restrições impostas no enunciado do problema, necessária para se calcular uma resposta.

Trabalhos (CROWDER, ET AL., 1978; BARR, ET AL., 1995; EIBEN, ET AL., 2001; COFFIN, ET AL., 2000; MGARBAGE COLLECTOREOCH, 2001; RARDIN, ET AL., 2001; JOHNSON, 2002; BARTZ-BEIELSTEIN, ET AL., 2010; COSTA, ET AL., 2011; PAIS, ET AL., 2014) apresentam a técnica de planejamento de experimentos com métodos rigorosos para a análise dos dados obtidos e com resultados que podem ser reproduzidos, Seção 4.

As teorias apresentadas são aplicadas no planejamento dos experimentos e estarão descritas no Capítulo 5. A técnica que foi utilizada para conduzir os experimentos com algoritmos está apresentada na Seção 3.3.

Capítulo 5

Metodologia para Comparação de Ambientes Virtuais

Este Capítulo apresenta de forma completa a metodologia proposta para a comparação de ambientes virtuais na computação em nuvem baseada na análise estatística. O processo utilizado para conduzir os experimentos é uma adaptação do método proposto por Barr et Al. (1995) apresentado na Seção 4.1.

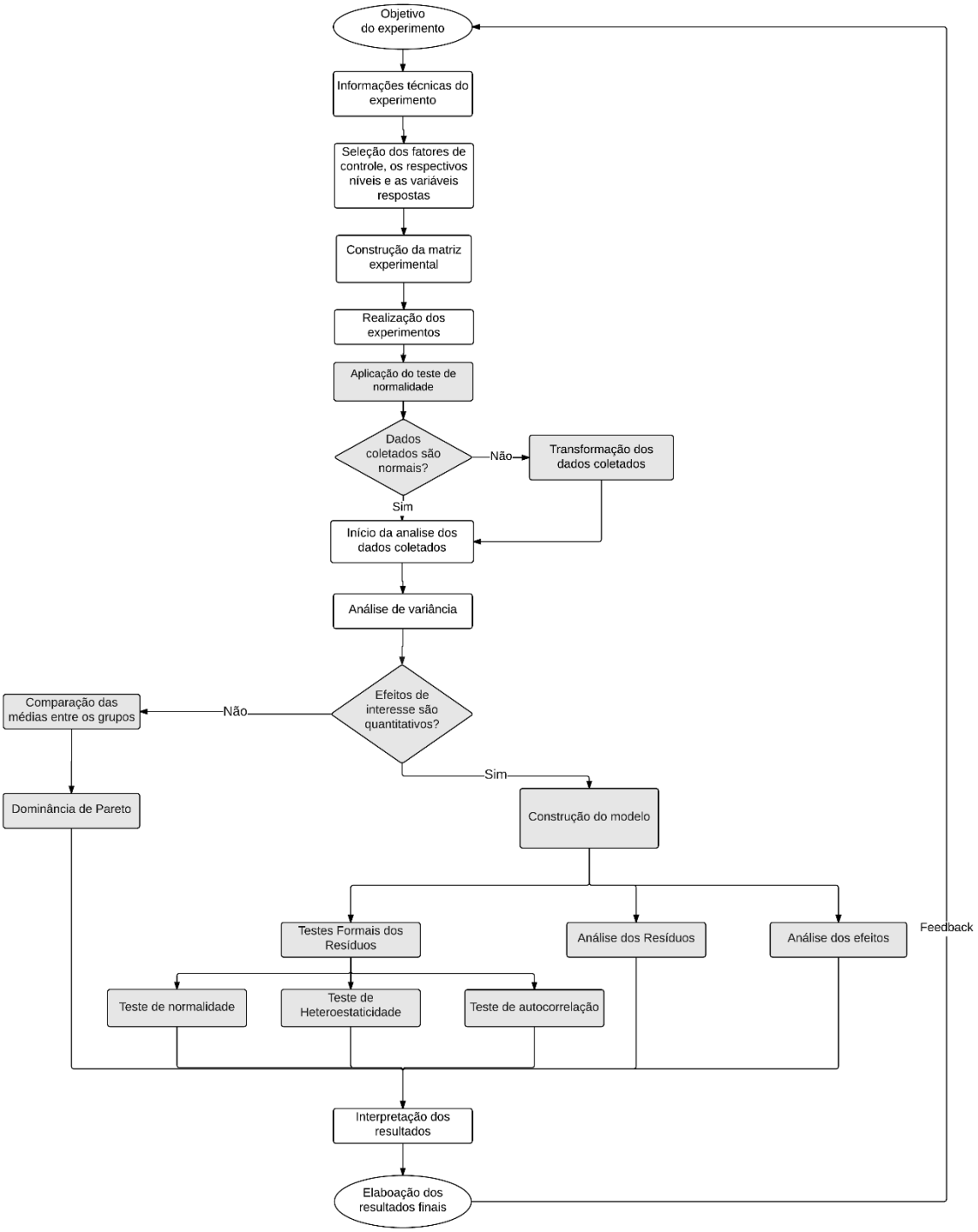
5.1 Metodologia proposta

Como apresentado na Seção 4.1, Barr, et. Al (1995) apresenta uma metodologia para conduzir um planejamento experimental em 5 etapas: (i) definir os objetivos dos experimentos; (ii) escolher medidas de desempenho e fatores a explorar; (iii) planejar e executar o experimento; (iv) analisar os dados e extrair conclusões; e (v) relatar os resultados.

A metodologia proposta nesse trabalho adapta o método de Barr, et At. (1995) para que o mesmo possa ser aplicado na comparação de ambientes virtuais na Computação em Nuvem. A metodologia proposta mostra de forma detalhada como deverá ser analisados os dados que foram coletados durante a execução dos experimentos. A Figura 10 apresenta as etapas da metodologia proposta. Os retângulos cinzas são etapas adicionadas ao modelo proposto por Barr et Al. (1995). Esses retângulos são responsáveis por guiar o usuário no processo de analise

dos dados coletados, verificação de normalidade dos cados coletados e transformação dos dados. Para a verificação do dados coletados e a análise dos dados pode ser utilizado o *software* R.

Figura 10: Fluxograma da metodologia proposta para comparação de ambientes virtuais



Fonte: Adaptação de Barr et Al., (1995)

5.2 Etapas da Metodologia proposta

5.2.1 Passo 01: Objetivo dos Experimentos

Nessa etapa como apresentada na Seção 4.2 devem ser apresentados de forma clara quais são os objetivos da experimentação. Nessa etapa pode ser delineado as hipóteses ou indagações que se desejam responder durante a etapa da elaboração dos resultados.

5.2.2 Passo 02: Informações técnicas do experimento

Depois de definir os objetivos dos experimentos (passo 01). Essa etapa (passo 02) deve conter informações técnicas como:

- *ambiente computacional*: etapa em que devem ser especificadas informações sobre a estrutura computacional utilizada para a execução dos experimentos, como: o tipo de processador, quantidade e o tipo e memória física, quantidade e o tipo do disco rígido e o sistema operacional juntamente com o *kernel*.
- *configuração do ambiente*: etapa responsável por detalhar todas as ferramentas (compiladores, linguagem de programação, *framework* instalados no ambiente computacional) utilizadas para a execução das instâncias de teste;
- *instância de teste*: que contém informações detalhadas sobre as instâncias que serão testadas (algoritmos);

5.2.3 Passo 03: Seleção dos fatores de controle, os respectivos níveis e variáveis respostas

Após definir as instâncias de teste do experimento (passo 02). O passo 03 tem como objetivo permitir ao experimentador selecionar quais serão os fatores de controle que serão utilizados para avaliação durante a experimentação. Os fatores a serem escolhidos devem corresponder ao parâmetros de configuração de um ambiente virtual.

Segundo Ziviani, (2008), a quantidade de dados a serem processados afeta o tempo de execução de um algoritmo de forma mais significativa, mas o custo de um algoritmo também é dependente do *hardware*, do compilador e da quantidade de memória.

Por fim Barr et al. (1995) indica que fatores do ambiente de teste como: tipo do processador, tamanho de memória, frequência do *clock*, sistema operacional, linguagem de programação, compilador, programas em execução em segundo plano e a habilidade do

programador devem ser informados. Mas o ideal seria que alguns desses fatores fossem idênticos ou isolados.

Selecionado os fatores que farão parte da experimentação, como segunda parte, cada fator pode ser observado em vários níveis. Como exemplo os níveis do fator quantidade de memória pode ser: 1GB, 3GB e 5GB.

Por fim, nessa etapa deve ser selecionado a variável resposta. A variável resposta é considerada uma variável dependente na experimentação podendo sofrer algum efeito quando alteração são aplicadas às variáveis de entrada. Como exemplo a variável reposta pode ser o temo de execução de um algoritmo.

5.2.4 Passo 04: Construção da matriz experimental

Definido os fatores e os níveis de cada fator (passo 03), o passo 04 tem como objetivo escolher o planejamento que mais se adeque, podendo, ser um planejamento aleatorizado com um único fator, planejamento fatorial completo n^k, planejamento fatorial em blocos, planejamento fatorial fracionado, entre outros.

Sabendo-se o planejamento que será utilizado, consegue-se construir a matriz experimental que é composta por todos os ensaios que deverão ser realizados. Por exemplo: a matriz experimental de um planejamento fatorial 2^4 possui 16 linhas, onde cada linha corresponde a um ensaio. Já uma matriz experimental de um planejamento fatorial 3^4 possui 81 ensaios.

5.2.5 Passo 05: Realização dos experimentos

Após o passo 04, na etapa de realização do experimento (Passo 05), é muito importante monitorar todo processo, para garantir que tudo esteja sendo feito de acordo com o planejamento. Erros no procedimento experimental nesse estágio, em geral, destruirão a validade do experimento. O planejamento geral, do início até o fim, é crucial para o sucesso.

5.2.6 Passo 06: Aplicação do teste de normalidade nos dados coletados no Passo 05

Após a coleta dos dados, para a utilização da estatística paramétrica nos próximos passos, a etapa de transformação dos dados é necessária quando requisitos de normalidade da

distribuição e homogeneidade das variâncias não puderem ser atendidos pelos dados da amostra experimental (CAMPOS, 2000).

Para confirmar se os dados coletados aproximam de uma distribuição normal pode ser aplicado o teste de Shapiro-Wilk, função *shapiro.test*, ou o teste de *Kolmogorov-Smirnov*, função *ks.test* ou o teste de *Anderson-Darling*, função *ad.test*, disponíveis no software R.

5.2.7 Passo 07: Transformação dos dados coletados no Passo 05

Caso os dados coletados não se aproximarem de uma distribuição normal (Passo 06). O passo 07 é necessário. Durante a etapa de transformações dos dados (Passo 07), não existem transformações específicas para cada conjunto de dados. Portanto, segundo Campos (2000), transformações dos dados mais comumente utilizadas são: logarítmica, a raíz quadrada dos dados, raíz cúbica dos dados, a transformação angular, a transformação hiperbólica de primeiro grau (ou o inverso dos dados) ou hiperbólica de segundo grau, a transformação percentual.

Além disso, transformações com distribuição empírica podem ser utilizadas, como a transformação de Johnson.

Segundo Harsteln, et al., (2010) e Slifker, et al., (1980) apresentam a transformação de Johnson ajustando os dados por meio de uma distribuição empírica, baseada em valores como γ, ε, η e λ. Esses valores são parâmetros de ajuste da curva da função que será normalizado.

Existem três famílias para a transformação de Johnson definidas como limitada (S_b), logonormal (S_l) e ilimitada (S_u) e é representada respectivamente através das Equações (1), (2) e (3).

$$z = \gamma + \eta sinh^{-1}\left(\frac{x-\varepsilon}{\lambda}\right) \quad (1)$$

$$z = \gamma + \eta ln\left(\frac{x-\varepsilon}{\lambda+\varepsilon-x}\right) \quad (2)$$

$$z = \gamma + \eta \ln(x - \varepsilon) \quad (3)$$

A partir dessas equações temos que reverter a transformação invertendo as funções que definem as famílias em transformação de Johnson. A Equação (4) representa a inversa da Equação (1) pertencente à família S_b.

$$x = \frac{(\lambda+\varepsilon)e^{\frac{z-\gamma}{\eta}} + \varepsilon}{1 - e^{\frac{z-\gamma}{\eta}}} \quad (4)$$

A Equação (5) representa a inversa da Equação (2) pertencente à família S_l.

$$x = \frac{\lambda\, e^{\frac{z-\gamma}{\eta}}}{1+ e^{\frac{z-\gamma}{\eta}}} + \varepsilon \qquad (5)$$

A Equação (6) representa a inversa da Equação (3) pertencente à família S_u.

$$x = e^{\frac{z-\gamma}{\eta}} + \varepsilon \qquad (6)$$

onde, γ, ε, η e λ são valores que são informados e z é o dado transformado. A transformação de Johnson pode ser utilizada no *software* R importando a biblioteca RE.Johnson (FERNANDEZ, 2015). Essa biblioteca retorna informações referente a qual família foi utilizada para a transformação juntamente com o valor-p, os dados transformados e os valores das variáveis , γ, ε, η e λ.

5.2.8 Passo 08: Início da análise dos dados coletados

Após a verificação da normalidade dos dados coletados, e a transformação caso necessário, Inicia a etapa de análise dos dados coleados.

Nessa etapa podem ser realizadas avaliações iniciais com os dados coletados como: cálculo de média, mediana, desvio padrão e coeficiente de variação. Além disso pode ser construído os gráficos de boxplot e o histograma dos dados coletados.

Gráfico de Boxplot

Na estatística descritiva o diagrama de caixa (*boxplot*) é um gráfico que permite no eixo vertical apresentar a variável a ser analisada e no eixo horizontal representar a variável dependente.

O *boxplot* é formado pelo primeiro e terceiro quartil e também pela mediana, além dos valores máximos, mínimos e *outliers* (BUSSAB, ET AL., 2009).

A interpretação do *boxplot* é da seguinte forma (Figura 11):

- A caixa denominada *box* contém 50% dos dados. O limite superior representa o percentil de 75% dos dados e o limite inferior representa o percentil de 25%;
- A linha na caixa representa a mediana;
- Os extremos do gráfico indicam os valores mínimo e máximo, exceto quando existir os *outliers*.

O *boxplot* também pode ser utilizado para comparar visualmente dois ou mais grupos. Então, dois ou mais *boxplots* podem ser colocados lado a lado e comparar a variabilidade entre eles, a mediana e assim por diante.

A construção do gráfico de caixa pode ser utilizado no *software* R utilizando a função disponível *boxplot().*

Figura 11: Representação de um diagrama de boxplot

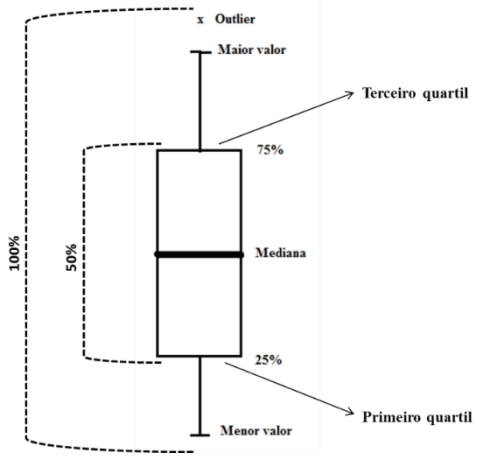


Gráfico de Histograma

Segundo Neto et al. (2010), histograma é um gráfico tradicional para uma determinada distribuição de frequência, em que cada intervalo dessa distribuição é representado por um retângulo. A base de cada retângulo representa uma classe e a altura de cada retângulo representa a quantidade ou frequência com que o valor dessa classe ocorreu no conjunto de dados.

Conhecendo a frequência pode-se determinar as probabilidades de certos valores de interesse testando hipóteses sobre uma determinada população.

A construção do gráfico de histograma pode ser utilizado no *software* R utilizando a função disponível *hist().*

5.2.9 Passo 09: Análise de Variância

A análise de variância (ANOVA) é executada para a matriz de experimental para confirmar a significância dos efeitos. A análise de variância é a melhor abordagem quando se quer comparar várias médias. Nos relatos apresentados dos resultados obtidos são mostrados nas tabelas os efeitos dos fatores principais e das interações de segunda ordem. As interações

de ordem maior que dois são, em geral, insignificantes e podem ser usadas para estimar o erro. Para a análise de variância e as demais análises e gráficos foi utilizado o *software* estatístico R (VENABLES, ET Al. 2015).

A análise de variância pode ser utilizada também no *software* R utilizando a função *lm()*, o resultado o teste de análise de variância pode ser verificada utilizando a função *summary.aov()*.

Após a execução da analise de variância deve-se verificar quais efeitos foram significativos e tomar uma decisão. Caso os efeitos significativos forem quantitativos deve-se realizar os Passos 10, 11 e 12 analisando assim os resíduos e os efeitos.

Se os efeitos significativos forem qualitativos deve-se realizar o Passo 13 e posteriormente o Passo 14.

5.2.10 Passo 10: Testes formais dos resíduos

Para confirmar que os resíduos estão distribuídos de forma homogênea é aplicado o teste de Breusch-Pagan, função *bpTest*. Para confirmar se os resíduos aproximam de uma distribuição normal é aplicado o teste de Shapiro-Wilk, função *shapiro.test*. E para verificar a auto correlação negativa e positiva nos resíduos é aplicado o teste de DurbinWatson, função *durbinWatsonTest*. Todas essas funções estão disponíveis no *software* R.

Teste de Shapiro-Wilk

O teste de Shapiro-Wilk é baseado na estatística W, utilizado para verificar se uma amostra aleatória provém de uma distribuição normal (ESTATCAMP, 2014; SHAPIRO ET AL (1965)). Para realizar o teste deverá então se formular duas hipóteses:

H_0 *: A amostra provém de uma população normal*

H_1 *: A amostra não provém de uma população normal*

Segundo Shapiro et al (1965) , baseado no cálculo retornado pelo teste, a decisão será: rejeitar H_0 a um nível de significância se $W_{calculado} < W\alpha$. A um nível de significância de 5% o valor de $W\alpha$ será de 0.934 tendo como base 35 repetições (Anexo A).

Teste de Breusch-Pagan

A análise de variância exige que os erros tenham distribuição normal e deve haver homoscedasticidade entre os tratamentos (variância homogênea). O teste de Breusch-Pagan é utilizado para testar a hipótese nula de que as variâncias dos erros são iguais

(homoscedasticidade) versus a hipótese alternativa de que as variâncias dos erros são uma função multiplicativa de uma ou mais variáveis, sendo que esta(s) variável(eis) pode(m) pertencer ou não ao modelo em questão (ESTATCAMP, 2014).

A Equação 7 representa a regressão linear com erros heteroscedásticos.

$$y_i = \beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots + \beta_k x_{ik} + e_i \qquad (7)$$

Com $\mathrm{var}(e_i) = \sigma_i^2$

A proposta é que a variância do erro é em função de um conjunto de variáveis explicativas (Equação 8).

$$\sigma_i^2 = h(\alpha_1 + \alpha_2 x_{i2} + \alpha_3 x_{i3} + \cdots + \alpha_k x_{ik}) \qquad (8)$$

Portanto, quando $\alpha_2 = \alpha_3 = \cdots = \alpha_k = 0$ os erros da equação são homoscedásticos.

Por fim, para testar se um modelo possui homoscedasticidade, deve-se testar as seguintes hipóteses:

$H_0 : \alpha_2 = \alpha_3 = \cdots = \alpha_k = 0$ homoscedasticidade

H_1 : *nem todos* α *em* H_0 *são zero* heteroscedasticidade

Portanto, H_0 não será rejeitada com o valor-p maior que o nível de significância adotado.

Teste de Durbin-Watson

O teste de Durbin-Watson é utilizado para detectar a presença de auto correlação (dependência) nos resíduos de uma análise de regressão. Segundo Rawlings (1998) as hipóteses que são consideradas no teste de Durbin-Watson são:

$H_0 : p = 0$ independentes

$H_1 : p > 0$ dependentes

onde p é o parâmetro de autocorrelação.

A Equação 9 representa o teste estatístico de Durbin-Watson.

$$dw = \frac{\sum_{i=2}^{n}(e_i - e_{i-1})^2}{\sum_{i=1}^{n} e_i^2} \approx 2(1 - \hat{p}) \qquad (9)$$

onde $e_i = y_i - \hat{y_i}$ e y_i e $\hat{y_i}$ são respectivamente valores preditos da variável resposta de um indivíduo i . Os valores críticos d_U (maior) e d_L (menor) são respectivamente valores tabulados (Anexo B) para diferentes valores de *k* e *n*. Então, se:

$dw < d_L$	rejeita H_0
$dw > d_U$	não rejeita H_0
$d_L < dw < d_U$	o teste foi inconclusivo.

5.2.11 Passo 11: Análise dos resíduos

Nessa etapa devem ser apresentados os gráfico dos resíduos versus valores previstos, histograma dos resíduos e o gráfico de probabilidade normal dos resíduos

No caso de um ajuste adequado, os resíduos devem apresentar comportamento aleatório e seus valores devem ser muito próximos do valor zero. Ou seja, ao observar o gráfico dos resíduos, não se pode identificar um padrão de comportamento que não o aleatório e nem valores muito distantes do zero. Caso esse comportamento ocorra, será normalmente necessário alterar o modelo, incluindo ou retirando variáveis independentes, ou até mesmo realizando alguma transformação que adeque melhor o modelo aos dados (NETO ET AL, 2010). Esse gráfico pode ser construído no *software* R através da função *plot(),* passando como parâmetro de entrada os resíduos e os valores preditos encontrados durante a análise de variância.

O gráfico do histograma dos resíduos tem como objetivo verificar e avaliar a dispersão e a distribuição dos resíduos. Esse gráfico pode ser construído no *software* R através da função *hist(),* passando como parâmetro de entrada os resíduos encontrados durante a análise de variância.

Segundo Simonoff (2014), o gráfico de probabilidade normal dos resíduos verifica visualmente se os erros seguem uma distribuição normal. Esse gráfico pode ser construído no *software* R através da função *qqnorm(),* passando como parâmetro de entrada os resíduos encontrados durante a análise de variância.

5.2.12 Passo 12: Análise dos efeitos

Nessa etapa são analisados os efeitos utilizando os gráficos de probabilidade normal dos efeitos, gráfico de pareto, gráfico de interação entre os efeitos e diagrama de interpretação dos resultados.

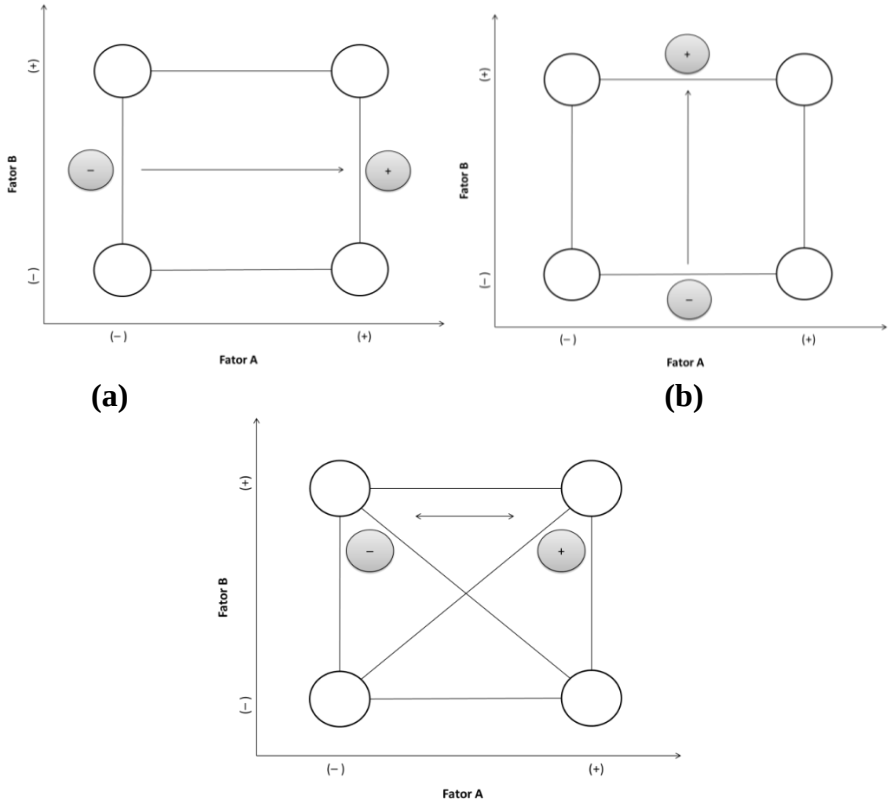
Gráfico de probabilidade normal dos efeitos

A análise utilizando gráficos normais é uma técnica que tem por objetivo distinguir, nos resultados de um planejamento experimental, os valores que correspondem realmente aos efeitos daqueles que são devidos apenas a ruídos (NETO ET AL, 2010). Também para Neto et al. (2010), os gráficos normais nos ajudam a avaliar a qualidade de um modelo qualquer, seja ele relacionado com um planejamento fatorial ou não. Um modelo bem ajustado deve ser capaz de representar toda informação sistêmica contida nos dados.

Diagrama de interpretação dos resultados e Gráfico de interação entre os fatores

Conforme Neto et al (2010), os efeitos podem ser interpretados geometricamente. A representação do planejamento experimental é feita em um sistema cartesiano, focando um eixo por fator. Portanto, para dois fatores, o espaço definido para esses fatores será um plano. Então a representação será na forma de um quadrado, sendo que cada ensaio do planejamento é representado por um vértice do quadrado (ver Figura 12).

Figura 12: Representação geométrica dos efeitos em um planejamento 2^2. Os efeitos principais são contrastes entre as arestas opostas [(a), (b)]. O efeito de interação é o contraste entre as duas diagonais [(c)] ou contraste das duas diagonais



(a) **(b)**

(c)

Fonte: Neto et. al (2010)

A Figura 12 representa a disposição do quadrado. Cada vértice desse quadrado é representado por um nível de cada fator, sendo o nível – e o nível +. A diferença entre os dois níveis do mesmo fator é denominado efeito principal. O efeito de interação é o contraste entre as duas diagonais, considerando + a diagonal que liga o ensaio (– , –) ao ensaio (+, +).

O gráfico de interação entre os fatores é utilizado para representar a diferença de comportamento de um determinado fator nos demais níveis de outro fator com respeito a característica de interesse (ruído).

A Figura 13 representa o gráfico de interação. O eixo X do plano cartesiano representa um fator A. O eixo Y representa a variável resposta. O outro fator B é representado pela retas, sendo uma para o nível – e a outra para o nível +. Neste caso se as linhas não forem paralelas existe interação entre os fatores (LOVIE, 2014).

Figura 13: Representação gráfica da interação entre os fatores A e B (Fator A:Fator B)

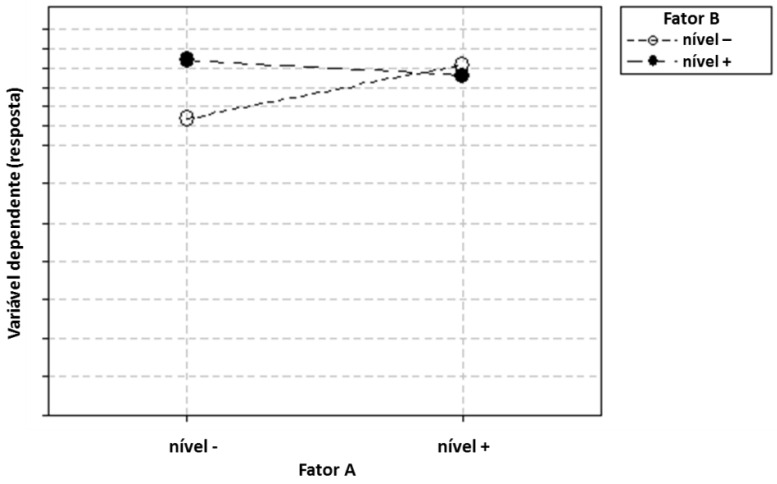


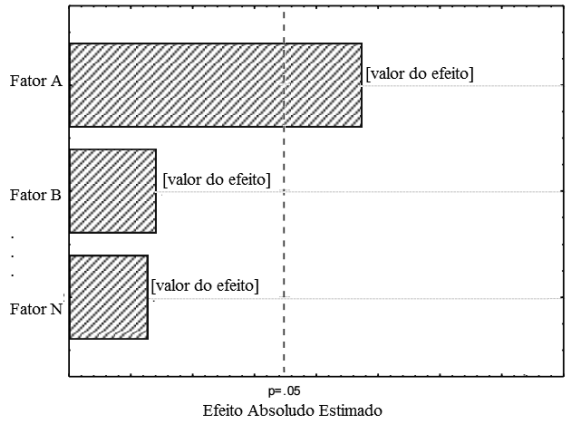
Gráfico de Pareto

O gráfico de Pareto é composto por um diagrama de barra horizontais, em que ordena os fatores e suas interações em ordem decrescente, permitindo identificar quais são os estatisticamente significativos ou não. O diagrama de Pareto é muito útil para analisar um grande número de fatores e para apresentar os resultados de um experimento para um público que não está familiarizado com a terminologia estatística padrão (STATISTICA, 2015).

A Figura 14 apresenta como deve ser construído um gráfico de Pareto. O eixo Y representa todos os fatores e interações entre os fatores, porém de forma ordenada do mais significativo para o menos significativo. O eixo X representa o valor de cada efeito absoluto estimado que é calculado via teste t-student, retirado da tabela ANOVA. A linha tracejada na vertical mostra o p-valor que foi aplicado durante a análise de variância. As barras que

ultrapassam a linha tracejada são estatisticamente significativas. Os efeitos que estão do lado esquerdo a linha vertical tracejada não são considerados estatisticamente significativos.

Figura 14: Representação do gráfico de Pareto



5.2.13 Passo 13: Comparação das médias entre os grupos

A média de uma população é uma de suas características mais importantes. É muito comum desejarmos tomar decisões a seu respeito, por exemplo, quando são comparadas duas amostras ou dois tratamentos.

Inicialmente, são formuladas as hipóteses nula (H_0) e alternativa (H_1), como se segue:

- H_0: As médias das aferições são iguais ($\mu_{amostra\ 1} = \mu_{amostra\ 2}$)
- H_1: As médias das aferições são diferentes ($\mu_{amostra\ 1} \neq \mu_{amostra\ 2}$)

A interpretação das variáveis enunciadas dá-se da seguinte forma: H_0 representa que as médias são significativamente iguais; logo, concluir-se-ia que não haveria diferenças entre as medidas obtidas entre as duas amostras. O inverso ocorre ao analisar H_1, na qual interpretar-se-ia que haveria diferenças significativas entre as médias obtidas entre as duas amostras (MARÔCO, 2011).

Em todos os tipos de testes t necessitamos relatar o valor de "t" que chamado de razão crítica assim como do p-valor, pois assim pode-se identificar a veracidade da probabilidade (p). Com essas informações pode afirmar se a diferença ocorre na realidade ($p < 0,05$) ou ela existe apenas ao acaso ($p > 0,05$).

A comparação de médias de 2 grupos de amostras normais pode também ser usado os testes de Tukey, Duncan ou testes não parametrizados como o de Kruskal Wallis. O teste de comparação de médias de 2 grupos t-student pode ser utilizado no software R, chamando a função t.test (VENABLES, ET Al. 2015).

5.2.14 Passo 14: Dominância de Pareto

Para analisar e verificar quais ambientes computacionais são melhores que os demais, utilizou-se a teoria denominada Dominância de Pareto (ZITZLER, ET AL., 2001). No contexto de dominância de Pareto, um elemento domina outros se ele for melhor que os demais em um critério e não for pior que eles em nenhum dos outros critérios. Porém, neste trabalho, o conceito foi estendido para o que será chamado de "dominância estatística" de Pareto.

Na dominância estatística de Pareto, dado duas soluções *a* e *b*, diz-se que *a* domina *b* ($a \prec b$) com um nível de 5% de significância se as seguintes condições forem satisfeitas:

- pode-se afirmar pelos indícios estatísticos que a solução *a* é superior a *b* em pelo menos um critério;
- não existe evidência estatística de que a solução *a* é diferente da solução *b* para os demais critérios.

5.3 Considerações finais do Capítulo

A metodologia proposta para comparação de ambientes virtuais na computação em nuvem usando análise estatística guia todos as etapas para a experimentação com algoritmos. A metodologia proposta é aplicada no Capítulo 6 em um estudo de caso.

Capítulo 6

Estudo de caso

Este capítulo apresenta um estudo de caso para a aplicação da metodologia proposta no Capítulo 5 para comparação de ambientes virtuais na Computação em Nuvem.

6.1 Objetivos da experimentação

O objetivo dos experimentos realizados é a determinação dos fatores que influenciam o desempenho de algoritmos computacionais em ambientes virtuais na Computação em Nuvem. Busca-se descobrir como, quando e quanto o aumento de desempenho dos algoritmos em ambientes virtuais é determinado pela configuração do ambiente e como os parâmetros de configuração podem influenciar-se mutuamente.

Com base nisso, queremos responder às seguintes indagações:

- I_0 – o fator *quantidade de núcleos* afeta o desempenho de um determinado algoritmo;
- I_1 – o fator *quantidade de memória (GB)* afeta o desempenho de um determinado algoritmo;
- I_2 – o fator *sistema operacional* afeta o desempenho de um determinado algoritmo;
- I_3 – o fator *máquina virtual* afeta o desempenho de um determinado algoritmo;

- I_4 – a interação do fator *quantidade de núcleos* com os demais fatores afeta o desempenho de um determinado algoritmo;
- I_5 – a interação do fator *quantidade de memória (GB)* com os demais fatores afeta o desempenho de um determinado algoritmo;
- I_6 – a interação do fator *sistema operacional* com os demais fatores afeta o desempenho de um determinado algoritmo;
- I_7 – a interação do fator *máquina virtual* com os demais fatores afeta o desempenho de um determinado algoritmo;

6.2 Informações técnicas do experimento

Como a metodologia proposta no Capítulo 5 é genérica. Pode ser aplicada para qualquer benchmark ou algoritmo. O *benchmark* utilizado foi o *Apache Hadoop.* Esse *benchmark* é um *framework* que permite o processamento distribuído de grandes conjuntos de dados em *clusters* de computadores que utilizam modelos de programação simples.

Foi escolhido o *benchmark Apache Hadoop*, primeiramente por ser um *framework* de código aberto e ser utilizado por várias organizações como ferramenta de pesquisa para obtenção e processamento de informações relevantes em grandes massas de dados. E segundo por lidar com o conceito de processamento distribuído em *cluster* de forma eficiente e e conceito de *Big data* de forma simples e robusta.

6.2.1 *Apache Hadoop*

Segundo Apache, (2014), o *Hadoop* foi projetado e desenvolvido para fornecer serviço de escalabilidade para milhares de máquinas, em que cada máquina oferece serviços de computação e armazenamento local. Em vez de confiar em *hardware* para proporcionar a alta disponibilidade, o *Hadoop* por si só é responsável por detectar e tratar falhas na camada de aplicação, de modo a fornecer um serviço altamente disponível no topo de um conjunto de computadores.

O projeto do *Apache Hadoop* inclui os seguintes módulos:

- *Hadoop Common*: Os utilitários que suportam acoplar outros módulos do *Hadoop*.

- *Hadoop Distributed File System (HDFS™)*: Um sistema de arquivos distribuído que fornece acesso de alto rendimento para os dados da aplicação.
- *Hadoop FIO*: *framework* para escalonamento de trabalho e gestão de recursos de *cluster*.
- *Hadoop MapReduce*: um sistema baseado em Yarn (Apache, 2014) para processamento paralelo de grandes conjuntos de dados.

6.2.2 Algoritmos de teste

Foram realizados testes em 5 algoritmos pertencentes ao *benchmark* do *Apache Hadoop.*

Algoritmo *Sudoku*

O *Apache Sudoku* tem como objetivo receber um arquivo de dados com as informações de um tabuleiro como apresentado na Figura 15a e apresentar uma solução para a entrada como a Figura 15b. O Algoritmo possui uma estrutura com complexidade algorítmica da ordem de $O(n^2)$, ou seja, dobrando o tamanho da entrada, o tempo de execução em média do algoritmo quadruplica (HADOOP EXAMPLES, 2014).

O algoritmo permite solucionar problemas com uma matriz de dimensão assimétrica (não quadrada). E a execução do algoritmo é dado pela seguinte linha `bin/hadoop jar hadoop-examples-1.1.1.jar sudoku puzzle1.dta`

Figura 15: Entrada para o algoritmo: (a) matriz de entrada para o algoritmo; (b) matriz de saída do algoritmo

8	5		3	9				
		2						
		6		1				2
		4			3		5	9
		8	9		1	4		
3	2		4			8		
9				8		5		
							2	
				4	5		7	8

(a)

8	5	1	3	9	2	6	4	7
4	3	2	6	7	8	1	9	5
7	9	6	5	1	4	3	8	2
6	1	4	8	2	3	7	5	9
5	7	8	9	6	1	4	2	3
3	2	9	4	5	7	8	1	6
9	4	7	2	8	6	5	3	1
1	8	5	7	3	9	2	6	4
2	6	3	1	4	5	9	7	8

(b)

Algoritmo *Pi*

O algoritmo *Pi* pertencente ao *benchmark* do *Hadoop* é um programa que usa o Map-Reduce para estimar o valor do *Pi* usando o método de quase-Monte Carlo.

A configuração utilizada do algoritmo é dado pela seguinte linha `hadoop jar /share/hadoop/mapreduce/hadoop-mapreduce-examples-2.4.1.jar pi 5 1000`. Pode-se observar que o problema em questão é resolvido usando 5 *Maps*, cada uma para computar 1000 pontos por valor parametrizados. Esses parâmetros podem ser alterados para melhorar o valor de *Pi* (HADOOP, 2014).

A Figura 16 apresenta a saída para a execução do exemplo acima. Pode-se verificar informações da quantidade de *Maps* utilizados, e informações de saída como: tempo utilizado de GC, tempo gasto de CPU, memória física, memória virtual, total *heap* utilizada e tempo gasto de execução.

Figura 16: Saída do algoritmo *Pi*

```
Number of Maps = 5
Samples per Map = 1000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Starting Job 14/09/03 09:54:35
INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
14/09/03 09:54:36 INFO input.FileInputFormat: Total input paths to process: 5
14/09/03 09:54:36 INFO mapreduce.JobSubmitter: number of splits:5
...
14/09/03 09:54:49 INFO mapreduce.Job: map 0% reduce 0%
14/09/03 09:55:30 INFO mapreduce.Job: map 40% reduce 0%
14/09/03 09:55:43 INFO mapreduce.Job: map 60% reduce 0%
14/09/03 09:55:44 INFO mapreduce.Job: map 100% reduce 0%
14/09/03 09:56:08 INFO mapreduce.Job: map 100% reduce 100%
14/09/03 09:56:11 INFO mapreduce.Job: Job job_140_0071 completed successfully
14/09/03 09:56:13 INFO mapreduce.Job: Counters: 49
     File System Counters
     ...
     Job Counters
     ...
     Map-Reduce
          Framework Map input records=5
          ...
          GC time elapsed (ms)=934
          CPU time spent (ms)=9352
          Physical memory (bytes) snapshot=880386048
          Virtual memory (bytes) snapshot=967434240
          Total committed heap usage (bytes)=622153728
     Shuffle Errors
Job Finished in 98.937 seconds
Estimated value of Pi is 3.14160000000000000000
```

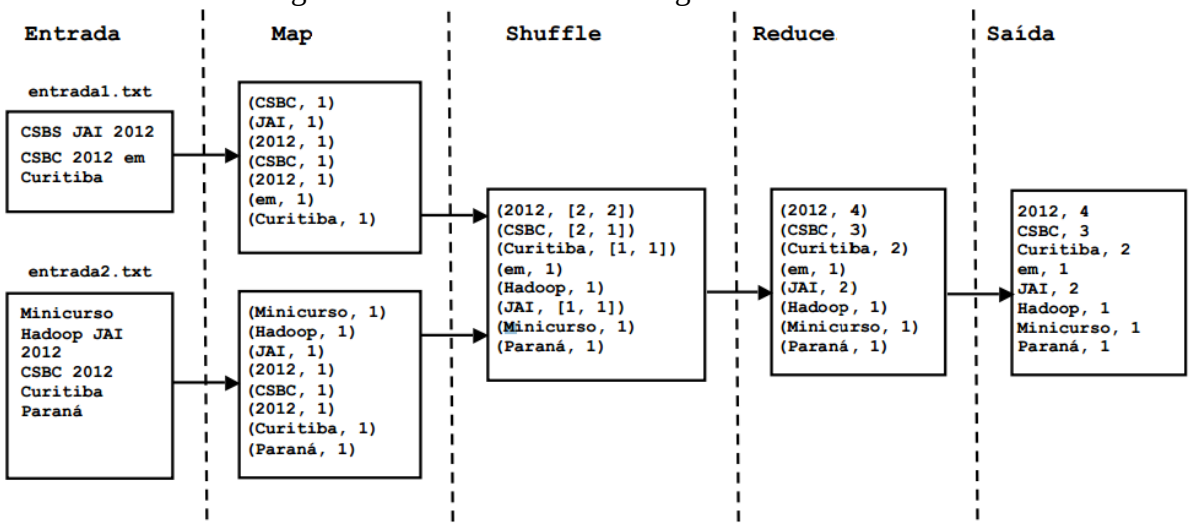
Algoritmo *WordCount*

O algoritmo *WordCount* pertencente ao *benchmark* do *Hadoop* é um programa que implementa a função de *Map* e *Reduce* que são utilizadas para contar a quantidade de palavras em um determinado texto.

Para cada arquivo de entrada (Figura 17), a etapa de *Map* contará as palavras existentes e agrupará em uma estrutura (palavra, quantidade), a etapa de *Shuffle* unirá as duas partes da etapa anterior em (`palavra, [quantidade1, ..., quantidadeN]`). Por fim, a etapa de *Reduce* somará as quantidades e apresentará para o usuário como reposta.

A execução do algoritmo é dado pela seguinte linha `hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.4.1.jar wordcount /texto.txt /saidaTexto`. O arquivo texto utilizado para os testes contém informações de aproximadamente 106MB. Ao todo o arquivo possui aproximadamente 19 milhões de palavras.

Figura 17: Funcionamento do algoritmo *WordCount*



Fonte: (Goldman, et al., 2012)

A Figura 18 apresenta a saída para a execução do exemplo acima. Pode-se verificar informações de saída como: tempo utilizado de GC, tempo gasto de CPU, memória física, memória virtual, total *heap* utilizada e tempo gasto de execução.

Algoritmo *TestDFSIO*

O algoritmo TestDFSIO é um teste de leitura e escrita para HDFS. É útil para tarefas como testes de esforço HDFS, para descobrir os gargalos de desempenho na rede, para produzir e verificar a *performance* do *hardware* e sistema em operações de leitura e escrita.

O teste de leitura de *TestDFSIO* não gera seus próprios arquivos de entrada. Por esta razão, é uma prática conveniente executar primeiro um teste de gravação via *–write* e em seguida um teste de leitura via *–read* (LI, J. ET AL., 2013).

Para o teste de escrita, a execução do algoritmo é dado pela seguinte linha `hadoop org.apache.hadoop.fs.TestDFSIO -write -nrFiles 1 -fileSize 1GB`. Já para o teste de leitura, a execução do algoritmo é dado pela seguinte linha `hadoop org.apache.hadoop.fs.TestDFSIO -read -nrFiles 1 -fileSize 1GB`. Em ambas as linhas o teste está manipulando um único arquivo com 1GB de informação.

Figura 18: Saída do algoritmo *WordCount*

```
14/09/03 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
14/09/03 11:14:12 INFO input.FileInputFormat: Total input paths to process : 1
14/09/03 11:14:12 INFO mapreduce.JobSubmitter: number of splits:1
...
11:14:13 INFO mapreduce.Job: Running job: job_1409725645049_0106
14/09/03 11:14:24 INFO mapreduce.Job: map 0% reduce 0%
14/09/03 11:14:37 INFO mapreduce.Job: map 13% reduce 0%
14/09/03 11:14:43 INFO mapreduce.Job: map 21% reduce 0%
14/09/03 11:14:46 INFO mapreduce.Job: map 25% reduce 0%
14/09/03 11:14:49 INFO mapreduce.Job: map 27% reduce 0%
14/09/03 11:14:52 INFO mapreduce.Job: map 36% reduce 0%
14/09/03 11:14:58 INFO mapreduce.Job: map 46% reduce 0%
14/09/03 11:15:01 INFO mapreduce.Job: map 47% reduce 0%
14/09/03 11:15:04 INFO mapreduce.Job: map 52% reduce 0%
14/09/03 11:15:07 INFO mapreduce.Job: map 58% reduce 0%
14/09/03 11:15:10 INFO mapreduce.Job: map 59% reduce 0%
14/09/03 11:15:13 INFO mapreduce.Job: map 67% reduce 0%
14/09/03 11:15:19 INFO mapreduce.Job: map 100% reduce 0%
14/09/03 11:15:43 INFO mapreduce.Job: map 100% reduce 100%
14/09/03 11:15:47 INFO mapreduce.Job: Job job_1449_0106 completed successfully
14/09/03 11:15:47 INFO mapreduce.Job: Counters: 49
      File System Counters
          ...
     Job Counters
          ...
     Map-Reduce Framework
          ...
          GC time elapsed (ms)=1899
          CPU time spent (ms)=55117
          Physical memory (bytes) snapshot=310185984
          Virtual memory (bytes) snapshot=332783616
          Total committed heap usage (bytes)=186957824
      Shuffle Errors
          ...
File Input Format Counters Bytes Read=111539491
File Output Format Counters Bytes Written=1254041
```

A Figura 19 apresenta a saída para a execução dos dois exemplos acima. Pode-se verificar informações de saída como: tempo utilizado de GC, tempo gasto de CPU, memória física, memória virtual, total *heap*. Informações referentes a I/O e vazão também podem ser observadas como saída.

Figura 19: Saída do algoritmo *TestDFSIO*

```
...
INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
INFO fs.TestDFSIO: Date & time: Wed Sep 03 03:34:35 BRT 2014
INFO fs.TestDFSIO: Number of files: 1
INFO fs.TestDFSIO: Total MBytes processed: 1024.0
INFO fs.TestDFSIO: Throughput mb/sec: 9.971759665011199
INFO fs.TestDFSIO: Average IO rate mb/sec: 9.971759796142578
INFO fs.TestDFSIO: IO rate std deviation: 0.001891378299681122
INFO fs.TestDFSIO: Test exec time sec: 220.75

INFO fs.TestDFSIO: ----- TestDFSIO ----- : read
INFO fs.TestDFSIO: Date & time: Wed Sep 03 03:37:14 BRT 2014
INFO fs.TestDFSIO: Number of files: 1
INFO fs.TestDFSIO: Total MBytes processed: 1024.0
INFO fs.TestDFSIO: Throughput mb/sec: 10.548218957951338
INFO fs.TestDFSIO: Average IO rate mb/sec: 10.548218727111816
INFO fs.TestDFSIO: IO rate std deviation: 0.0016878044420488402
INFO fs.TestDFSIO: Test exec time sec: 149.969
```

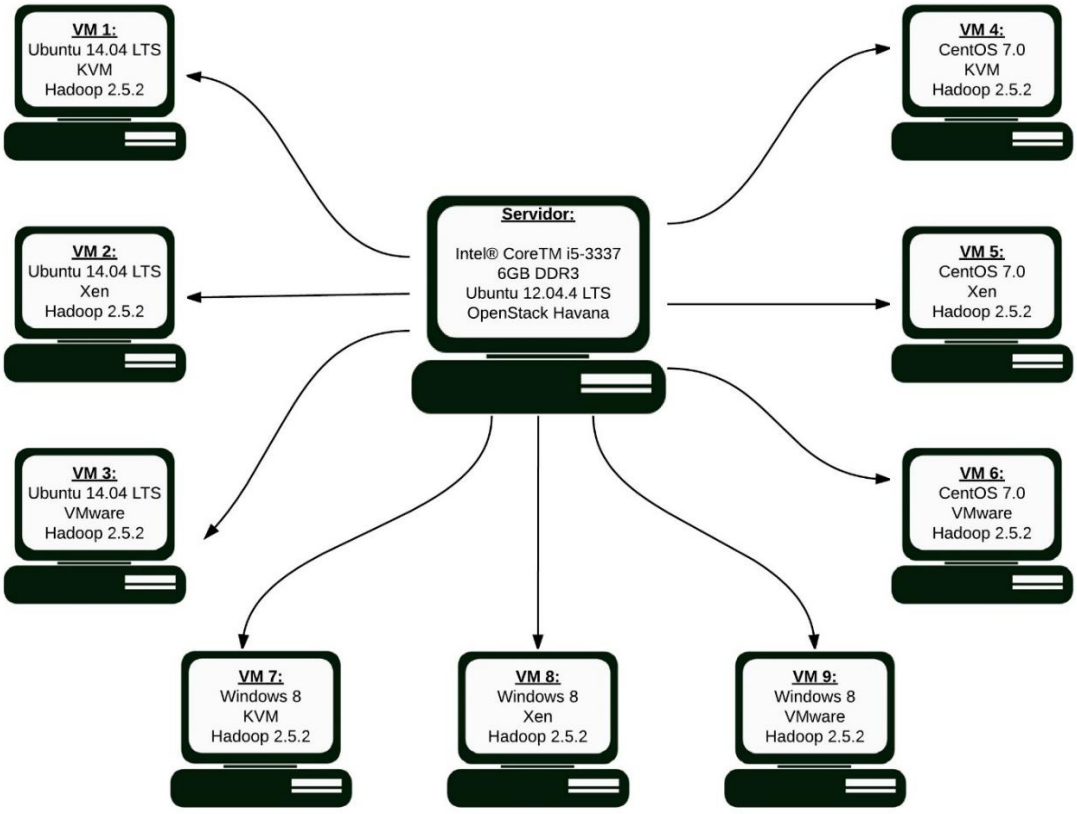
5.2.3 Ambiente Computacional

O computador onde estão sendo realizados os ensaios é composto por um computador com processador Intel® Core™ i5-3337U 1.8GHz, com uma CPU com quatro núcleos, 32K de cache L1, 256k de cache L2, 3072k de cache L3, 6GB de memória DDR3 1600MHz, disco de 1.0 TB ATA e com o sistema operacional Ubuntu 12.04.4 LTS (Kernel 3.8.0-44 generic x86-64).

Para o desenvolvimento da nuvem privada foi instalado o *software* OpenStack Havana no ambiente físico, apresentado na Seção 2.7.3, como plataforma para gerenciamento de toda a infraestrutura computacional de forma totalmente integrada. Segundo OpenStack.org, (2013) o OpenStack permite que seja criado ambientes virtuais em Hyper-V, KVM, Xen, XenServer, VMware, QEMU e LXC.

Dentro do OpenStack foi criado 9 ambientes virtuais. Em cada ambiente virtual foi configurado o *Hadoop* 2.5.2 no modo pseudo-distribuído e por fim, executado de forma centralizada utilizando somente um *DataNode* e um *TaskTracer*. A Figura 20 apresenta a visão geral do sistema composto pelo 9 ambientes virtuais.

Figura 20: Visão geral do sistema



6.3 Seleção dos fatores de controle, os respectivos níveis e variáveis respostas

Alguns fatores foram fixados, tais como: o tipo do processador, a frequência do *clock*, a linguagem de programação e o compilador. Os fatores não controláveis, como sistemas em execução no segundo plano que podem interferir no tempo de execução de um determinado algoritmo, foram contornados executando os algoritmos aleatoriamente utilizando a repetição (Seção 2.3). Nesse trabalho foi utilizado 35 réplicas.

Para um ambiente virtual a ser analisado foram escolhidos fatores que correspondem a parâmetros de configuração de um ambiente virtual em Computação em Nuvem. Os fatores e os seus respectivos níveis estão apresentados na Tabela 12, e são:

- os níveis para o fator **Núcleo** (quantidade de núcleos de processamento) foram definidos em função do número de núcleos existentes no processador;
- os níveis para o fator **Memória** (quantidade de memória) foram também definidos em função da quantidade de memória existentes no ambiente físico;

- o fator **SO** (tipo de sistema operacional) representa três tipos de sistemas diferentes: o Ubuntu 14.04 LTS 64bits (*kernel* 3.13.6-24-generic), o CentOS 7.0 64bits (*kernel* 3.10.0-123.4.2.el7.x86-64) e o Windows 8.0 64bit;
- o fator **VM** (tipo de máquina virtual) representa três tipos de ambientes de virtualização de hardware: KVM (*Kernel Virtual Machine*), Xen, VMware;

Tabela 12: Níveis dos fatores analisados pelo planejamento experimental

#	Fator	Descrição	Nível 1	Nível 2	Nível 3
1	Núcleo	Quantidade de núcleos de processamento	1	2	3
2	Memória	Quantidade de memória	1	3	5
3	SO	Tipo de Sistema Operacional	Ubuntu	CentOS	Windows
4	VM	Tipo de Máquina Virtual	KVM	Xen	VMware

A resposta dos experimentos foi o tempo (*wall clock*) em segundos. Além de informações que foram informadas pelo *Apache Hadoop*, como:

- Tempo em milissegundos gasto com o *Garbage Collector*;
- Tempo em milissegundos gasto com o processamento de CPU;
- Quantidade em bytes de memória física;
- Quantidade em bytes de memória virtual;
- Quantidade em bytes comprometida do uso do Heap;
- Taxa de transferência (*throughput*) em megabytes por segundo (MB/seg);
- Taxa média de I/O em megabytes por segundo (MB/seg).

6.4 Construção da matriz experimental

O planejamento adotado foi o planejamento fatorial completo 3^k, totalmente aleatorizado. A matriz de planejamento do experimento fatorial 3^4 tem 81 linhas (Anexo C) para as corridas experimentais que correspondem às execuções de cada algoritmo (Seção 5.1.2).

6.5 Realização dos experimentos

Para os nove ambientes foram realizados aleatoriamente as execuções de todos os algoritmos seguindo a matriz experimental proposta na Seção 6.4. As variáveis repostas foram coletadas e armazenadas para posteriores análises.

6.6 Transformação dos dados

Para cada algoritmo, foram analisados todas as variáveis respostas para verificar se os dados coletados seguiam uma distribuição normal. Foi verificado que nenhum dos dados coletados se aproximavam de uma distribuição normal. Inicialmente foram realizados testes com outras transformações nos dados como: log, $\log_{10}$, ln, raiz quadrada, raiz cúbica, raiz quarta, inversa. Todas essas transformações não resultaram em uma distribuição normal nos dados coletados.

Todo o processo de transformação dos dados seguem a mesma metodologia. Por isso, nesse momento focaremos somente na variável tempo. Para as demais variáveis respostas a transformação dos dados está apresentada no Anexo D.

Para a variável tempo, a transformação de dados utilizada nesse trabalho foi a transformação de Johnson. Lembrando que o teste de normalidade retornado pelo pacote RE.Johnson é o teste de *Andserson Darling*. Para cada algoritmo o resultado da transformação foi:

- Algoritmo *Sudoku*: família = SU, valor-p = 0.4282, γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048;
- Algoritmo *Pi:* família = SB, valor-p = 0.4587, γ = 1.878965, λ = 753.9305, ε = 30.97788 e η = 0.5000765;
- Algoritmo *WordCount:* família = SB, valor-p = 0.0627, γ = 2.667033, λ = 359.489, ε = 60.50103 e η = 0.8657382;
- Algoritmo *TestDFSIO read:* família = SL, valor-p = 0.2815, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845;
- Algoritmo *TestDFSIO write:* família = SU, valor-p = 0.1176, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894;

6.7 Considerações finais do Capítulo

Os experimentos foram realizados para eliminar erros sistemáticos, determinar os níveis dos fatores, identificar fontes potenciais de variabilidade, identificar as fontes que são controláveis e as não-controláveis e identificar as variáveis importantes. O planejamento adotado foi o planejamento fatorial completo com três níveis (Seção 6.4).

Para cada algoritmo (Seção 5.2.2) foram executados 81 experimentos com 35 repetições. As variáveis repostas foram coletadas e no final foi calculada a média de cada experimento. A transformação dos dados aplicada foi a transformação de Johnson. No final, para cada algoritmo, os dados serão analisados, interpretados e apresentados no Capítulo 6.

Capítulo 7

Análise dos resultados obtidos no estudo de caso

Esse Capítulo apresenta a análise e a interpretação dos resultados obtidos por meio das execuções dos experimentos para cada um dos algoritmos propostos na Seção 6.2.2. A análise e a interpretação dos resultados apresentados nesse Capítulo leva em consideração somente a variável resposta Tempo. As análises das demais variáveis respostas estão apresentadas no Anexo D.

7.1 Algoritmo *Sudoku*

Nesta seção são apresentados os resultados e análise do experimento para o algoritmo *Sudoku*. A variável resposta é constituída pelos tempos que foram obtidos através da execução do algoritmo para encontrar a solução apresentada na Figura 15. Os níveis dos fatores utilizados para a execução da matriz experimental estão apresentados na Tabela 12.

7.1.1 Início da análise dos dados coletados

A Tabela 13 apresenta um resumo dos tempos de execução, em segundos, para cada ambiente virtual, obtidos com o tamanho da amostra de n = 35 replicações: a média $\overline{X}$, o desvio

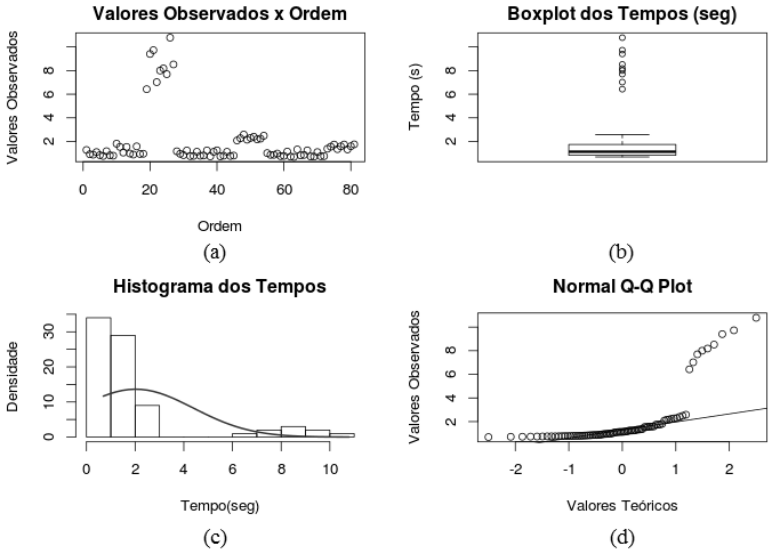
padrão S, a mediana, o valor mínimo, o valor máximo, o coeficiente de variação $\hat{\delta}_x$ de X, o coeficiente de variação $\hat{\delta}_{\overline{X}}$ de $\overline{X}$ e o erro padrão SE. Pode-se verificar que, em média, o ambiente virtual que gastou mais tempo de processamento foi o Windows e KVM. O ambiente virtual que gastou, em média, o menor tempo de processamento foi o Ubuntu e VMware.

Tabela 13: Tempos de execução (*seg*) do algoritmo *Sudoku*

	$\overline{X}$	*S*	*Mediana*	*Min*	*Max*	$\hat{\delta}_x$	$\hat{\delta}_{\overline{X}}$	*SE*
Ubuntu x KVM	0.9575	0.1842	0.8819	0.7756	1.2827	0.1923	0.0108	0.0104
CentOS x KVM	1.2588	0.3561	1.0608	0.8907	1.8147	0.2829	0.0159	0.0201
Windows x KVM	8.4210	1.3681	8.1862	6.4278	10.7880	0.1625	0.0092	0.0771
Ubuntu x Xen	0.9445	0.1886	0.8486	0.7629	1.2403	0.1997	0.0113	0.0106
CentOS x Xen	0.9576	0.2242	0.8060	0.7409	1.2536	0.2342	0.0132	0.0126
Windows x Xen	2.3029	0.1634	2.2776	2.0877	2.5874	0.0709	0.0040	0.0092
Ubuntu x VMware	0.8705	0.1524	0.8667	0.6933	1.1474	0.1751	0.0099	0.0086
CentOS x VMware	0.9273	0.2362	0.8256	0.7239	1.3282	0.2547	0.0144	0.0133
Windows x VMware	1.5620	0.1712	1.5845	1.3307	1.7568	0.1096	0.0062	0.0096

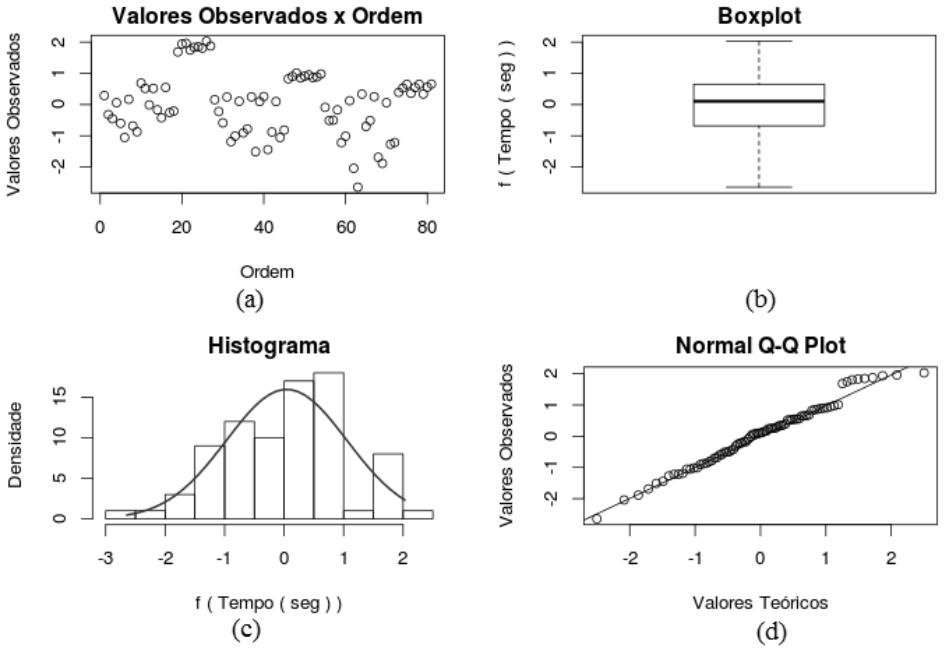
A Figura 21 apresenta, da esquerda para a direita, informações exploratórias sobre os tempos encontrados durante a execução do algoritmo *Sudoku* nos ambientes virtuais analisados. O primeiro gráfico (Figura 21a) apresenta o diagrama da média dos tempos por ordem de execução dos experimentos. O segundo gráfico (Figura 21b) apresenta o diagrama de caixa com todos os tempos, em segundos. Pode-se verificar que a maioria dos tempos (75%) ficou abaixo dos 2 segundos. Os próximos gráficos (Figura 21c e 21d) são respectivamente o histograma e o gráfico de probabilidade normal dos tempos. Pode-se verificar pelo histograma (Figura 21c) que os dados coletados não são normais. O histograma em questão apresenta uma assimetria a direita o que implica na obrigatoriedade de aplicar uma transformação nos dados.

Figura 21: Gráficos exploratórios dos tempos para o algoritmo *Sudoku*



Baseado nessa assimetria, os dados foram transformados utilizando a transformação de Johnson (Seção 5.2.7). A transformação de Johnson foi realizada no software R usando a função RE.Johnson. O resultado da transformação foi: família SU na Equação (3) com as seguintes valores: γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048. Essa transformação é validada com um valor-p = 0.4282 (teste de normalidade *Anderson Darling*, função *ad.test*). A Figura 22 mostra os mesmos gráficos, porém após a transformação sobre os tempos de execução do algoritmo. Essa transformação possibilita uma distribuição mais simétrica (Figura 22c). O diagrama de caixa mostra (Figura 22b) menos valores discrepantes, e os pontos se aproximam da reta no gráfico de probabilidade normal (Figura 22d).

Figura 22: Gráficos exploratórios dos tempos para o algoritmo *Sudoku*, após transformação de Johnson (família = SU, valor-p = 0.4282, γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048)



7.1.2 Análise de Variância

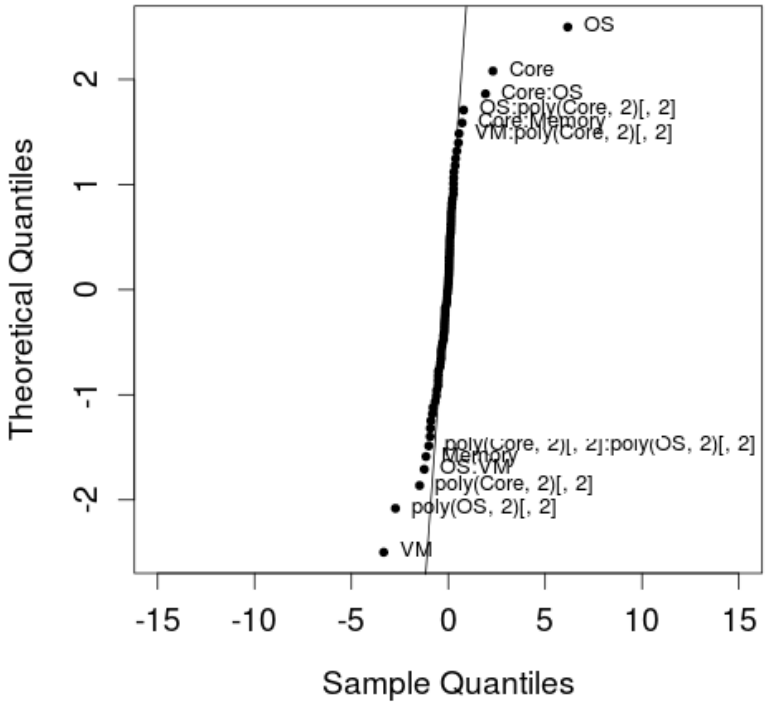
Essas nove configurações dos ambientes virtuais, variando a quantidade de núcleos (unidades de processamento) e a quantidade de memória, pertencem à matriz experimental do fatorial de 3^4, totalizando 81 corridas experimentais testadas.

Os tempos medidos foram os tempos de execução total (*wall clock time*). O número de repetições n = 35 foi previamente estabelecido. O total de execuções do algoritmo *Sudoku* foi de 2835 execuções e o tempo gasto aproximado foi de 1.59 horas de teste.

A análise do fatorial 3^4 se iniciou com a construção do gráfico de probabilidade normal dos efeitos (apresentado na Seção 5.8.3) estimados para a identificação dos efeitos significativos (Figura 23). Mostrando como efeito mais significativo o SO, seguido do efeito principal VM.

A significância estatística dos efeitos foi confirmada através do modelo com interação de segunda ordem e aproximação linear combinada com aproximação quadrática. As interações de ordem maior que dois são, em geral, insignificantes e podem ser usadas para estimar o erro. Portanto, a verificação dos resíduos do modelo de ordem dois mostrou que a variância dos resíduos sob a medida que os valores previstos crescem (Figura 23).

Figura 23: Gráfico de probabilidade normal dos efeitos estimados do fatorial 3^4 dos tempos para o algoritmo *Sudoku,* após transformação de Johnson (família = SU, valor-p = 0.4282, γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048)



Para a interpretação dos efeitos encontrados durante análise de variância é necessário reverter à transformação de Johnson que foi aplicada. A reversão da transformação de Johnson está apresentada na Seção 5.2.7.

A Tabela ANOVA, apresentada na Tabela 14, possui um coeficiente de determinação R^2 igual a 0.9495 com 48 graus de liberdade. Baseado na reversão da transformação de Johnson (Equação 6), o efeito de um determinado fator pode ser interpretado como uma variação percentual do tempo médio (PAIS, 2014). Por exemplo, o efeito principal do fator VM foi igual a -3.3315 (Tabela 14). Agora para reverter essa transformação, esse valor deverá ser informado

na variável *z* da Equação (6) resultando no valor de 0.6330. Ou seja, quando o fator VM muda do nível – para o nível 0 o tempo médio é multiplicado por 0.6330. Contudo, a interpretação de VM com SO foi significativa (Tabela 14), obrigando então, interpretar o efeito de VM juntamente com o fator SO.

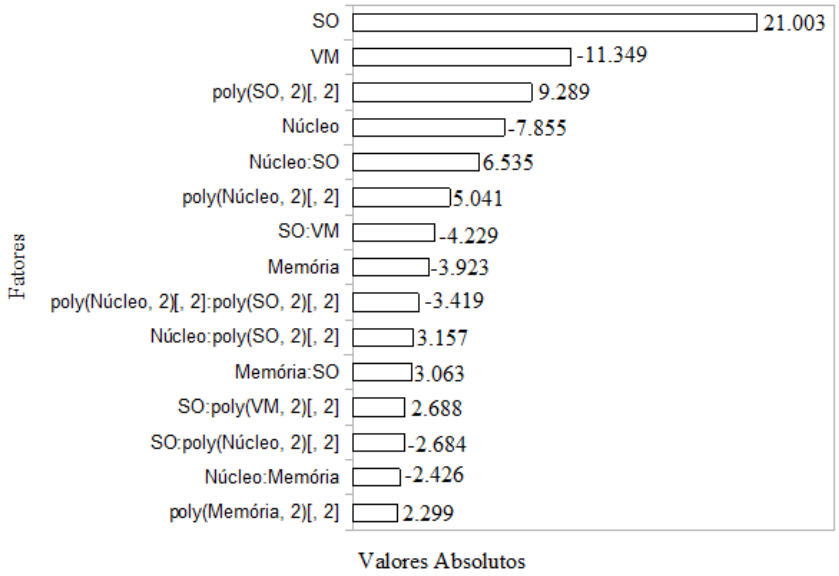
Tabela 14: Modelo para o fatorial 3^4 para o algoritmo *Sudoku* após a transformação de Johnson (família = SU, valor-p = 0.4282, γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048)

	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	-3.87E-01	0.04297	0.03262	1.318	0.193916
Núcleo	2.31E+00	-0.31379	0.03995	-7.855	3.65E-10
Memória	-1.15E+00	-0.15673	0.03995	-3.923	0.000277
SO	6.17E+00	0.83902	0.03995	21.003	< 2e-16
VM	-3.33E+00	-0.45336	0.03995	-11.349	3.43E-15
poly(Núcleo, 2)[, 2]	-1.48E+00	147.995	0.29355	5.041	7.00E-06
poly(Memória, 2)[, 2]	-6.75E-01	0.67502	0.29355	2.299	0.025872
poly(SO, 2)[, 2]	-2.73E+00	272.691	0.29355	9.289	2.67E-12
poly(VM, 2)[, 2]	-5.09E-01	0.50918	0.29355	1.735	0.089241
Núcleo:Memória	7.12E-01	-0.11868	0.04893	-2.426	0.019088
Núcleo:SO	1.92E+00	0.31973	0.04893	6.535	3.80E-08
Núcleo:VM	-4.29E-01	-0.07155	0.04893	-1.462	0.150139
Núcleo:poly(Memória, 2)[, 2]	-3.70E-01	0.45278	0.35953	1.259	0.213982
Núcleo:poly(SO, 2)[, 2]	-9.27E-01	11.349	0.35953	3.157	0.002757
Núcleo:poly(VM, 2)[, 2]	2.80E-01	-0.34322	0.35953	-0.955	0.344544
Memória:SO	-8.99E-01	0.14986	0.04893	3.063	0.003586
Memória:VM	-5.23E-01	-0.08711	0.04893	-1.78	0.081339
Memória:poly(Núcleo, 2)[, 2]	-1.58E-01	0.19319	0.35953	0.537	0.593519
Memória:poly(SO, 2)[, 2]	1.30E-01	0.15894	0.35953	0.442	0.660421
Memória:poly(VM, 2)[, 2]	-3.69E-01	-0.4519	0.35953	-1.257	0.214868
SO:VM	-1.24E+00	-0.20693	0.04893	-4.229	0.000105
SO:poly(Núcleo, 2)[, 2]	7.88E-01	-0.96497	0.35953	-2.684	0.009954
SO:poly(Memória, 2)[, 2]	-1.60E-01	-0.19561	0.35953	-0.544	0.588914
SO:poly(VM, 2)[, 2]	-7.89E-01	0.96652	0.35953	2.688	0.009844
VM:poly(Núcleo, 2)[, 2]	5.55E-01	0.67983	0.35953	1.891	0.064682
VM:poly(Memória, 2)[, 2]	6.71E-02	-0.08222	0.35953	-0.229	0.820082
VM:poly(SO, 2)[, 2]	-9.56E-02	-0.1171	0.35953	-0.326	0.74607
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	1.78E-01	-159.814	264.199	-0.605	0.548096
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	-1.00E+00	-903.398	264.199	-3.419	0.001289
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	-5.97E-01	-537.673	264.199	-2.035	0.047383
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-5.14E-01	-4.63	264.199	-1.752	0.086079
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	7.91E-03	-0.07121	264.199	-0.027	0.978608
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	4.05E-01	-3.64739	264.199	-1.379	0.174279

O nível de significância adotado nos experimentos foi de 5% de significância estatística. A Figura 24 apresenta o gráfico de Pareto para os valores absolutos do percentual estimado de cada efeito no tempo médio do algoritmo *Sudoku*. Os valores apresentados no gráfico foram somente aqueles que ficaram abaixo do nível de significância (valor-p < 0.05). Pode-se verificar que os fatores que tiveram mais impacto no tempo de execução do algoritmo foram o SO, VM,

e a interação entre o sistema operacional e a máquina virtual (SO:VM). Os valores do gráfico de Pareto são referentes ao valor-t (Tabela 14) e foram calculados baseado na Seção 5.2.12.

Figura 24: Gráfico de Pareto com os efeitos estimados em valor percentual absoluto para o tempo de execução para o algoritmo *Sudoku* (*valor-p = 0.05*), após a transformação de Johnson (família = SU, valor-p = 0.4282, γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048)



7.1.3 Testes formais de homoscedasticidade, normalidade e independências

O teste de *Breusch-Pagan* confirmou que os resíduos estavam distribuídos de forma homogênea com *valor-p = 0.0801*. Outros testes de homoscedasticidade também foram aplicados para confirmar a homogeneidade dos dados (teste de *Goldfeld-Quandt* com *valor-p = 0.411* e o teste de *Harrison-McCabe* com *valor-p = 0.064*).

O teste normalidade confirmou que os resíduos se aproximam de uma distribuição normal (teste de *Shapiro-Wilk*, *valor-p = 0.0651*). Outros testes também foram aplicados para confirmar a distribuição normal. O teste de *Kolmogorov-Smirnov* apresentou *valor-p = 0.5715*. Para o teste de normalidade *Anderson-Darling* o *valor-p = 0.1631*.

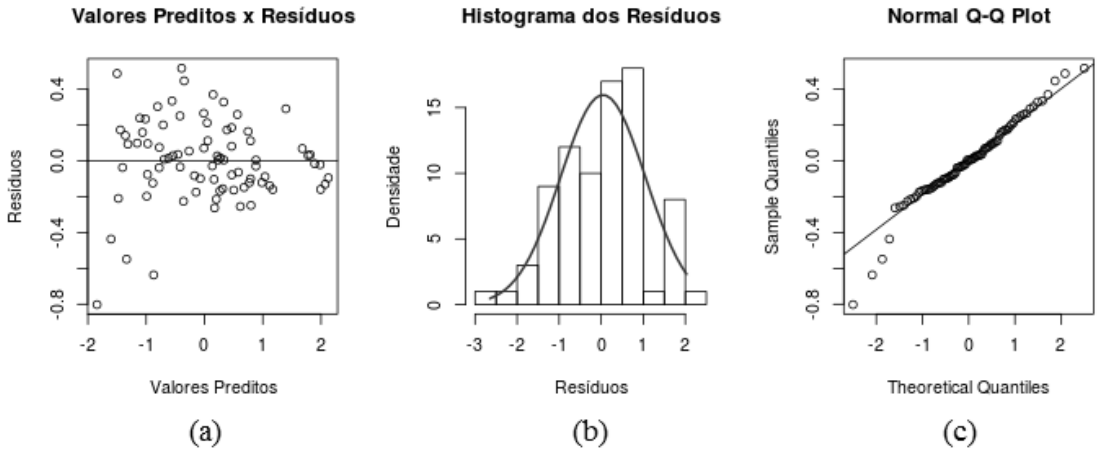
Segundo o teste de auto correlação negativa e positiva (teste de *DurbinWatson*) o *dw = 1.951* e *valor-p = 0.174* , em que, não rejeita o H_0, ou seja, os resíduos são independentes.

7.1.4 Análise dos resíduos

A Figura 25 apresenta a verificação dos resíduos para a variável resposta tempo com os dados transformados. O gráfico, da esquerda para direita, representa a relação entre os resíduos

pelos valores preditos e não apresenta estrutura ou padrão (Figura 25a). O histograma dos resíduos (Figura 25b) que apresenta a distribuição da frequência dos resíduos apresentou aproximadamente simétrico em forma de sino. O gráfico à direita (Figura 25c) que representa a probabilidade normal dos resíduos mostrou que os pontos estão próximos da reta.

Figura 25: Verificação dos resíduos: gráfico resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem de dois do fatorial 3^4 do algoritmo *Sudoku*, após a transformação de Johnson (família = SU, valor-p = 0.4282, γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048)



7.1.5 Análise dos efeitos

Os fatores com efeito principal significativo também fazem parte das interações significativas e, portanto, não devem ser interpretados isoladamente, como por exemplo SO, VM, Núcleo e Memória (Tabela 14). Eles devem ser interpretados em conjunto com os fatores que se interagem.

De acordo com a Tabela 14, as interações que foram estatisticamente significativas foram: núcleo:SO, memória:SO e SO:VM, pois tiveram valores-p inferiores a 0.05. A interação núcleo:memória pode ser considerada significante na prática pois apresentou um valor-p igual a 0.064766, próximo de 0.05 (Tabela 14).

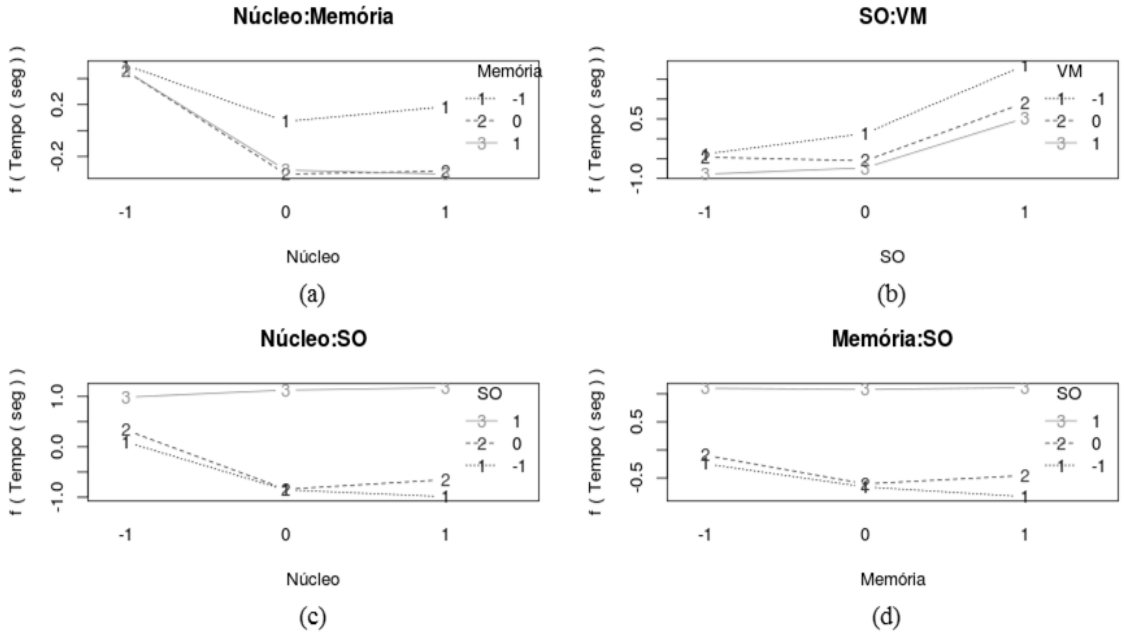
Para o algoritmo *Sudoku* as interações mostram que, para diminuir o tempo de execução do algoritmo basta aumentar a quantidade de memória (Figura 26a). Também pode-se verificar que o tempo de execução depende do tipo de Sistema Operacional e Máquina Virtual (Figuras 26a, 26b, 26c). Os gráficos da parte inferior mostraram que a alteração dos níveis não causou

grande impacto no tempo transformado de execução do algoritmo, nos dois gráficos a linha (3) representada pelo sistema operacional Windows foi superior às demais linhas.

O gráfico da Figura 26b mostra que a mudança do sistema operacional em conjunto com a máquina virtual também afeta a transformação do tempo de execução do algoritmo. Para o nível – (Ubuntu) do fator SO, o tempo transformado de execução é praticamente igual, o que deixa o usuário livre para escolher qualquer nível do fator VM. Já o fator SO no nível 0 (CentOS) o tempo de execução se mostrou praticamente igual, exceto para o VM no nível – (KVM) o qual também apresentou resultado inferior no nível + do fator SO.

Finalmente, o fator SO no nível + (Windows) apresentou todos os resultados maiores sobre os demais níveis. Porém, pode-se dizer que a combinação KVM *versus* Windows apresentou os piores resultados e muito superiores às demais interações analisadas. Essas interpretações podem ser confirmadas com o teste-*t.*

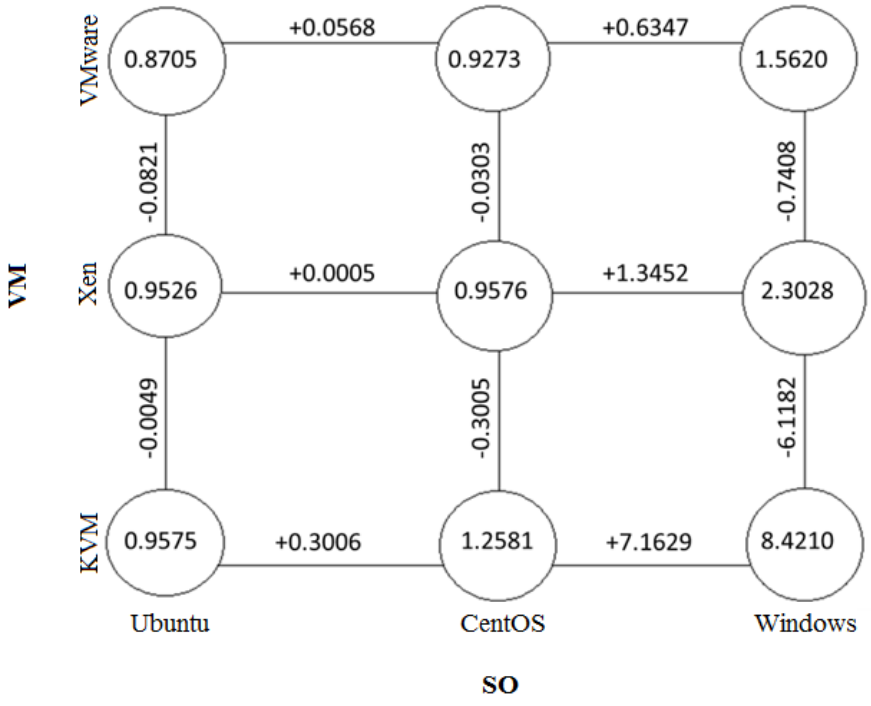
Figura 26: Gráficos de interação dos efeitos significativos, de cima para baixo e da esquerda para direita: núcleo:memória, núcleo:SO, memória:SO e SO:VM, após a transformação de Johnson (família = SU, valor-p = 0.4282, γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048)



Como o efeito de interação SO:VM é significativo, os efeitos principais devem ser interpretados conjuntamente (Figura 27). A melhor forma é criar um diagrama contendo as respostas médias em todas as combinações, podendo concluir que:

- Alterando o SO de Ubuntu para CentOS e CentOS para Windows o tempo de execução do algoritmo aumenta, mas esse efeito é maior pronunciado com a máquina virtual KVM do que com as máquinas virtuais Xen e VMware;
- Trocando-se a máquina virtual VMware por KVM ou Xen diminui-se o tempo de execução do algoritmo e esse efeito é muito mais significativo com o sistema operacional no nível + (Windows) com uma diminuição de aproximadamente 6.859 segundos (6.1182 + 0.7408);
- O melhor desempenho é obtido usando a máquina virtual VMware juntamente com o sistema operacional Ubuntu.
- O pior desempenho do algoritmo foi encontrado no ambiente virtual com a máquina virtual KVM e o sistema operacional Windows com um desempenho 967.37% superior ao melhor ambiente virtual (8.4210 / 0.8705).

Figura 27: Diagrama para interpretação dos resultados do planejamento fatorial 3^4. Os valores dos vértices são os tempos médios.

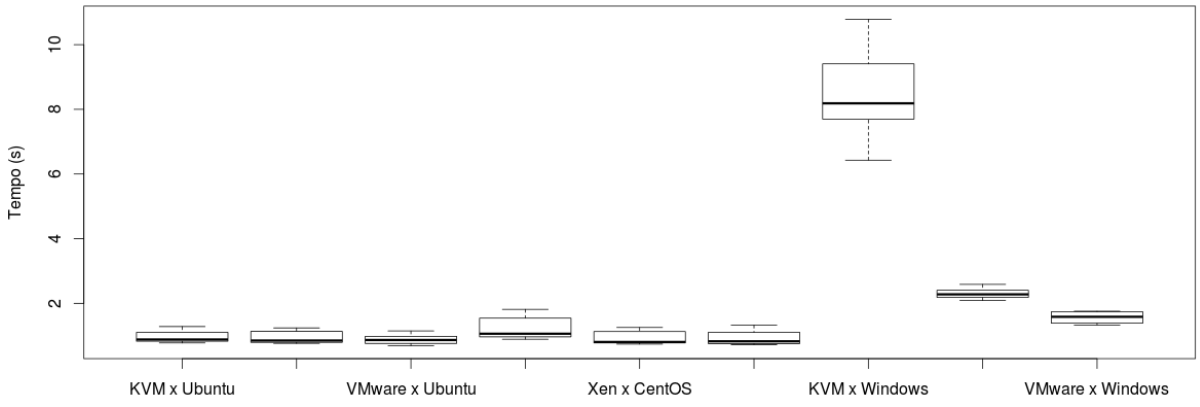


Avaliando as indagações apresentadas na Seção 6.1 e baseando-se nas análises do algoritmo *Sudoku* apresentadas anteriormente pode-se concluir que aceita-se as indagações I_0, I_1, I_2 e I_3. As indagações relacionadas com as interações entre os fatores I_4, I_5, I_6, e I_7 também são aceitas, pois as interações entre núcleo:memória, núcleo:SO, memória:SO e SO:VM foram consideradas estatisticamente significativas a um nível de significância de 5%.

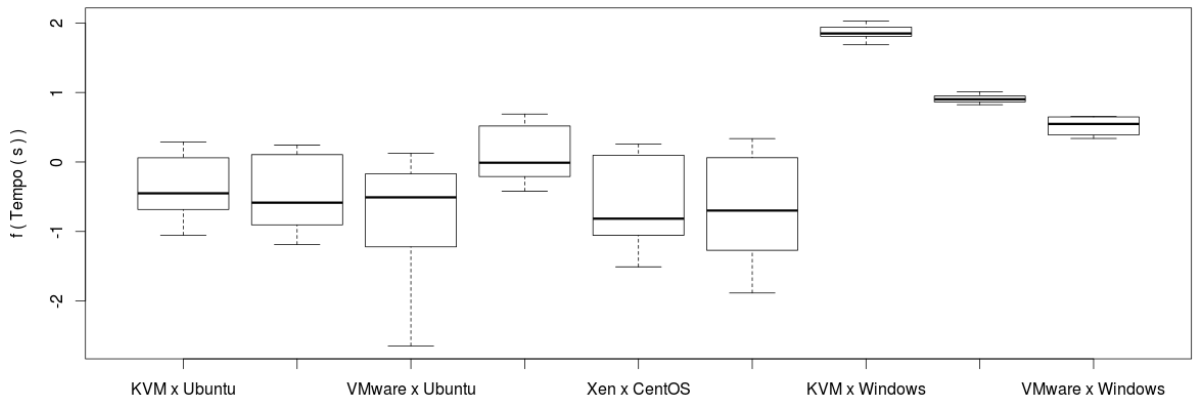
7.1.6 Comparação das médias entre os grupos

Os diagramas de caixa após a transformação de Johnson foram então agrupados pelos fatores SO:VM e são mostrados na Figura 28. Para a interpretação dos dados foi utilizado o teste *t* para a comparação das médias (média dos tempos transformados pelo método de Johnson) de cada fator (WINTER, 2013). Pode-se observar que todos dos tempos coletados para a combinação KVM com Windows e Xen com Windows estão superiores aos demais.

Figura 28: Diagramas de caixa agrupados pela combinação do sistema operacional pela máquina virtual: (a) tempos sem transformação (b) após a transformação de Johnson (família = SU, valor-p = 0.4282, γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048), para o algoritmo *Sudoku*



(a)



(b)

A Tabela 15 apresenta o resultado da aplicação do teste *t* e os valores nela contidos possuem os seguintes significados para cada um dos quesitos analisados:

- +1 : O *fator* da linha é maior que o *fator* da coluna;
- -1: O *fator* da linha é menor que o *fator* da coluna;

- 0: Não tem evidências estatísticas que suportem afirmar que o *fator* da linha é diferente do *fator* da coluna, com 95% de confiança.

Tabela 15: Comparação entre os fatores sistema operacional *versus* máquina virtual, após transformação de Johnson (família = SU, valor-p = 0.4282, γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048), para o algoritmo *Sudoku*

Ambiente	KVM x Ubuntu	Xen x Ubuntu	VMware x Ubuntu	KVM x CentOS	Xen x CentOS	VMware x CentOS	KVM x Windows	Xen x Windows	VMware x Windows
KVM x Ubuntu	--	0	0	-1	0	-1	-1	-1	-1
Xen x Ubuntu	0	--	0	-1	0	-1	-1	-1	-1
VMware x Ubuntu	0	0	--	-1	0	-1	-1	-1	-1
KVM x CentOS	+1	+1	+1	--	+1	-1	-1	-1	-1
Xen x CentOS	0	0	0	-1	--	-1	-1	-1	-1
VMware x CentOS	+1	+1	+1	+1	+1	--	-1	-1	-1
KVM x Windows	+1	+1	+1	+1	+1	+1	--	+1	+1
Xen x Windows	+1	+1	+1	+1	+1	+1	-1	--	+1
VMware x Windows	+1	+1	+1	+1	+1	+1	-1	-1	--

Baseado na Tabela 15, conclui-se que o ambiente virtual (KVM com Windows) possui os piores tempos de execução para o algoritmo *Sudoku*, pois na linha somente aparece o valor +1. Pode-se verificar também que todos dos tempos para o sistema operacional Windows são superiores às distribuições Linux analisadas (Ubuntu e CentOS).

7.1.7 Dominância de Pareto

Focando na variável resposta Tempo (seg) foram aplicados os conceitos da dominância de Pareto aos dados apresentados na Tabelas 15. A Tabela 16 apresenta para cada ambiente virtual quantos ambientes o dominam e quais ambientes virtuais são por ele dominados. Como por exemplo pode-se observar que para o ambiente virtual KVM x Ubuntu não existe nenhum valor positivo na linha (+1), ou seja, não existe nenhum ambiente virtual que o domina. Portanto, existem 5 valores negativos (-1), ou seja, esses valores correspondem aos ambientes virtuais qual são dominados por ele (KVM x CentOS, VMware x Windows, KVM x Windows, Xen x Windows e VMware x Windows). Essa mesma ideia é aplicada para todas as linhas da Tabela 15.

Tabela 16: Dominância estatística de Pareto aplicado a variável resposta Tempo da Tabela 15 para o algoritmo *Sudoku*, após a transformação de Johnson (família = SU, valor-p = 0.4282, γ = -2.096445, ε = 0.7161157, η = 0.6047109 e λ = 0.02192048)

Ambientes virtuais	Quantos o dominam	Quais são por ele dominados
KVM x Ubuntu	0	KVM x CentOS, VMware x Windows, KVM x Windows, Xen x Windows e VMware x Windows
Xen x Ubuntu	0	KVM x CentOS, VMware x Windows, KVM x Windows, Xen x Windows e VMware x Windows
VMware x Ubuntu	0	KVM x CentOS, VMware x Windows, KVM x Windows, Xen x Windows e VMware x Windows
KVM x CentOS	5	VMware x CentOS, KVM x Windows, Xen x Windows, VMware x Windows
Xen x CentOS	0	KVM x CentOS, VMware x Windows, KVM x Windows, Xen x Windows e VMware x Windows
VMware x CentOS	5	KVM x Windows, Xen x Windows e VMware x Windows
KVM x Windows	8	∅
Xen x Windows	7	KVM x Windows
VMware x Windows	6	KVM x Windows e Xen x Windows

Na Tabela 17 foram aplicados os conceitos de dominância de Pareto e foram encontradas 6 fronteiras para o algoritmo *Sudoku* para a variável resposta Tempo. Os ambientes pertencentes a primeira fronteira são considerados em média dominantes aos demais, ou seja, gastam em média gastam menos tempo de processamento. Pode-se observar pelas fronteiras de Pareto que o sistema virtual Windows obteve em média os maiores tempos de processamento independe de máquina virtual.

Tabela 17: Classificação dos ambientes virtuais nas fronteiras de Pareto para a variável resposta Tempo

Fronteira	Ambientes virtuais
1	KVM x Ubuntu, Xen x Ubuntu, VMware x Ubuntu, Xen x CentOS
2	KVM x CentOS
3	VMware x CentOS
4	VMware x Windows
5	Xen x Windows
6	KVM x Windows

7.2 Algoritmo *Pi*

Nesta seção são apresentados os resultados e análise do experimento para o algoritmo *Pi*. Os níveis dos fatores utilizados para a execução da matriz experimental estão apresentados na Tabela 12. Nesse algoritmo foi analisado como variáveis respostas além do tempo de

execução do algoritmo informações como: tempo gasto com o uso do *Garbage Collector*, tempo despendido como o uso da CPU (*CPU*), total de memória física, total de memória virtual e o total de *Heap*, porém a análise dessas outras variáveis respostas está apresentada no Anexo D.

7.2.1 Início da análise dos dados coletados

A Tabela 18 apresenta um resumo das informações coletadas durante a execução do algoritmo *Pi* obtidas com o tamanho da amostra de n = 35 repetições: a média $\bar{X}$, o desvio padrão *S*, a mediana, o valor mínimo e o valor máximo, o coeficiente de variação $\hat{\delta}_x$ de *X*, o coeficiente de variação $\hat{\delta}_{\bar{X}}$ de $\bar{X}$ e o erro padrão *SE*. Pode-se verificar que, em média, o ambiente virtual que gastou mais tempo de processamento foi o CentOS e KVM. O ambiente virtual que gastou, em média, o menor tempo de processamento foi o Windows e VMware.

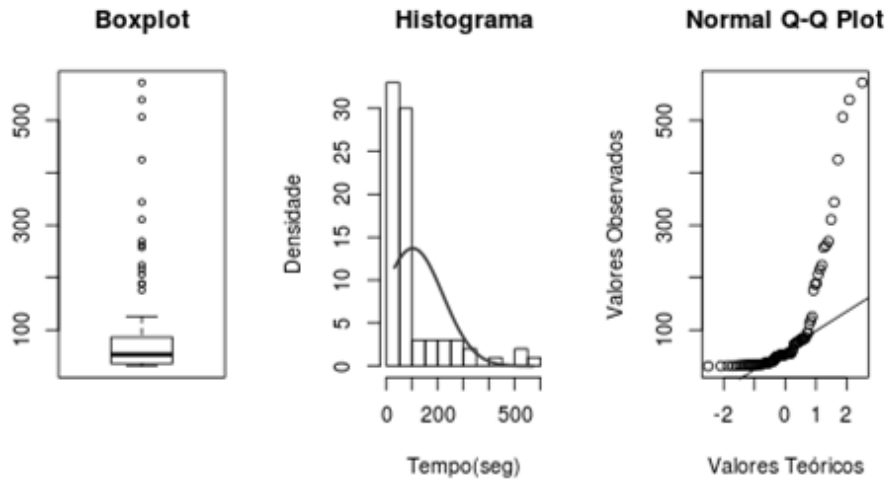
Tabela 18: Tempos de execução gasto (*seg*) para o algoritmo *Pi*

	$\bar{X}$	*S*	*Mediana*	*Min*	*Max*	$\hat{\delta}_x$	$\hat{\delta}_{\bar{X}}$	*SE*
Ubuntu x KVM	132.762	148.288	52.558	32.473	425.346	1.117	0.372	49.429
CentOS x KVM	208.462	249.006	54.628	35.352	572.384	1.194	0.398	83.002
Windows x KVM	125.669	46.745	116.876	82.462	188.876	0.372	0.124	15.582
Ubuntu x Xen	103.029	96.942	51.168	31.344	261.629	0.941	0.314	32.314
CentOS x Xen	122.547	128.255	49.268	31.049	344.094	1.047	0.349	42.752
Windows x Xen	61.162	25.007	55.468	37.789	111.325	0.409	0.136	8.336
Ubuntu x VMware	74.670	14.439	75.264	53.032	94.352	0.193	0.064	4.813
CentOS x VMware	47.206	16.375	50.507	31.605	80.826	0.347	0.116	5.458
Windows x VMware	40.710	7.979	36.569	33.159	54.484	0.196	0.065	2.660

A Figura 29 apresenta, da esquerda para a direita, informações exploratórias sobre os tempos encontrados durante a execução do algoritmo *Pi* nos ambientes virtuais analisados. O primeiro gráfico (Figura 29a) apresenta o diagrama de caixa com as respectivas respostas e que as maiorias dos ensaios (mais que 75%) tiveram tempo inferior a 150 segundos. O gráfico da normal (Figura 29c) mostra que os dados coletados não se aproximam de uma distribuição normal o que caracteriza a necessidade de se aplicar uma transformação nos dados.

Baseado nessa assimetria, os dados foram transformados utilizando a transformação de Johnson como resultado apresentado na Seção 5.2.7. A transformação de Johnson foi realizada no software R usando a função RE.Johnson. O resultado da transformação foi: família = SB, γ = 1.878965, λ = 753.9305, ε = 30.97788 e η = 0.5000765. Essa transformação é validada com um p-valor = 0.4587 (teste de normalidade de *Anderson Darling*, função *ad.test*).

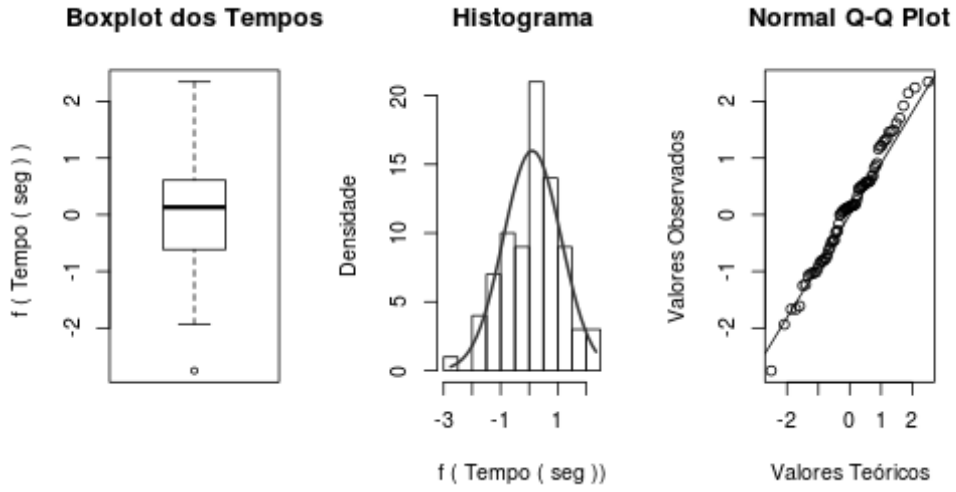
Figura 29: Gráficos exploratórios dos tempos para o algoritmo *Pi*



(a) (b) (c)

A Figura 30 mostra os mesmos gráficos, porém após transformação sobre os tempos de execução do algoritmo. Essa transformação possibilita uma distribuição mais simétrica (Figura 30b). O diagrama de caixa mostra (Figura 30a) menos valores discrepantes, e os pontos se aproximam da reta no gráfico de probabilidade normal (Figura 30c).

Figura 30: Gráficos exploratórios dos tempos para o algoritmo *Pi*, após transformação de Johnson (família = SB, $\gamma = 1.878965$, $\lambda = 753.9305$, $\varepsilon = 30.97788$ e $\eta = 0.5000765$)



7.2.2 Análise de Variância

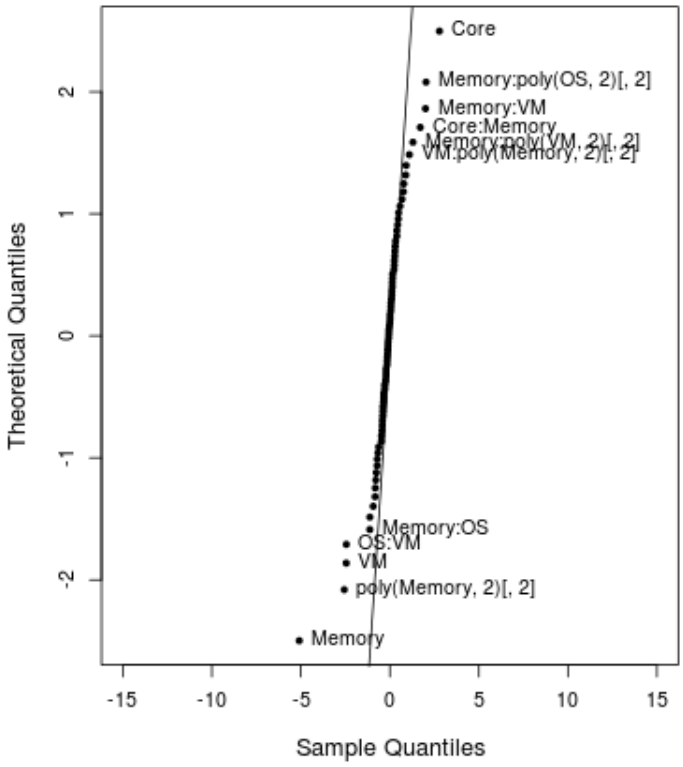
Essas nove configurações dos ambientes virtuais, variando a quantidade de núcleos (unidades de processamento) e a quantidade de memória pertencem a matriz experimental do fatorial de 3^4, totalizando 81 corridas experimentais a serem testadas.

Os tempos medidos foram os tempos de execução total (*wall clock time*). O número de repetições n = 35 foi previamente estabelecido. O total de execuções do algoritmo *Pi* foi de 2835 execuções e o tempo gasto aproximado foi de 80.17 horas de teste.

A análise do fatorial 3^4 se iniciou com a construção do gráfico de probabilidade normal dos efeitos (apresentado na Seção 5.8.3) estimados para a identificação dos efeitos significativos (Figura 31). O efeito mais significativo foi a memória seguida do núcleo.

A significância estatística dos efeitos foi confirmada através do modelo com interação de segunda ordem e aproximação linear combinada com aproximação quadrática. As interações de ordem maior que dois são, em geral, insignificantes e podem ser usadas para estimar o erro. Portanto, a verificação dos resíduos do modelo de ordem dois mostrou que a variância dos resíduos sob a medida que os valores previstos crescem (Figura 31).

Figura 31: Gráfico de probabilidade normal dos efeitos estimados do fatorial 3^4 dos tempos para o algoritmo *Pi,* após transformação de Johnson (família = SB, $\gamma = 1.878965$, $\lambda = 753.9305$, $\varepsilon = 30.97788$ e $\eta = 0.5000765$)



Para a interpretação dos efeitos encontrados durante análise de variância é necessário reverter à transformação de Johnson que foi aplicada. A reversão da transformação de Johnson está apresentada na Seção 5.2.7.

A Tabela ANOVA é apresentada na Tabela 19, possuindo um coeficiente de determinação R^2 igual a 0.9093 com 48 graus de liberdade. A interpretação de VM com SO foi

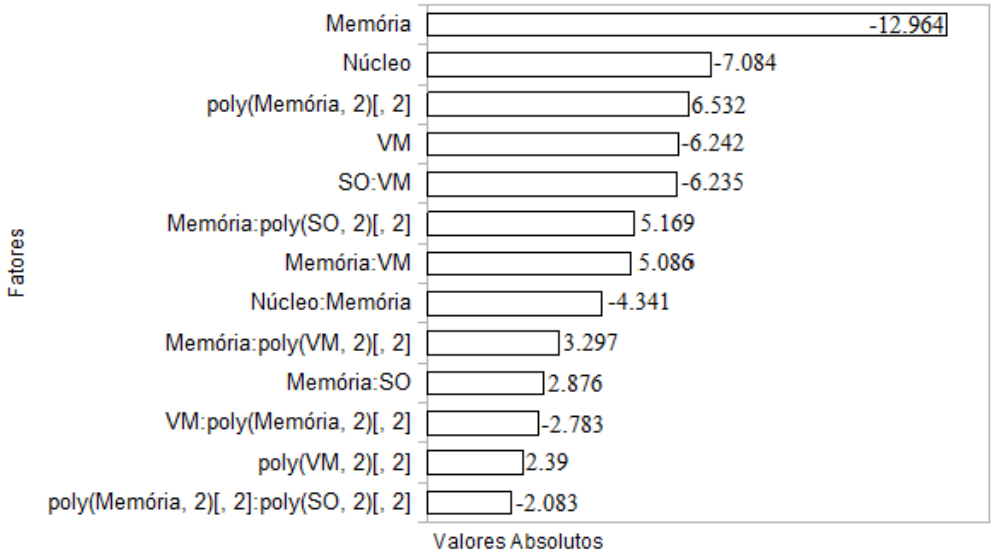
significativa (Tabela 19), portanto, o efeito de VM deve ser interpretado em conjunto com o fator SO.

Tabela 19: Modelo para o fatorial 3^4 para o algoritmo *Pi* após a transformação de Johnson dos tempos para a variável resposta Tempo (seg), após transformação de Johnson (família = SB, $\gamma = 1.878965$, $\lambda = 753.9305$, $\varepsilon = 30.97788$ e $\eta = 0.5000765$)

	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	-0.983950512	0.10933	0.04364	2.506	0.01567
Núcleo	2.781922044	-0.37857	0.05344	-7.084	5.47E-09
Memória	-5.091068573	-0.69281	0.05344	-12.964	< 2e-16
SO	-0.033070053	-0.0045	0.05344	-0.084	9.33E-01
VM	-2.451435063	-0.3336	0.05344	-6.242	1.07E-07
poly(Núcleo, 2)[, 2]	-0.437347566	0.43735	0.39271	1.114	2.71E-01
poly(Memória, 2)[, 2]	-2.565206213	2.56521	0.39271	6.532	3.84E-08
poly(SO, 2)[, 2]	-0.75853713	0.75854	0.39271	1.932	5.93E-02
poly(VM, 2)[, 2]	-0.938524486	0.93852	0.39271	2.39	0.02083
Núcleo:Memória	1.704915357	-0.28415	0.06545	-4.341	7.27E-05
Núcleo:SO	0.435275429	0.07255	0.06545	1.108	0.27322
Núcleo:VM	-0.406293399	-0.06772	0.06545	-1.035	0.30605
Núcleo:poly(Memória, 2)[, 2]	-0.673236459	0.82454	0.48097	1.714	0.09292
Núcleo:poly(SO, 2)[, 2]	-0.712201837	0.87227	0.48097	1.814	0.076
Núcleo:poly(VM, 2)[, 2]	-0.787632947	0.96465	0.48097	2.006	0.05055
Memória:SO	-1.129499887	0.18825	0.06545	2.876	0.00599
Memória:VM	1.997539393	0.33292	0.06545	5.086	6.00E-06
Memória:poly(Núcleo, 2)[, 2]	-0.225763607	0.2765	0.48097	0.575	0.56806
Memória:poly(SO, 2)[, 2]	2.02983318	2.48603	0.48097	5.169	4.53E-06
Memória:poly(VM, 2)[, 2]	1.2948079	1.58581	0.48097	3.297	0.00184
SO:VM	-2.448572812	-0.40809	0.06545	-6.235	1.10E-07
SO:poly(Núcleo, 2)[, 2]	-0.392502955	0.48072	0.48097	0.999	0.32258
SO:poly(Memória, 2)[, 2]	-0.716452795	-0.87747	0.48097	-1.824	0.07433
SO:poly(VM, 2)[, 2]	0.571304394	-0.6997	0.48097	-1.455	0.15225
VM:poly(Núcleo, 2)[, 2]	0.297905944	0.36486	0.48097	0.759	0.45181
VM:poly(Memória, 2)[, 2]	1.093097159	-1.33876	0.48097	-2.783	0.00767
VM:poly(SO, 2)[, 2]	0.778396691	0.95334	0.48097	1.982	0.05321
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.250553501	-2.25498	3.53443	-0.638	0.5265
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.060145476	0.54131	3.53443	0.153	0.87892
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.486980935	4.38283	3.53443	1.24	0.22099
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-0.817884039	-7.36096	3.53443	-2.083	0.04264
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	0.678644383	-6.1078	3.53443	-1.728	0.0904

O nível de significância adotados nos experimentos foi de 5% de significância estatística. A Figura 32 apresenta o gráfico de Pareto para os valores absolutos do percentual estimado de cada efeito no tempo médio do algoritmo *Pi*. Os valores apresentados no gráfico foram somente aqueles que ficaram abaixo do nível de significância. Pode-se verificar que os fatores que tiveram mais impacto no tempo de execução do algoritmo foram o Memória e Núcleo. A interação entre o sistema operacional e a máquina virtual (SO:VM) também foi estatisticamente significativa. Os valores do gráfico de Pareto são referentes ao valor-t (Tabela 19) e foram calculados baseado na Seção 5.2.12.

Figura 32: Gráfico de Pareto com os efeitos estimados em valor percentual absoluto para a variável resposta tempo para o algoritmo *Pi* (*valor-p = 0.05*), após transformação de Johnson (família = SB, $\gamma = 1.878965$, $\lambda = 753.9305$, $\varepsilon = 30.97788$ e $\eta = 0.5000765$)



7.2.3 Testes formais de homoscedasticidade, normalidade e independências

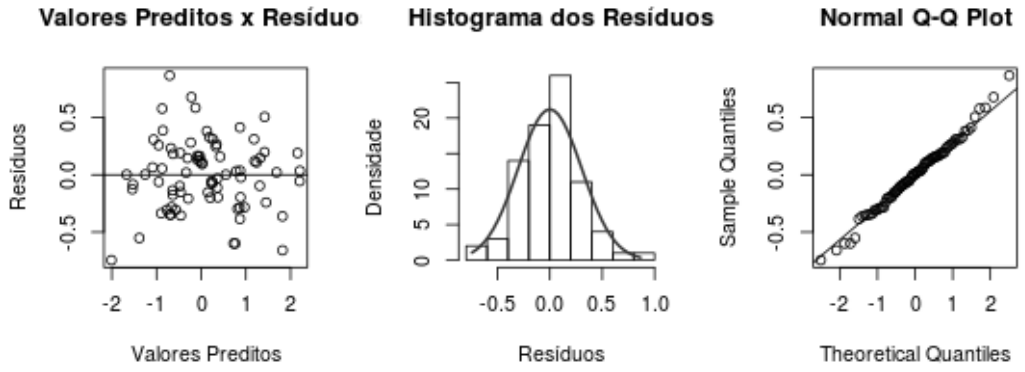
O teste de *Breusch-Pagan* confirmou que os resíduos estavam distribuídos de forma homogênea com *valor-p = 0.2051*. O teste normalidade confirmou que os resíduos se aproximam de uma distribuição normal (teste de *Shapiro-Wilk*, *valor-p = 0.8867*). Outros testes também foram aplicados para confirmar a distribuição normal. O teste de *Kolmogorov-Smirnov* apresentou *valor-p = 0.9912*. Para o teste de normalidade *Anderson-Darling* o *valor-p = 0.8441*.

Segundo o teste de auto correlação negativa e positiva (teste de *DurbinWatson*) o *dw = 1.344* (teste inconclusivo).

7.2.4 Análise dos resíduos

A Figura 33 apresenta a verificação dos resíduos para a variável resposta tempo com os dados transformados. O gráfico, da esquerda para direita, representa a relação entre os resíduos pelos valores preditos e não apresenta estrutura ou padrão (Figura 33a). O histograma dos resíduos (Figura 33b) que apresenta a distribuição da frequência dos resíduos apresentou aproximadamente simétrico em forma de sino. O gráfico à direita (Figura 33c) que representa a probabilidade normal dos resíduos mostrou que os pontos estão próximos da reta.

Figura 33: Verificação dos resíduos: gráfico resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem de dois do fatorial 3^4 do algoritmo *Pi,* após a transformação de Johnson (família = SB, $\gamma = 1.878965$, $\lambda = 753.9305$, $\varepsilon = 30.97788$ e $\eta = 0.5000765$)



(a) (b) (c)

7.2.5 Análise dos efeitos

Levando em consideração somente a variável reposta *Tempo,* somente o fator principal SO não foi estatisticamente significativo, porém o SO pode ser significativo na prática por seu valor-p estar próximo de 0.05. Porém os fatores analisados não podem ser interpretados independentes por fazerem parte de interações significativas. Eles devem ser interpretados em conjunto com os fatores com o qual interagem. De acordo com a ANOVA representada na Tabela 19, as interações que foram estatisticamente significativas, com aproximação linear foram: núcleo:memória, memória:SO, memória:VM e SO:VM (Figura 34).

Para o algoritmo *Pi,* a interação Núcleo:Memória mostrou que para diminuir o tempo de execução do algoritmo basta aumentar a quantidade de memória. Para a interação Memória:SO mostrou que para diminuir o tempo de execução do algoritmo basta aumentar a quantidade de memória. O SO que obteve o melhor desempenho analisando a interação Núcelo:SO foi o Windows em todas as configurações.

O gráfico SO:VM mostra que a mudança do sistema operacional em conjunto com a máquina virtual afeta o tempo de execução do algoritmo. Para qualquer nível do fator SO a máquina virtual com maior tempo de execução em média é o KVM e a máquina virtual com menor tempo de execução em média é o VMware.

Como o efeito de interação SO:VM é significativo, os tempos de execução devem ser interpretados conjuntamente (Figura 35). A melhor forma é criar um diagrama contendo as respostas médias em todas as combinações, podendo concluir que:

- Trocando a máquina virtual KVM para o Xen e depois para o VMware o tempo de execução do algoritmo diminui e esse efeito é muito mais significativo com o sistema operacional no nível 0 (CentOS) com uma diminuição de aproximadamente 161.25 segundos (85.915 + 75.3414);
- Os melhores desempenhos são obtidos usando a máquina virtual VMware juntamente com o sistema operacional Windows 8.1, porém os demais tempos ficaram muitos próximos.
- O pior desempenho do algoritmo foi encontrado no ambiente virtual com a máquina virtual KVM e o sistema operacional CentOS com um desempenho 512.06% superior ao melhor ambiente virtual (208.462 / 40.71).
- Analisando o desempenho do SO e variando a VM, em média o Ubuntu gastou 103.4866 segundos, o CentOS gastou 126.071 segundos e o Windows gastou 75.8469 segundos.

Figura 34: Gráficos de interação, de cima para baixo e da esquerda para direita: núcleo:memória, memória:SO, memória:VM e SO:VM, para o algoritmo *Pi*, após a transformação de Johnson (família = SB, $\gamma = 1.878965$, $\lambda = 753.9305$, $\varepsilon = 30.97788$ e $\eta = 0.5000765$)

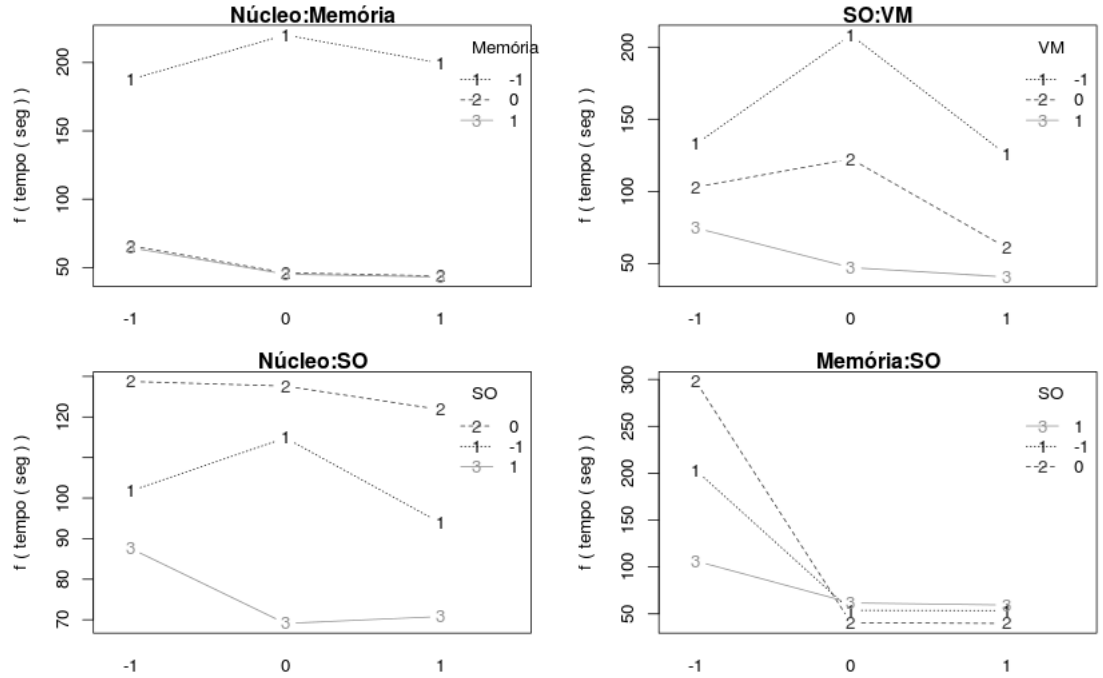
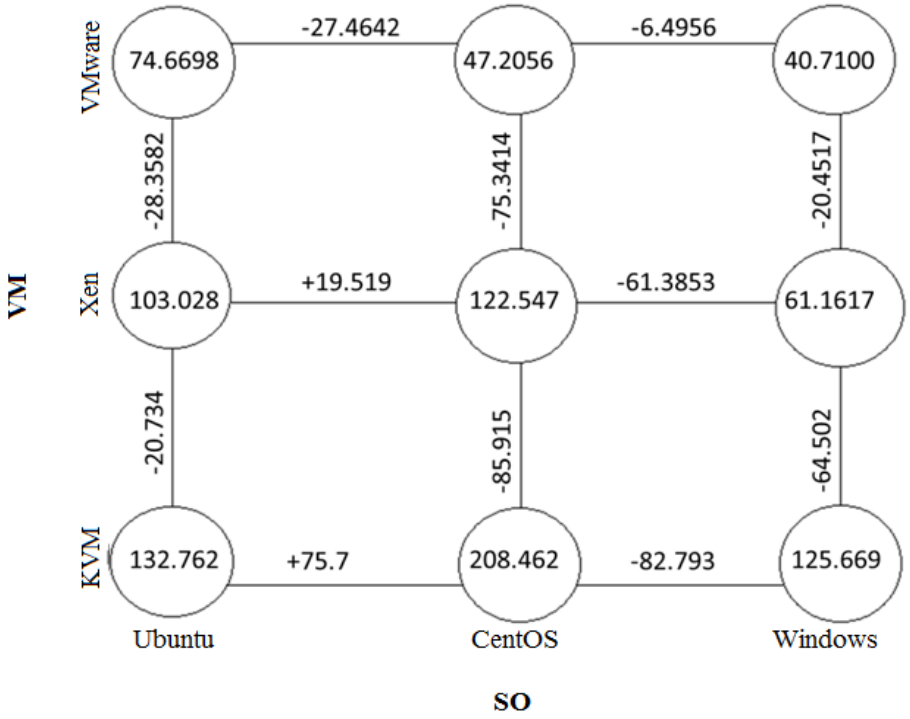


Figura 35: Diagrama para interpretação dos resultados do planejamento fatorial 3^4 para o algoritmo *Pi*. Os valores dos vértices são os tempos médios sem a transformação de Johnson



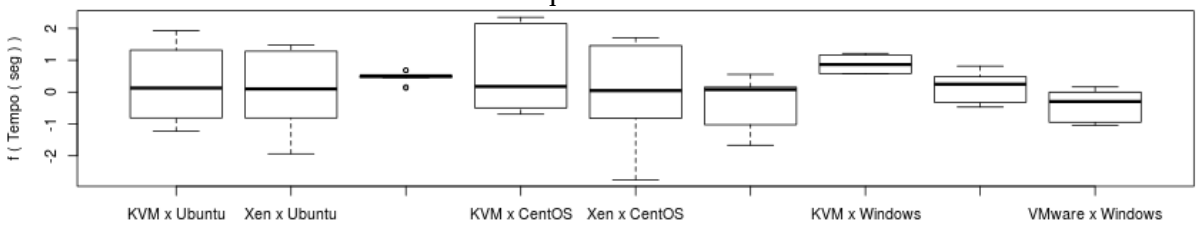
Avaliando as indagações apresentadas na Seção 6.1 e baseada nas análises do algoritmo *Pi* apresentadas anteriormente pode-se concluir que aceita-se as indagações I_0, I_1, I_2 e I_3. As indagações relacionadas com as interações entre os fatores I_4, I_5, I_6, e I_7 também são aceitas, pois as interações entre núcleo:memória, núcleo:SO, memória:SO, e SO:VM foram consideradas estatisticamente significativas a um nível de significância de 5%.

7.2.6 Comparação das médias entre os grupos

Como apresentado na Tabela 19 o efeito de interação entre o SO e VM foi estatisticamente significativo.

Os diagramas de caixa após a transformação de Johnson foram então agrupados pelos fatores SO:VM e são mostrados na Figura 36. Para a interpretação dos dados foi utilizado o teste *t* para a comparação das médias (média dos tempos transformados pelo método de Johnson) de cada fator (WINTER, 2013). Pode-se concluir que a combinação com o fator sistema operacional no nível + (Windows) também possui uma maior estabilidade. Porém a pior combinação (maior tempo gasto em segundos) é a composta pela máquina virtual KVM (nível –) e o sistema operacional Windows (nível +). A melhor combinação é composta pela máquina virtual VMware (nível +) e o sistema operacional Windows (nível +).

Figura 36: Diagrama de caixa (boxplot) para a variável resposta Tempo, para o algoritmo *Pi*, após a transformação de Johnson (família = SB, γ = 1.878965, λ = 753.9305, ε = 30.97788 e η = 0.5000765): variável resposta agrupada pela combinação do sistema operacional pela máquina virtual



A Tabela 20 apresenta o resultado da aplicação do teste *t* e os valores nela contidos possuem os seguintes significados para cada um dos quesitos analisados:

- +1 : O *fator* da linha é maior que o *fator* da coluna;
- -1: O *fator* da linha é menor que o *fator* da coluna;
- 0: Não tem evidências estatísticas que suportem afirmar que o *fator* da linha é diferente do *fator* da coluna, com 95% de confiança.

Tabela 20: Comparação entre os fatores sistema operacional *versus* máquina virtual, após transformação de Johnson (família = SB, γ = 1.878965, λ = 753.9305, ε = 30.97788 e η = 0.5000765), para o algoritmo *Pi*

Teste *t* aplicado para a variável dependente Tempo (seg)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	0	0	0
Xen x Ubuntu	0	--	0	0	0	0	-1	0	0
VMware x Ubuntu	0	0	--	0	0	+1	-1	0	+1
KVM x CentOS	0	0	0	--	0	0	0	0	+1
Xen x CentOS	0	0	0	0	--	0	0	0	0
VMware x CentOS	0	0	-1	0	0	--	-1	0	0
KVM x Windows	0	+1	+1	0	0	+1	--	+1	+1
Xen x Windows	0	0	0	0	0	0	-1	--	+1
VMware x Windows	0	0	-1	-1	0	0	-1	-1	--

Baseado na Tabela 15, conclui-se que o ambiente virtual (KVM com Windows) possui os piores tempos de execução para o algoritmo *Sudoku*, pois na linha somente aparece o valor +1.

7.2.7 Dominância de Pareto

Focando na variável resposta Tempo (seg) foram aplicados os conceitos da dominância de Pareto aos dados apresentados na Tabelas 20. A Tabela 21 apresenta para cada ambiente

virtual quantos ambientes o dominam e quais ambientes virtuais são por ele dominados. Como por exemplo pode-se observar que para o ambiente virtual KVM x Ubuntu não existe nenhum valor positivo na linha (+1), ou seja, não existe nenhum ambiente virtual que o domina. Já a linha VMware x CentOS, existem 2 valores negativos (-1), ou seja, esses valores correspondem aos ambientes virtuais qual são dominados por ele (VMware x Ubuntu, KVM x Windows). Essa mesma ideia é aplicada para todas as linhas da Tabela 20.

Tabela 21: Dominância estatística de Pareto aplicado a variável resposta Tempo da Tabela 20 para o algoritmo *Pi,* , após transformação de Johnson (família = SB, $\gamma = 1.878965$, $\lambda = 753.9305$, $\varepsilon = 30.97788$ e $\eta = 0.5000765$)

Ambientes virtuais	**Quantos o dominam**	**Quais são por ele dominados**
KVM x Ubuntu	0	∅
Xen x Ubuntu	0	KVM x Windows
VMware x Ubuntu	2	KVM x Windows
KVM x CentOS	1	∅
Xen x CentOS	0	∅
VMware x CentOS	0	KVM x Windows e VMware x Ubuntu
KVM x Windows	5	∅
Xen x Windows	1	KVM x Windows
VMware x Windows	0	KVM x Windows, Xen x Windows, VMware x Ubuntu e KVM x CentOS

Na Tabela 22 foram aplicados os conceitos de dominância de Pareto e foram encontradas 3 fronteiras para o algoritmo *Pi* para a variável resposta Tempo. Os ambientes pertencentes a primeira fronteira são considerados em média dominantes aos demais, ou seja, gastam em média gastam menos tempo de processamento. Pode-se observar pelas fronteiras de Pareto que o sistema virtual Windows obteve em média os maiores tempos de processamento independe de máquina virtual.

Tabela 22: Classificação dos ambientes virtuais nas fronteiras de Pareto para a variável resposta Tempo para o algoritmo *Pi,* , após transformação de Johnson (família = SB, $\gamma = 1.878965$, $\lambda = 753.9305$, $\varepsilon = 30.97788$ e $\eta = 0.5000765$)

Fronteira	**Ambientes virtuais**
1	KVM x Ubuntu, Xen x Ubuntu, Xen x CentOS, VMware x CentOS, VMware x Windows
2	VMware x Ubuntu, KVM x CentOS, Xen x Windows
3	KVM x Windows

7.3 Algoritmo *WordCount*

Nesta seção são apresentados os resultados e análise do experimento para o algoritmo *WordCount.* Os níveis dos fatores utilizados para a execução da matriz experimental está apresentada na Tabela 12. Nesse algoritmo foi analisado como variáveis respostas além do tempo de execução do algoritmo informações como: tempo gasto com o uso do *Garbage Collector*, tempo despendido como o uso da CPU (*CPU*), total de memória física, total de memória virtual e o total de *Heap*, porém a análise dessas outras variáveis respostas está apresentada no Anexo D.

7.3.1 Início da análise dos dados coletados

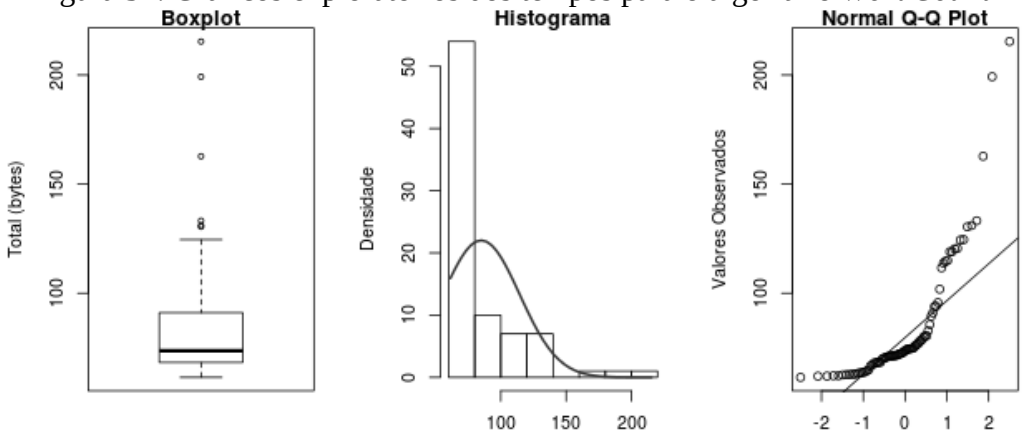
A Tabela 23 apresenta um resumo das informações coletadas durante a execução do algoritmo *WordCount* obtidos com o tamanho da amostra de n = 35 repetições: a média $\bar{X}$, o desvio padrão *S*, a mediana, o valor mínimo e o valor máximo, o coeficiente de variação $\hat{\delta}_x$ de *X*, o coeficiente de variação $\hat{\delta}_{\bar{X}}$ de $\bar{X}$ e o erro padrão *SE*. O ambiente virtual que apresenta, em média, o maior tempo de execução é o Windows x KVM. E o ambiente virtual que apresenta, em média, o menor tempo de execução é o Ubuntu x Xen.

Tabela 23: Tempos de execução gasto (*seg*) para o algoritmo *WordCount*

	$\bar{X}$	*S*	*Mediana*	*Min*	*Max*	$\hat{\delta}_x$	$\hat{\delta}_{\bar{X}}$	*SE*
Ubuntu x KVM	69.677	6.467	72.632	62.191	78.956	0.093	0.031	2.156
CentOS x KVM	79.844	11.530	76.858	68.054	95.693	0.144	0.048	3.843
Windows x KVM	142.390	40.153	124.433	111.641	215.393	0.282	0.094	13.384
Ubuntu x Xen	69.275	9.339	69.695	61.438	91.049	0.135	0.045	3.113
CentOS x Xen	69.661	7.880	71.298	62.001	85.649	0.113	0.038	2.627
Windows x Xen	78.096	9.859	75.195	70.231	101.856	0.126	0.042	3.286
Ubuntu x VMware	113.156	23.917	120.375	68.017	133.231	0.211	0.070	7.972
CentOS x VMware	72.092	9.429	70.355	62.922	89.448	0.131	0.044	3.143
Windows x VMware	71.083	3.916	71.031	66.650	78.227	0.055	0.018	1.305

A Figura 37 apresenta, da esquerda para a direita, informações exploratórias sobre os tempos encontrados durante a execução do algoritmo *Pi* nos ambientes virtuais analisados. O primeiro gráfico (Figura 37a) apresenta o diagrama de caixa com as respectivas respostas e que as maiorias dos ensaios (mais que 75%) tiveram tempo inferior a 100 segundos. O gráfico da normal (Figura 37c) mostra que os dados coletados não se aproximam de uma distribuição normal o que caracteriza a necessidade de se aplicar uma transformação nos dados.

Figura 37: Gráficos exploratórios dos tempos para o algoritmo *WordCount*

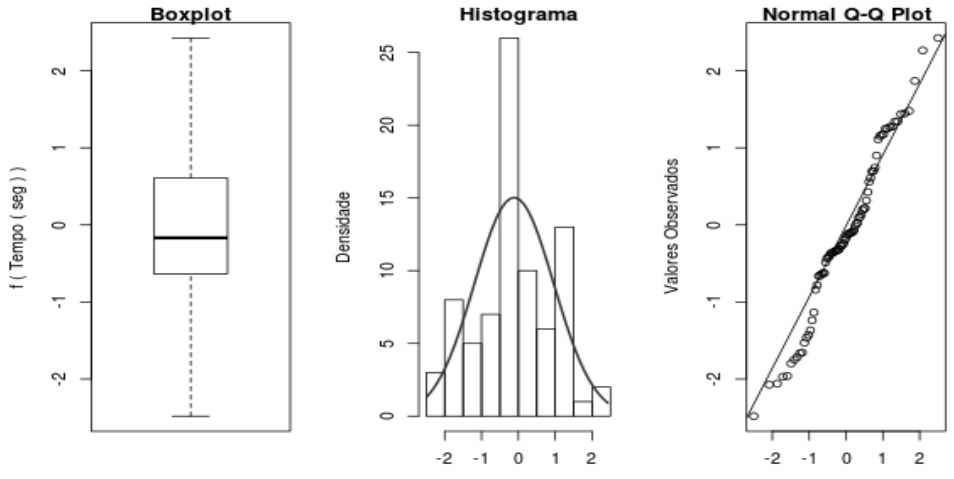


(a) (b) (c)

Baseado nessa assimetria, os dados foram transformados utilizando a transformação de Johnson como resultado apresentado na Seção 5.2.7 A transformação de Johnson foi realizada no software R usando a função RE.Johnson. O resultado da transformação foi: família = SB, $\gamma = 2.667033$, $\lambda = 359.489$, $\varepsilon = 60.50103$ e $\eta = 0.8657382$. Essa transformação é validada com um p-valor = 0.0627 (teste de normalidade de *Anderson Darling*, função *ad.test*).

A Figura 30 mostra os mesmos gráficos, porém após transformação sobre os tempos de execução do algoritmo. Essa transformação possibilita uma distribuição mais simétrica (Figura 30b). O diagrama de caixa mostra (Figura 30a) menos valores discrepantes, e os pontos se aproximam da reta no gráfico de probabilidade normal (Figura 30c).

Figura 38: Gráficos exploratórios dos tempos para o algoritmo *WordCount* após transformação de Johnson (família = SB, $\gamma = 2.667033$, $\lambda = 359.489$, $\varepsilon = 60.50103$ e $\eta = 0.8657382$)



(a) (b) (c)

7.3.2 Análise de variância

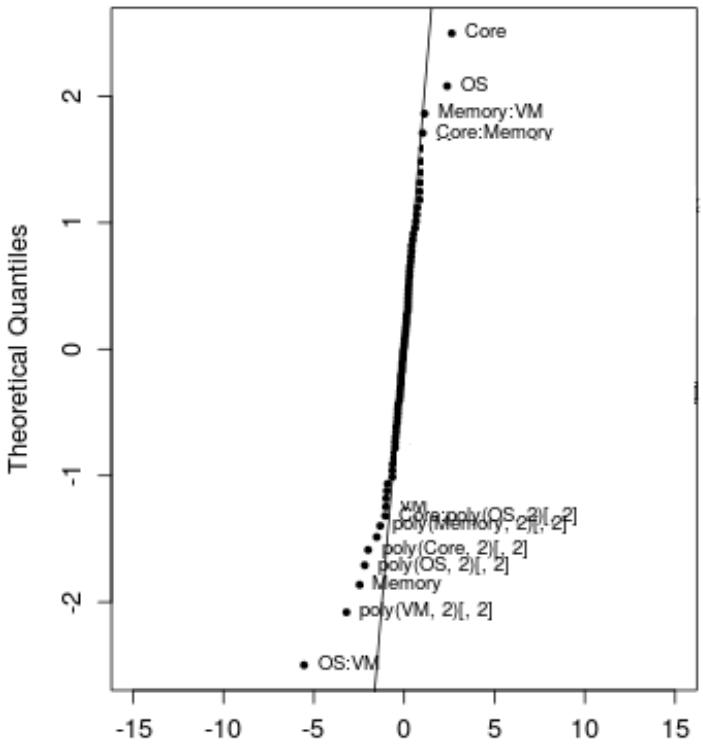
Essas nove configurações dos ambientes virtuais, o qual variou a quantidade de núcleos (unidades de processamento) e a quantidade de memória pertencem a matriz experimental do fatorial de 3^4, totalizando 81 corridas experimentais a serem testadas.

Os tempos medidos foram os tempos de execução total (*wall clock time*). O número de repetições n = 35 foi previamente estabelecido. O total de execuções do algoritmo *WordCount* foi de 2835 execuções e o tempo gasto aproximado foi de 66.96 horas de teste.

A análise do fatorial 3^4 se iniciou com a construção do gráfico de probabilidade normal dos efeitos (apresentado na Seção 5.8.3) estimados para a identificação dos efeitos significativos (Figura 39). A interação entre os efeitos SO e VM foi altamente significativa.

A significância estatística dos efeitos foi confirmada através do modelo com interação de segunda ordem e aproximação linear combinada com aproximação quadrática. As interações de ordem maior que dois são, em geral, insignificantes e podem ser usadas para estimar o erro. Portanto, a verificação dos resíduos do modelo de ordem dois mostrou que a variância dos resíduos sob a medida que os valores previstos crescem (Figura 39).

Figura 39: Gráfico de probabilidade normal dos efeitos estimados do fatorial 3^4 dos tempos para o algoritmo *WordCount*, após transformação de Johnson (família = SB, $\gamma = 2.667033$, $\lambda = 359.489$, $\varepsilon = 60.50103$ e $\eta = 0.8657382$)



Para a interpretação dos efeitos encontrados durante análise de variância é necessário reverter à transformação de Johnson que foi aplicada. A reversão da transformação de Johnson está apresentada na Seção 5.2.7.

A Tabela ANOVA é apresentada na Tabela 24, possuindo um coeficiente de determinação R^2 igual a 0.8904 com 48 graus de liberdade. A interpretação de VM com SO foi significativa (Tabela 24), portanto, o efeito de VM deve ser interpretado em conjunto com o fator SO.

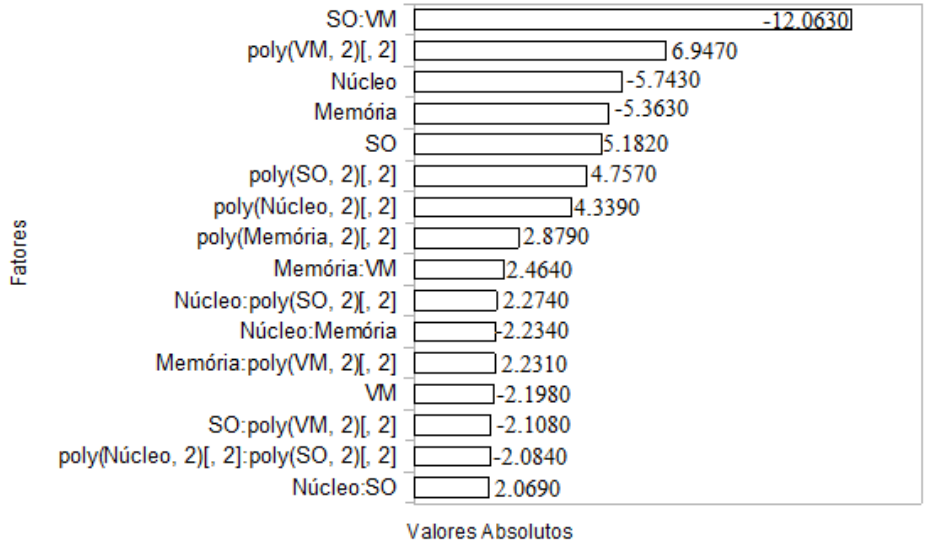
Tabela 24: Modelo para o fatorial 3^4 para o algoritmo *WordCount* dos tempos para a variável resposta Tempo (seg), após transformação de Johnson (família = SB, $\gamma = 2.667033$, $\lambda = 359.489$, $\varepsilon = 60.50103$ e $\eta = 0.8657382$)

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	1.0523	-0.1169	0.0510	-2.2910	0.0264
Núcleo	2.6379	-0.3590	0.0625	-5.7430	0.0000
Memória	-2.4633	-0.3352	0.0625	-5.3630	0.0000
SO	2.3802	0.3239	0.0625	5.1820	0.0000
VM	-1.0094	-0.1374	0.0625	-2.1980	0.0328
poly(Núcleo, 2)[, 2]	-1.9928	1.9928	0.4593	4.3390	0.0001
poly(Memória, 2)[, 2]	-1.3225	1.3225	0.4593	2.8790	0.0059
poly(SO, 2)[, 2]	-2.1848	2.1848	0.4593	4.7570	0.0000
poly(VM, 2)[, 2]	-3.1905	3.1905	0.4593	6.9470	0.0000
Núcleo:Memória	1.0262	-0.1710	0.0766	-2.2340	0.0302
Núcleo:SO	0.9503	0.1584	0.0766	2.0690	0.0439
Núcleo:VM	0.3817	0.0636	0.0766	0.8310	0.4100
Núcleo:poly(Memória, 2)[, 2]	-0.4980	0.6099	0.5625	1.0840	0.2837
Núcleo:poly(SO, 2)[, 2]	-1.0445	1.2793	0.5625	2.2740	0.0275
Núcleo:poly(VM, 2)[, 2]	-0.9140	1.1194	0.5625	1.9900	0.0523
Memória:SO	0.4068	-0.0678	0.0766	-0.8860	0.3801
Memória:VM	1.1315	0.1886	0.0766	2.4640	0.0174
Memória:poly(Núcleo, 2)[, 2]	-0.0584	0.0715	0.5625	0.1270	0.8994
Memória:poly(SO, 2)[, 2]	0.2599	0.3183	0.5625	0.5660	0.5742
Memória:poly(VM, 2)[, 2]	1.0248	1.2551	0.5625	2.2310	0.0304
SO:VM	-5.5403	-0.9234	0.0766	-12.0630	0.0000
SO:poly(Núcleo, 2)[, 2]	0.6087	-0.7456	0.5625	-1.3250	0.1913
SO:poly(Memória, 2)[, 2]	0.2305	0.2824	0.5625	0.5020	0.6180
SO:poly(VM, 2)[, 2]	0.9681	-1.1857	0.5625	-2.1080	0.0403
VM:poly(Núcleo, 2)[, 2]	0.2145	0.2627	0.5625	0.4670	0.6426
VM:poly(Memória, 2)[, 2]	0.4789	-0.5865	0.5625	-1.0430	0.3023
VM:poly(SO, 2)[, 2]	0.8865	1.0858	0.5625	1.9300	0.0595
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.2925	-2.6326	4.1336	-0.6370	0.5272
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	-0.9572	-8.6145	4.1336	-2.0840	0.0425
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	-0.2250	-2.0247	4.1336	-0.4900	0.6265
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-0.2316	-2.0845	4.1336	-0.5040	0.6164
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	0.7340	-6.6062	4.1336	-1.5980	0.1166
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	0.5034	4.5306	4.1336	1.0960	0.2785

O nível de significância adotados nos experimentos foi de 5% de significância estatística. A Figura 40 apresenta o gráfico de Pareto para os valores absolutos do percentual

estimado de cada efeito no tempo médio do algoritmo *WordCount*. Os valores apresentados no gráfico foram somente aqueles que ficaram abaixo do nível de significância. Pode-se verificar que os fatores que tiveram mais impacto no tempo de execução do algoritmo foram a interação entre SO:VM. Os fatores principais Núcleo e Memória também foi altamente estatisticamente significativos. Os valores do gráfico de Pareto são referentes ao valor-t (Tabela 24) e foram calculados baseado na Seção 5.2.12.

Figura 40: Gráfico de Pareto com os efeitos estimados em valor percentual absoluto para a variável resposta tempo para o algoritmo *WordCount* (*valor-p = 0.05*), após transformação de Johnson (família = SB, γ = 2.667033, λ = 359.489, ε = 60.50103 e η = 0.8657382)



7.3.3 Testes formais de homoscedasticidade, normalidade e independências

O teste heteroestaticidade de *Breusch-Pagan* não confirmou que os resíduos estavam distribuídos de forma homogênea com *valor-p = 0.02117*. Porém, os testes de *Goldfeld-Quandt* (Goldfeld, et al., 1965) e *Harrison-McCabe* (Harrison, et al., 1979) confirmam que os resíduos aproximam de uma distribuição normal, aceitando a hipótese de Homoscedasticidade. Eles tiveram respectivamente valores-p igual a: 0.1471 e 0.061.

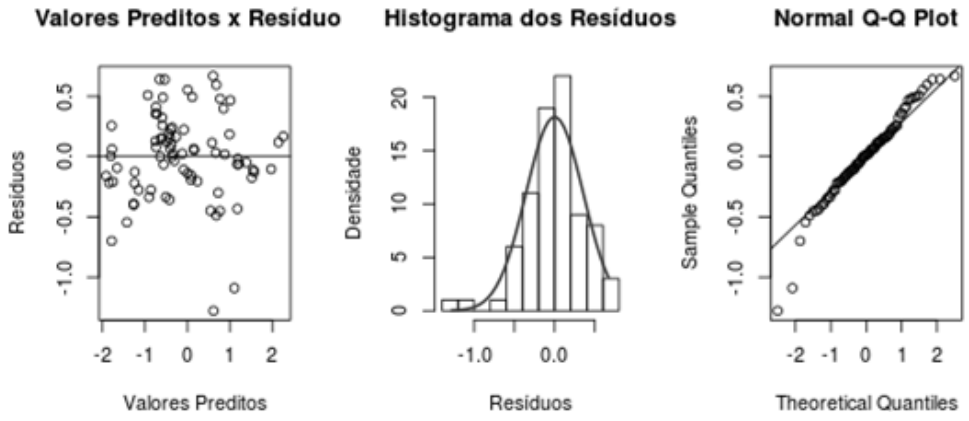
O teste normalidade *Shapiro-Wilk* não confirmou que os resíduos se aproximam de uma distribuição normal (teste de *Shapiro-Wilk*, *valor-p = 0.01284*). Porém, os testes de Kolmogorov-Smirnov (Lilliefors, 1967) e Andereson-Darling (Razali, et al., 2011) confirmam que os resíduos aproximam de uma distribuição normal, aceitando a hipótese de Normalidade. Eles tiveram respectivamente valores-p igual a: 0.7888 e 0.2066.

Segundo o teste de auto correlação negativa e positiva (teste de *DurbinWatson*) o *dw = 1.441*, em que, resultado como um teste inconclusivo.

7.3.4 Análise dos resíduos

A Figura 41 apresenta a verificação dos resíduos para a variável resposta tempo com os dados transformados. O gráfico, da esquerda para direita, representa a relação entre os resíduos pelos valores preditos e não apresenta estrutura ou padrão (Figura 41a). O histograma dos resíduos (Figura 41b) que apresenta a distribuição da frequência dos resíduos apresentou aproximadamente simétrico em forma de sino. O gráfico à direita (Figura 41c) que representa a probabilidade normal dos resíduos mostrou que os pontos estão próximos da reta.

Figura 41: Verificação dos resíduos: gráfico resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem de dois do fatorial 3^4 do algoritmo *WordCount*, após a transformação de Johnson (família = SB, $\gamma = 2.667033$, $\lambda = 359.489$, $\varepsilon = 60.50103$ e $\eta = 0.8657382$)



(a) (b) (c)

7.3.5 Análise dos efeitos

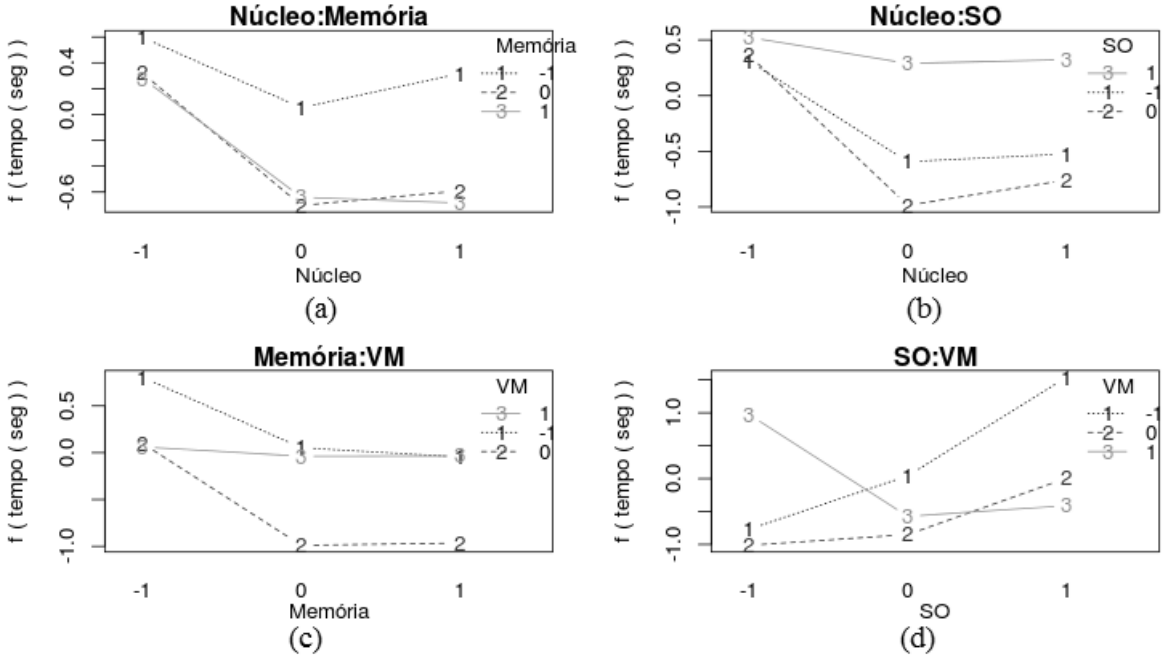
Levando em consideração somente a variável reposta *Tempo*, todos os fatores principais foram estatisticamente significativos. Porém esses fatores não podem ser interpretados independentes por fazerem parte de interações significativas. Eles devem ser interpretados em conjunto com os fatores com o qual interagem. De acordo com a Tabela ANOVA (Tabela 24), as interações que foram estatisticamente significativas, com aproximação linear foram: núcleo:memória, núcleo:SO, memória:VM e SO:VM (Figura 42).

Para o algoritmo *WordCount*, a interação Núcleo:Memória (Figura 42a) mostrou que para diminuir o tempo de execução do algoritmo basta aumentar a quantidade de memória. Para a interação Memória:SO (Figura 42c) mostrou que para diminuir o tempo de execução do algoritmo basta aumentar a quantidade de memória. O SO que obteve o melhor desempenho analisando a interação Memória:SO foi o CentOS praticamente em todas as configurações.

O gráfico Núcleo:SO (Figura 42b) o SO que também obteve o melhor desempenho foi o CentOS praticamente em todas as configurações.

O gráfico SO:VM (Figura 42d) mostra que a mudança do sistema operacional em conjunto com a máquina virtual afeta o tempo de execução do algoritmo. Para qualquer nível – do fator SO, a máquina virtual com maior tempo de execução em média é o VMware e a máquina virtual com menor tempo de execução em média é o Xen. Para o nível 0 do fator SO ainda a máquina virtual Xen obteve em média o menor tempo de execução do algoritmo. Já o KVM obteve o pior desempenho.

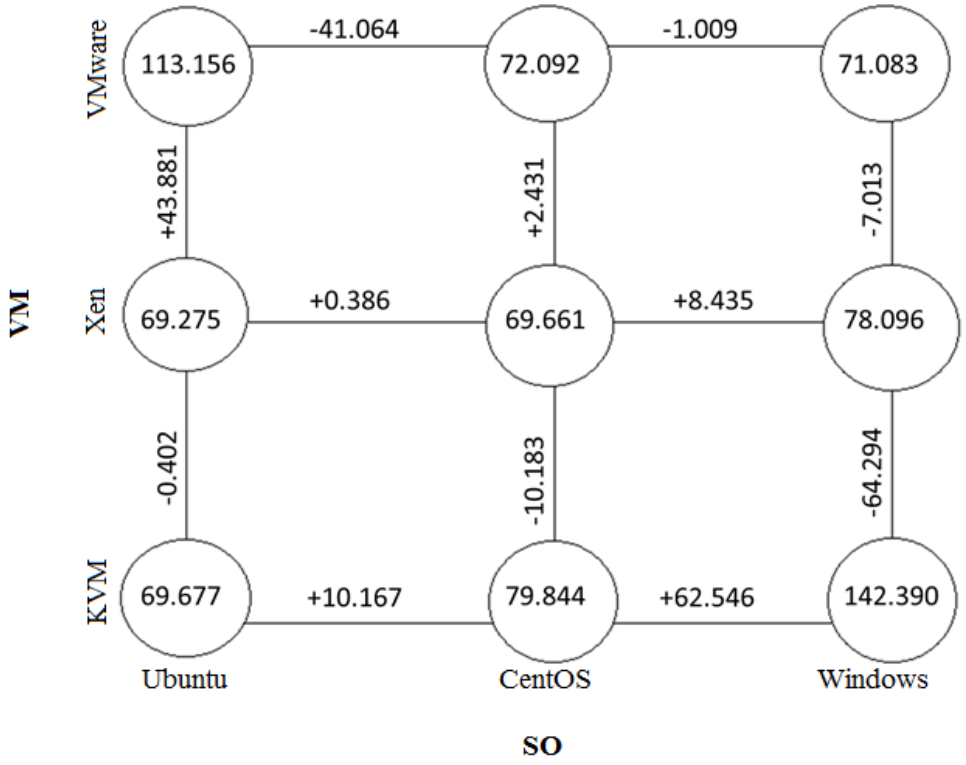
Figura 42: Gráficos de interação, de cima para baixo e da esquerda para direita: núcleo:memória, memória:SO, memória:VM e SO:VM, para o algoritmo *WordCount*, após a transformação de Johnson (família = SB, $\gamma = 2.667033$, $\lambda = 359.489$, $\varepsilon = 60.50103$ e $\eta = 0.8657382$)



Como o efeito de interação SO:VM é significativo, os tempos de execução devem ser interpretados conjuntamente (Figura 43). A melhor forma é criar um diagrama contendo as respostas médias em todas as combinações, podendo concluir que:

- Trocando a máquina virtual KVM para o Xen e depois para o VMware o tempo de execução do algoritmo diminui e esse efeito é muito mais significativo com o sistema operacional no nível + (Windows) com uma diminuição de aproximadamente 71.31 (64.294 + 7.07) segundos;
- Os melhores desempenhos são obtidos usando a máquina virtual Xen juntamente com o sistema operacional Ubuntu, porém outros tempos ficaram muitos próximos.
- O pior desempenho do algoritmo foi encontrado no ambiente virtual com a máquina virtual KVM e o sistema operacional Windows com um desempenho 205.54% (142.390 / 69.275) superior ao melhor ambiente virtual.
- O sistema operacional mais estável para as configurações analisadas foi o CentOS. E a máquina virtual mais estável para as configurações analisadas foi o Xen.

Figura 43: Diagrama para interpretação dos resultados do planejamento fatorial 3^4. Os valores dos vértices são os tempos médios sem a transformação de Johnson para o algoritmo *WordCount*



Avaliando as indagações apresentadas na Seção 6.1 e baseada nas análises do algoritmo *WordCount* apresentadas anteriormente pode-se concluir que aceita-se as indagações I_0, I_1, I_2 e I_3. As indagações relacionadas com as interações entre os fatores I_4, I_5, I_6, e I_7 também são aceitas,

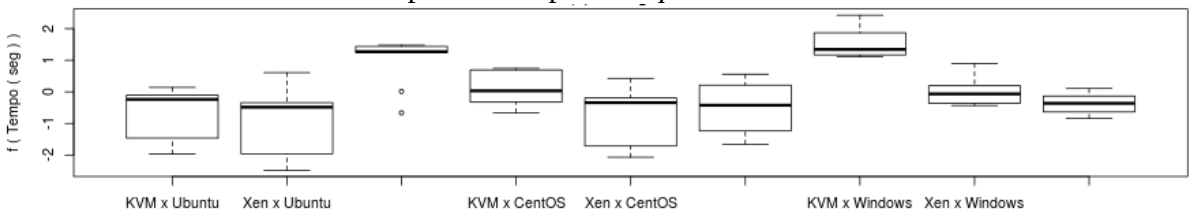
pois as interações entre núcleo:memória, núcleo:SO, memória:SO, e SO:VM foram consideradas estatisticamente significativas a um nível de significância de 5%.

7.3.6 Comparação das médias entre os grupos

Como apresentado na Tabela 24 o efeito de interação entre o SO e VM foi estatisticamente significativo.

Os diagramas de caixa após a transformação de Johnson foram então agrupados pelos fatores SO:VM e são mostrados na Figura 44. Para a interpretação dos dados foi utilizado o teste *t* para a comparação das médias (média dos tempos transformados pelo método de Johnson) de cada fator (WINTER, 2013). Pode-se concluir que a combinação com o fator sistema operacional no nível + (Windows) também possui uma maior estabilidade. A pior combinação (maior tempo gasto) é a composta pela máquina virtual KVM (nível –) e o sistema operacional Windows (nível +).

Figura 44: Diagrama de caixa (boxplot) para a variável resposta Tempo, para o algoritmo *WordCount*, após a transformação de Johnson (família = SB, $\gamma = 2.667033$, $\lambda = 359.489$, $\varepsilon = 60.50103$ e $\eta = 0.8657382$): variável resposta agrupada pela combinação do sistema operacional pela máquina virtual



A Tabela 25 apresenta o resultado da aplicação do teste *t* e os valores nela contidos possuem os seguintes significados para cada um dos quesitos analisados:

- +1 : O *fator* da linha é maior que o *fator* da coluna;
- -1: O *fator* da linha é menor que o *fator* da coluna;
- 0: Não tem evidências estatísticas que suportem afirmar que o *fator* da linha é diferente do *fator* da coluna, com 95% de confiança.

Baseado na Tabela 25, conclui-se que o ambiente virtual (KVM com Windows) possui os piores tempos de execução para o algoritmo *WordCount*, pois na linha somente aparece o valor +1.

Tabela 25: Comparação entre os fatores sistema operacional *versus* máquina virtual, após transformação de Johnson (família = SB, $\gamma = 2.667033$, $\lambda = 359.489$, $\varepsilon = 60.50103$ e $\eta = 0.8657382$), para o algoritmo *WordCount*

Teste *t* aplicado para a variável dependente Tempo (seg)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	-1	-1	+1	0	-1	-1	0
Xen x Ubuntu	0	--	-1	-1	0	0	-1	-1	0
VMware x Ubuntu	+1	+1	--	+1	+1	+1	0	+1	+1
KVM x CentOS	+1	+1	-1	--	+1	0	-1	0	0
Xen x CentOS	-1	0	-1	-1	--	0	-1	-1	0
VMware x CentOS	0	0	-1	0	0	--	-1	0	0
KVM x Windows	+1	+1	0	+1	+1	+1	--	+1	+1
Xen x Windows	+1	+1	-1	0	+1	0	-1	--	+1
VMware x Windows	0	0	-1	0	0	0	-1	-1	--

7.3.7 Dominância de Pareto

Focando na variável resposta Tempo (seg) foram aplicados os conceitos da dominância de Pareto aos dados apresentados na Tabelas 26. A Tabela 27 apresenta para cada ambiente virtual quantos ambientes o dominam e quais ambientes virtuais são por ele dominados. Como por exemplo pode-se observar que para o ambiente virtual KVM x Ubuntu não existe nenhum valor positivo na linha (+1), ou seja, não existe nenhum ambiente virtual que o domina. Já a linha Xen x CentOS, existem 5 valores negativos (-1), ou seja, esses valores correspondem aos ambientes virtuais qual são dominados por ele (KVM x Ubuntu, VMware x Ubuntu, KVM x CentOS, KVM x Windows e Xen x Windows). Essa mesma ideia é aplicada para todas as linhas da Tabela 25.

Tabela 26: Dominância estatística de Pareto aplicado a variável resposta Tempo da Tabela 25 para o algoritmo *WordCount*, após transformação de Johnson (família = SB, $\gamma = 2.667033$, $\lambda = 359.489$, $\varepsilon = 60.50103$ e $\eta = 0.8657382$)

Ambientes virtuais	**Quantos o dominam**	**Quais são por ele dominados**
KVM x Ubuntu	1	VMware x Ubuntu, KVM x CentOS, KVM x Windows e Xen x Windows
Xen x Ubuntu	0	VMware x Ubuntu, KVM x CentOS, KVM x Windows e Xen x Windows
VMware x Ubuntu	7	∅
KVM x CentOS	3	VMware x Ubuntu e KVM x Windows
Xen x CentOS	0	KVM x Ubuntu, VMware x Ubuntu, KVM x CentOS, KVM x Windows e Xen x Windows
VMware x CentOS	0	VMware x Ubuntu e KVM x Windows
KVM x Windows	7	∅
Xen x Windows	4	VMware x Ubuntu e KVM x Windows
VMware x Windows	0	VMware x Ubuntu, KVM x Windows e Xen x Windows

Na Tabela 27 foram aplicados os conceitos de dominância de Pareto e foram encontradas 4 fronteiras para o algoritmo *WordCount* para a variável resposta Tempo. Os ambientes pertencentes a primeira fronteira são considerados em média dominantes aos demais, ou seja, gastam em média gastam menos tempo de processamento.

Tabela 27: Classificação dos ambientes virtuais nas fronteiras de Pareto para a variável resposta Tempo para o algoritmo *WordCount,* após transformação de Johnson (família = SB, $\gamma = 2.667033$, $\lambda = 359.489$, $\varepsilon = 60.50103$ e $\eta = 0.8657382$)

Fronteira	Ambientes virtuais
1	Xen x Ubuntu, Xen x CentOS, VMware x CentOS, VMware x Windows
2	KVM x Ubuntu
3	KVM x CentOS, Xen x Windows
4	VMware x Ubuntu e KVM x Ubuntu

7.4 Algoritmo *TestDFSIO Read*

Nesta seção são apresentados os resultados e análise do experimento para o algoritmo *TestDFSIO Read.* Os níveis dos fatores utilizados para a execução da matriz experimental está apresentada na Tabela 12. Nesse algoritmo foi analisado como variáveis respostas além do tempo de execução informações como: tempo gasto com o uso do *Garbage Collector*, tempo despendido como o uso da CPU (*CPU*), total de memória física, total de memória virtual, o total de Heap, o *throughput* (megabytes por segundo) e a taxa média de megabytes por segundo, porém a análise dessas outras variáveis respostas estão apresentadas no Anexo D.

7.4.1 Início da análise dos dados coletados

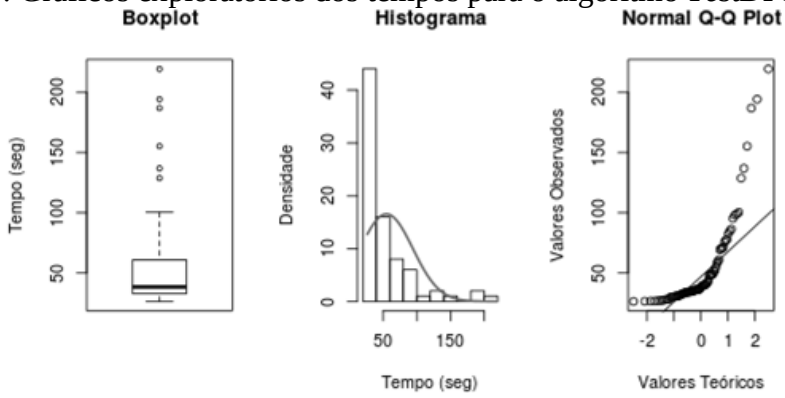
A Tabela 28 apresenta um resumo das informações coletadas durante a execução do algoritmo *TestDFSIO Read* obtidos com o tamanho da amostra de n = 35: a média $\bar{X}$, o desvio padrão *S*, a mediana, o valor mínimo e o valor máximo, o coeficiente de variação $\hat{\delta}_x$ de *X*, o coeficiente de variação $\hat{\delta}_{\bar{X}}$ de $\bar{X}$ e o erro padrão *SE*.

Tabela 28: Tempos de execução gasto (*seg*) para o algoritmo *TestDFSIO Read*

	$\bar{X}$	*S*	*Mediana*	*Min*	*Max*	$\hat{\delta}_x$	$\hat{\delta}_{\bar{X}}$	*SE*
Ubuntu x KVM	33.619	4.094	34.141	27.515	40.603	0.122	0.041	1.365
CentOS x KVM	38.863	7.336	37.062	30.195	52.142	0.189	0.063	2.445
Windows x KVM	128.895	54.311	98.948	82.675	219.160	0.421	0.140	18.104
Ubuntu x Xen	40.925	12.513	39.251	26.853	60.976	0.306	0.102	4.171
CentOS x Xen	38.796	10.393	36.133	26.563	54.872	0.268	0.089	3.464
Windows x Xen	79.893	48.438	59.504	34.438	155.151	0.606	0.202	16.146
Ubuntu x VMware	64.839	16.900	70.801	32.767	77.820	0.261	0.087	5.633
CentOS x VMware	31.452	4.896	30.332	26.821	41.744	0.156	0.052	1.632
Windows x VMware	37.622	7.854	36.200	30.068	49.488	0.209	0.070	2.618

A Figura 45 apresenta, da esquerda para a direita, informações exploratórias sobre os tempos encontrados durante a execução do algoritmo *TestDFSIO Read* nos ambientes virtuais analisados. O primeiro gráfico (Figura 45a) apresenta o diagrama de caixa com as respectivas respostas e que as maiorias dos ensaios (mais que 75%) tiveram tempo inferior a 100 segundos. O gráfico da normal (Figura 45c) mostra que os dados coletados não se aproximam de uma distribuição normal o que caracteriza a necessidade de se aplicar uma transformação nos dados.

Figura 45: Gráficos exploratórios dos tempos para o algoritmo *TestDFSIO Read*



(a) (b) (c)

Baseado nessa assimetria, os dados foram transformados utilizando a transformação de Johnson como resultado apresentado na Seção 6.5. A transformação de Johnson foi realizada no software R usando a função RE.Johnson. O resultado da transformação foi: família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845. Essa transformação é validada com um p-valor = 0.8649 (teste de normalidade de *Anderson Darling*, função *ad.test*).

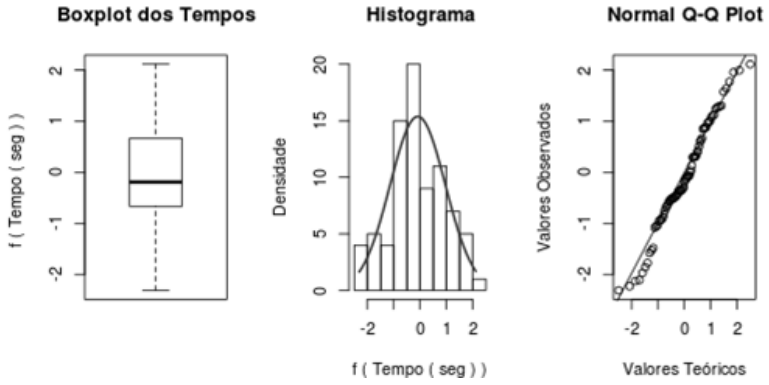
A Figura 46 mostra os mesmos gráficos, porém após transformação sobre os tempos de execução do algoritmo. Essa transformação possibilita uma distribuição mais simétrica (Figura 46b). O diagrama de caixa mostra (Figura 46a) menos valores discrepantes, e os pontos se aproximam da reta no gráfico de probabilidade normal (Figura 46c).

7.4.2 Análise de variância

Essas nove configurações do ambiente virtual, o qual variou a quantidade de núcleos (unidades de processamento) e a quantidade de memória pertencem a matriz experimental do fatorial de 3^4, totalizando 81 corridas experimentais a serem testadas.

Os tempos medidos foram os tempos de execução total (*wall clock time*). O número de repetições n = 35 foi previamente estabelecido. O total de execuções do algoritmo *TestDFSIO Read* foi de 2835 execuções e o tempo gasto aproximado foi de 43.31 horas de teste.

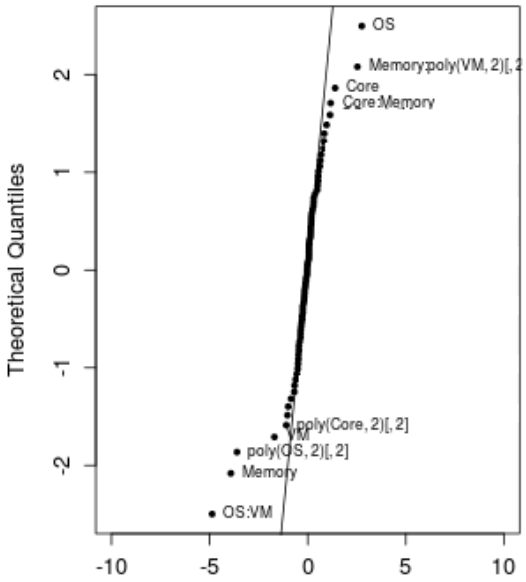
Figura 46: Gráficos exploratórios dos tempos para o algoritmo *TestDFSIO Read* após transformação de Johnson (família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845)



(a) (b) (c)

A análise do fatorial 3^4 se iniciou com a construção do gráfico de probabilidade normal dos efeitos (apresentado na Seção 5.8.3) estimados para a identificação dos efeitos significativos (Figura 47). A interação entre os efeitos SO e VM foi altamente significativa, seguido pelos efeitos principais memória e SO.

Figura 47: Gráfico de probabilidade normal dos efeitos estimados do fatorial 3^4 dos tempos para o algoritmo *TestDFSIO Read*, após transformação de Johnson (família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845)



Para a interpretação dos efeitos encontrados durante análise de variância é necessário reverter à transformação de Johnson que foi aplicada. A reversão da transformação de Johnson está apresentada na Seção 5.2.7.

A Tabela ANOVA é apresentada na Tabela 29, possuindo um coeficiente de determinação R^2 igual a 0.9208 com 48 graus de liberdade. A interpretação de VM com SO foi significativa (Tabela 29), portanto, o efeito de VM deve ser interpretado em conjunto com o fator SO.

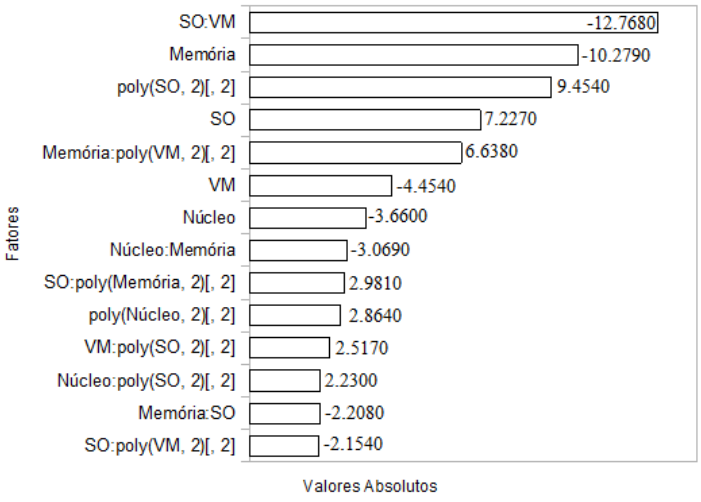
Tabela 29: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Read* dos tempos para a variável resposta Tempo (seg), após transformação de Johnson (família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845)

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	0.8141	-0.0905	0.0424	-2.1340	0.0380
Núcleo	1.3963	-0.1900	0.0519	-3.6600	0.0006
Memória	-3.9207	-0.5335	0.0519	-10.2790	0.0000
SO	2.7567	0.3751	0.0519	7.2270	0.0000
VM	-1.6990	-0.2312	0.0519	-4.4540	0.0001
poly(Núcleo, 2)[, 2]	-1.0923	1.0923	0.3815	2.8640	0.0062
poly(Memória, 2)[, 2]	-0.0156	0.0156	0.3815	0.0410	0.9675
poly(SO, 2)[, 2]	-3.6062	3.6062	0.3815	9.4540	0.0000
poly(VM, 2)[, 2]	-0.1863	0.1863	0.3815	0.4890	0.6274
Núcleo:Memória	1.1706	-0.1951	0.0636	-3.0690	0.0035
Núcleo:SO	0.1719	0.0287	0.0636	0.4510	0.6542
Núcleo:VM	-0.0796	-0.0133	0.0636	-0.2090	0.8356
Núcleo:poly(Memória, 2)[, 2]	0.6117	-0.7492	0.4672	-1.6040	0.1153
Núcleo:poly(SO, 2)[, 2]	-0.8506	1.0418	0.4672	2.2300	0.0305
Núcleo:poly(VM, 2)[, 2]	0.0651	-0.0797	0.4672	-0.1710	0.8653
Memória:SO	0.8424	-0.1404	0.0636	-2.2080	0.0320
Memória:VM	0.6819	0.1137	0.0636	1.7880	0.0801
Memória:poly(Núcleo, 2)[, 2]	-0.2861	0.3504	0.4672	0.7500	0.4569
Memória:poly(SO, 2)[, 2]	0.5028	0.6158	0.4672	1.3180	0.1937
Memória:poly(VM, 2)[, 2]	2.5319	3.1009	0.4672	6.6380	0.0000
SO:VM	-4.8704	-0.8117	0.0636	-12.7680	< 2e-16
SO:poly(Núcleo, 2)[, 2]	0.4826	-0.5911	0.4672	-1.2650	0.2119
SO:poly(Memória, 2)[, 2]	1.1371	1.3927	0.4672	2.9810	0.0045
SO:poly(VM, 2)[, 2]	0.8215	-1.0061	0.4672	-2.1540	0.0363
VM:poly(Núcleo, 2)[, 2]	0.2504	0.3067	0.4672	0.6560	0.5147
VM:poly(Memória, 2)[, 2]	0.1843	-0.2257	0.4672	-0.4830	0.6312
VM:poly(SO, 2)[, 2]	0.9601	1.1759	0.4672	2.5170	0.0152
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.6837	6.1530	3.4330	1.7920	0.0794
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	-0.6704	-6.0332	3.4330	-1.7570	0.0852
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.6422	5.7794	3.4330	1.6830	0.0988
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.1458	1.3121	3.4330	0.3820	0.7040
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.3724	3.3519	3.4330	0.9760	0.3338
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	0.5316	4.7847	3.4330	1.3940	0.1698

O nível de significância adotados nos experimentos foi de 5% de significância estatística. A Figura 48 apresenta o gráfico de Pareto para os valores absolutos do percentual

estimado de cada efeito no tempo médio do algoritmo *TestDFSIO Read*. Os valores apresentados no gráfico foram somente aqueles que ficaram abaixo do nível de significância. Pode-se verificar que os fatores que tiveram mais impacto no tempo de execução do algoritmo foram a interação entre SO:VM. Os fatores principais Núcleo e Memória também foi altamente estatisticamente significativos. Os valores do gráfico de Pareto são referentes ao valor-t (Tabela 29) e foram calculados baseado na Seção 5.2.12.

Figura 48: Gráfico de Pareto com os efeitos estimados em valor percentual absoluto para a variável resposta tempo para o algoritmo *TestDFSIO* (*valor-p = 0.05*), após transformação de Johnson (família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845)



7.4.3 Testes formais de homoscedasticidade, normalidade e independências

O teste heteroestaticidade de *Breusch-Pagan* confirmou que os resíduos estavam distribuídos de forma homogênea com *valor-p = 0.0678*. O teste normalidade *Shapiro-Wilk* confirmou que os resíduos se aproximam de uma distribuição normal (teste de *Shapiro-Wilk*, *valor-p = 0.0562*). Ainda, os testes de Kolmogorov-Smirnov (Lilliefors, 1967) e Andereson-Darling (Razali, et al., 2011) também confirmam que os resíduos aproximam de uma distribuição normal, aceitando a hipótese de Normalidade. Eles tiveram respectivamente valores-p igual a: 0.7931 e 0.0853.

Segundo o teste de auto correlação negativa e positiva (teste de *DurbinWatson*) o *dw = 1.378*, em que, resultado como um teste inconclusivo.

7.4.4 Análise dos resíduos

A Figura 49 apresenta a verificação dos resíduos para a variável resposta tempo com os dados transformados. O gráfico, da esquerda para direita, representa a relação entre os resíduos

pelos valores preditos e não apresenta estrutura ou padrão (Figura 49a). O histograma dos resíduos (Figura 49b) que apresenta a distribuição da frequência dos resíduos apresentou aproximadamente simétrico em forma de sino. O gráfico à direita (Figura 49c) que representa a probabilidade normal dos resíduos mostrou que os pontos estão próximos da reta.

Figura 49: Verificação dos resíduos: gráfico resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem de dois do fatorial 3^4 do algoritmo *TestDFSIO Read*, após a transformação de Johnson (família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845)

7.4.5 Análise dos efeitos

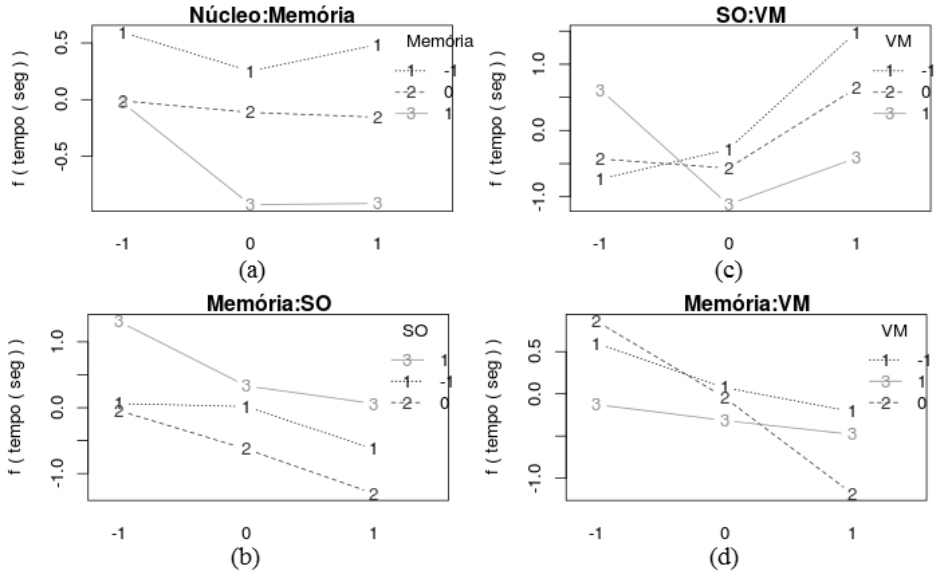
Levando em consideração somente a variável reposta *Tempo*, todos os fatores principais foram estatisticamente significativos. Porém esses fatores não podem ser interpretados independentes por fazerem parte de interações significativas. Eles devem ser interpretados em conjunto com os fatores com o qual interagem. De acordo com o modelo apresentado na Equação 30, as interações que foram estatisticamente significativas, com aproximação linear foram: núcleo:memória, memória:SO e SO:VM (Figura 50). A interação memória:VM pode ser considerada significativa na prática pois o seu valor-p ficou próximo do nível de significância de 5% (valor-p = 0.0801).

Para o algoritmo *TestDFSIO Read*, a interação Núcleo:Memória (Figura 50a) mostrou que para diminuir o tempo de execução do algoritmo basta aumentar a quantidade de memória. O gráfico SO:VM (Figura 50c) mostra que a mudança do sistema operacional em conjunto com a máquina virtual afeta o tempo de execução do algoritmo. Para o nível – do fator SO, a máquina virtual com maior tempo de execução em média é o VMware e a máquina virtual com menor tempo de execução em média é o KVM. Para o demais níveis do fator SO a inferência acima inverte, a máquina virtual com melhor desempenho em média é o VMware e a pior é o KVM.

O gráfico Memória:SO (Figura 50b) o SO que também obteve o melhor desempenho em média foi o CentOS, em todas as configurações. E o sistema operacional que obteve os pior desempenho em média foi o sistema operacional Windows.

Para a interação Memória:VM (Figura 50d) mostrou que para diminuir o tempo de execução do algoritmo basta aumentar a quantidade de memória. A máquina virtual que obteve o melhor desempenho em média analisando a interação Memória:VM foi o VMware no nível – e nível 0, e a máquina virtual Xen no nível +.

Figura 50: Gráficos de interação, de cima para baixo e da esquerda para direita: núcleo:memória, memória:SO, memória:VM e SO:VM, após a transformação de Johnson (família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845) para o algoritmo *TestDFSIO Read*

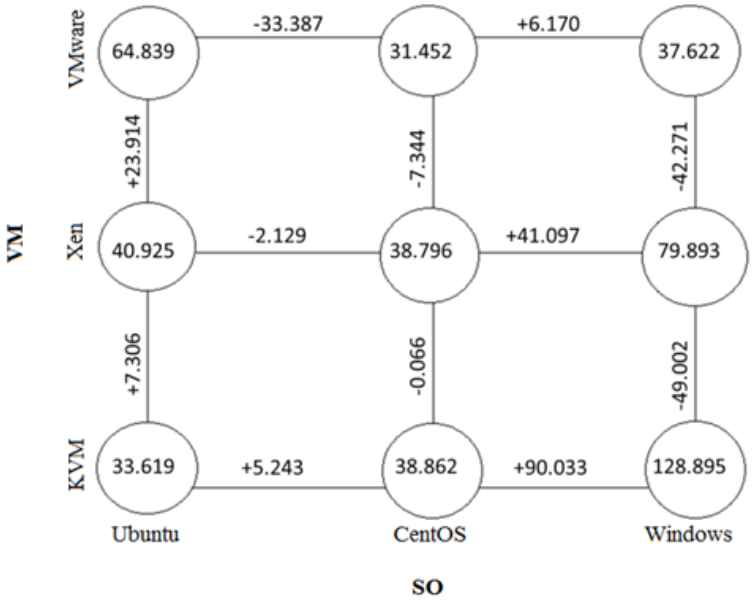


Como o efeito de interação SO:VM é significativo, os tempos de execução devem ser interpretados conjuntamente (Figura 51). A melhor forma é criar um diagrama contendo as respostas médias em todas as combinações, podendo concluir que:

- Trocando a máquina virtual KVM para o Xen e depois para o VMware o tempo de execução do algoritmo diminui para os sistemas operacionais CentOS e Windows e esse efeito é muito mais significativo com o sistema operacional no nível + (Windows) com uma diminuição de aproximadamente 91.273 (49.002 + 42.271) segundos. Já o sistema operacional Ubuntu com as mesmas alterações das máquinas virtuais o tempo aumente em média 31.220 (7.306 + 23.914) segundos;
- Os melhores desempenhos são obtidos usando a máquina virtual Xen juntamente com o sistema operacional CentOS, porém a configuração Ubuntu com KVM ficou muito próximo com uma diferença de 2.166 (33.619 – 31.452) segundos.

- O pior desempenho do algoritmo foi encontrado no ambiente virtual com a máquina virtual KVM e o sistema operacional Windows com um desempenho 409.81% (128.895 / 31.452) superior ao melhor ambiente virtual.
- O sistema operacional mais estável para as configurações analisadas foi o CentOS. E a máquina virtual mais estável para as configurações analisadas foi o VMware.

Figura 51: Diagrama para interpretação dos resultados do planejamento fatorial 3^4. Os valores dos vértices são os tempos médios sem a transformação de Johnson para o algoritmo *TestDFSIO Read*

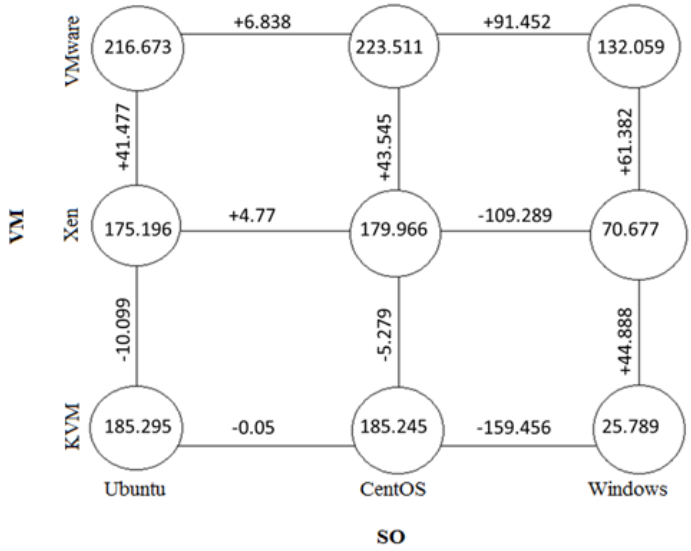


Levando em consideração somente a variável reposta *Throughput*, como o efeito de interação SO:VM é significativo, as respostas (MB/seg) da execução devem ser interpretadas conjuntamente (Figura 52). A melhor forma é criar um diagrama contendo as respostas médias em todas as combinações, podendo concluir que:

- A configuração que possui a melhor configuração em média é composta pelo sistema operacional CentOS e a máquina virtual VMware com aproximadamente 223.511 MB/seg;
- A configuração que possui a pior configuração em média é composta pelo sistema operacional Windows e a máquina virtual KVM com aproximadamente 25.789 MB/seg, ou seja, é inferior em 866.69% (223.511 / 25.789) à melhor configuração;

- Para os ambientes analisados, os melhores desempenhos são obtidos usando a máquina virtual VMware, ou seja, com uma vazão de aproximadamente 572.243 MB/seg (216.673 + 223.511 + 132.059).
- Para os ambientes analisados, os melhores desempenhos são obtidos usando o sistema operacional CentOS, ou seja, com uma vazão de aproximadamente 588.2722 MB/seg (185.245 + 179.966 + 223.511), entretanto o sistema operacional Ubuntu aproximou-se com uma vazão de 577.164 MB/seg (185.295 + 175.196 + 216.673).

Figura 52: Diagrama para interpretação dos resultados do planejamento fatorial 3^4. Os valores dos vértices são os *Throughput* sem a transformação de Johnson para o algoritmo *TestDFSIO Read*



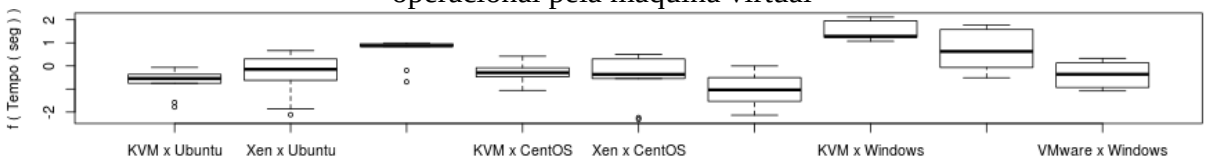
Avaliando as indagações apresentadas na Seção 6.1 e baseada nas análises do algoritmo *TestDFSIO Read* apresentadas anteriormente pode-se concluir que se aceita as indagações $I_{0,}$ $I_{1,}$ I_2 e $I_{3.}$ As indagações relacionadas com as interações entre os fatores $I_{4,}$ $I_{5,}$ $I_{6,}$ e I_7 também são aceitas, pois as interações entre núcleo:memória, memória:SO, memória:VM, e SO:VM foram consideradas estatisticamente significativas a um nível de significância de 5%, em relação à variável resposta Tempo (seg).

7.4.6 Comparação das médias entre os grupos

Como apresentado na Tabela 29 o efeito de interação entre o SO e VM foi estatisticamente significativo.

Os diagramas de caixa após a transformação de Johnson foram então agrupados pelos fatores SO:VM e são mostrados na Figura 53. Para cada diagrama são apresentadas as médias, desvio padrão e mediana. Para a interpretação dos dados foi utilizado o teste *t* para a comparação das médias (média dos tempos transformados pelo método de Johnson) de cada fator (WINTER, 2013). Pode-se concluir que a combinação com o fator sistema operacional no nível + (Windows) também possui uma maior estabilidade. A pior combinação (maior tempo gasto) é a composta pela máquina virtual KVM (nível –) e o sistema operacional Windows (nível +).

Figura 53: Diagrama de caixa (boxplot) para a variável resposta Tempo, para o algoritmo *TestDFSIO Read*, após a transformação de Johnson (família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845): variável resposta agrupada pela combinação do sistema operacional pela máquina virtual



A Tabela 30 apresenta o resultado da aplicação do teste *t* e os valores nela contidos possuem os seguintes significados para cada um dos quesitos analisados:

- +1 : O *fator* da linha é maior que o *fator* da coluna;
- -1: O *fator* da linha é menor que o *fator* da coluna;
- 0: Não tem evidências estatísticas que suportem afirmar que o *fator* da linha é diferente do *fator* da coluna, com 95% de confiança.

Tabela 30: Comparação entre os fatores sistema operacional *versus* máquina virtual, após transformação de Johnson (família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845), para o algoritmo *TestDFSIO Read*

Teste *t* aplicado para a variável dependente Tempo (seg)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	-1	0	0	0	-1	-1	0
Xen x Ubuntu	0	--	-1	0	0	0	-1	-1	0
VMware x Ubuntu	+1	+1	--	+1	+1	+1	-1	0	+1
KVM x CentOS	0	0	-1	--	0	+1	-1	-1	0
Xen x CentOS	0	0	-1	0	--	0	-1	-1	0
VMware x CentOS	0	0	-1	-1	0	--	-1	-1	+1
KVM x Windows	+1	+1	+1	+1	+1	+1	--	+1	+1
Xen x Windows	+1	+1	0	+1	+1	+1	-1	--	+1
VMware x Windows	0	0	-1	0	0	-1	-1	-1	--

Baseado na Tabela 30, conclui-se que o ambiente virtual (KVM com Windows) possui os piores tempos de execução para o algoritmo *TestDFSIO Read*, pois na linha somente aparece

o valor +1. A melhor combinação é composta pela máquina virtual Xen (nível 0) e o sistema operacional CentOS (nível +).

7.4.7 Dominância de Pareto

Focando na variável resposta Tempo (seg) foram aplicados os conceitos da dominância de Pareto aos dados apresentados na Tabelas 31. A Tabela 32 apresenta para cada ambiente virtual quantos ambientes o dominam e quais ambientes virtuais são por ele dominados. Como por exemplo pode-se observar que para o ambiente virtual KVM x Ubuntu não existe nenhum valor positivo na linha (+1), ou seja, não existe nenhum ambiente virtual que o domina. Já a linha Xen x CentOS, existem 3 valores negativos (-1), ou seja, esses valores correspondem aos ambientes virtuais qual são dominados por ele (VMware x Ubuntu, KVM x Windows e Xen x Windows). Essa mesma ideia é aplicada para todas as linhas da Tabela 30.

Tabela 31: Dominância estatística de Pareto aplicado a variável resposta Tempo da Tabela 30 para o algoritmo *TestDFSIO Read*, após transformação de Johnson (família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845)

Ambientes virtuais	Quantos o dominam	Quais são por ele dominados
KVM x Ubuntu	0	VMware x Ubuntu, KVM x Windows e Xen x Windows
Xen x Ubuntu	0	VMware x Ubuntu, KVM x Windows e Xen x Windows
VMware x Ubuntu	6	KVM x Windows
KVM x CentOS	1	VMware x Ubuntu, KVM x Windows e Xen x Windows
Xen x CentOS	0	VMware x Ubuntu, KVM x Windows e Xen x Windows
VMware x CentOS	1	VMware x Ubuntu, KVM x CentOS, KVM x Windows e Xen x Windows
KVM x Windows	8	∅
Xen x Windows	6	KVM x Windows
VMware x Windows	0	VMware x Ubuntu, VMware x CentOS, KVM x Windows e Xen x Windows

Tabela 32: Classificação dos ambientes virtuais nas fronteiras de Pareto para a variável resposta Tempo para o algoritmo TestDFISO Read, , após transformação de Johnson (família = SL, γ = -2.387549, λ = 0, ε = 25.47198 e η = 0.8549845)

Fronteira	Ambientes virtuais
1	KVM x Ubuntu, Xen x Ubuntu, Xen x CentOS, VMware x Windows
2	VMware x CentOS
3	KVM x CentOS
4	VMware x Ubuntu e Xen x Ubuntu
5	KVM x Windows

Na Tabela 32 foram aplicados os conceitos de dominância de Pareto e foram encontradas 5 fronteiras para o algoritmo *TestDFSIO* para a variável resposta Tempo. Os ambientes pertencentes a primeira fronteira são considerados dominantes aos demais, ou seja, gastam em média gastam menos tempo de processamento.

7.5 Algoritmo *TestDFSIO Write*

Nesta seção são apresentados os resultados e análise do experimento para o algoritmo *TestDFSIO Write*. Os níveis dos fatores utilizados para a execução da matriz experimental estão apresentada na Tabela 12. Nesse algoritmo foi analisado como variáveis respostas além do tempo de execução do algoritmo informações como: tempo gasto com o uso do *Garbage Collector*, tempo despendido como o uso da CPU (CPU), total de memória física, total de memória virtual, o total de Heap, o *throughput* (megabytes por segundo) e a taxa média de megabytes por segundo, porém a análise dessas outras variáveis respostas estão apresentadas no Anexo D.

7.5.1 Início da análise dos dados coletados

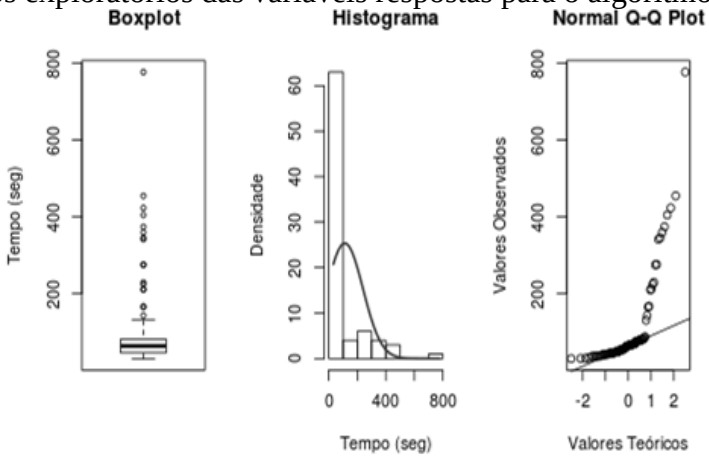
A Tabela 33 apresenta um resumo das informações coletadas durante a execução do algoritmo *TestDFSIO Write* obtidos com o tamanho da amostra de n = 35: a média $\bar{X}$, o desvio padrão *S*, a mediana, o valor mínimo e o valor máximo, o coeficiente de variação $\hat{\delta}_x$ de *X*, o coeficiente de variação $\hat{\delta}_{\bar{X}}$ de $\bar{X}$ e o erro padrão *SE*. Em média, os maiores tempos gastos com a execução do algoritmos estão com a máquina virtual KVM.

Tabela 33: Tempos de execução gasto (*seg*) para o algoritmo *TestDFSIO Write*

	$\bar{X}$	*S*	*Mediana*	*Min*	*Max*	$\hat{\delta}_x$	$\hat{\delta}_{\bar{X}}$	*SE*
Ubuntu x KVM	170.892	153.926	70.367	62.481	405.006	0.901	0.300	51.309
CentOS x KVM	202.737	250.148	63.760	49.648	777.189	1.234	0.411	83.383
Windows x KVM	264.961	114.518	229.693	78.367	454.915	0.432	0.144	38.173
Ubuntu x Xen	54.361	15.896	45.996	38.209	81.747	0.292	0.097	5.299
CentOS x Xen	61.148	40.774	43.857	38.308	165.141	0.667	0.222	13.591
Windows x Xen	84.325	48.496	55.763	44.939	167.338	0.575	0.192	16.165
Ubuntu x VMware	71.863	17.393	77.557	37.061	87.467	0.242	0.081	5.798
CentOS x VMware	39.325	7.813	41.941	31.156	54.507	0.199	0.066	2.604
Windows x VMware	60.382	13.587	58.910	36.761	81.085	0.225	0.075	4.529

A Figura 54 apresenta, da esquerda para a direita, informações exploratórias sobre os tempos encontrados durante a execução do algoritmo *TestDFSIO Write* nos ambientes virtuais analisados. O primeiro gráfico (Figura 54a) apresenta o diagrama de caixa com as respectivas respostas e que as maiorias dos ensaios (mais que 75%) tiveram tempo inferior a 200 segundos. O gráfico da normal (Figura 54c) mostra que os dados coletados não se aproximam de uma distribuição normal o que caracteriza a necessidade de se aplicar uma transformação nos dados.

Figura 54: Gráficos exploratórios das variáveis respostas para o algoritmo *TestDFSIO Write*



(a) (b) (c)

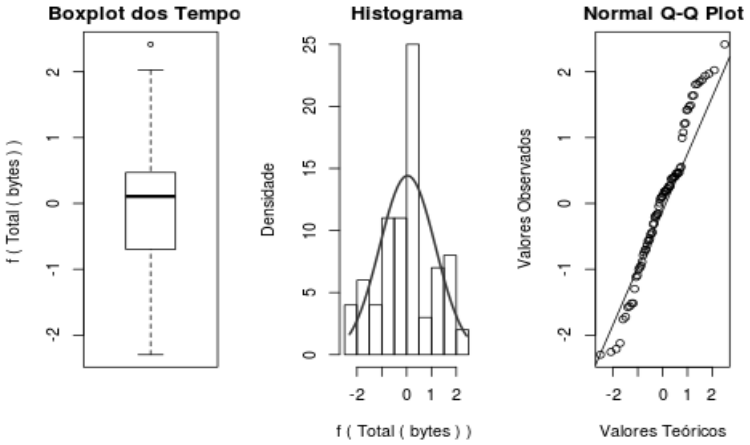
Baseado nessa assimetria, os dados foram transformados utilizando a transformação de Johnson como resultado apresentado na Seção 6.5. A transformação de Johnson foi realizada no software R usando a função RE.Johnson. O resultado da transformação foi: família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894. Essa transformação é validada com um p-valor = 0.6839 (teste de normalidade de *Anderson Darling*, função *ad.test*).

A Figura 55 mostra os mesmos gráficos, porém após transformação sobre os tempos de execução do algoritmo. Essa transformação possibilita uma distribuição mais simétrica (Figura 55b). O diagrama de caixa mostra (Figura 55a) menos valores discrepantes, e os pontos se aproximam da reta no gráfico de probabilidade normal (Figura 55c).

7.5.2 Análise de variância

Essas nove configurações do ambiente virtual, o qual variou a quantidade de núcleos (unidades de processamento) e a quantidade de memória pertencem a matriz experimental do fatorial de 3^4, totalizando 81 corridas experimentais a serem testadas.

Figura 55: Gráficos exploratórios dos tempos para o algoritmo *TestDFSIO Write* após transformação de Johnson (família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894)

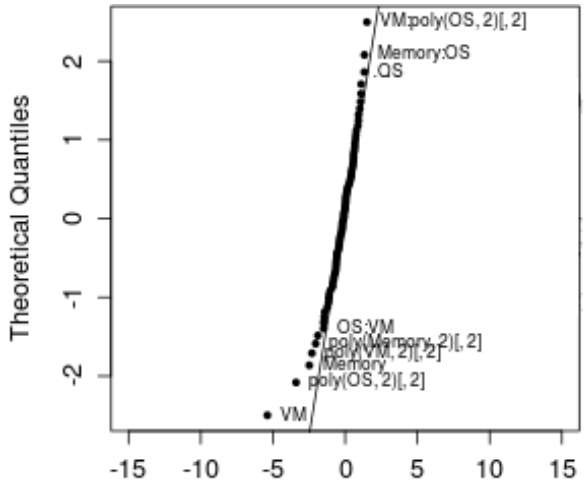


(a) (b) (c)

Os tempos medidos foram os tempos de execução total (*wall clock time*). O número de repetições n = 35 foi previamente estabelecido. O total de execuções do algoritmo *TestDFSIO Write* foi de 2835 execuções e o tempo gasto aproximado foi de 88.37 horas de teste.

A análise do fatorial 3^4 se iniciou com a construção do gráfico de probabilidade normal dos efeitos (apresentado na Seção 5.8.3) estimados para a identificação dos efeitos significativos (Figura 47). O efeito VM foi altamente significativa, seguido pelos efeitos principais memória e SO.

Figura 56: Gráfico de probabilidade normal dos efeitos estimados do fatorial 3^4 dos tempos para o algoritmo *TestDFSIO Write,* após transformação de Johnson (família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894)



Para a interpretação dos efeitos encontrados durante análise de variância é necessário reverter à transformação de Johnson que foi aplicada. A reversão da transformação de Johnson está apresentada na Seção 5.2.7.

A Tabela ANOVA é apresentada na Tabela 34, possuindo um coeficiente de determinação R^2 igual a 0.7828 com 48 graus de liberdade. A interpretação de VM com SO foi significativa (Tabela 34), portanto, o efeito de VM deve ser interpretado em conjunto com o fator SO.

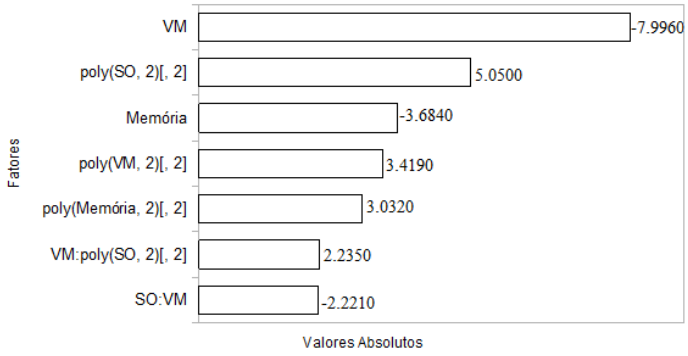
Tabela 34: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Write* dos tempos para a variável resposta Tempo (seg), após transformação de Johnson (família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894)

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	0.3420	0.0380	0.0749	0.5080	0.6141
Núcleo	0.6307	-0.0858	0.0917	-0.9360	0.3538
Memória	-2.4818	-0.3377	0.0917	-3.6840	0.0006
SO	1.3245	0.1802	0.0917	1.9660	0.0551
VM	-5.3870	-0.7331	0.0917	-7.9960	0.0000
poly(Núcleo, 2)[, 2]	-0.8577	0.8577	0.6737	1.2730	0.2091
poly(Memória, 2)[, 2]	-2.0426	2.0426	0.6737	3.0320	0.0039
poly(SO, 2)[, 2]	-3.4023	3.4023	0.6737	5.0500	0.0000
poly(VM, 2)[, 2]	-2.3033	2.3033	0.6737	3.4190	0.0013
Núcleo:Memória	-1.2279	0.2047	0.1123	1.8230	0.0746
Núcleo:SO	-1.1088	-0.1848	0.1123	-1.6460	0.1063
Núcleo:VM	-0.5690	-0.0948	0.1123	-0.8450	0.4025
Núcleo:poly(Memória, 2)[, 2]	0.0438	-0.0536	0.8251	-0.0650	0.9485
Núcleo:poly(SO, 2)[, 2]	0.2804	-0.3435	0.8251	-0.4160	0.6791
Núcleo:poly(VM, 2)[, 2]	0.7744	-0.9484	0.8251	-1.1490	0.2561
Memória:SO	1.3311	-0.2219	0.1123	-1.9760	0.0539
Memória:VM	0.7383	0.1230	0.1123	1.0960	0.2786
Memória:poly(Núcleo, 2)[, 2]	-1.0266	1.2574	0.8251	1.5240	0.1341
Memória:poly(SO, 2)[, 2]	0.5658	0.6929	0.8251	0.8400	0.4052
Memória:poly(VM, 2)[, 2]	0.4563	0.5588	0.8251	0.6770	0.5015
SO:VM	-1.4965	-0.2494	0.1123	-2.2210	0.0311
SO:poly(Núcleo, 2)[, 2]	0.9162	-1.1221	0.8251	-1.3600	0.1802
SO:poly(Memória, 2)[, 2]	-0.0066	-0.0081	0.8251	-0.0100	0.9922
SO:poly(VM, 2)[, 2]	0.6627	-0.8117	0.8251	-0.9840	0.3302
VM:poly(Núcleo, 2)[, 2]	0.9217	1.1288	0.8251	1.3680	0.1777
VM:poly(Memória, 2)[, 2]	1.1061	-1.3547	0.8251	-1.6420	0.1072
VM:poly(SO, 2)[, 2]	1.5055	1.8439	0.8251	2.2350	0.0301
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.1901	1.7106	6.0633	0.2820	0.7791
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	-0.3711	-3.3396	6.0633	-0.5510	0.5843
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	-0.4813	-4.3320	6.0633	-0.7140	0.4784
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-1.1564	-10.4075	6.0633	-1.7160	0.0925
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	1.0082	-9.0739	6.0633	-1.4970	0.1411
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	1.0624	9.5620	6.0633	1.5770	0.1214

O nível de significância adotados nos experimentos foi de 5% de significância estatística. A Figura 57 apresenta o gráfico de Pareto para os valores absolutos do percentual estimado de cada efeito no tempo médio do algoritmo *TestDFSIO Write*. Os valores

apresentados no gráfico foram somente aqueles que ficaram abaixo do nível de significância. Pode-se verificar que os fatores que tiveram mais impacto no tempo de execução do algoritmo foi o fator VM. O fator principal Memória e a interação SO:VM também foram estatisticamente significativos. Os valores do gráfico de Pareto são referentes ao valor-t (Tabela 29) e foram calculados baseado na Seção 5.2.12.

Figura 57: Gráfico de Pareto com os efeitos estimados em valor percentual absoluto para a variável resposta tempo para o algoritmo *TestDFSIO Write* (*valor-p = 0.05*), após transformação de Johnson (família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894)



7.5.3 Testes formais de homoscedasticidade, normalidade e independências

O teste heteroestaticidade de *Breusch-Pagan* confirmou que os resíduos estavam distribuídos de forma homogênea com *valor-p = 0.05334*. O teste normalidade *Shapiro-Wilk* confirmou que os resíduos se aproximam de uma distribuição normal (teste de *Shapiro-Wilk*, *valor-p = 0.999*). Ainda, os testes de Kolmogorov-Smirnov (Lilliefors, 1967) e Andereson-Darling (Razali, et al., 2011) também confirmam que os resíduos aproximam de uma distribuição normal, aceitando a hipótese de Normalidade. Eles tiveram respectivamente valores-p igual a: 0.9984 e 0.9944.

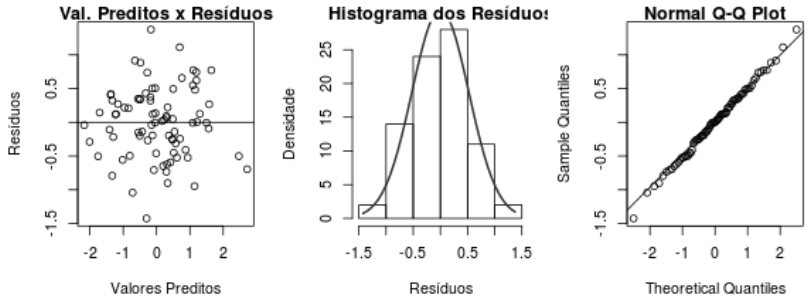
Segundo o teste de auto correlação negativa e positiva (teste de *DurbinWatson*) o *dw = 1.909*, confirmando a independência dos resíduos.

7.5.4 Análise dos resíduos

A Figura 58 apresenta a verificação dos resíduos para a variável resposta tempo com os dados transformados. O gráfico, da esquerda para direita, representa a relação entre os resíduos pelos valores preditos e não apresenta estrutura ou padrão (Figura 58a). O histograma dos resíduos (Figura 58b) que apresenta a distribuição da frequência dos resíduos apresentou

aproximadamente simétrico em forma de sino. O gráfico à direita (Figura 58c) que representa a probabilidade normal dos resíduos mostrou que os pontos estão próximos da reta.

Figura 58: Verificação dos resíduos: gráfico resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem de dois do fatorial 3^4 do algoritmo *TestDFSIO Write*, após a transformação de Johnson (família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894)

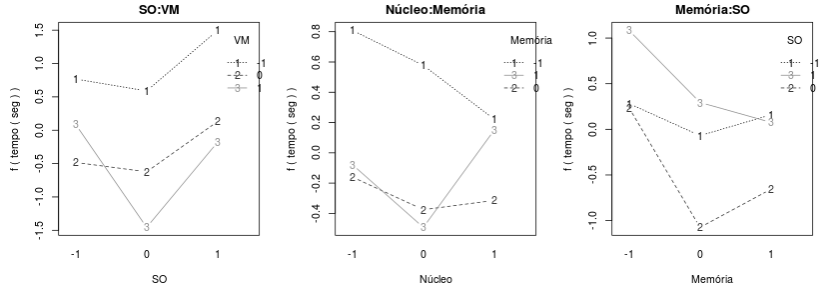


7.5.5 Análise dos efeitos

Levando em consideração somente a variável reposta *Tempo*, apenas os fatores principais *Memória* e *VM* foram estatisticamente significativos. O fator *SO* pode ser considerado significativo na prática pois o seu valor-p ficou muito próximo do nível de significância de 5%. Porém esses fatores não podem ser interpretados independentes por fazerem parte de interações significativas. Eles devem ser interpretados em conjunto com os fatores com o qual interagem.

De acordo com o modelo apresentado na Tabela 34, as interações que foram estatisticamente significativas, com aproximação linear foram SO:VM (Figura 59). A interação núcleo:memória e memória:SO podem ser consideradas significativas na prática pois os seus valor-p ficaram próximo do nível de significância de 5% (valores-p = 0.0745 e 0.0539).

Figura 59: Gráficos de interação, de cima para baixo e da esquerda para direita: SO:VM, núcleo:memória e memória:SO, após a transformação de Johnson (família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894) para o algoritmo *TestDFSIO Write*



(a) (b) (c)

O gráfico SO:VM (Figura 59a) mostra que a mudança do sistema operacional em conjunto com a máquina virtual afeta o tempo de execução do algoritmo. Para qualquer nível do fator SO, a máquina virtual com maior tempo de execução em média é o KVM. Para o níveis 0 e + do fator SO o tempo de execução do algoritmo na máquina virtual VMware é em média menor que as demais.

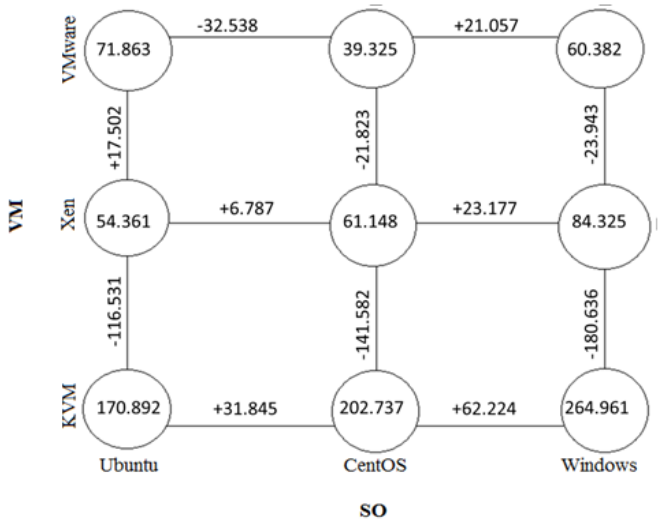
O gráfico Núcleo:Memória (Figura 59b) mostra que para o nível – do fator memória o tempo de execução do algoritmo é, em média, maior independentemente da quantidade de núcleo. O níveis 0 do fator memória foi o que obteve em média o melhor desempenho no geral.

Para a interação Memória:SO (Figura 59c) mostrou que o SO que obteve o melhor desempenho em média analisando a interação Memória:SO foi o CentOS e o sistema operacional que obteve em média os maiores tempos foi o Windows.

Como o efeito de interação SO:VM é significativo, os tempos de execução devem ser interpretados conjuntamente (Figura 60). A melhor forma é criar um diagrama contendo as respostas médias em todas as combinações, podendo concluir que:

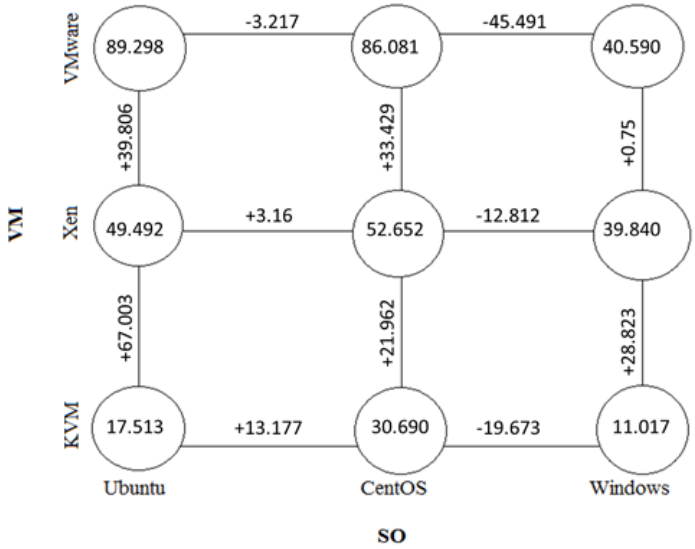
- Trocando a máquina virtual KVM para o Xen e depois para o VMware o tempo de execução do algoritmo diminui para os sistemas operacionais CentOS e Windows e esse efeito é muito mais significativo com o sistema operacional no nível + (Windows) com uma diminuição de aproximadamente 204.579 (180.636 + 23.943) segundos. Já o sistema operacional Ubuntu com as mesmas alterações das máquinas virtuais o tempo diminui do nível – para o nível 0 do fator VM e aumento do nível 0 para o nível +.
- Os melhores desempenhos no nível – e o do fator VM é obtido usando o sistema operacional Ubuntu, Já a melhor configuração foi utilizando a máquina virtual VMware (nível +) com o sistema operacional CentOS.
- O pior desempenho do algoritmo foi encontrado no ambiente virtual com a máquina virtual KVM e o sistema operacional Windows com um desempenho 673.77% (264.961 / 39.325) superior ao melhor ambiente virtual.
- O sistema operacional mais estável para as configurações analisadas foi o Ubuntu. E a máquina virtual mais estável para as configurações analisadas foi o VMware.

Figura 60: Diagrama para interpretação dos resultados do planejamento fatorial 3^4. Os valores dos vértices são os tempos médios sem a transformação de Johnson para o algoritmo *TestDFSIO Write*



Levando em consideração somente a variável reposta *Throughput*, como o efeito de interação SO:VM é significativo, as respostas (MB/seg) da execução devem ser interpretados conjuntamente (Figura 61).

Figura 61: Diagrama para interpretação dos resultados do planejamento fatorial 3^4. Os valores dos vértices são os *Throughput* sem a transformação de Johnson para o algoritmo *TestDFSIO Write*



A melhor forma é criar um diagrama contendo as respostas médias em todas as combinações, podendo concluir que:

- A configuração que possui a melhor média é composta pelo sistema operacional Ubuntu e a máquina virtual VMware com aproximadamente 89.298 MB/seg, porém a configuração VMware: CentOS possui um desempenho aproximadamente igual (86.081 MB/seg);
- A configuração que possui a pior configuração em média é composta pelo sistema operacional Windows e a máquina virtual KVM com aproximadamente 11.017 MB/seg, ou seja, é inferior em 810.55% (89.298 / 11.017) à melhor configuração;
- Para os ambientes analisados, os melhores desempenhos são obtidos usando a máquina virtual VMware, ou seja, com uma vazão de aproximadamente 215.969 MB/seg (89.298 + 86.081 + 40.590).
- Para os ambientes analisados, os melhores desempenhos são obtidos usando o sistema operacional CentOS, ou seja, com uma vazão de aproximadamente 169.423 MB/seg (30.690 + 49.492 + 86.081), entretanto o sistema operacional Ubuntu aproximou-se com uma vazão de 156.303 MB/seg (17.513 + 49.492 + 89.289). Já o sistema operacional Windows em média possui a pior vazão em megabytes por segundo com aproximadamente 91.447 MB/seg (11.017 + 39.840 + 40.590);

Avaliando as indagações apresentadas na Seção 6.1 e baseada nas análises do algoritmo *TestDFSIO Write* apresentadas anteriormente pode-se concluir que aceita-se as indagações $I_{0,}$ $I_{1,}$ I_2 e $I_{3.}$ As indagações relacionadas com as interações entre os fatores $I_{4,}$ $I_{5,}$ $I_{6,}$ e I_7 também são aceitas, pois as interações entre núcleo:memória, memória:SO e SO:VM foram consideradas estatisticamente significativas a um nível de significância de 5%, em relação à variável resposta Tempo (seg).

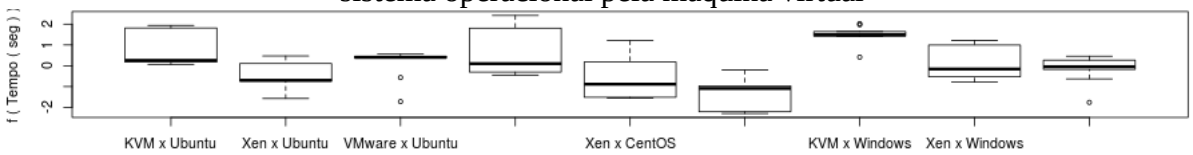
7.5.6 Comparação das médias entre os grupos

Como apresentado na Tabela 34 o efeito de interação entre o SO e VM foi estatisticamente significativo.

Os diagramas de caixa após a transformação de Johnson foram então agrupados pelos fatores SO:VM e são mostrados na Figura 62. Para cada diagrama são apresentados as médias, desvio padrão e mediana. Para a interpretação dos dados foi utilizado o teste *t* para a comparação das médias (média dos tempos transformados pelo método de Johnson) de cada fator (WINTER,

2013). Pode-se concluir que a combinação com o fator sistema operacional no nível + (Windows) também possui uma maior estabilidade. A pior combinação (maior tempo gasto) é a composta pela máquina virtual KVM (nível –) e o sistema operacional Windows (nível +).

Figura 62: Diagrama de caixa (boxplot) para a variável resposta Tempo, para o algoritmo *TestDFSIO Write*, após a transformação de Johnson (família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894): variável resposta agrupada pela combinação do sistema operacional pela máquina virtual



A Tabela 35 apresenta o resultado da aplicação do teste *t* e os valores nela contidos possuem os seguintes significados para cada um dos quesitos analisados:

- +1 : O *fator* da linha é maior que o *fator* da coluna;
- -1: O *fator* da linha é menor que o *fator* da coluna;
- 0: Não tem evidências estatísticas que suportem afirmar que o *fator* da linha é diferente do *fator* da coluna, com 95% de confiança.

Tabela 35: Comparação entre os fatores sistema operacional *versus* máquina virtual, após transformação de Johnson (família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894), para o algoritmo *TestDFSIO Write*

Teste *t* aplicado para a variável dependente Tempo (seg)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	+1	0	0	+1	+1	-1	0	+1
Xen x Ubuntu	-1	--	0	-1	0	+1	-1	0	0
VMware x Ubuntu	0	0	--	0	0	+1	-1	0	0
KVM x CentOS	0	+1	0	--	+1	+1	-1	0	0
Xen x CentOS	-1	0	0	-1	--	0	-1	0	0
VMware x CentOS	-1	-1	-1	-1	0	--	-1	-1	-1
KVM x Windows	+1	+1	+1	+1	+1	+1	--	+1	+1
Xen x Windows	0	0	0	0	0	+1	-1	--	0
VMware x Windows	-1	0	0	0	0	+1	-1	0	--

Baseado na Tabela 35, conclui-se que o ambiente virtual (KVM com Windows) possui os piores tempos de execução para o algoritmo *TestDFSIO Write*, pois na linha somente aparece o valor +1. A melhor combinação é composta pela máquina virtual Xen (nível 0) ou VMware (nível +) e o sistema operacional CentOS (nível +).

7.5.7 Dominância de Pareto

Focando na variável resposta Tempo (seg) foram aplicados os conceitos da dominância de Pareto aos dados apresentados na Tabelas 36. A Tabela 37 apresenta para cada ambiente virtual quantos ambientes o dominam e quais ambientes virtuais são por ele dominados. Como por exemplo pode-se observar que para o ambiente virtual VMware x CentOS não existe nenhum valor positivo na linha (+1), ou seja, não existe nenhum ambiente virtual que o domina. Já a linha Xen x CentOS, existem 3 valores negativos (-1), ou seja, esses valores correspondem aos ambientes virtuais qual são dominados por ele (VMware x Ubuntu, KVM x CentOS e KVM x Windows). Essa mesma ideia é aplicada para todas as linhas da Tabela 30.

Tabela 36: Dominância estatística de Pareto aplicado a variável resposta Tempo da Tabela 305 para o algoritmo *TestDFSIO Write*, após transformação de Johnson (família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894)

Ambientes virtuais	Quantos o dominam	Quais são por ele dominados
KVM x Ubuntu	0	KVM x Windows
Xen x Ubuntu	0	VMware x Ubuntu, KVM x CentOS e KVM x Windows
VMware x Ubuntu	6	KVM x Windows
KVM x CentOS	1	KVM x Windows
Xen x CentOS	0	VMware x Ubuntu, KVM x CentOS e KVM x Windows
VMware x CentOS	1	KVM x Ubuntu, Xen x Ubuntu, VMware x Ubuntu, KVM x CentOS, KVM x Windows, Xen x Windows e VMware x Windows
KVM x Windows	8	∅
Xen x Windows	6	KVM x Windows
VMware x Windows	0	KVM x Ubuntu e KVM x Windows

Na Tabela 37 foram aplicados os conceitos de dominância de Pareto e foram encontradas 4 fronteiras para o algoritmo *TestDFSIO Write* para a variável resposta Tempo. Os ambientes pertencentes a primeira fronteira são considerados em média dominantes aos demais, ou seja, gastam em média gastam menos tempo de processamento.

Tabela 37: Classificação dos ambientes virtuais nas fronteiras de Pareto para a variável resposta Tempo para o algoritmo TestDFISO Write, após transformação de Johnson (família = SU, γ = -1.459791, λ = 5.092776, ε = 39.06827e η = 0.6837894)

Fronteira	Ambientes virtuais
1	Xen x CentOS, VMware x CentOS
2	Xen x Ubuntu, VMware x Ubuntu, Xen x Windows e VMware x Windows
3	KVM x Ubuntu e KVM x CentOS
4	KVM x Windows

7.6 Considerações finais do Capítulo

Baseado nas Tabelas17, 22, 27, 32 e 37 foram aplicados os conceitos de dominância estatística de Pareto e foram encontradas 6 fronteiras para o algoritmo *Sudoku*, 3 fronteiras para o algoritmo *Pi*, 4 fronteiras para o algoritmo *WordCount* e *TestDFSIO Write* e 5 fronteiras para o algoritmo *TestDFSIO Read.* Os ambientes pertencentes a primeira fronteira são considerados em média dominantes aos demais, ou seja, gastam em média gastam menos tempo de processamento. A Tabela 38 mostra uma visão Geral de cada fronteira para cada algoritmo. Pode-se verificar que o ambiente Xen x CentOS apareceu na primeira fronteira em todos os algoritmos e o ambiente virtual KVM x Windows apareceu na última fronteira em todos os algoritmos.

Tabela 38: Classificação nas fronteiras de Pareto para cada algoritmo

Ambientes virtuais	**Algoritmos**				
	Sudoku	**Pi**	**WordCount**	**TestDFSIO Read**	**TestDFSIO Write**
KVM x Ubuntu	1	1	2	1	3
Xen x Ubuntu	1	1	1	1	2
VMware x Ubuntu	1	2	4	4	2
KVM x CentOS	2	2	3	3	3
Xen x CentOS	1	1	1	1	1
VMware x CentOS	3	1	1	2	1
KVM x Windows	6	3	4	5	4
Xen x Windows	5	2	3	4	2
VMware x Windows	4	1	1	1	2

A Tabela 39, apresentam os mesmos resultados da Tabela 103, porém organizados pela ideia de dominância estatística de Pareto. Agora, aplicando novamente o conceito de dominância estatística de Pareto na Tabela 39, o resultados está apresentado na Tabela 40.

Tabela 39: Dominância estatística de Pareto aplicado a variável resposta Tempo da Tabela 38

Ambientes virtuais	**Quantos o dominam**	**Quais são por ele dominados**
KVM x Ubuntu	2	KVM x CentOS e KVM e Windows
Xen x Ubuntu	1	KVM x Ubuntu, VMware x Ubuntu, KVM x CentOS, KVM x Windows, Xen x Windows e VMware x Windows
VMware x Ubuntu	2	KVM e Windows
KVM x CentOS	3	KVM e Windows
Xen x CentOS	0	KVM x Ubuntu, Xen x Ubuntu, VMware x Ubuntu, KVM x CentOS, VMware x CentOS, KVM x Windows, Xen x Windows e VMware x Windows
VMware x CentOS	1	KVM x Windows e Xen x Windows
KVM x Windows	8	∅
Xen x Windows	4	KVM e Windows
VMware x Windows	2	KVM x Windows e Xen x Windows

Tabela 40: Classificação dos ambientes virtuais nas fronteiras de Pareto para a variável resposta Tempo

Fronteira	Ambientes virtuais
1	Xen x CentOS
2	Xen x Ubuntu e VMware x CentOS
3	KVM x Ubuntu, VMware Ubuntu e VMware x Windows
4	Xen x Windows e KVM x CentOS
5	KVM x Windows

Levando em consideração o tempo de execução do algoritmo, a dominância de Pareto apresentou o ambiente virtual Xen:CentOS na primeira fronteira como o ambiente virtual que em média obteve os melhores desempenhos computacionais para os algoritmos analisados. Os ambientes virtuais que ocuparam a segunda fronteira foram os ambientes Xen:Ubuntu e VMware:CentOS, ou seja eles tiveram em média tempos inferiores à primeira fronteira e entre si eles foram considerados equivalentes. Os ambientes pertencentes à terceira fronteira foram KVM:Ubuntu, VMware:Ubuntu e VMware:Windows. Os ambientes pertencentes à quarta fronteira foram Xen:Windows, KVM:CentOS e o ambiente que obteve em média tempos inferiores aos demais foi o KVM:Windows. Pode-se concluir que a máquina virtual Xen e o sistema operacional CentOS, em média, obtiveram os melhores desempenhos. Porém, se o usuário quiser utilizar o sistema operacional Ubuntu, aconselha-se instalá-lo na máquina virtual Xen. E caso o usuário deseje usar o sistema operacional Windows, aconselha ser instalado sobre a máquina virtual VMware.

Pode-se verificar que a metodologia proposta para a comparação de ambientes virtuais apresentada no Capítulo 5, foi validada com a utilização de um estudo de caso.

Capítulo 8

Considerações finais e trabalhos futuros

Computação em Nuvem é um conjunto de recursos virtuais (*hardware*, plataformas de desenvolvimento e serviços) que podem ser facilmente utilizados e acessados via *Internet*. Os recursos computacionais (físicos ou virtuais) do provedor são utilizados para servir a múltiplos usuários, sendo alocados e realocados dinamicamente conforme demanda. Os sistemas de gerenciamento utilizados na Computação em Nuvem controlam e monitoram automaticamente todos os recursos para cada tipo de serviço (armazenamento, processamento e largura de banda). O monitoramento do uso dos recursos deve ser transparente para o provedor do serviço, assim como também, para o consumidor do serviço utilizado.

Na Computação em Nuvem a infraestrutura pode ser disponibilizada como serviço através da virtualização com o uso de máquinas virtuais. A virtualização é um mecanismo que permite a abstração dos recursos de *hardware* e de sistema de um dado sistema operacional. Portanto, esse tipo de tecnologia é utilizada em ambientes em nuvens por meio de um grande conjunto de servidores usando monitores de máquinas virtuais, que estão localizadas entre o *hardware* e o sistema operacional. No entanto, existe uma grande disseminação de máquinas virtuais; cada uma possuindo vantagens e desvantagens próprias. Essas características

específicas de cada máquina virtual permitem a existência de desempenhos computacionais diferentes.

Esse trabalho teve como objetivo propor uma metodologia que busca-se descobrir como, quando e quanto o aumento do desempenho dos algoritmos em ambientes virtuais é determinado pela configuração do ambiente e como os parâmetros de configuração podem influenciar-se mutuamente, e por fim, descobrir através de métodos estatísticos qual configuração de ambiente virtual obteve os melhores resultados em média. Para cada ambiente virtual, foram coletados informações como: tempo de uso de *Garbage Collector*, tempo de uso de CPU, quantidade utilizada de memória principal, memória virtual e heap, *throughput*, taxa média de IO e os tempos de execução dos algoritmos. Essas respostas foram analisadas utilizando ANOVA com o objetivo de descobrir qual configuração (quantidade de núcleo de processamento, quantidade de memória, máquina virtual e sistema operacional) de um ambiente virtualizado permitirá usar todos os aplicativos de forma mais ágil sem perda de recursos computacionais, ou seja, utilizando menos recursos computacionais. Além disso, os resultados foram analisados utilizando a técnica de comparação de médias entre grupos (teste *t-student*) e dominância de Pareto.

As etapas da metodologia de desenvolvimento deste trabalho estão definidas no Capítulo 5. Neste trabalho foi utilizado a técnica de planejamento fatorial completo 3^4. O planejamento experimental representa um conjunto de ensaios estabelecidos com critérios científicos e estatísticos, com o objetivo de determinar a influência de diversas variáveis nos resultados de um sistema ou processo.

Os algoritmos utilizados para os testes foram os pertencentes ao *benchmark* do Apache Hadoop, apresentados na Seção 6.2. Segundo Ziviani, (2008) o que afeta o tempo de processamento de um algoritmo é o *hardware*, compilador e a quantidade de memória. Já para Barr et a. (1995), as variáveis que afetam o desempenho de um algoritmo são o tipo de processador, tamanho de memória, frequência do *clock*, sistema operacional, linguagem de programação, compilador, programas em execução em segundo plano e a habilidade do programador. A matriz de experimentos utilizada possui 81 corridas experimentais para cada algoritmo, composta por 4 (quatro) fatores (quantidade núcleo, quantidade de memória, sistema operacional e máquina virtual), sendo cada fator composto por 3 (três) níveis, apresentado na Tabela 12.

Para a análise dos dados foi utilizado à análise de variância (ANOVA), o teste de comparação de médias (teste *t*) e dominância de Pareto, com um nível de 5% de significância

estatística, ou seja, 95% de confiança. Os dados coletados foram transformados utilizando a transformação de Johnson (Seção 5.8.3), para que os mesmos pudessem estar aproximadamente distribuídos de forma normal. Para confirmar se os resíduos estavam distribuídos homogeneamente foi aplicado o teste de Breusch-Pagan. Caso a hipótese H_0 fosse rejeitada, demais testes menos rigorosos foram aplicados como: teste de GoldFeld-Quandt e teste de Harrison-McCabe. Para confirmar se os resíduos aproximavam de uma distribuição normal foi aplicado o teste de Shapiro-Wilk. Caso a hipótese H_0 fosse rejeitada, demais testes menos rigorosos foram aplicados como: teste de Kolmogorov-Smirnov e teste de Anderson-Darling. Por fim, para verificar a presença de auto correlação positiva ou negativa nos resíduos foi aplicado o teste de DurbinWatson.

Para a variável resposta *Tempo,* os fatores núcleo e memória no nível – em média para quase todos os algoritmos gasta mais tempo para executar o algoritmo, que os demais níveis (nível 0 e +). O fator sistema operacional Windows (nível +), em média, para quase todos os algoritmos, gasta mais tempo para executar o algoritmo, que os demais níveis (nível 0 e –). Já para o fator máquina virtual quase em todos os algoritmos o ambiente Xen (nível 0) obteve a melhor desempenho que o KVM (nível –). E em alguns algoritmos a máquina virtual KVM (nível –) em média gastou mais tempo para resolver o problema que a máquina virtual VMware (nível +). Já entre a máquina virtual Xen e a máquina virtual VMware não existem evidências estatísticas que suportem afirmar que sejam diferentes, exceto para o algoritmo *WordCount*.

Já a Tabela 40, somente para a variável resposta Tempo, mostrou que para os ambientes virtuais (máquina virtual x sistema operacional) analisados, o que obteve em média as melhores configurações em relação ao tempo de processamento dos algoritmos foi o sistema operacional CentOS com a máquina virtual Xen, pois segundo a teoria de dominância de Pareto, esse ambiente virtual ficou classificado na primeira fronteira. Porém não se pode deixar de lado o desempenho conseguido pelas combinações entre o sistema operacional Ubuntu e a máquina virtual Xen e o sistema operacional CentOS e a máquina virtual VMware, esses dois ambientes ficaram classificados na segunda fronteira. Ficou claro que, em média o sistema operacional Windows possui tempos superiores aos outros sistemas operacionais, entretanto, caso deseje utilizá-lo, a máquina virtual mais adequada em média é a VMware (terceira fronteira), contudo esses mesmos ambientes virtuais, em média, possuiu o mesmo desempenho que os ambientes KVM x Ubuntu e VMware x Ubuntu.

Além do teste de médias para os ambientes computacionais foi aplicado também o teste de variância (ANOVA), sobre os dados transformados pela Transformação de Johnson, para

confirmar a significância dos efeitos e responder às 8 Indagações apresentadas na Seção 6.1. Sendo assim, levando em consideração a variável resposta *Garbage Collector* (Tabela 106), pode-se verificar que os efeitos principais Memória, SO e VM foram estatisticamente significativos para a maiorias dos algoritmos analisados aceitando então as Indagações (I_1, I_2, I_3). Já as interações entre núcleo:memória e memória:SO foram também na maioria dos casos significativos aceitando então as indagações (I_4, I_5, I_6). Pode-se verificar também que o efeito mais impactante nessa variável resposta foi o efeito da memória.

Por fim, os efeitos mais significativos para a variável resposta *Tempo*, para todos os algoritmos analisados foram memória, SO e VM, aceitando então as Indagações (I_1, I_2, I_3). Já o núcleo não foi significativo para o algoritmo *TestDFSIO Write*, mas não será por isso que o núcleo seja não significativo, pois nos demais algoritmos ele foi bastante significativo. As interações entre núcleo:memória e SO:VM foram significativos em todos os algoritmos, aceitando então as indagações (I_4, I_5, I_6, I_7). Porém, não pode-se deixar de lado algumas interações importantes como: núcleo:poly(SO, 2), memória:SO, memória:VM, memória:poly(VM, 2), VM:poly(SO, 2) que foram também significativas em quase todos os algoritmos analisados.

Para o estudo de caso utilizado, pode-se concluir então que todos os efeitos analisados são importantes para o desempenho de um determinado algoritmo. Portanto, os fatores núcleo, memória, SO e VM não podem ser analisados separadamente, pois, existem interações estatisticamente significativas em que os mesmos estão presentes. Por isso, deve-se, então para tirar um maior proveito do ambiente virtual, analisar a interação entre núcleo com memória, memória juntamente com o SO e VM e principalmente a interação entre a o SO com a VM que pode substancialmente interferir na desempenho de um determinado algoritmo.

Por fim, pode-se concluir que a metodologia proposta para a comparação de ambientes virtuais conseguiu responder todos as indagações elencadas nos objetivos do estudo de caso.

7.1 Trabalhos futuros

Como trabalhos futuros propomos:

- Aplicar o teste de dominância de Pareto para as demais variáveis respostas (*Garbage Collector*, *CPU*, *Memória Física*, *Memória Virtual*, *Heap*, *Throughput e Taxa média de IO*);

- Realizar uma avaliação mais rigorosa para as demais variáveis respostas (*Garbage Collector, CPU, Memória Física, Memória Virtual, Heap, Throughput e Taxa média de IO*);
- Analisar outros algoritmos pertencentes ao *benchmark* do Apache Hadoop analisando o impacto desses nas mesmas variáveis respostas utilizadas;
- Verificar a possibilidade de existir outros fatores controláveis que poderão interferir no desempenho de um determinado algoritmo;
- Realizar a mesma metodologia proposta na tese em uma nuvem pública como: Amazon, Azure e/ou Google;
- Comparar os resultados encontrados entre os dois tipos de nuvens (pública e privada) verificando quais pontos existem diferenças de performance;
- Realizar a mesma metodologia proposta na tese em um ambiente físico e comparar com os resultados encontrados utilizando a Computação em Nuvem pública e privada;
- Realizar testes usando a rede com a necessidade de descobrir informações que possam divergir entre os ambientes virtuais analisados;

7.2 Publicações

As publicações que foram realizadas durante a execução do projeto:

- BÔAVENTURA, R. S. ; YAMANAKA, K. ; PRADO, G. . Performance Analysis of Algorithms for Virtualized Environments on Cloud Computing. Revista IEEE América Latina, v. 12, p. 792-797, 2014.
- BÔAVENTURA, R. S. ; YAMANAKA, K. ; PRADO, G. ; PINTO, B. Q. ; MACIEL, M. B. ; ROCHA, H. X. . Técnica de Planejamento Experimental aplicada à análise de desempenho de algoritmos na computação em nuvem. In: 2nd Ibero-Americana Computação Aplicada 2014 (CIACA 2014), 2014, Porto. Anais da 2nd Ibero-Americana Computação Aplicada 2014 (CIACA 2014). Lisboa: IADIS Press, 2014. p. 203-207.
- BÔAVENTURA, R. S. ; YAMANAKA, K. ; PINTO, B. Q. . Análise de Desempenho de Algoritmos em Ambientes Virtualizados em Cloud Computing. In: Conferência Iberoamericana WWW/Internet, 2013, Porto Alegre. Anais da Conferência Iberoamericana WWW/Internet. Portugal: IADIS Press, 2013. p. 2013-2017.

- PRADO, G. ; BÔAVENTURA, R. S. ; PINTO, B. Q. . Avaliando o Desempenho da Azure utilizando planejamento experimental sobre a execução de algoritmos determinísticos. In: 12th Conference Ibero-Americana WWW/Internet 2014 (CIAWI 2014), 2014, Porto. Anais da 12th Conference Ibero-Americana WWW/Internet 2014 (CIAWI 2014). Lisboa: IADIS Press, 2014. p. 27-34.
- PRADO, G. ; DRUMMOND, L. M. A. ; BÔAVENTURA, R. S. ; YAMANAKA, K. . Evaluating performance of deterministic algorithms on a multicore processor of a public cloud. In: IEEE 26th International Symposium on Computer Architecture and High Performance Computing Workshops, 2014. P. 42-47. DOI: 10.1109/SBAC-PADW.2014.11

Referências

ANTONY, J.; KATE M.; FRANGOU, A. A strategic methodology to the use of advanced statistical quality improvement techniques. The TQM Magazine, v.10 n.3, p.169 - 176, 1998.

APACHE HADOOP 2.5.2 The Apache Software Foundation. Disponivel em: <http://hadoop.apache.org/>. Acessado em novembro de 2013.

ARMBRUST, M.; FOX, A.; GRIFFITH, G.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. A view of cloud computing. *Commun. ACM* 53, DOI=10.1145/1721654.1721672 Disponivel em: <http://doi.acm.org/10.1145/1721654.1721672>, v.53 n.4, p.50-58. 2010.

AT&T. Businesses Consider Mobile Security Services & Cloud A Top Priority for Disaster Planning: AT&T Study. AT&T Corporation. - News Release Archives. Disponível em: <http://www.att.com/gen/pressroom?pid=22861&cdvn=news&newsarticleid=34475&mapcode=mk-att-business-continuit|business-continuity-2012>. 2012. Acessado em: 15 de julho de 2013.

BARR, R. S.; GOLDEN, B. L.; KELLY, J. P.; RESENDE, M. G.; STEWART JR, W. R. Designing and reporting on computational experiments with heuristic methods. Journal of Heuristics, v.1, n.1, p. 9-32, 1995.

BELOGLAZOV, A.; BUYYA, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers.Concurrency and Computation: Practice and Experience, v. 24, n. 13, p. 1397-1420, 2012.

BESERRA, D.; BORBA, A.; SOUTO, S. C. R. A.; DE ANDRADE, M. J. P.; DE ARAÚJO, A. E. P. Desempenho de Ferramentas de Virtualização na Implementação de Clusters Beowulf Virtualizados em Hospedeiros Windows. In: X Workshop em Clouds, Grids e Aplicações-SBRC 2012. p. 86-95. 2012.

BHARDWAJ, S; JAIN, L; JAIN, S. Cloud computing: A study of infrastructure as a service (IAAS). International Journal of engineering and information Technology, v. 2, n. 1, p. 60-63, 2010.

BONIFACE, M.; NASSER, B.; PAPAY, J.; PHILLIPS, S. C.; SERVIN, A.; YANG, X.; KYRIAZIS, D. Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds. Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on. - Barcelona, Spain. p. 155-160. 2010.

BOUÇAS, C. Computação em nuvem crescerá 74% em 2013, diz consultoria. Jornal Valor Economico, 2012. Disponível em: <http://www.valor.com.br/empresas/2936640/computacao-em-nuvem-crescera-74-em-2013-no-brasil-diz-consultoria>. Acessado em: Jul/2013.

BOX, G.; HUNTER, J. S.; HUNTER, J. S. Statistics for experimenters An introduction to design, data analysis, and model building. Wiley series in probability and mathematical statistics., 1978.

BRENDEL, J. Infrastructure as a Service (IaaS): Am Beispiel von Amazon EC2 und Eucalyptus. Distributed Systems and Operating Systems. 2010. Disponível em: < http://www4.cs.fau.de/Lehre/SS10/HS_AKSS/papers/02_Ausarbeitung_Joerg_Brendel.pdf >. Acessado em: Ago/2014.

BUSSAB, W O.; MORETTIN, P. A. Estatística básica. Saraiva, 2010.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on. Ieee, 2008. p. 5-13.

BUYYA, R. High Performance Cluster Computing: Architecture and Systems, Prentice Hall, Upper SaddleRiver, NJ, USA, v. 1, p. 999, 1999.

CALHEIROS, R. N.; RANJAN, R.; BELOGLAZOV, A.; DE ROSE, C. A.; BUYYA, R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, v. 41, n. 1, p. 23-50, 2011.

CAMPOS, V.; VASCONCELOS, A. B. Teste de desempenho em desktops virtuais. Centro Universitário Ritter dos Reis. 2009.

CAMPOS, G. M. Estatística Prática para Docentes e Pós-Graduandos: Transformação dos dados amostrais. Disponível em: < http://www.forp.usp.br/restauradora/gmc/gmc_livro/ gmc_livro_cap13.html>. Acessado em: Mar/2015. 2000.

HO, H. E. T. VOLTAIC: Um Sistema De Gerência Automatizada De Recursos De Nós Virtuais Para Computação Em Nuvens. 2012. Tese de Doutorado. Universidade Federal do Rio de Janeiro.

CHIARANDINI, M.; PAQUETE, L.; PREUSS, M. eds. Experimental methods for the analysis of optimization algorithms. Germany: Springer, 2010.

COFFIN, M.; SALTZMAN, M. J. Statistical analysis of computational tests of algorithms and heuristics. INFORMS Journal on Computing, v. 12, n. 1, p. 24-44, 2000.

ConVirt: Overview. Disponível em: < http://www.convirture.com/products.php>. 2013. Acessado em: jul/2013.

COSTA, C. R.; LONGO, H. Condução de Experimentos Computacionais com Métodos Heurísticos. Anais XLIII Simpósio Brasileiro de Pesquisa Operacional. - Ubatuba, SP. 2011.

COUTINHO, E. F.; REGO, P. A.; GOMES, D. G.; DE SOUZA, J. N. Análise de Desempenho com Benchmarks em um Ambiente Publico de Computaç ao em Nuvem. Proc. of the X Workshop em Clouds e Aplicações. Belo Horizonte. 2012. p.96-109.

CREASY, R. J. The origin of the VM/370 time-sharing system. IBM Journal of Research and Development, v. 25, n. 5, p. 483-490, 1981.

CROWDER, H. P.; DEMBO, R. S.; MULVEY, J. M. Reporting computational experiments in mathematical programming. Mathematical Programming, v. 15, n. 1, p. 316-329, 1978.

DEAN, A.; VOSS, D. Design and Analysis of Experiments. New York, USA : Springer Link. 1999.

DESHANE, T.; SHEPHERD, Z.; MATTHEWS, J.; BEN-YEHUDA, M.; SHAH, A.; RAO, B. Quantitative comparison of Xen and KVM. *Xen Summit, Boston, MA, USA*, p.1-2. 2008.

EIBEN, A. E.; HINTERDING, R.; MICHALEWICZ, Z. Parameter control in evolutionary algorithms. Evolutionary Computation, IEEE Transactions on, v. 3, n. 2, p. 124-141, 1999.

ENDO, P. T.; GONÇALVES, G. E.; KELNER, J.; SADOK, D. A Survey on Open-source Cloud Computing Solutions. VIII Workshop em Clouds, Grids e Aplicações. - Gramado/RS, Brasil. p. 3-16. 2010.

ERCAN, T. Effective use of cloud computing in educational institutions. Procedia - Social and Behavioral Sciences. - Izmir, Turkey. v.2, n.2, p.938-942. 2010.

ESTATCAMP. Teste de Shapiro-Wilk. Disponível em : <http://www.portalaction. com.br/content/64-teste-de-shapiro-wilk>. Acessado em: Nov/2014.

Examples Apache Sudoku Code. Disponível em: <https://svn.apache.org/repos/asf/hadoop/ common/branches/MAPREDUCE-233/src/examples/org/apache/hadoop/examples/dancing/ Sudoku.java. Acessado em: nov/2014.

FERNANDEZ A.; MARCELINO J.; MARQUES P. Cloud computing. INPI – Instituto Nacional de propriedade industrial: Departamento de Patentes de Modelos de Utilidade. - 2011.

Fernandez E. S. Package 'Johnson'. Disponível em: <http://cran.r-project.org/web/packages/ Johnson/Johnson.pdf> Acessado em: Mar/2015. 2015.

FIGUEIREDO, R.; DINDA, P. A. FORTES, J.; Guest Editors' Introduction: Resource Virtualization Renaissance. Computer, v. 38, n. 5, p. 28-31, 2005.

FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. Cloud computing and grid computing 360-degree compared. In: Grid Computing Environments Workshop, 2008. GCE'08. IEEE, p. 1-10. 2008.

GENT, I. P.; KOTTHOFF, L. Reliability of Computational Experiments on Virtualised Hardware. In: AI for Data Center Management and Cloud Computing. 2011.

GHOSH, R.; TRIVEDI, K. S.; NAIK, V. K., KIM, D. S. End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach. In: Dependable Computing (PRDC), 2010 IEEE 16th Pacific Rim International Symposium on. IEEE, p. 125-132. 2010.

GOERKE, F.; HEISE, A.; JAEGER, D.; PÖPKE, C.; REICHEL, S. Security Management Platform: Documentation. Network Security in Practice - Winter Term 09/10. Potsdam, Alemanha. 2010.

GOLDFELD, S. M.; QUANDT, R. E. Some tests for homoscedasticity. Journal of the American Statistical Association, v. 60, n. 310, p. 539-547, 1965.

GOLDMAN, A.; KON, F.; JUNIOR, F. P.; POLATO, I.; DE FÁTIMA PEREIRA, R. Apache Hadoop: Conceitos teóricos e práticos, evoluçao e novas possibilidades. XXXI Jornadas de atualizaçoes em informatica. p. 88-136. 2012.

Hadoop Getting Started with Hadoop Core. Disponível em: <http://www.odbms.org/wp-content/uploads/2014/02/Pro-Hadoop-Ch.-1.pdf>. Acessado em: nov/2014.

HARRISON, M. J.; MCCABE, B. PM. A test for heteroscedasticity based on ordinary least squares residuals. Journal of the American Statistical Association, v. 74, n. 366a, p. 494-499, 1979.

HARSTELN, R. E.; DO AMARAL FILHO, J. R.; WERNER, L. Análise de capacidade de dados não normais de um sistema de tratamento de efluente industrial. INGEPRO-Inovação, Gestão e Produção, v. 2, n. 11, p. 013-025, 2010.

HAYES, B. Cloud computing. Communications of the ACM, v. 7, p. 9-11. 2008.

HEO, J.; ZHU, X.; PADALA, P.; WANG, Z. Memory overbooking and dynamic control of Xen virtual machines in consolidated environments. In: Integrated Network Management, 2009. IM'09. IFIP/IEEE International Symposium on. IEEE, p. 630-637. 2009.

HOOKER, J. N. Needed: An empirical science of algorithms. Operations Research, v. 42, n. 2, p. 201-212, 1994.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K.; STÜTZLE, T. ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research, v. 36, n. 1, p. 267-306, 2009.

HWANG, J.; ZENG, S.; WOOD, T. A component-based performance comparison of four hypervisors. In: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on. IEEE, p. 269-276. 2013.

IBM Virtualization in education, IBM Global Education. 2007. Disponível em: <http://www-07.ibm.com/solutions/in/education/download/Virtualization>. Acessado em: nov/2014.

JAGANNATHAN, Srivatsan. Comparison and Evaluation of Open-Source Cloud Management Software. KTH Royal Institute of Technology. Stockholm, Sweden, 2012.

JAIN, R. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. New York : John Wiley and Sons Inc, 1991.

JAMSA, K. A. Cloud computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security, and More. Burlington, MA : Jones & Bartlett Learning, 2013.

JOHNSON, D. S. A theoretician's guide to the experimental analysis of algorithms. Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges, v. 59, p. 215-250, 2002.

JUNGELS, G. M. Virtualization in high-performance computing: An analysis of physical and virtual node performance. College of Technology Masters Theses. Purdue University Libraries. - 2012.

KELLER, E.; REXFORD J. The Platform as a Service Model for Networking [Periódico] // Internet Networking Management Workshop / Workshop on Research on Enterprise Networking. - 2010. - http://static.usenix.org/event/inmwren10/tech/full_papers/Keller.pdf.

KENNEDY, P. A Guide to Econometrics. Massachusetts, USA : Wiley-Blackwell, v. 6. 2008.

KHAJEH-HOSSEINI, A.; GREENWOOD, D.; SOMMERVILLE, I. Cloud migration: A case study of migrating an enterprise it system to iaas. In: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on. IEEE, p. 450-457. 2010.

KUNDU, S.; RANGASWAMI, R.; GULATI, A.; ZHAO, M.; DUTTA, K. Modeling virtualized applications using machine learning techniques. SIGPLAN Not.v. 47, n.7, p. 3-14. 2012.

LAUREANO, M. Máquinas Virtuais e Emuladores. Conceitos, Técnicas e Aplicações. São Paulo: Novatec, 2006.

LI, J.; WANG, Q.; JAYASINGHE, D.; PARK, J.; ZHU, T.; PU, C. Performance Overhead Among Three Hypervisors: An Experimental Study using Hadoop Benchmarks. In: Big Data (BigData Congress), 2013 IEEE International Congress on. IEEE, p. 9-16. 2013.

LILLIEFORS, H. W. On the Kolmogorov-Smirnov test for normality with mean and variance unknown. Journal of the American Statistical Association, v. 62, n. 318, p. 399-402, 1967.

LONGO, F.; GHOSH, R.; NAIK, V. K.; TRIVEDI, K. S. A scalable availability model for infrastructure-as-a-service cloud. In: Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on. IEEE, p. 335-346. 2011.

LOVIE, P. Interaction Plot. Wiley StatsRef: Statistics Reference Online, 2014.

LUNA, A. J. Degradação Fotoquímica do Fenol, 2,4- Diclorofenol e Ácido 2,4-Diclorofenoxiacético em Meio de Alta Salinidade. Tese de doutorado apresentada à Escola Politécnica da Universidade de São Paulo; Disponível em: <http://www.nupeg.ufrn.br/foteq/pdfs/TESIS-ultimaversao.pdf>. Acessada em Mar/2015. 2004.

MACHADO, C. P.; LOUREIRO C. A. H. Comparação de ferramentas de software para administração de nuvem privada. Universidade Luterana do Brasil (Ulbra) - Campus Canoas. - Canoas, RS, 2011.

MBI Mayer&Bunge 2ª Pesquisa Anual sobre Cloud Computing. MBI Mayer&Bunge Informática.. - MBI Mayer&Bunge, 2011 Disponível em: <http://www.mbi.com.br/mbi/biblioteca/relatorios/2011-03-2a-pesquisa-anual-sobre-cloud-computing/>. Acessado em: 15 jul. 2013.

MENASCÉ, D. A. Virtualization: Concepts, applications, and performance modeling. In 31th International Computer Measurement Group Conference. - Orlando, Florida, USA : Computer Measurement Group, p. 407–414. 2005.

MERGEN, M. F.; UHLIG, V.; KRIEGER, O.; XENIDIS, J. Virtualization for high-performance computing. ACM SIGOPS Operating Systems Review, v. 40, n. 2, p. 8-11, 2006.

Microsystems Sun Introduction to Cloud Computing Architecture Disponível em: <http://pt.scribd.com/doc/17028937/Cloud-Computing> 2009. Acessado em: nov/2014.

MONTGOMERY, D. Design and Analysis of Experiments. John Wiley and Sons, v. 7. 2010.

MONTGOMERY, D.; RUNGER G. Estatística aplicada e probabilidade para engenheiros LTC, 2009.

MARÔCO, J. Análise estatística com o SPSS Statistics. ReportNumber, Lda, 2011.

NANDA, S.; CHIUEH, T. A survey of virtualization technologies, Technical report. 2008 Disponível em: <http://www.ittc.ku.edu/kulkarni/teaching/archieve/EECS800-Spring-2008/surveyvirt ualizationtechnologies.pdf>. Acessado em: nov/2014.

NETO, B. B.; ISCARMINIO I. S.; BRUNS R. E. Como fazer experimentos: pesquisa e desenvolvimento na ciência e na indústria. Porto Alegre : Bookman, v. 4. 2010.

NURMI, D.; WOLSKI, R.; GRZEGORCZYK, C.; OBERTELLI, G.; SOMAN, S.; YOUSEFF, L.; ZAGORODNOV, D. The eucalyptus open-source cloud-computing system. In: Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on. IEEE, p. 124-131. 2009.

OPENNEBULA.ORG An Overview of OpenNebula 4.0. 2013. Disponível em:< http://opennebula.org/documentation:rel4.0:intro> Acessado em: nov/2014.

OpenQRM. Concepts: Complete separation of hardware (physical servers and virtual machines) from software (server-images). 2013. Disponível em: <http://www.openqrm-enterprise.com/about-openqrm/concepts.html> Acessado em nov/2014.

OpenStack.org OpenStack: An Overview. 2013. Disponível em: < http://www.openstack.org/downloads/openstack-overview-datasheet.pdf.> Acessado em nov/2014.

PAIS, M. S.; PERETTA, I. S.; YAMANAKA, K.; PINTO, E. R. Factorial design analysis applied to the performance of parallel evolutionary algorithms. Journal of the Brazilian Computer Society, v. 20, n. 1, p. 1-17, 2014.

PAIS, M. S. Estudo da Influência dos Parâmetros de Algoritmos Paralelos da Computação Evolutiva no seu Desempenho em Plataformas Multicore. Uberlândia : Tese de doutorado: Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia, 2014.

PALLISTER, J.; HOLLIS, S. J.; BENNETT, J. Identifying Compiler Options to Minimize Energy Consumption for Embedded Platforms.The Computer Journal, p. bxt129, 2013.

PASAMONTES, A.; CALLAO, P.. Fractional factorial design and simplex algorithm for optimizing sequential injection analysis (SIA) and second order calibration. Chemometrics and intelligent laboratory systems, v. 83, n. 2, p. 127-132, 2006.

PENG, J.; ZHANG, X.; LEI, Z.; ZHANG, B.; ZHANG, W.; & LI, Q. Comparison of Several Cloud Computing Platforms. Information Science and Engineering (ISISE), 2009 Second International Symposium on. - Shanghai, China. pp. 23-27. 2009.

PETROVSKI, A.; BROWNLEE, A.; MCCALL, J. Statistical optimisation and tuning of GA factors. In: Proceedings of the IEEE Congress on Evolutionar Computation (CEC 2005), v. 1. IEEE, 2005.

PFITSCHER, R. J.; PILLON, M. A.; OBELHEIRO, R. R. Diagnóstico do provisionamento de recursos para máquinas virtuais em nuvens IaaS. 31º Simpósio Brasileiro de Redes de Computadores (SBRC), p. 599-612. 2013.

PiEstimulator Hadoop Examples. 2013. Disponível em: <http://svn.apache.org/repos/asf/hadoop/common/branches/branch0.20/src/examples/org/apache/hadoop/examples/PiEstimator.java>. Acessado em: nov/2014.

POPEK, G. J.; GOLDBERG, R. P. Formal requirements for virtualizable third generation architectures. Communications of the ACM, v. 17, n. 7, p. 412-421, 1974.

RAO, J.; BU, X.; XU, C. Z.; WANG, K. A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In: Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on. IEEE. 45-54. 2011.

RARDIN, R. L.; UZSOY, R.. Experimental evaluation of heuristic optimization algorithms: A tutorial. Journal of Heuristics, v. 7, n. 3, p. 261-304, 2001.

RAWLINGS, J. O.; PANTULA S. G.; DICKEY D. A. Applied Regression Analysis: A Research Tool. New York : Springer. v. 5. 1998.

RAZALI, N. M.; WAH, Y. B. x. Journal of Statistical Modeling and Analytics, v. 2, n. 1, p. 21-33, 2011.

RIDGE, E.; KUDENKO, D. Tuning an algorithm using design of experiments. In: Experimental methods for the analysis of optimization algorithms. Springer Berlin Heidelberg, p. 265-286. 2010.

RODERO-MERINO, L., VAQUERO, L. M., GIL, V., GALÁN, F., FONTÁN, J., MONTERO, R. S., & LLORENTE, I. M. From infrastructure delivery to service management in clouds. Future Generation Computer Systems, v. 26, n. 8, p. 1226-1240, 2010.

RODRIGUES, M. I.; IEMMA, A. F. Planejamento de experimentos e otimização de processos. Campinas, SP : Casa do Espírito Amigo Fraternidade Fé e Amor, 2009.

ROLZ, C. E. Computer and Information Science Applications in Bioprocess Engineering NATO ASI Series. Statistical Design and Analysis of Experiments.v. 305. p. 143-156. 1996.

ROSE, R. Survey of system virtualization techniques. 2004. Disponível em: <http://ir.library.oregonstate.edu/xmlui/bitstream/handle/1957/9907/rose-virtualization.pdf ?sequence=1>. Acessado em jan/2015.

ROSENBLUM, M.; GARFINKEL, T. Virtual machine monitors: Current technology and future trends. Computer, v. 38, n. 5, p. 39-47, 2005.

SCHAD, J.; DITTRICH, J.; QUIANÉ-RUIZ, J. Runtime measurements in the cloud: observing, analyzing, and reducing variance.Proceedings of the VLDB Endowment, v. 3, n. 1-2, p. 460-471, 2010.

SHAPIRO, S. S.; WILK, M. B. An analysis of variance test for normality (complete samples). Biometrika, p. 591-611, 1965.

SIMONOFF, J. S. Regression diagnostics Disponível em: <http://people.stern. nyu.edu/jsimonof/ classes/2301/pdf/diagnost.pdf>. Acessado em: nov/2014.

SLIFKER, J. F.; SHAPIRO, S. S. The johnson system: selection and parameter estimation. Technometrics, v. 22, n. 2, p. 239-246, 1980.

SOTOMAYOR, B.; MONTERO, R. S.; LLORENTE, I. M.; FOSTER, I. Virtual infrastructure management in private and hybrid clouds. Internet Computing, IEEE, v. 13, n. 5, p. 14-22, 2009.

SOUSA, F. R. C.; MOREIRA, L. O.; MACHADO J. C. Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios. UFPI: Universidade Federal do Piaui : ERCEMAPI'09, 2009.

SOUTHERN, G.; HWANG, D.; BARNES, R. SMP virtualization performance evaluation. In: Proc of 2nd Int Workshop on Virtualization Performance: Analysis, Characterization, and Tools. Boston: ISPASS. p. 1-15. 2009.

SOUZA, F. B. Uma arquitetura para Monitoramento e Medição de Desempenho para Ambientes Virtuais. Tese de Doutorado da Universidade Federal de Minas Gerais. - Belo Horizonte. 2006.

SRIDHARAN, S. A Performance Comparison of Hypervisors for Cloud Computing [Artigo] // A thesis submitted to the School of Computing. University of Noth Florida. Florida, USA, 2012.

STATISTICA. Analysis of an experiment with To-factors - ANOVA/Effects tab. Disponível em: < http://documentation.statsoft.com/STATISTICAHelp.aspx?path=Experimental/Doe/Dialogs/TwoLevel/AnalysisofanExperimentwithTwoLevelFactorsANOVAEffectsTab>. Acessado em Mar/2015. 2015.

SUN, X. H.; HE, D.; CAMERON, K. W.; LUO, Y. A factorial performance evaluation for hierarchical memory systems. In: Parallel Processing, 1999. 13th International and 10th Symposium on Parallel and Distributed Processing, 1999. 1999 IPPS/SPDP. Proceedings. IEEE, p. 70-74. 1999.

SYED, R.; ROBINSON, B.; WILLIAMS, L. Does hardware configuration and processor load impact software fault observability?. Software Testing, Verification and Validation (ICST). Third International Conference on, p. 285–294. 2010.

TAMHANE, A. C. High Performance Cluster Computing: Architectures and Systems. John Wiley & Sons, Inc. 1999.

TestDFSIO Teste de TestDFSIO. Disponível em: <https://github.com/intel-hadoop/HiBench/blob/master/common/autogen/src/org/apache/hadoop/fs/dfsioe/TestDFSIO.java>. Acessado em: dz/2014.

TIAN, W.; SU, S.; LU, G.. A framework for implementing and managing platform as a service in a virtual cloud computing lab. In:Education Technology and Computer Science (ETCS), 2010 Second International Workshop on. IEEE, p. 273-276. 2010.

Tutorial MapReduce WordCount v2. Disponível em: <http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v2.0>. Acessado em: nov/2014.

VAQUERO, L. M. RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds: towards a cloud definition.ACM SIGCOMM Computer Communication Review, v. 39, n. 1, p. 50-55, 2008.

VASIĆ, N.; NOVAKOVIĆ, D.; MIUČIN, S.; KOSTIĆ, D.; BIANCHINI, R. Dejavu: accelerating resource allocation in virtualized environments. ACM SIGARCH Computer Architecture News, v. 40, n. 1, p. 423-436, 2012.

VECCHIOLA, C.; CHU, X.; BUYYA, R.. Aneka: a software platform for .NET-based cloud computing. High Speed and Large Scale Scientific Computing, p. 267-295, 2009.

VENABLES W. N.; SMITH, D. M. An Introduction to R Notes on R: A Programming Environment for Data Analysis and Graphics. Version: 3.1.3. Disponível em: < http://cran.r-project.org/doc/manuals/R-intro.pdf>. Acessado em: Março/2015.

VERAS, M. Datacenter - Componente Central da Infraestrutura de TI.São Paulo:Brasport, 2009.

VERAS, M. Virtualização: Componente Central do Datacenter. Rio de Janeiro:Brasport, 2011.

VOUK, M. Cloud computing–issues, research and implementations.CIT. Journal of Computing and Information Technology, v. 16, n. 4, p. 235-246, 2008.

WINTER, J. C. F. Using the Student's t-test with extremely small sample sizes. Practical Assessment, Research & Evaluation, v. 18, n. 10, p. 2, 2013.

YOUSEFF, L.; BUTRICO, M.; DA SILVA, D. Toward a unified ontology of cloud computing. In: Grid Computing Environments Workshop, 2008. GCE'08. IEEE, p. 1-10. 2008.

ZHANG, L.; ZHOU, Q. CCOA: Cloud computing open architecture. In: Web Services, 2009. ICWS 2009. IEEE International Conference on. Ieee, p. 607-616. 2009.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications, v. 1, n. 1, p. 7-18, 2010.

ZITZLER, E., Laumanns M. e Thiele L. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH). - Zurich. 2001.

ZIVIANI, N. Projeto de Algoritmos: com implementações em Pascal e C. São Paulo:Pioneira Thomson Learning, v. 2. 2008.

Anexos

Anexo A – Teste de normalidade Shapiro-Wilk

Segundo Shapiro et al (1965) os valores críticos da estatística W de Shapiro-Wilk são dados na Tabela 41. A letra *N* representa a quantidade de repetições.

Tabela 41: Estatística W para o teste de Shapiro-Wilk

	Nível de significância								
N	**0,01**	**0,02**	**0,05**	**0,1**	**0,5**	**0,9**	**0,95**	**0,98**	**0,99**
3	0,753	0,756	0,767	0,789	0,959	0,998	0,999	1	1
4	0,687	0,707	0,748	0,792	0,935	0,987	0,992	0,996	0,997
5	0,686	0,715	0,762	0,806	0,927	0,979	0,986	0,991	0,993
6	0,713	0,743	0,788	0,826	0,927	0,974	0,981	0,986	0,989
7	0,73	0,76	0,803	0,838	0,928	0,972	0,979	0,985	0,988
8	0,749	0,778	0,818	0,851	0,932	0,972	0,978	0,984	0,987
9	0,764	0,791	0,829	0,859	0,935	0,972	0,978	0,984	0,986
10	0,781	0,806	0,842	0,869	0,938	0,972	0,978	0,983	0,986
11	0,792	0,817	0,85	0,876	0,94	0,973	0,979	0,984	0,986
12	0,805	0,828	0,859	0,883	0,943	0,973	0,979	0,984	0,986
13	0,814	0,837	0,866	0,889	0,945	0,974	0,979	0,984	0,986
14	0,825	0,846	0,874	0,895	0,947	0,975	0,98	0,984	0,986
15	0,835	0,855	0,881	0,901	0,95	0,975	0,98	0,984	0,987
16	0,844	0,863	0,887	0,906	0,952	0,976	0,981	0,985	0,987
17	0,851	0,869	0,892	0,91	0,954	0,977	0,981	0,985	0,987
18	0,858	0,874	0,897	0,914	0,956	0,978	0,982	0,986	0,988
19	0,863	0,879	0,901	0,917	0,957	0,978	0,982	0,986	0,988
20	0,868	0,884	0,905	0,92	0,959	0,979	0,983	0,986	0,988
21	0,873	0,888	0,908	0,923	0,96	0,98	0,983	0,987	0,989
22	0,878	0,892	0,911	0,926	0,961	0,98	0,984	0,987	0,989

23	0,881	0,895	0,914	0,928	0,962	0,981	0,984	0,987	0,989
24	0,884	0,898	0,916	0,93	0,963	0,981	0,984	0,987	0,989
25	0,888	0,901	0,918	0,931	0,964	0,981	0,985	0,988	0,989
26	0,891	0,904	0,92	0,933	0,965	0,982	0,985	0,988	0,989
27	0,894	0,906	0,923	0,935	0,965	0,982	0,985	0,988	0,99
28	0,896	0,908	0,924	0,936	0,966	0,982	0,985	0,988	0,99
29	0,898	0,91	0,926	0,937	0,966	0,982	0,985	0,988	0,99
30	0,9	0,912	0,927	0,939	0,967	0,983	0,985	0,988	0,99
31	0,902	0,914	0,929	0,94	0,967	0,983	0,986	0,988	0,99
32	0,904	0,915	0,93	0,941	0,968	0,983	0,986	0,988	0,99
33	0,906	0,917	0,931	0,942	0,968	0,983	0,986	0,989	0,99
34	0,908	0,919	0,933	0,943	0,969	0,983	0,986	0,989	0,99
35	0,91	0,92	0,934	0,944	0,969	0,984	0,986	0,989	0,99
36	0,912	0,922	0,935	0,945	0,97	0,984	0,986	0,989	0,99
37	0,914	0,924	0,936	0,946	0,97	0,984	0,987	0,989	0,99
38	0,916	0,925	0,938	0,947	0,971	0,984	0,987	0,989	0,99
39	0,917	0,927	0,939	0,948	0,971	0,984	0,987	0,989	0,991
40	0,919	0,928	0,94	0,949	0,972	0,985	0,987	0,989	0,991
41	0,92	0,929	0,941	0,95	0,972	0,985	0,987	0,989	0,991
42	0,922	0,93	0,942	0,951	0,972	0,985	0,987	0,989	0,991
43	0,923	0,932	0,943	0,951	0,973	0,985	0,987	0,99	0,991
44	0,924	0,933	0,944	0,952	0,973	0,985	0,987	0,99	0,991
45	0,926	0,934	0,945	0,953	0,973	0,985	0,988	0,99	0,991
46	0,927	0,935	0,945	0,953	0,974	0,985	0,988	0,99	0,991
47	0,928	0,936	0,946	0,954	0,974	0,985	0,988	0,99	0,991
48	0,929	0,937	0,947	0,954	0,974	0,985	0,988	0,99	0,991
49	0,929	0,938	0,947	0,955	0,974	0,985	0,988	0,99	0,991
50	0,93	0,939	0,947	0,955	0,974	0,985	0,988	0,99	0,991

Anexo B –Teste de Correlação de Durbin-Watson

Segundo Rawlings, et al., (1998) apresenta na Tabela 42 os valores críticos d_L e d_U a um nível de significância de 5% para o teste de correlação Durbin-Watson.

Tabela 42: Pontos significantes de dL e dU para o teste de correlação de Durbin-Watson

Nível de significância de 5%										
n	***k = 1***		***k = 2***		***k = 3***		***k = 4***		***k = 5***	
	d_L	d_U	d_L	d_U	d_L	d_U	d_L	d_U	d_L	d_U
15	1.08	1.36	.95	1.54	.82	1.75	.69	1.97	.56	2.21
16	1.10	1.37	.98	1.54	.86	1.73	.74	1.93	.62	2.15
17	1.13	1.38	1.02	1.54	.90	1.71	.78	1.90	.67	2.10
18	1.16	1.39	1.05	1.53	.93	1.69	.82	1.87	.71	2.06
19	1.18	1.40	1.08	1.53	.97	1.68	.86	1.85	.75	2.02
20	1.20	1.41	1.10	1.54	1.00	1.68	.90	1.83	.79	1.99
21	1.22	1.42	1.13	1.54	1.03	1.67	.93	1.81	.83	1.96
22	1.24	1.43	1.15	1.54	1.05	1.66	.96	1.80	.86	1.94
23	1.26	1.44	1.17	1.54	1.08	1.66	.99	1.79	.90	1.92
24	1.27	1.45	1.19	1.55	1.10	1.66	1.01	1.78	.93	1.90
25	1.29	1.45	1.21	1.55	1.12	1.66	1.04	1.77	.95	1.89
26	1.30	1.46	1.22	1.55	1.14	1.65	1.06	1.76	.98	1.88
27	1.32	1.47	1.24	1.56	1.16	1.65	1.08	1.76	1.01	1.86
28	1.33	1.48	1.26	1.56	1.18	1.65	1.10	1.75	1.03	1.85
29	1.34	1.48	1.27	1.56	1.20	1.65	1.12	1.74	1.05	1.84
30	1.35	1.49	1.28	1.57	1.21	1.65	1.14	1.74	1.07	1.83
31	1.36	1.50	1.30	1.57	1.23	1.65	1.16	1.74	1.09	1.83
32	1.37	1.50	1.31	1.57	1.24	1.65	1.18	1.73	1.11	1.82
33	1.38	1.51	1.32	1.58	1.26	1.65	1.19	1.73	1.13	1.81
34	1.39	1.51	1.33	1.58	1.27	1.65	1.21	1.73	1.15	1.81
35	1.40	1.52	1.34	1.58	1.28	1.65	1.22	1.73	1.16	1.80
36	1.41	1.52	1.35	1.59	1.29	1.65	1.24	1.73	1.18	1.80
37	1.42	1.53	1.36	1.59	1.31	1.66	1.25	1.72	1.19	1.80
38	1.43	1.54	1.37	1.59	1.32	1.66	1.26	1.72	1.21	1.79
39	1.43	1.54	1.38	1.60	1.33	1.66	1.27	1.72	1.22	1.79
40	1.44	1.54	1.39	1.60	1.34	1.66	1.29	1.72	1.23	1.79
45	1.48	1.57	1.43	1.62	1.38	1.67	1.34	1.72	1.29	1.78
50	1.50	1.59	1.46	1.63	1.42	1.67	1.38	1.72	1.34	1.77
55	1.53	1.60	1.49	1.64	1.45	1.68	1.41	1.72	1.38	1.77
60	1.55	1.62	1.51	1.65	1.48	1.69	1.44	1.73	1.41	1.77
65	1.57	1.63	1.54	1.66	1.50	1.70	1.47	1.73	1.44	1.77
70	1.58	1.64	1.55	1.67	1.52	1.70	1.49	1.74	1.46	1.77
75	1.60	1.65	1.57	1.68	1.54	1.71	1.51	1.74	1.49	1.77
80	1.61	1.66	1.59	1.69	1.56	1.72	1.53	1.74	1.51	1.77
85	1.62	1.67	1.60	1.70	1.57	1.72	1.55	1.75	1.52	1.77
90	1.63	1.68	1.61	1.70	1.59	1.73	1.57	1.75	1.54	1.78
95	1.64	1.69	1.62	1.71	1.60	1.73	1.58	1.75	1.56	1.78
100	1.65	1.69	1.63	1.72	1.61	1.74	1.59	1.76	1.57	1.7

Anexo C – Matriz de experimentos

A tabela 43 apresenta a matriz de experimentos que foi utilizada planejamento fatorial de 3^4.

Tabela 43: Matriz de experimentos

	Valores codificados				Valores Reais			
Exp.	**Núcleo**	**Memória**	**SO**	**VM**	**Núcleo**	**Memória**	**SO**	**VM**
1	–	–	–	–	1	1	Ubuntu	KVM
2	0	–	–	–	2	1	Ubuntu	KVM
3	+	–	–	–	3	1	Ubuntu	KVM
4	–	0	–	–	1	3	Ubuntu	KVM
5	0	0	–	–	2	3	Ubuntu	KVM
6	+	0	–	–	3	3	Ubuntu	KVM
7	–	+	–	–	1	5	Ubuntu	KVM
8	0	+	–	–	2	5	Ubuntu	KVM
9	+	+	–	–	3	5	Ubuntu	KVM
10	–	–	0	–	1	1	CentOS	KVM
11	0	–	0	–	2	1	CentOS	KVM
12	+	–	0	–	3	1	CentOS	KVM
13	–	0	0	–	1	3	CentOS	KVM
14	0	0	0	–	2	3	CentOS	KVM
15	+	0	0	–	3	3	CentOS	KVM
16	–	+	0	–	1	5	CentOS	KVM
17	0	+	0	–	2	5	CentOS	KVM
18	+	+	0	–	3	5	CentOS	KVM
19	–	–	+	–	1	1	Windows	KVM
20	0	–	+	–	2	1	Windows	KVM
21	+	–	+	–	3	1	Windows	KVM
22	–	0	+	–	1	3	Windows	KVM
23	0	0	+	–	2	3	Windows	KVM
24	+	0	+	–	3	3	Windows	KVM
25	–	+	+	–	1	5	Windows	KVM
26	0	+	+	–	2	5	Windows	KVM
27	+	+	+	–	3	5	Windows	KVM
28	–	–	–	0	1	1	Ubuntu	Xen
29	0	–	–	0	2	1	Ubuntu	Xen
30	+	–	–	0	3	1	Ubuntu	Xen
31	–	0	–	0	1	3	Ubuntu	Xen
32	0	0	–	0	2	3	Ubuntu	Xen
33	+	0	–	0	3	3	Ubuntu	Xen
34	–	+	–	0	1	5	Ubuntu	Xen
35	0	+	–	0	2	5	Ubuntu	Xen
36	+	+	–	0	3	5	Ubuntu	Xen
37	–	–	0	0	1	1	CentOS	Xen
38	0	–	0	0	2	1	CentOS	Xen
39	+	–	0	0	3	1	CentOS	Xen
40	–	0	0	0	1	3	CentOS	Xen
41	0	0	0	0	2	3	CentOS	Xen
42	+	0	0	0	3	3	CentOS	Xen

43	–	+	0	0	1	5	CentOS	Xen
44	0	+	0	0	2	5	CentOS	Xen
45	+	+	0	0	3	5	CentOS	Xen
46	–	–	+	0	1	1	Windows	Xen
47	0	–	+	0	2	1	Windows	Xen
48	+	–	+	0	3	1	Windows	Xen
49	–	0	+	0	1	3	Windows	Xen
50	0	0	+	0	2	3	Windows	Xen
51	+	0	+	0	3	3	Windows	Xen
52	–	+	+	0	1	5	Windows	Xen
53	0	+	+	0	2	5	Windows	Xen
54	+	+	+	0	3	5	Windows	Xen
55	–	–	–	+	1	1	Ubuntu	VMware
56	0	–	–	+	2	1	Ubuntu	VMware
57	+	–	–	+	3	1	Ubuntu	VMware
58	–	0	–	+	1	3	Ubuntu	VMware
59	0	0	–	+	2	3	Ubuntu	VMware
60	+	0	–	+	3	3	Ubuntu	VMware
61	–	+	–	+	1	5	Ubuntu	VMware
62	0	+	–	+	2	5	Ubuntu	VMware
63	+	+	–	+	3	5	Ubuntu	VMware
64	–	–	0	+	1	1	CentOS	VMware
65	0	–	0	+	2	1	CentOS	VMware
66	+	–	0	+	3	1	CentOS	VMware
67	–	0	0	+	1	3	CentOS	VMware
68	0	0	0	+	2	3	CentOS	VMware
69	+	0	0	+	3	3	CentOS	VMware
70	–	+	0	+	1	5	CentOS	VMware
71	0	+	0	+	2	5	CentOS	VMware
72	+	+	0	+	3	5	CentOS	VMware
73	–	–	+	+	1	1	Windows	VMware
74	0	–	+	+	2	1	Windows	VMware
75	+	–	+	+	3	1	Windows	VMware
76	–	0	+	+	1	3	Windows	VMware
77	0	0	+	+	2	3	Windows	VMware
78	+	0	+	+	3	3	Windows	VMware
79	–	+	+	+	1	5	Windows	VMware
80	0	+	+	+	2	5	Windows	VMware
81	+	+	+	+	3	5	Windows	VMware

Anexo D – Análise das demais variáveis respostas

Nessa seção são apresentados alguns resultados e análises do experimento para os 5 algoritmos analisados para as demais variáveis respostas: tempo gasto com o uso do *Garbage Collector*, tempo despendido como o uso da CPU (*CPU*), total de memória física, total de memória virtual e o total de *Heap, throughput* e taxa média de *I/O*. Os níveis dos fatores utilizados para a execução da matriz experimental está apresentada na Tabela 12.

D.1 Algoritmo *Pi*

A Figura 63 apresenta, da esquerda para a direita, informações exploratórias sobre as variáveis dependentes (*Garbage Collector*, CPU, memória física, memória virtual, *heap* e o tempo) respectivamente, encontradas durante a execução do algoritmo *Pi* nos ambientes virtuais analisados. O primeiro gráfico de cada variável resposta apresenta o diagrama de caixa com as respectivas respostas.

Pode-se verificar que para esse algoritmo o uso de *Garbage Collector* (Figura 63a) foi baixo, exceto alguns *outliers*. O uso da CPU (Figura 63b) na maioria dos ensaios (75%) ficou abaixo de 10000 milissegundos, Para as variáveis respostas: memória física (Figura 32c), virtual (Figura63d) e heap (Figura 63e) representou um grande uso durante a execução do algoritmo.

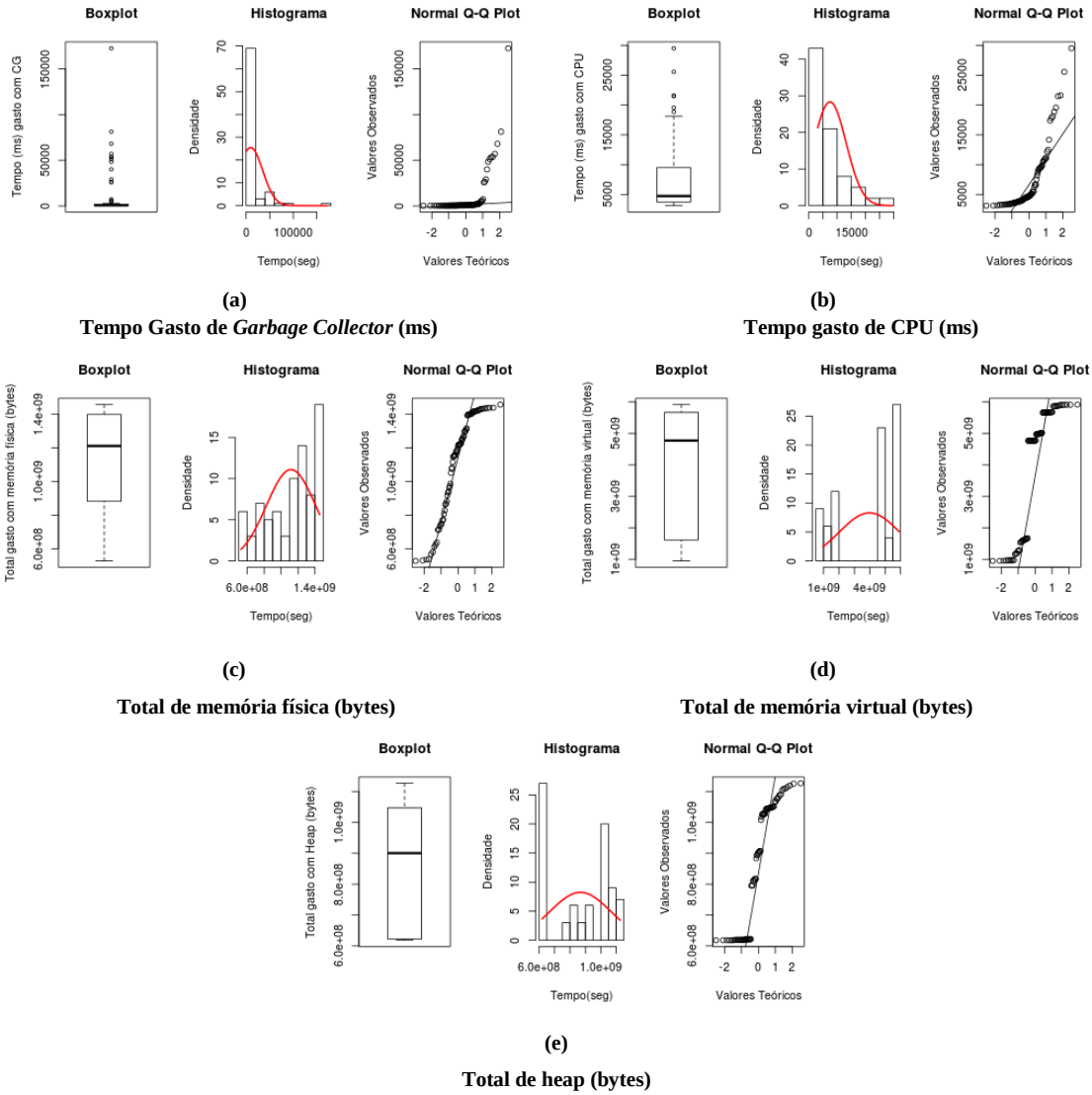
Os gráficos das normais da Figura 63 mostram que os dados coletados não se aproximam de uma distribuição normal o que caracteriza a necessidade de se aplicar uma transformação nos dados. Para cada conjunto de dados de cada variável resposta foi aplicada a transformação de Johnson.

Para cada variável resposta, o resultado da transformação de Johnson foi:

- *Garbage Collector*: família = SU, γ = -1.757143, λ = 17.78508, ε = 486.494 e η = 0.4363912;
- CPU: família = SB, γ = 1.477327, λ = 30306.15, ε = 3115.443 e η = 0.560928;
- Memória física: família = SB, γ = -0.4627434, λ = 941365392, ε = 515968988 e η = 0.4562478;

- Memória virtual: família = SB, γ = -0.1651402, λ = 4968229851, ε = 944335268 e η = 0.3164529;
- Heap: família = SB, γ = 0.480076, λ = 507740665, ε = 618614567 e η = 0.202535;

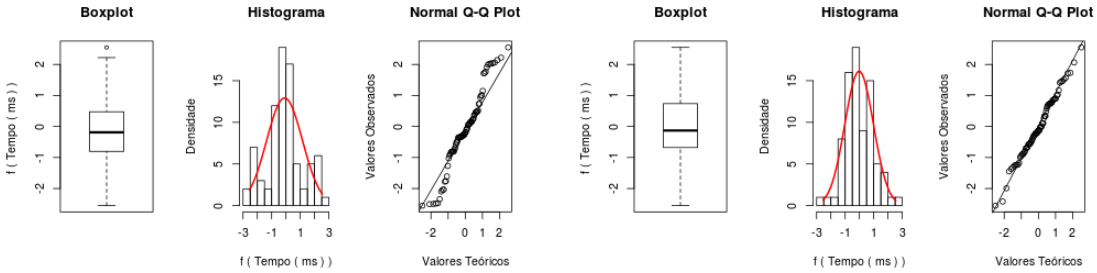
Figura 63: Gráficos exploratórios das variáveis respostas para o algoritmo *Pi*



A Figura 64 mostra os mesmos gráficos, porém após a transformação de Johnson. Os diagramas de caixas e os histogramas apresentam mais simétricos em relação aos apresentados na Figura 63. Por fim, em todos os gráficos os dados aproximam de uma distribuição normal.

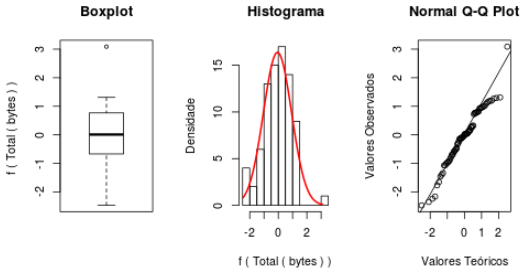
A Figura 65 mostra as mesmas informações, porém agrupadas através da combinação do fator sistema operacional pelo fator máquina virtual. As análises baseadas em *boxplot* são limitadas, podendo tirar conclusões somente quando os agrupamentos são óbvios. A Tabela 44 mostra o resultado da aplicação do teste *t*.

Figura 64: Gráficos exploratórios das variáveis respostas para o algoritmo *Pi* após transformação de Johnson

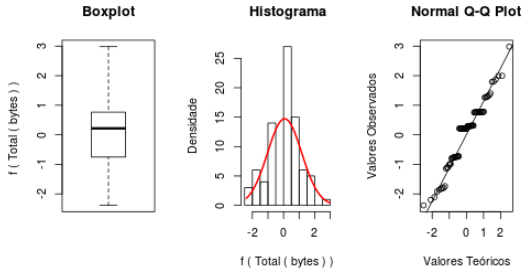


(a)
f(Tempo Gasto de Garbage Collector (ms))

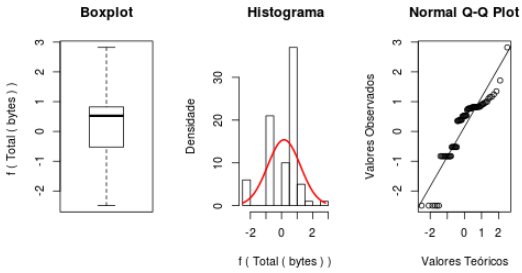
(b)
f(Tempo gasto de CPU (ms))



(c)
f(Total de Memória Física (bytes))



(d)
f(Total de Memória Virtual (bytes))



(e)
f(Total de Heap (bytes))

Portanto, pode-se verificar que não existem evidências estatísticas que a memória física (Figura 65c) e o heap (Figura 65e) diferem o desempenho entre os ambientes virtuais (Tabela 44). Analisando a variável resposta *Garbage Collector*, pode-se concluir que a melhor combinação é o sistema operacional Windows (nível +) com a máquina virtual VMware (nível +), ou seja o tempo (ms) gasto com o *Garbage Collector* é menor (Figura 65a).

Na Figura 65b, para o tempo gasto com o uso da CPU o fator sistema operacional no nível + (Windows) juntamente com o fator máquina virtual no nível – (KVM) teve o pior desempenho , ou seja utilizou a CPU por mais tempo. Já a combinação (Tabela 44) do sistema operacional Windows (nível +) juntamente com a máquina virtual VMware (nível +) para executar o mesmo algoritmo obteve o melhor desempenho, ou seja, utilizou menos CPU.

Figura 65: Diagrama de caixa (*boxplot*) das variáveis de respostas para o algoritmo *Pi*, após a transformação de Johnson: variáveis respostas agrupadas pela combinação do sistema operacional pela máquina virtual

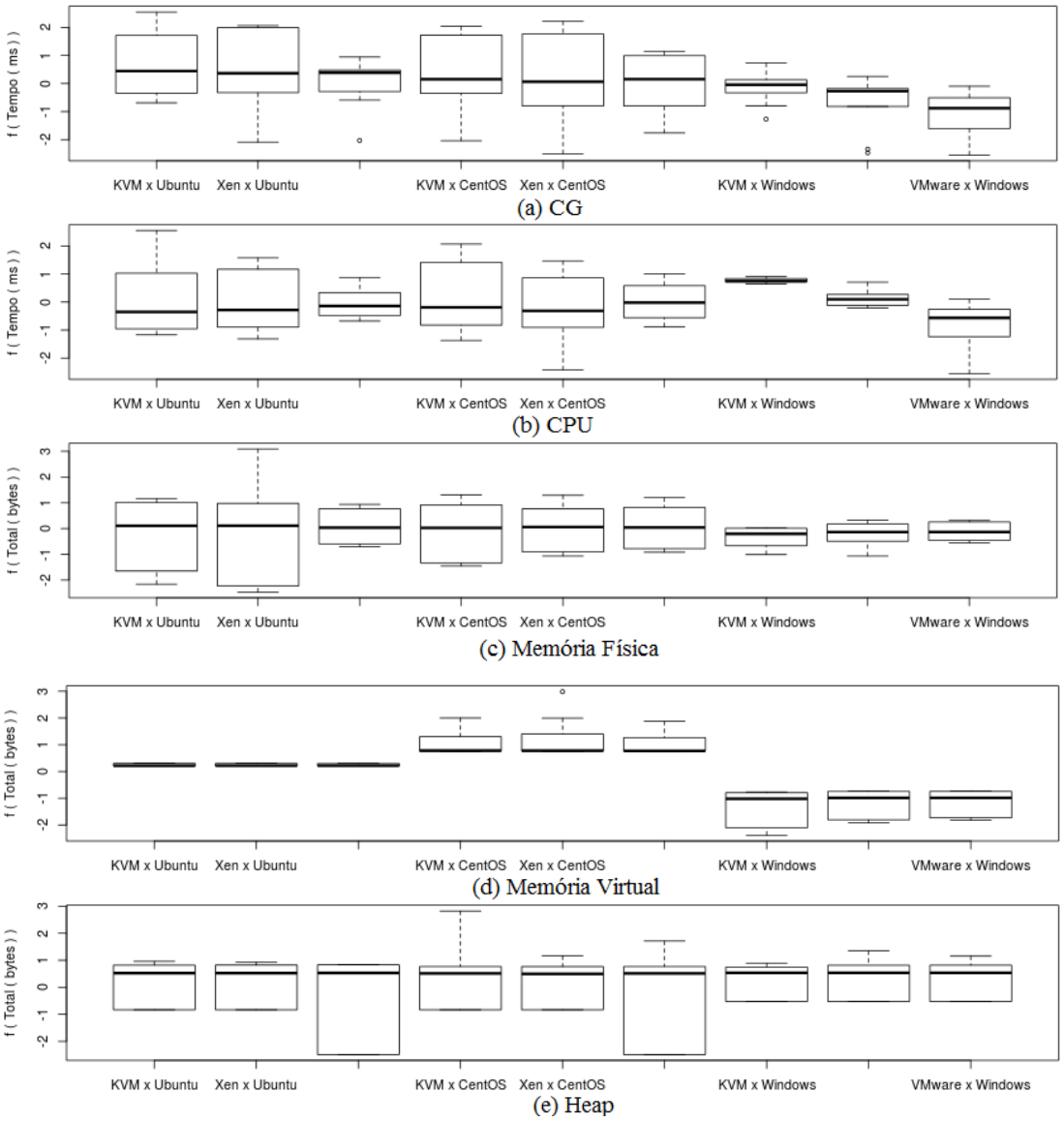


Tabela 44: Comparação entre os fatores após a transformação logarítmica: variáveis respostas agrupadas pela combinação do sistema operacional pela máquina virtual

Teste *t* aplicado para a variável dependente *Garbage Collector* (ms)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	0	+1	+1
Xen x Ubuntu	0	--	0	0	0	0	0	0	+1
VMware x Ubuntu	0	0	--	0	0	0	0	0	+1
KVM x CentOS	0	0	0	--	0	0	0	0	+1
Xen x CentOS	0	0	0	0	--	0	0	0	0
VMware x CentOS	0	0	0	0	0	--	0	0	+1
KVM x Windows	0	0	0	0	0	0	--	0	+1
Xen x Windows	-1	0	0	0	0	0	0	--	0
VMware x Windows	-1	-1	-1	-1	0	-1	-1	0	--
Teste *t* aplicado para a variável dependente CPU (ms)									

Ambiente	KVM x Ubuntu	Xen x Ubuntu	VMware x Ubuntu	KVM x CentOS	Xen x CentOS	VMware x CentOS	KVM x Windows	Xen x Windows	VMware x Windows
KVM x Ubuntu	--	0	0	0	0	0	0	0	0
Xen x Ubuntu	0	--	0	0	0	0	0	0	0
VMware x Ubuntu	0	0	--	0	0	0	-1	0	+1
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	-1	0	0
VMware x CentOS	0	0	0	0	0	--	-1	0	+1
KVM x Windows	0	0	+1	0	+1	+1	--	+1	+1
Xen x Windows	0	0	0	0	0	0	-1	--	+1
VMware x Windows	0	0	-1	0	0	-1	-1	-1	--
Teste *t* aplicado para a variável dependente Memória Física (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	0	0	0
Xen x Ubuntu	0	--	0	0	0	0	0	0	0
VMware x Ubuntu	0	0	--	0	0	0	0	0	0
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	0	0	0
VMware x CentOS	0	0	0	0	0	--	0	0	0
KVM x Windows	0	0	0	0	0	0	--	0	0
Xen x Windows	0	0	0	0	0	0	0	--	0
VMware x Windows	0	0	0	0	0	0	0	0	--
Teste *t* aplicado para a variável dependente Memória Virtual (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	-1	-1	-1	+1	+1	+1
Xen x Ubuntu	0	--	0	-1	-1	-1	+1	+1	+1
VMware x Ubuntu	0	0	--	-1	-1	-1	+1	+1	+1
KVM x CentOS	+1	+1	+1	--	0	0	+1	+1	+1
Xen x CentOS	+1	+1	+1	0	--	0	+1	+1	+1
VMware x CentOS	+1	+1	+1	0	0	--	+1	+1	+1
KVM x Windows	-1	-1	-1	-1	-1	-1	--	0	0
Xen x Windows	-1	-1	-1	-1	-1	-1	0	--	0
VMware x Windows	-1	-1	-1	-1	-1	-1	0	0	--
Teste *t* aplicado para a variável dependente Heap (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	0	0	0
Xen x Ubuntu	0	--	0	0	0	0	0	0	0
VMware x Ubuntu	0	0	--	0	0	0	0	0	0
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	0	0	0
VMware x CentOS	0	0	0	0	0	--	0	0	0
KVM x Windows	0	0	0	0	0	0	--	0	0
Xen x Windows	0	0	0	0	0	0	0	--	0
VMware x Windows	0	0	0	0	0	0	0	0	--

Em relação à utilização da memória virtual (Figura 65d), a máquina virtual não interfere no desempenho de um algoritmo. A maior utilização de memória virtual está com o sistema operacional CentOS e a menor utilização de memória virtual está com o sistema operacional Windows (Tabela 44).

Com 5% de significância as Tabelas 45, 46, 47, 48 e 49 apresentam a Tabela ANOVA para as variáveis dependentes (*Garbage Collector, CPU, Memória Física, Memória Virtual, e Heap*, respectivamente) para o algoritmo *Pi.*

Tabela 45: Modelo para o fatorial 3^4 para o algoritmo *Pi* após a transformação Johnson dos tempos para a variável dependente *Garbage Collector*

	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	0.881427617	-0.09794	0.06214	-1.576	0.121607
Núcleo	0.161186059	-0.02193	0.07611	-0.288	0.774441
Memória	-5.763081447	-0.78426	0.07611	-10.304	9.41E-14
SO	-3.690118357	-0.50216	0.07611	-6.598	3.04E-08
VM	-2.254042559	-0.30674	0.07611	-4.03	1.98E-04
poly(Núcleo, 2)[, 2]	-0.713196427	0.7132	0.5593	1.275	0.208391
poly(Memória, 2)[, 2]	-4.678024483	4.67802	0.5593	8.364	6.24E-11
poly(SO, 2)[, 2]	0.892156491	-0.89216	0.5593	-1.595	1.17E-01
poly(VM, 2)[, 2]	-0.112720784	0.11272	0.5593	0.202	0.841129
Núcleo:Memória	0.426428847	-0.07107	0.09322	-0.762	0.449533
Núcleo:SO	3.308081814	0.55135	0.09322	5.915	3.38E-07
Núcleo:VM	-0.211633323	-0.03527	0.09322	-0.378	0.70681
Núcleo:poly(Memória, 2)[, 2]	-1.463444343	1.79235	0.685	2.617	0.011841
Núcleo:poly(SO, 2)[, 2]	-1.714392325	2.09969	0.685	3.065	0.003565
Núcleo:poly(VM, 2)[, 2]	-0.231832557	0.28394	0.685	0.415	0.680352
Memória:SO	-1.078138712	0.17969	0.09322	1.928	5.98E-02
Memória:VM	1.177499562	0.19625	0.09322	2.105	4.05E-02
Memória:poly(Núcleo, 2)[, 2]	-0.341896943	0.41874	0.685	0.611	0.54389
Memória:poly(SO, 2)[, 2]	1.360173245	1.66587	0.685	2.432	1.88E-02
Memória:poly(VM, 2)[, 2]	0.825540116	1.01108	0.685	1.476	0.146469
SO:VM	-0.522823726	-0.08714	0.09322	-0.935	0.354581
SO:poly(Núcleo, 2)[, 2]	0.84217553	-1.03145	0.685	-1.506	0.138681
SO:poly(Memória, 2)[, 2]	-1.967824451	-2.41008	0.685	-3.518	9.61E-04
SO:poly(VM, 2)[, 2]	-0.273660458	0.33516	0.685	0.489	0.626865
VM:poly(Núcleo, 2)[, 2]	-0.42759061	-0.52369	0.685	-0.765	0.448306
VM:poly(Memória, 2)[, 2]	-0.125051036	0.15316	0.685	0.224	0.82403
VM:poly(SO, 2)[, 2]	-1.016528176	-1.24499	0.685	-1.817	0.075386
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.19242448	-1.73182	5.03371	-0.344	0.732315
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	-1.599689833	-14.39721	5.03371	-2.86	0.006251
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.35432704	3.18894	5.03371	0.634	0.529403
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-1.736257359	-15.62632	5.03371	-3.104	0.003195
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	1.099067891	-9.89161	5.03371	-1.965	0.055207

Tabela 46: Modelo para o fatorial 3^4 para o algoritmo *Pi* após a transformação de Johnson para a variável resposta *CPU*

	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	0.186371341	-0.020708	0.043449	-0.477	0.635811
Núcleo	-2.793281849	0.380118	0.053215	7.143	4.44E-09
Memória	-5.48070096	-0.745829	0.053215	-14.016	< 2e-16
SO	0.008463511	0.001152	0.053215	0.022	9.83E-01
VM	-2.519939914	-0.34292	0.053215	-6.444	5.24E-08
poly(Núcleo, 2)[, 2]	-0.114377257	0.114377	0.391045	0.292	7.71E-01
poly(Memória, 2)[, 2]	-2.846198269	2.846198	0.391045	7.278	2.76E-09
poly(SO, 2)[, 2]	-0.283651141	0.283651	0.391045	0.725	4.72E-01
poly(VM, 2)[, 2]	-0.390110821	0.390111	0.391045	0.998	3.23E-01
Núcleo:Memória	-0.51254358	0.085424	0.065174	1.311	0.196196
Núcleo:SO	-0.404220916	-0.06737	0.065174	-1.034	0.30646
Núcleo:VM	-0.044736446	-0.007456	0.065174	-0.114	0.909396
Núcleo:poly(Memória, 2)[, 2]	0.302571502	-0.370573	0.478931	-0.774	0.442873
Núcleo:poly(SO, 2)[, 2]	0.907394109	-1.111326	0.478931	-2.32	0.024614
Núcleo:poly(VM, 2)[, 2]	0.415018399	-0.508292	0.478931	-1.061	0.293862
Memória:SO	-2.236868786	0.372811	0.065174	5.72	6.69E-07

Memória:VM	1.11363702	0.185606	0.065174	2.848	0.006461
Memória:poly(Núcleo, 2)[, 2]	-0.453362635	0.555254	0.478931	1.159	0.252044
Memória:poly(SO, 2)[, 2]	1.552719107	1.901685	0.478931	3.971	0.000239
Memória:poly(VM, 2)[, 2]	0.589231527	0.721658	0.478931	1.507	0.138413
SO:VM	-2.129653089	-0.354942	0.065174	-5.446	1.74E-06
SO:poly(Núcleo, 2)[, 2]	-0.62501564	0.765485	0.478931	1.598	0.116535
SO:poly(Memória, 2)[, 2]	-0.908395104	-1.112552	0.478931	-2.323	0.024464
SO:poly(VM, 2)[, 2]	0.404056159	-0.494866	0.478931	-1.033	0.306655
VM:poly(Núcleo, 2)[, 2]	0.701413791	0.859053	0.478931	1.794	0.079164
VM:poly(Memória, 2)[, 2]	0.024397633	-0.029881	0.478931	-0.062	0.950511
VM:poly(SO, 2)[, 2]	-1.282095333	-1.57024	0.478931	-3.279	0.001945
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.379996624	-3.41997	3.519408	-0.972	0.336049
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.368589172	3.317303	3.519408	0.943	0.350619
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.362029899	3.258269	3.519408	0.926	0.359181
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-0.508221101	-4.57399	3.519408	-1.3	0.199929
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	0.105051021	-0.945459	3.519408	-0.269	0.789357

Tabela 47: Modelo para o fatorial 3^4 para o algoritmo *Pi* após a transformação de Johnson para a variável resposta *Memória Física*

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	0.501330303	-0.055703	0.038633	-1.442	0.15583
Núcleo	-2.070870293	0.28181	0.047315	5.956	2.93E-07
Memória	5.694539162	0.774929	0.047315	16.378	< 2e-16
SO	-0.64835613	-0.08823	0.047315	-1.865	6.83E-02
VM	0.863488933	0.117506	0.047315	2.483	1.66E-02
poly(Núcleo, 2)[, 2]	0.919083259	-0.919083	0.347696	-2.643	1.11E-02
poly(Memória, 2)[, 2]	3.857573588	-3.857574	0.347696	-11.095	7.58E-15
poly(SO, 2)[, 2]	0.877936422	-0.877936	0.347696	-2.525	1.49E-02
poly(VM, 2)[, 2]	0.033956257	-0.033956	0.347696	-0.098	9.23E-01
Núcleo:Memória	-1.175084347	0.195847	0.057949	3.38	0.00145
Núcleo:SO	-1.019776376	-0.169963	0.057949	-2.933	0.00513
Núcleo:VM	0.16006343	0.026677	0.057949	0.46	0.64734
Núcleo:poly(Memória, 2)[, 2]	1.615763233	-1.978898	0.425838	-4.647	2.65E-05
Núcleo:poly(SO, 2)[, 2]	0.251121936	-0.30756	0.425838	-0.722	0.47365
Núcleo:poly(VM, 2)[, 2]	0.361306846	-0.442509	0.425838	-1.039	0.30394
Memória:SO	2.273303845	-0.378884	0.057949	-6.538	3.76E-08
Memória:VM	-1.110569538	-0.185095	0.057949	-3.194	0.00248
Memória:poly(Núcleo, 2)[, 2]	0.59458858	-0.728219	0.425838	-1.71	0.09371
Memória:poly(SO, 2)[, 2]	-0.213617778	-0.261627	0.425838	-0.614	0.54186
Memória:poly(VM, 2)[, 2]	-0.619223117	-0.75839	0.425838	-1.781	0.08125
SO:VM	-0.034778703	-0.005796	0.057949	-0.1	0.92074
SO:poly(Núcleo, 2)[, 2]	-0.126619827	0.155077	0.425838	0.364	0.71733
SO:poly(Memória, 2)[, 2]	1.789100355	2.191191	0.425838	5.146	4.90E-06
SO:poly(VM, 2)[, 2]	0.222645088	-0.272683	0.425838	-0.64	0.52499
VM:poly(Núcleo, 2)[, 2]	-0.018442924	-0.022588	0.425838	-0.053	0.95792
VM:poly(Memória, 2)[, 2]	-0.526064049	0.644294	0.425838	1.513	0.13684
VM:poly(SO, 2)[, 2]	0.112397788	0.137659	0.425838	0.323	0.7479
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.43754855	3.937937	3.12926	1.258	0.21433
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.482416666	4.34175	3.12926	1.387	0.17171
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	-0.125754961	-1.131795	3.12926	-0.362	0.71918
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.341798678	3.076188	3.12926	0.983	0.33052
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.911632334	8.204691	3.12926	2.622	0.01168

Tabela 48: Modelo para o fatorial 3^4 para o algoritmo *Pi* após a transformação de Johnson para a variável resposta *Memória Virtual*

	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	-6.33E-01	0.070284	0.020838	3.373	0.001478
Núcleo	-7.48E-01	0.101844	0.025521	3.991	0.000225
Memória	2.28E+00	0.310734	0.025521	12.176	2.75E-16
SO	-5.43E+00	-0.739492	0.025521	-28.976	< 2e-16
VM	1.87E-01	0.025482	0.025521	0.998	3.23E-01
poly(Núcleo, 2)[, 2]	1.10E+00	-1.100764	0.187541	-5.869	3.97E-07
poly(Memória, 2)[, 2]	1.08E+00	-1.08001	0.187541	-5.759	5.84E-07
poly(SO, 2)[, 2]	7.06E+00	-7.060092	0.187541	-37.646	< 2e-16
poly(VM, 2)[, 2]	3.11E-01	-0.310976	0.187541	-1.658	1.04E-01
Núcleo:Memória	-4.53E-01	0.075549	0.031257	2.417	1.95E-02
Núcleo:SO	2.15E-01	0.035899	0.031257	1.149	2.56E-01
Núcleo:VM	2.57E-02	0.004276	0.031257	0.137	8.92E-01
Núcleo:poly(Memória, 2)[, 2]	3.32E-01	-0.406632	0.22969	-1.77	8.30E-02
Núcleo:poly(SO, 2)[, 2]	3.99E-01	-0.488974	0.22969	-2.129	0.038426
Núcleo:poly(VM, 2)[, 2]	2.93E-02	-0.035859	0.22969	-0.156	0.876595
Memória:SO	-1.61E+00	0.268313	0.031257	8.584	2.93E-11
Memória:VM	-2.43E-01	-0.040512	0.031257	-1.296	0.201135
Memória:poly(Núcleo, 2)[, 2]	8.72E-01	-1.0675	0.22969	-4.648	2.65E-05
Memória:poly(SO, 2)[, 2]	-1.47E-01	-0.179937	0.22969	-0.783	0.437246
Memória:poly(VM, 2)[, 2]	-1.47E-01	-0.179827	0.22969	-0.783	0.437524
SO:VM	2.69E-01	0.044785	0.031257	1.433	0.158395
SO:poly(Núcleo, 2)[, 2]	3.80E-02	-0.04648	0.22969	-0.202	0.84049
SO:poly(Memória, 2)[, 2]	-8.38E-01	-1.02609	0.22969	-4.467	4.81E-05
SO:poly(VM, 2)[, 2]	9.97E-02	-0.122158	0.22969	-0.532	0.597291
VM:poly(Núcleo, 2)[, 2]	-1.80E-02	-0.02206	0.22969	-0.096	0.923886
VM:poly(Memória, 2)[, 2]	-8.00E-02	0.097986	0.22969	0.427	0.671578
VM:poly(SO, 2)[, 2]	1.84E-01	0.22549	0.22969	0.982	0.331164
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-2.80E-01	2.516937	1.687868	1.491	0.142454
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	1.28E+00	11.561958	1.687868	6.85	1.25E-08
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	2.56E-01	2.303993	1.687868	1.365	0.178607
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-9.81E-02	-0.88294	1.687868	-0.523	0.603304
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	1.09E-01	-0.984133	1.687868	-0.583	0.56258

Tabela 49: Modelo para o fatorial 3^4 para o algoritmo *Pi* após a transformação de Johnson para a variável resposta *Heap*

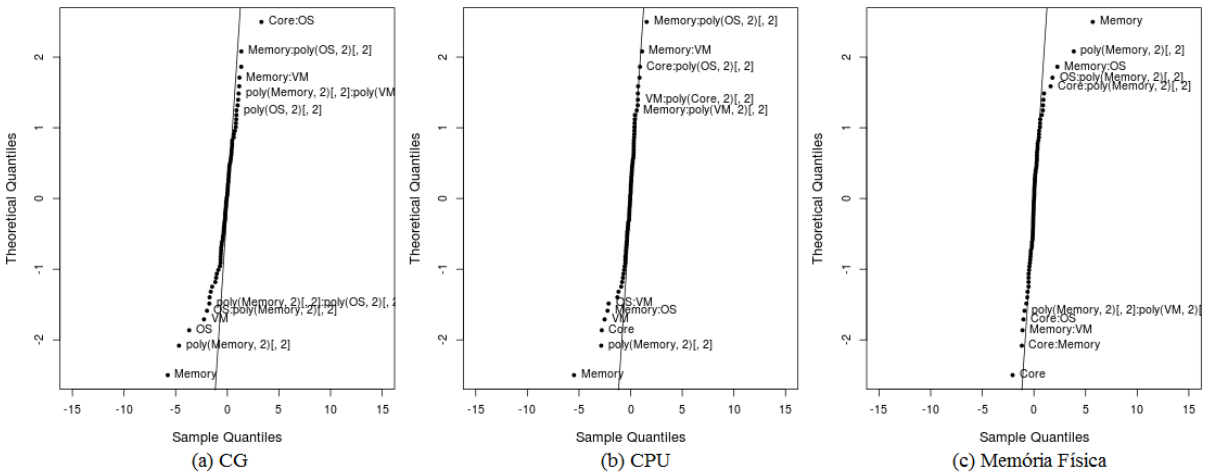
	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	-1.46E+00	1.62E-01	3.35E-02	4.836	1.40E-05
Núcleo	-1.15E+00	1.57E-01	4.10E-02	3.826	0.000376
Memória	7.45E+00	1.01E+00	4.10E-02	24.749	< 2e-16
SO	1.17E+00	1.59E-01	4.10E-02	3.871	0.000327
VM	-1.47E+00	-2.00E-01	4.10E-02	-4.892	1.16E-05
poly(Núcleo, 2)[, 2]	9.10E-01	-9.10E-01	3.01E-01	-3.022	0.004021
poly(Memória, 2)[, 2]	3.11E+00	-3.11E+00	3.01E-01	-10.343	8.30E-14
poly(SO, 2)[, 2]	-1.40E-01	1.40E-01	3.01E-01	0.466	0.643004
poly(VM, 2)[, 2]	6.53E-01	-6.53E-01	3.01E-01	-2.169	0.035057
Núcleo:Memória	-7.90E-01	1.32E-01	5.02E-02	2.623	0.011661
Núcleo:SO	8.60E-02	1.43E-02	5.02E-02	0.286	0.77644
Núcleo:VM	1.80E-02	3.01E-03	5.02E-02	0.06	0.952489
Núcleo:poly(Memória, 2)[, 2]	2.62E-01	-3.20E-01	3.69E-01	-0.869	0.389406
Núcleo:poly(SO, 2)[, 2]	6.79E-02	-8.32E-02	3.69E-01	-0.226	0.822531
Núcleo:poly(VM, 2)[, 2]	4.58E-02	-5.60E-02	3.69E-01	-0.152	0.879866
Memória:SO	1.12E+00	-1.86E-01	5.02E-02	-3.708	0.000541
Memória:VM	1.48E+00	2.47E-01	5.02E-02	4.925	1.04E-05

Memória:poly(Núcleo, 2)[, 2]	7.51E-01	-9.20E-01	3.69E-01	-2.496	0.016067
Memória:poly(SO, 2)[, 2]	-1.37E+00	-1.68E+00	3.69E-01	-4.565	3.48E-05
Memória:poly(VM, 2)[, 2]	1.12E+00	1.38E+00	3.69E-01	3.733	0.000501
SO:VM	9.22E-01	1.54E-01	5.02E-02	3.061	0.003604
SO:poly(Núcleo, 2)[, 2]	-2.17E-02	2.66E-02	3.69E-01	0.072	0.942816
SO:poly(Memória, 2)[, 2]	8.81E-01	1.08E+00	3.69E-01	2.926	0.005228
SO:poly(VM, 2)[, 2]	-3.92E-01	4.80E-01	3.69E-01	1.302	0.198987
VM:poly(Núcleo, 2)[, 2]	1.99E-01	2.44E-01	3.69E-01	0.66	0.512121
VM:poly(Memória, 2)[, 2]	1.10E+00	-1.35E+00	3.69E-01	-3.648	0.00065
VM:poly(SO, 2)[, 2]	7.58E-01	9.28E-01	3.69E-01	2.516	0.015277
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	1.47E-02	-1.32E-01	2.71E+00	-0.049	0.961241
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	6.50E-01	5.85E+00	2.71E+00	2.158	0.03597
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	-2.49E-01	-2.24E+00	2.71E+00	-0.828	0.411741
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-2.35E-03	-2.12E-02	2.71E+00	-0.008	0.993795
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	4.49E-01	-4.04E+00	2.71E+00	-1.492	0.142282

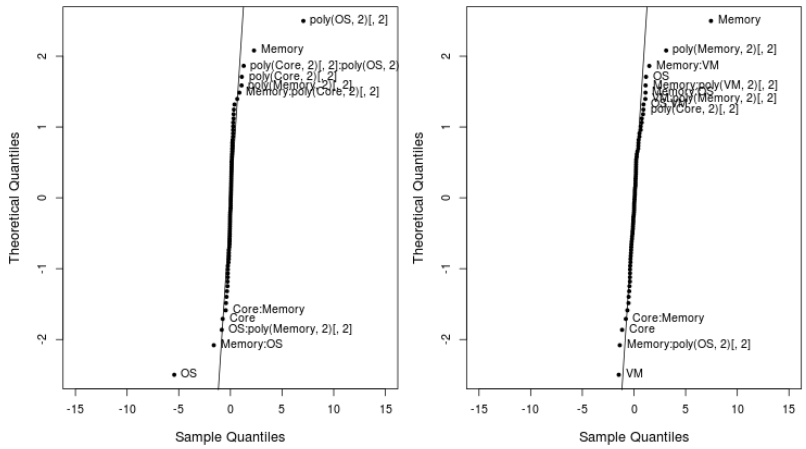
A análise do fatorial 3^4 se iniciou com a projeção do gráfico de probabilidade normal dos efeitos estimados para a identificação dos efeitos significativos (Figura 66) para cada variável resposta. Pode-se verificar que em todos os gráficos de probabilidade normal os fatores SO e/ou VM (efeito principal e/ou efeito de segunda ordem) foram estatisticamente significativos.

A Figura 67 mostra o gráfico de Pareto que apresenta os valores absolutos estimados de cada efeito em cada variável resposta para o algoritmo *Pi*. Pode-se verificar que na maioria dos gráficos (Figuras 67a, 67b e 76c), a Memória é o fator com maior efeito significativo. Os gráficos de Pareto para as variáveis respostas *Memória Virtual* (Figura 67d) *e Heap* (Figura 67e) mostram que o fator VM não foi estatisticamente significativo.

Figura 66: Gráfico de probabilidade normal dos efeitos estimados do fatorial 3^4 dos tempos para o algoritmo *Pi* para as variáveis respostas, após a transformação de Johnson: *Garbage Collector* (ms), CPU (ms), memória física (bytes), memória virtual (bytes) e heap (bytes), respectivamente.



(a) CG (b) CPU (c) Memória Física



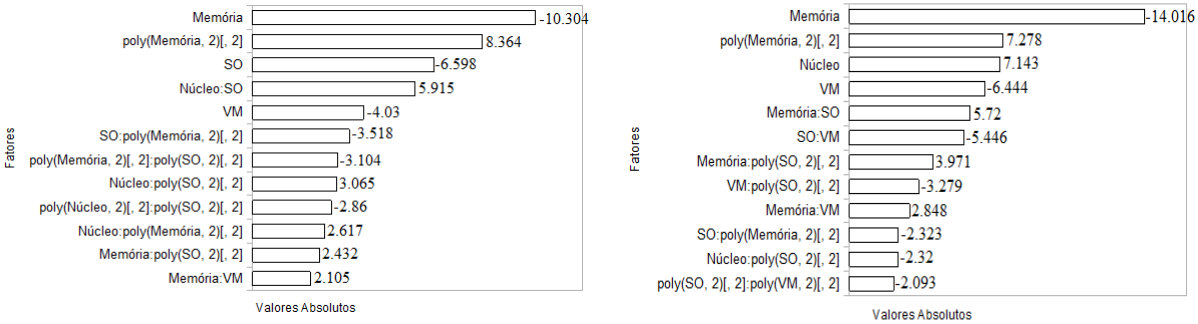
(d) Memória Virtual (e) Heap

Os coeficientes de determinação R^2 com 48 graus de liberdade, apresentado na Tabela 50.

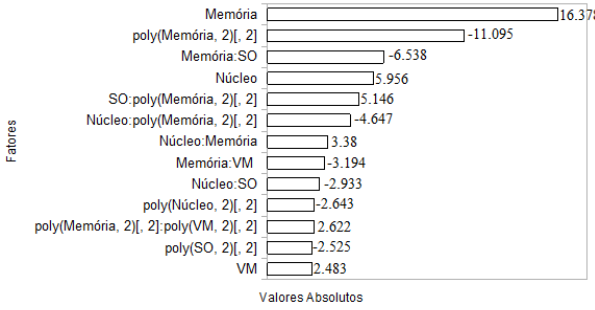
Tabela 50: Coeficientes de determinação para o algoritmo *Pi* após a transformação de Johnson para cada variável resposta

Variáveis respostas	**R^2**
Garbage Collector	0.8789
CPU	0.9083
Memória Física	0.9250
Memória Virtual	0.9824
Heap	0.9505

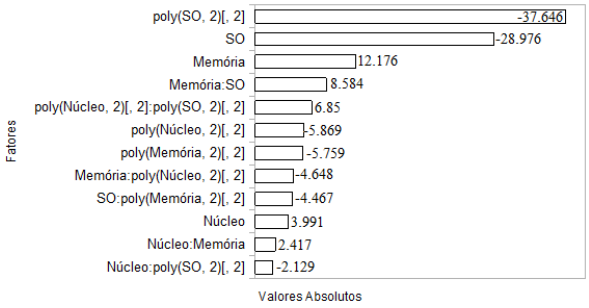
Figura 67: Gráfico de Pareto com os efeitos estimados em valor percentual absoluto para o tempo de execução para o algoritmo *Pi*: *Garbage Collector* (ms), CPU (ms), memória física (bytes), memória virtual (bytes), heap (bytes) e tempo (seg), respectivamente, após a transformação de Johnson.



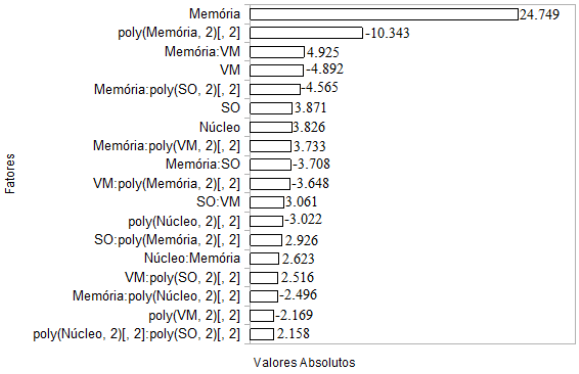
(a) Garbage Collector (b) CPU



(c) Memória Física



(d) Memória Virtual



(e) Heap

Testes formais foram realizados (Tabela 51) e confirmaram que para as variáveis respostas *Garbage Collector*, *CPU*, *Memória Física*, *Memória Virtual e Heap*, os resíduos estavam distribuídos de forma homogênea (teste de Breusch-Pagan, função *bpTest*). Já para o teste de distribuição normal (teste de Shapiro-Wilk, função *shapiro.test*), somente as variáveis respostas *Memória Física e Memória Virtual* os resíduos não se aproximam de uma distribuição normal. Portanto, de acordo com Montgomery (2009), desvios de normalidade moderados podem ocorrer contanto que pontos discrepantes sejam investigados. Por fim, para o teste o teste de auto correlação negativa e positiva (teste de DurbinWatson, função durbinWatsonTest), as variáveis dependentes *CPU*, *Memória Física e Heap* não pode-se confluir a dependência dos resíduos, as variáveis dependentes *Garbage Collector e Memória Virtual* não existe auto correlação dos resíduos.

Como apresentado na Tabela 51, a hipótese de distribuição normal foi rejeitada pelo teste de Shapiro-Wilk para as variáveis respostas (*Memória Física* e *Memória Virtual*), porém na Tabela 52, os testes de Kolmogorov-Smirnov (Lilliefors, 1967) e Andereson-Darling (Razali, et al., 2011) confirmam que os resíduos aproximam de uma distribuição normal, aceitando a hipótese de Normalidade.

Tabela 51: Testes formais de Breusch-Pagan, Shapiro-Wilk, DurbinWatson para o algoritmo Pi após a transformação de Johnson para cada variável resposta

Variável dependente	Breusch-Pagan valor-p	Shapiro-Wilk valor-p	DurbinWatson Dw
GARBAGE COLLECTOR	0.0697 **Aceita H_0**	0.2052 **Aceita H_0**	2.077 **Aceita H_0**
CPU	0.07393 **Aceita H_0**	0.0984 **Aceita H_0**	1.591 **Teste inconclusivo**
Memória física	0.261 **Aceita H_0**	7.906e-06 **Rejeita H_0**	1.455 **Teste inconclusivo**
Memória Virtual	0.1614 **Aceita H_0**	4.552e-06 **Rejeita H_0**	2.721 **Aceita H_0**
Heap	0.1074 **Aceita H_0**	0.0553 **Aceita H_0**	1.666 **Teste inconclusivo**

Tabela 52: Testes formais de normalidade para o algoritmo *Pi* após a transformação de Johnson (H_0: variância dos erros são iguais)

Variável dependente	Shapiro-Wilk valor-p	Kolmogorov-Smirnov valor-p	Anderson-Darling valor-p
Garbage Collector	0.2052 **Aceita H_0**	0.5338 **Aceita H_0**	0.2337 **Aceita H_0**
CPU	0.0984 **Aceita H_0**	0.9436 **Aceita H_0**	0.3715 **Aceita H_0**
Memória física	7.906e-06 **Rejeita H_0**	0.3008 **Aceita H_0**	0.0826 **Aceita H_0**
Memória Virtual	4.552e-06 **Rejeita H_0**	0.2476 **Aceita H_0**	0.0609 **Aceita H_0**
Heap	0.0553 **Aceita H_0**	0.819 **Aceita H_0**	0.1391 **Aceita H_0**

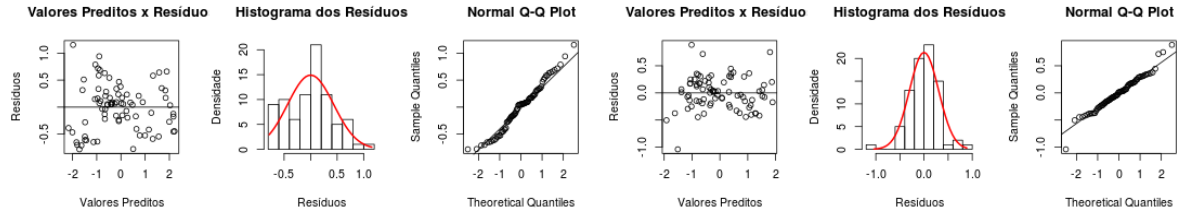
A Figura 68 apresenta a verificação do modelo através da análise dos resíduos para cada variável resposta. O gráfico da esquerda para direita representa a relação entre os resíduos pelos valores preditos e não apresenta estrutura ou padrão. O histograma dos resíduos que apresenta a distribuição da frequência dos resíduos apresentou aproximadamente simétrico em forma de sino. O gráfico à direita representa a probabilidade normal dos resíduos apresentando pontos próximos da reta.

D.2 Algoritmo *Pi*

A Figura 69 apresenta da esquerda para a direita, informações exploratórias sobre as variáveis respostas (*Garbage Collector, CPU, memória física, memória virtual e heap)* respectivamente, encontradas durante a execução do algoritmo *WordCount* nos ambientes

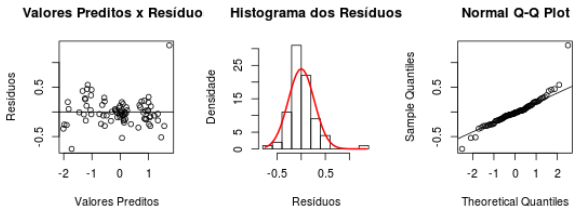
virtuais analisados. O primeiro gráfico de cada variável resposta apresenta o diagrama de caixa com as respectivas respostas.

Figura 68: Verificação dos resíduos: gráfico resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem de dois do fatorial 3^4 do algoritmo *Pi*, após a transformação de Johnson para cada variável resposta

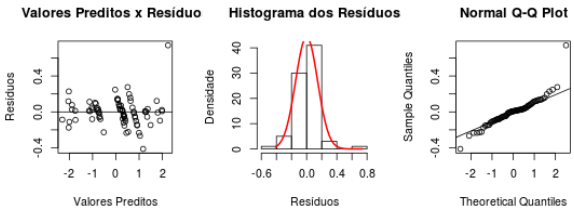


(a) *Garbage Collector*

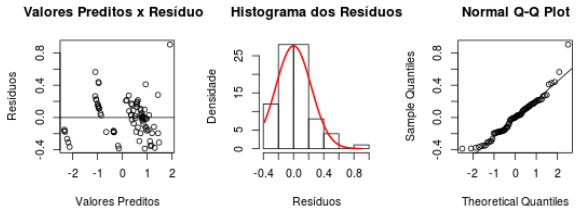
(b) CPU



(c) Memória Física



(d) Memória Virtual



(e) Heap

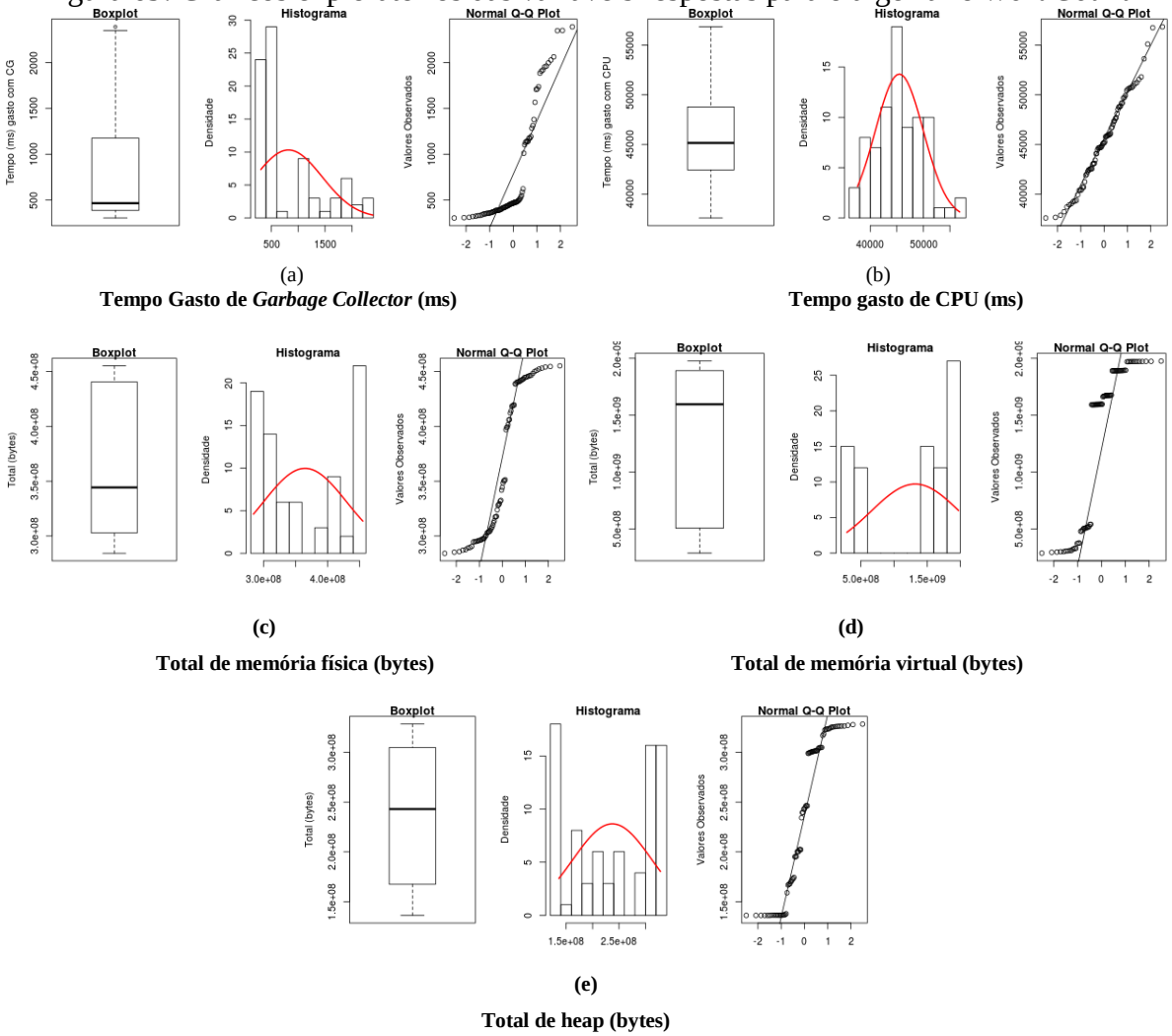
Os gráficos das normais da Figura 69 mostram que os dados coletados não se aproximam de uma distribuição normal o que caracteriza a necessidade de se aplicar uma transformação nos dados. Para cada conjunto de dados de cada variável resposta foi aplicada a transformação de Johnson (JOHNSON, 2002).

Para cada variável resposta o resultado da transformação de Johnson foi:

- *Garbage Collector*: família = SU, γ = -1.441924, λ = 25.80153, ε = 338.4527 e η = 0.5616037;
- CPU: família = SB, γ = 0.5008697, λ = 29916.74, ε = 33110.74 e η = 1.413889;
- Memória física: família = SB, γ = -0.04282915, λ = 171048537, ε = 284195594 e η = 0.4416898;
- Memória virtual: família = SB, γ = -0.07596994, λ = 1686402543, ε = 289314265 e η = 0.2927233;

- Heap: família = SB, γ = -0.0674148, λ = 193190596, ε = 135197440 e η = 0.379073;

Figura 69: Gráficos exploratórios das variáveis respostas para o algoritmo *WordCount*



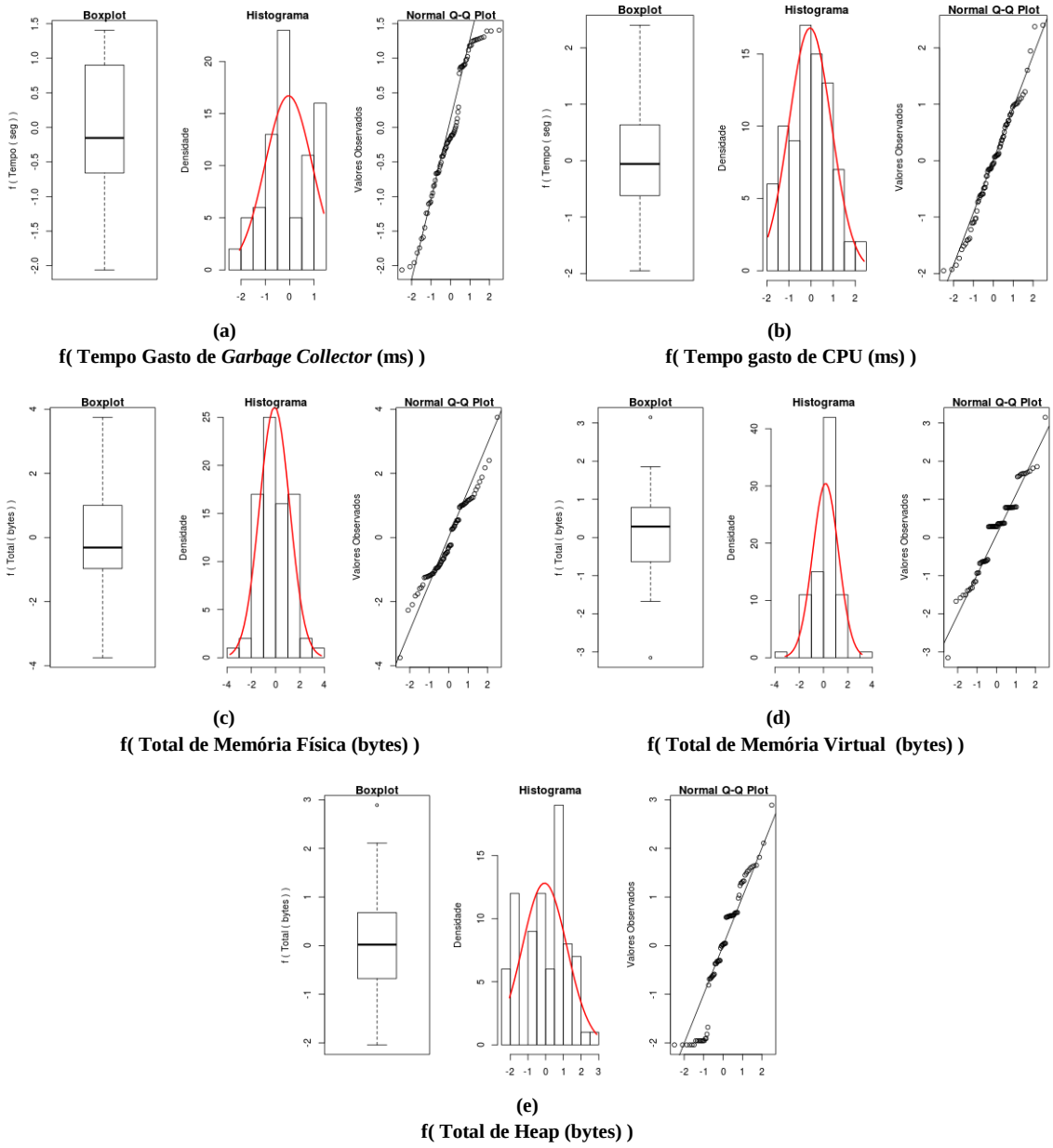
A Figura 70 mostra os mesmos gráficos, porém após a transformação de Johnson. Os diagramas de caixas e os histogramas apresentam ser mais simétricos em relação aos apresentados na Figura 69. E por fim, pode-se observar que todos os gráficos os dados aproximam de uma distribuição normal.

A Figura 71 mostra as mesmas informações, porém agrupadas através da combinação do fator sistema operacional pelo fator máquina virtual. As análises baseadas em *boxplot* são limitadas, podendo tirar conclusões somente quando os agrupamentos são óbvios. A Tabela 53 mostra o resultado da aplicação do teste *t*. Pode-se então verificar que não existem evidências estatísticas que a *Garbage Collector* (Figura 71a), memória física (Figura 71c) e o heap (Figura 71e) que confirmem a diferença de desempenho entre os ambientes virtuais (Tabelas 53a, 53c e 53e). Analisando a variável resposta CPU (Tabela 53b), pode-se concluir que a pior

combinação é o sistema operacional Windows (nível +) com a máquina virtual KVM (nível -), ou seja o tempo (ms) gasto com o *Garbage Collector* é maior (Figura 71b).

Em relação à utilização da memória virtual (Figura 71d) a máquina virtual não interfere no desempenho de um algoritmo. A maior utilização de memória virtual está com o sistema operacional CentOS e a menor utilização de memória virtual está com o sistema operacional Windows (Tabela 53d).

Figura 70: Gráficos exploratórios das variáveis respostas para o algoritmo *WordCount* após transformação de Johnson



(a)
f(Tempo Gasto de *Garbage Collector* (ms))

(b)
f(Tempo gasto de CPU (ms))

(c)
f(Total de Memória Física (bytes))

(d)
f(Total de Memória Virtual (bytes))

(e)
f(Total de Heap (bytes))

Figura 71: Diagrama de caixa (*boxplot*) das variáveis de respostas para o algoritmo *WordCount*, após a transformação de Johnson: variáveis respostas agrupadas pela combinação do sistema operacional pela máquina virtual

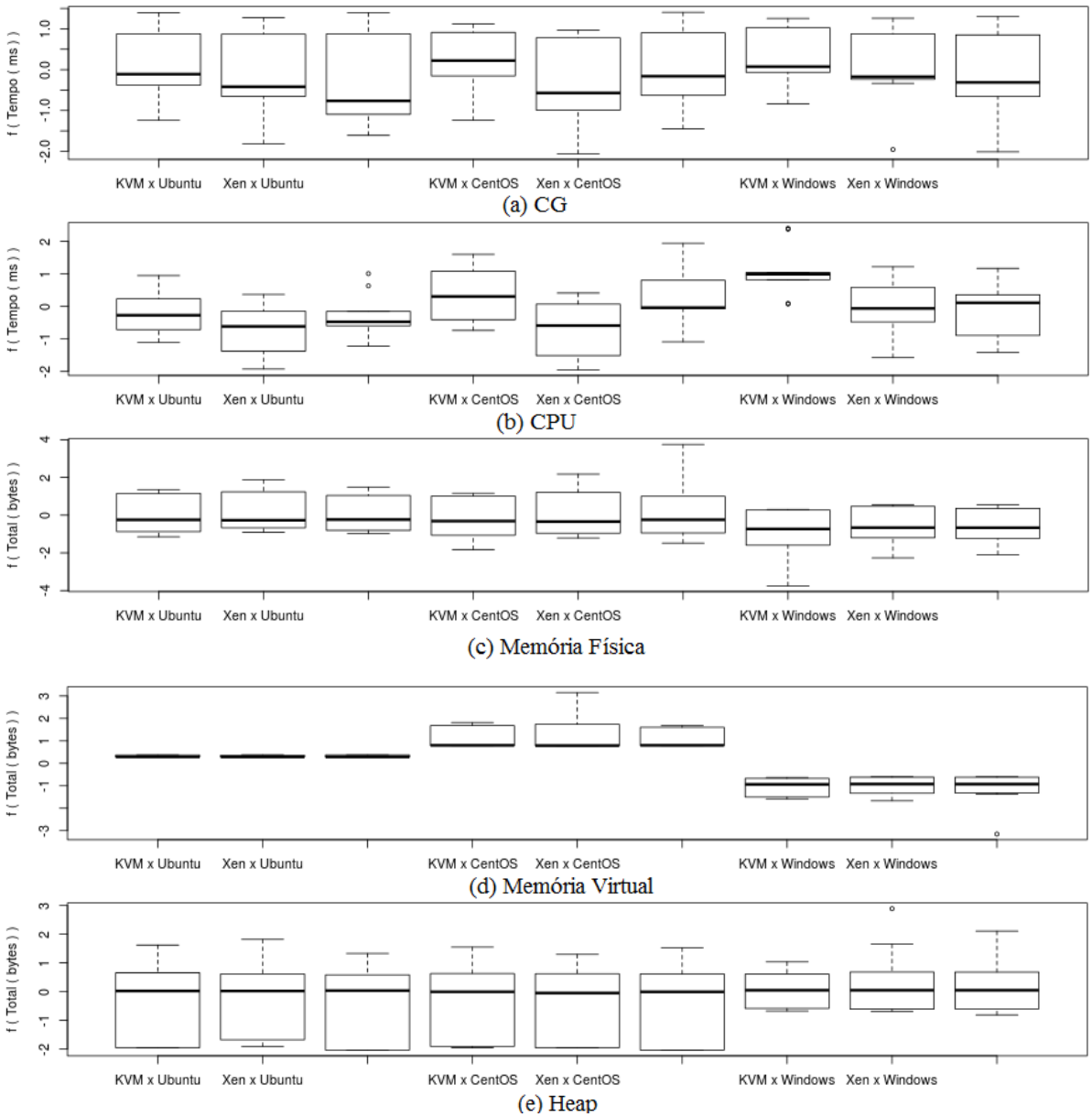


Tabela 53: Comparação entre os fatores após a transformação de Johnson: variáveis respostas agrupadas entre sistema operacional pela máquina virtual para o algoritmo *WordCount*

Teste *t* aplicado para a variável dependente *Garbage Collector* (ms)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	0	0	0
Xen x Ubuntu	0	--	0	0	0	0	0	0	0
VMware x Ubuntu	0	0	--	0	0	0	0	0	0
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	0	0	0
VMware x CentOS	0	0	0	0	0	--	0	0	0
KVM x Windows	0	0	0	0	0	0	--	0	0
Xen x Windows	0	0	0	0	0	0	0	--	0
VMware x Windows	0	0	0	0	0	0	0	0	--

Teste *t* aplicado para a variável dependente CPU (ms)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--		0	0	0	0	-1	0	0
Xen x Ubuntu	0	--	0	-1	0	-1	-1	0	0
VMware x Ubuntu	0	0	--	0	0	0	-1	0	0
KVM x CentOS	0	+1	0	--	+1	0	0	0	0
Xen x CentOS	0	0	0	-1	--	0	-1	0	0
VMware x CentOS	0	+1	0	0	0	--	0	0	0
KVM x Windows	+1	+1	+1	0	+1	0	--	+1	+1
Xen x Windows	0	0	0	0	0	0	-1	--	0
VMware x Windows	0	0	0	0	0	0	-1	0	--
Teste *t* aplicado para a variável dependente Memória Física (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	0	0	0
Xen x Ubuntu	0	--	0	0	0	0	0	0	0
VMware x Ubuntu	0	0	--	0	0	0	0	0	0
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	0	0	0
VMware x CentOS	0	0	0	0	0	--	0	0	0
KVM x Windows	0	0	0	0	0	0	--	0	0
Xen x Windows	0	0	0	0	0	0	0	--	0
VMware x Windows	0	0	0	0	0	0	0	0	--
Teste *t* aplicado para a variável dependente Memória Virtual (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	-1	-1	-1	+1	+1	+1
Xen x Ubuntu	0	--	0	-1	-1	-1	+1	+1	+1
VMware x Ubuntu	0	0	--	-1	-1	-1	+1	+1	+1
KVM x CentOS	+1	+1	+1	--	0	0	+1	+1	+1
Xen x CentOS	+1	+1	+1	0	--	0	+1	+1	+1
VMware x CentOS	+1	+1	+1	0	0	--	+1	+1	+1
KVM x Windows	-1	-1	-1	-1	-1	-1	--	0	0
Xen x Windows	-1	-1	-1	-1	-1	-1	0	--	0
VMware x Windows	-1	-1	-1	-1	-1	-1	0	0	--
Teste *t* aplicado para a variável dependente Heap (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	0	0	0
Xen x Ubuntu	0	--	0	0	0	0	0	0	0
VMware x Ubuntu	0	0	--	0	0	0	0	0	0
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	0	0	0
VMware x CentOS	0	0	0	0	0	--	0	0	0
KVM x Windows	0	0	0	0	0	0	--	0	0
Xen x Windows	0	0	0	0	0	0	0	--	0
VMware x Windows	0	0	0	0	0	0	0	0	--

Com 5% de significância as Tabelas 54, 55, 56, 57 e 58 apresentam a Tabela ANOVA para as variáveis dependentes (*Garbage Collector, CPU, Memória Física, Memoria Virtual, Heap e Tempo,* respectivamente) para o algoritmo *WordCount.*

A análise do fatorial 3^4 se iniciou com a projeção do gráfico de probabilidade normal dos efeitos estimados para a identificação dos efeitos significativos (Figura 72) para cada variável resposta. Pode-se verificar que todos os gráficos de probabilidade normal os fatores SO ou VM (efeito principal ou interação) foram estatisticamente significativos.

Tabela 54: Modelo para o fatorial 3^4 para o algoritmo *WordCount* após a transformação Johnson dos tempos para a variável dependente *Garbage Collector*

	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	0.3908	-0.0434	0.0238	-1.8270	0.0739
Núcleo	-1.9295	0.2626	0.0291	9.0220	0.0000
Memória	-7.0445	-0.9586	0.0291	-32.9400	< 2e-16
SO	0.8124	0.1106	0.0291	3.7990	0.0004
VM	-1.2569	-0.1710	0.0291	-5.8770	0.0000
poly(Núcleo, 2)[, 2]	-0.0547	0.0547	0.2139	0.2560	0.7992
poly(Memória, 2)[, 2]	-2.4373	2.4373	0.2139	11.3970	0.0000
poly(SO, 2)[, 2]	0.0054	-0.0054	0.2139	-0.0250	0.9801
poly(VM, 2)[, 2]	-0.8921	0.8921	0.2139	4.1710	0.0001
Núcleo:Memória	-1.3420	0.2237	0.0356	6.2750	0.0000
Núcleo:SO	0.2369	0.0395	0.0356	1.1080	0.2735
Núcleo:VM	-0.1015	-0.0169	0.0356	-0.4750	0.6372
Núcleo:poly(Memória, 2)[, 2]	-1.5614	1.9123	0.2619	7.3010	0.0000
Núcleo:poly(SO, 2)[, 2]	0.1102	-0.1349	0.2619	-0.5150	0.6088
Núcleo:poly(VM, 2)[, 2]	0.1963	-0.2404	0.2619	-0.9180	0.3632
Memória:SO	-0.4360	0.0727	0.0356	2.0390	0.0470
Memória:VM	-0.9655	-0.1609	0.0356	-4.5150	0.0000
Memória:poly(Núcleo, 2)[, 2]	0.5285	-0.6473	0.2619	-2.4710	0.0171
Memória:poly(SO, 2)[, 2]	-0.1633	-0.2000	0.2619	-0.7640	0.4488
Memória:poly(VM, 2)[, 2]	0.2422	0.2966	0.2619	1.1330	0.2630
SO:VM	-0.0798	-0.0133	0.0356	-0.3730	0.7108
SO:poly(Núcleo, 2)[, 2]	0.7191	-0.8808	0.2619	-3.3630	0.0015
SO:poly(Memória, 2)[, 2]	-0.5330	-0.6528	0.2619	-2.4920	0.0162
SO:poly(VM, 2)[, 2]	0.1509	-0.1848	0.2619	-0.7060	0.4839
VM:poly(Núcleo, 2)[, 2]	0.1750	0.2143	0.2619	0.8180	0.4173
VM:poly(Memória, 2)[, 2]	-0.3448	0.4223	0.2619	1.6120	0.1135
VM:poly(SO, 2)[, 2]	-0.4067	-0.4982	0.2619	-1.9020	0.0632
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	1.5679	-14.1108	1.9247	-7.3310	0.0000
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	-0.0463	-0.4169	1.9247	-0.2170	0.8294
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.0463	0.4169	1.9247	0.2170	0.8294
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.4249	3.8242	1.9247	1.9870	0.0527
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	0.1797	-1.6172	1.9247	-0.8400	0.4049
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.7601	-6.8412	1.9247	-3.5540	0.0009

Tabela 55: Modelo para o fatorial 3^4 para o algoritmo *WordCount* após a transformação de Johnson para a variável resposta *CPU*

	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	0.3920	-0.0436	0.0224	-1.9440	0.0578
Núcleo	-6.0801	0.8274	0.0274	30.1510	< 2e-16
Memória	-2.7055	-0.3682	0.0274	-13.4160	< 2e-16
SO	2.5113	0.3418	0.0274	12.4530	< 2e-16
VM	-1.6958	-0.2308	0.0274	-8.4090	0.0000
poly(Núcleo, 2)[, 2]	1.3293	-1.3293	0.2017	-6.5920	0.0000
poly(Memória, 2)[, 2]	-1.4514	1.4514	0.2017	7.1970	0.0000
poly(SO, 2)[, 2]	0.0733	-0.0733	0.2017	-0.3640	0.7178
poly(VM, 2)[, 2]	-2.8322	2.8322	0.2017	14.0450	< 2e-16
Núcleo:Memória	0.2866	-0.0478	0.0336	-1.4210	0.1617
Núcleo:SO	0.3163	0.0527	0.0336	1.5680	0.1233
Núcleo:VM	0.3031	0.0505	0.0336	1.5030	0.1394
Núcleo:poly(Memória, 2)[, 2]	-0.3255	0.3987	0.2470	1.6140	0.1130
Núcleo:poly(SO, 2)[, 2]	0.6501	-0.7962	0.2470	-3.2240	0.0023
Núcleo:poly(VM, 2)[, 2]	0.6001	-0.7350	0.2470	-2.9760	0.0046

Memória:SO	0.2007	-0.0335	0.0336	-0.9950	0.3246
Memória:VM	-0.2136	-0.0356	0.0336	-1.0590	0.2949
Memória:poly(Núcleo, 2)[, 2]	-0.0409	0.0501	0.2470	0.2030	0.8400
Memória:poly(SO, 2)[, 2]	-0.2501	-0.3063	0.2470	-1.2400	0.2210
Memória:poly(VM, 2)[, 2]	-0.4818	-0.5901	0.2470	-2.3890	0.0209
SO:VM	-1.6354	-0.2726	0.0336	-8.1100	0.0000
SO:poly(Núcleo, 2)[, 2]	0.3730	-0.4569	0.2470	-1.8500	0.0705
SO:poly(Memória, 2)[, 2]	0.2836	0.3473	0.2470	1.4060	0.1661
SO:poly(VM, 2)[, 2]	-0.1797	0.2201	0.2470	0.8910	0.3772
VM:poly(Núcleo, 2)[, 2]	-0.0654	-0.0801	0.2470	-0.3240	0.7470
VM:poly(Memória, 2)[, 2]	-0.1620	0.1984	0.2470	0.8030	0.4257
VM:poly(SO, 2)[, 2]	-0.9272	-1.1356	0.2470	-4.5980	0.0000
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.0849	-0.7640	1.8149	-0.4210	0.6757
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	-0.4961	-4.4650	1.8149	-2.4600	0.0175
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.0516	0.4642	1.8149	0.2560	0.7992
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.2154	1.9387	1.8149	1.0680	0.2908
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.2725	2.4529	1.8149	1.3520	0.1829
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.9033	-8.1297	1.8149	-4.4790	0.0000

Tabela 56: Modelo para o fatorial 3^4 para o algoritmo *WordCount* após a transformação de Johnson para a variável resposta *Memória Física*

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	0.7853	-0.0873	0.0419	-2.0820	0.0427
Núcleo	-4.8957	0.6662	0.0513	12.9790	< 2e-16
Memória	6.3685	0.8666	0.0513	16.8840	< 2e-16
SO	-3.0832	-0.4196	0.0513	-8.1740	0.0000
VM	1.1582	0.1576	0.0513	3.0710	0.0035
poly(Núcleo, 2)[, 2]	2.7339	-2.7339	0.3772	-7.2480	0.0000
poly(Memória, 2)[, 2]	4.3857	-4.3857	0.3772	-11.6270	0.0000
poly(SO, 2)[, 2]	1.7640	-1.7640	0.3772	-4.6770	0.0000
poly(VM, 2)[, 2]	0.5741	-0.5741	0.3772	-1.5220	0.1346
Núcleo:Memória	-1.0330	0.1722	0.0629	2.7390	0.0086
Núcleo:SO	0.4922	0.0820	0.0629	1.3050	0.1982
Núcleo:VM	0.1306	0.0218	0.0629	0.3460	0.7307
Núcleo:poly(Memória, 2)[, 2]	2.0119	-2.4641	0.4620	-5.3340	0.0000
Núcleo:poly(SO, 2)[, 2]	0.4413	-0.5405	0.4620	-1.1700	0.2478
Núcleo:poly(VM, 2)[, 2]	-0.1481	0.1814	0.4620	0.3930	0.6963
Memória:SO	-0.4793	0.0799	0.0629	1.2710	0.2100
Memória:VM	-0.5983	-0.0997	0.0629	-1.5860	0.1193
Memória:poly(Núcleo, 2)[, 2]	0.4125	-0.5052	0.4620	-1.0940	0.2796
Memória:poly(SO, 2)[, 2]	-0.1939	-0.2375	0.4620	-0.5140	0.6095
Memória:poly(VM, 2)[, 2]	0.3079	0.3771	0.4620	0.8160	0.4184
SO:VM	0.5687	0.0948	0.0629	1.5080	0.1382
SO:poly(Núcleo, 2)[, 2]	0.0285	-0.0350	0.4620	-0.0760	0.9400
SO:poly(Memória, 2)[, 2]	0.3537	0.4332	0.4620	0.9380	0.3531
SO:poly(VM, 2)[, 2]	0.1006	-0.1233	0.4620	-0.2670	0.7908
VM:poly(Núcleo, 2)[, 2]	0.2447	0.2997	0.4620	0.6490	0.5197
VM:poly(Memória, 2)[, 2]	0.6628	-0.8118	0.4620	-1.7570	0.0853
VM:poly(SO, 2)[, 2]	-0.4995	-0.6117	0.4620	-1.3240	0.1917
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.9361	8.4249	3.3948	2.4820	0.0166
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.0255	0.2294	3.3948	0.0680	0.9464
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.2803	2.5226	3.3948	0.7430	0.4611
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	1.3047	11.7424	3.3948	3.4590	0.0012
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	0.0551	-0.4957	3.3948	-0.1460	0.8845
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.3018	-2.7162	3.3948	-0.8000	0.4276

Tabela 57: Modelo para o fatorial 3^4 para o algoritmo *WordCount* após a transformação de Johnson para a variável resposta *Memória Virtual*

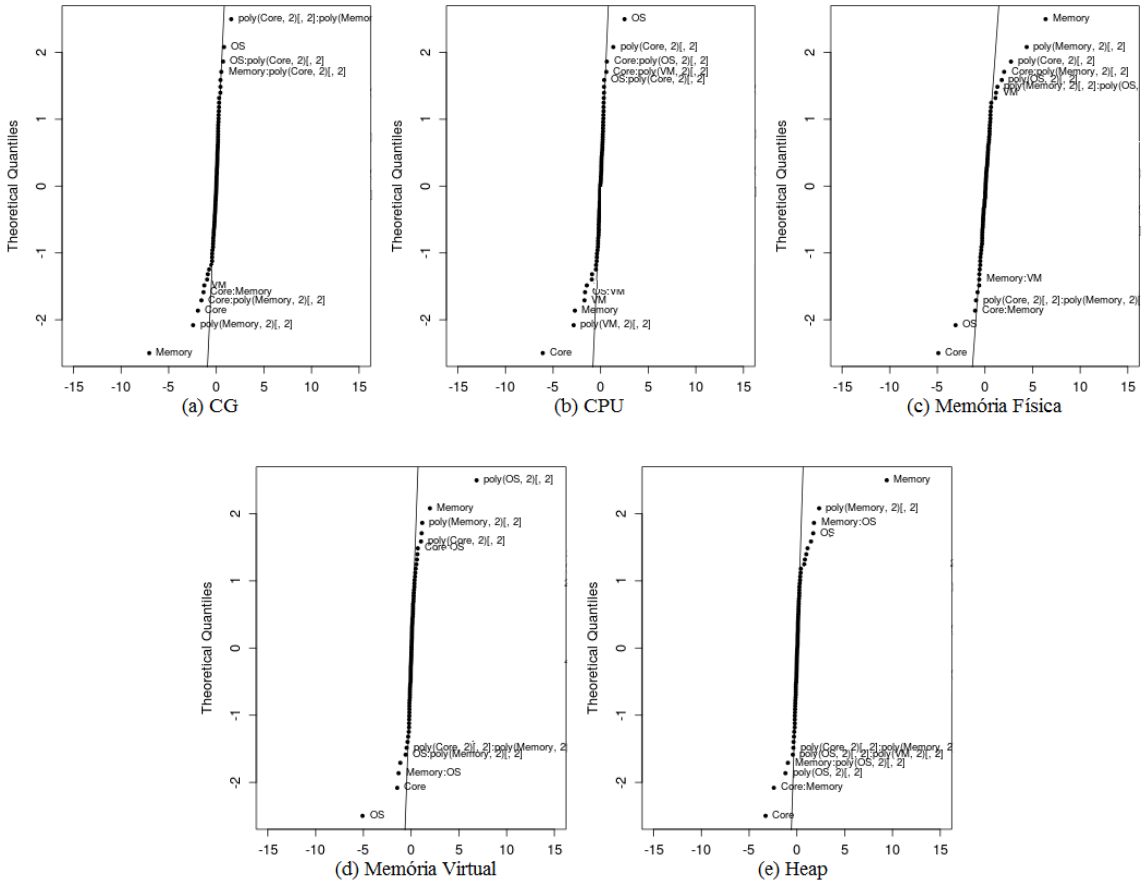
	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	-1.5131	0.1681	0.0318	5.2810	0.0000
Núcleo	-1.4504	0.1974	0.0390	5.0630	0.0000
Memória	1.9696	0.2680	0.0390	6.8750	0.0000
SO	-5.0848	-0.6920	0.0390	-17.7480	< 2e-16
VM	-0.1858	-0.0253	0.0390	-0.6490	0.5197
poly(Núcleo, 2)[, 2]	1.0346	-1.0346	0.2865	-3.6110	0.0007
poly(Memória, 2)[, 2]	1.1651	-1.1651	0.2865	-4.0670	0.0002
poly(SO, 2)[, 2]	6.8516	-6.8516	0.2865	-23.9150	< 2e-16
poly(VM, 2)[, 2]	0.4056	-0.4056	0.2865	-1.4160	0.1633
Núcleo:Memória	-0.3096	0.0516	0.0478	1.0800	0.2853
Núcleo:SO	0.6979	0.1163	0.0478	2.4360	0.0186
Núcleo:VM	0.2674	0.0446	0.0478	0.9330	0.3552
Núcleo:poly(Memória, 2)[, 2]	0.2685	-0.3288	0.3509	-0.9370	0.3534
Núcleo:poly(SO, 2)[, 2]	0.5494	-0.6729	0.3509	-1.9180	0.0611
Núcleo:poly(VM, 2)[, 2]	-0.1283	0.1571	0.3509	0.4480	0.6564
Memória:SO	-1.3137	0.2190	0.0478	4.5850	0.0000
Memória:VM	0.1896	0.0316	0.0478	0.6620	0.5113
Memória:poly(Núcleo, 2)[, 2]	0.1995	-0.2444	0.3509	-0.6960	0.4895
Memória:poly(SO, 2)[, 2]	-0.2201	-0.2696	0.3509	-0.7680	0.4461
Memória:poly(VM, 2)[, 2]	0.1050	0.1286	0.3509	0.3670	0.7156
SO:VM	-0.1769	-0.0295	0.0478	-0.6170	0.5399
SO:poly(Núcleo, 2)[, 2]	0.3649	-0.4470	0.3509	-1.2740	0.2089
SO:poly(Memória, 2)[, 2]	-0.5801	-0.7105	0.3509	-2.0250	0.0485
SO:poly(VM, 2)[, 2]	0.1869	-0.2289	0.3509	-0.6520	0.5173
VM:poly(Núcleo, 2)[, 2]	-0.1810	-0.2217	0.3509	-0.6320	0.5306
VM:poly(Memória, 2)[, 2]	0.1374	-0.1683	0.3509	-0.4800	0.6336
VM:poly(SO, 2)[, 2]	-0.0382	-0.0468	0.3509	-0.1330	0.8944
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.4734	4.2605	2.5785	1.6520	0.1050
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.6561	5.9049	2.5785	2.2900	0.0265
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.3469	3.1224	2.5785	1.2110	0.2319
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.4784	4.3053	2.5785	1.6700	0.1015
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.2155	1.9397	2.5785	0.7520	0.4556
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	0.2596	2.3362	2.5785	0.9060	0.3694

Tabela 58: Modelo para o fatorial 3^4 para o algoritmo *WordCount* após a transformação de Johnson para a variável resposta *Heap*

	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	0.5373	-0.0597	0.0221	-2.7010	0.0095
Núcleo	-3.2752	0.4457	0.0271	16.4620	< 2e-16
Memória	9.4062	1.2800	0.0271	47.2790	< 2e-16
SO	1.7113	0.2329	0.0271	8.6010	0.0000
VM	0.0270	0.0037	0.0271	0.1360	0.8926
poly(Núcleo, 2)[, 2]	1.4804	-1.4804	0.1990	-7.4410	0.0000
poly(Memória, 2)[, 2]	2.3320	-2.3320	0.1990	-11.7210	0.0000
poly(SO, 2)[, 2]	-1.1995	1.1995	0.1990	6.0290	0.0000
poly(VM, 2)[, 2]	0.4123	-0.4123	0.1990	-2.0720	0.0436
Núcleo:Memória	-2.4131	0.4022	0.0332	12.1290	0.0000
Núcleo:SO	0.1928	0.0321	0.0332	0.9690	0.3373
Núcleo:VM	0.1646	0.0274	0.0332	0.8270	0.4121
Núcleo:poly(Memória, 2)[, 2]	0.1919	-0.2350	0.2437	-0.9640	0.3397
Núcleo:poly(SO, 2)[, 2]	-0.1611	0.1973	0.2437	0.8100	0.4221
Núcleo:poly(VM, 2)[, 2]	0.3289	-0.4028	0.2437	-1.6530	0.1049

Memória:SO	1.7868	-0.2978	0.0332	-8.9810	0.0000
Memória:VM	0.2558	0.0426	0.0332	1.2860	0.2046
Memória:poly(Núcleo, 2)[, 2]	0.9821	-1.2028	0.2437	-4.9360	0.0000
Memória:poly(SO, 2)[, 2]	-0.9402	-1.1515	0.2437	-4.7260	0.0000
Memória:poly(VM, 2)[, 2]	-0.2209	-0.2705	0.2437	-1.1100	0.2724
SO:VM	0.3953	0.0659	0.0332	1.9870	0.0526
SO:poly(Núcleo, 2)[, 2]	-0.1915	0.2345	0.2437	0.9620	0.3406
SO:poly(Memória, 2)[, 2]	1.1147	1.3652	0.2437	5.6030	0.0000
SO:poly(VM, 2)[, 2]	0.1519	-0.1860	0.2437	-0.7630	0.4490
VM:poly(Núcleo, 2)[, 2]	0.0241	0.0295	0.2437	0.1210	0.9042
VM:poly(Memória, 2)[, 2]	-0.0063	0.0077	0.2437	0.0310	0.9751
VM:poly(SO, 2)[, 2]	0.1063	0.1302	0.2437	0.5340	0.5956
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.3698	3.3282	1.7906	1.8590	0.0692
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.1046	0.9418	1.7906	0.5260	0.6013
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	-0.0457	-0.4115	1.7906	-0.2300	0.8192
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.7737	6.9637	1.7906	3.8890	0.0003
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	0.2593	-2.3339	1.7906	-1.3030	0.1986
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.4259	-3.8333	1.7906	-2.1410	0.0374

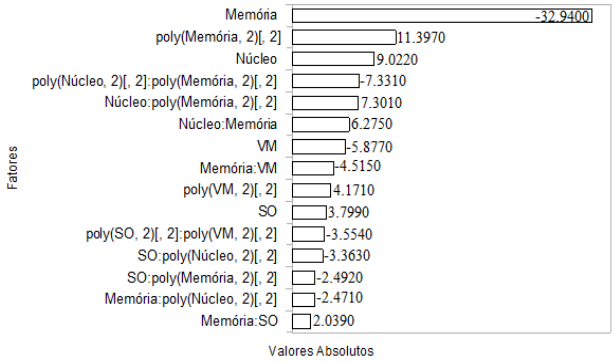
Figura 72: Gráfico da probabilidade normal dos efeitos estimados do fatorial 3^4 dos tempos para o algoritmo *WordCount* para as variáveis respostas, após a transformação de Johnson



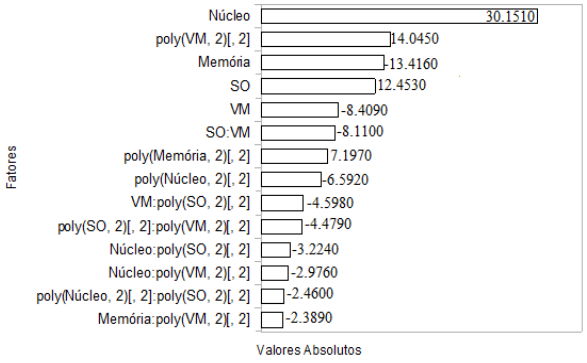
Como descrito na Seção 6.3, os níveis de significância adotados nos experimentos foi de 5% de significância estatística. A Figura 73 mostra o gráfico de Pareto que apresenta os valores absolutos estimados de cada efeito em cada variável resposta para o algoritmo

WordCount. Pode-se verificar que na maioria dos gráficos (Figuras 73ª e 73c), a Memória é o fator com maior efeito significativo. Idêntico a análise feita para o algoritmo *Pi*, os gráficos de Pareto para as variáveis respostas *Memória Virtual* (Figura 73d) *e Heap* (Figura 73e) também mostraram que o fator VM não foi estatisticamente significativo.

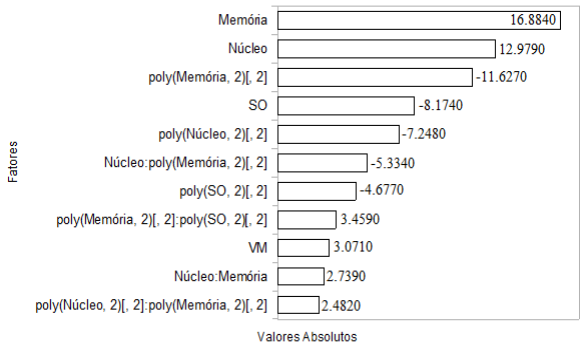
Figura 73: Gráfico de Pareto com os efeitos estimados em valor percentual absoluto para o tempo de execução para o algoritmo *Pi*, após a transformação de Johnson.



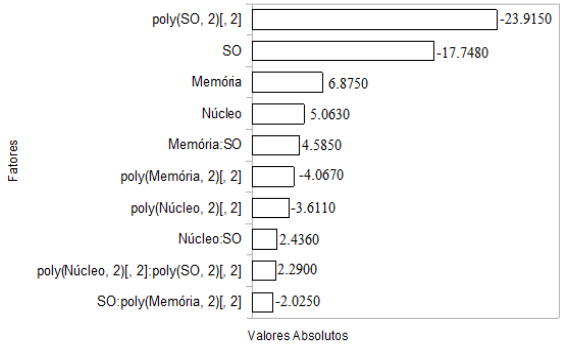
(a) *Garbage Collector*



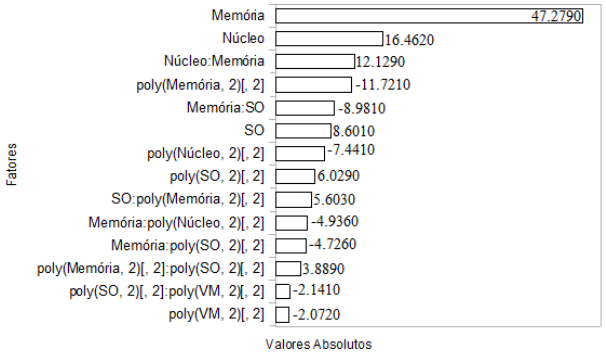
(b) CPU



(c) Memória Física



(d) Memória Virtual



(e) Heap

Os coeficientes de determinação R^2 com 48 graus de liberdade, apresentado na Tabela 59.

Tabela 59: Coeficientes de determinação para o algoritmo *WordCount* após a transformação de Johnson para cada variável resposta

Variáveis respostas	**R^2**
Garbage Collector	0.9706
CPU	0.9736
Memória Física	0.9444
Memória Virtual	0.9562
Heap	0.9850
Tempo	0.8904

Testes formais foram realizados (Tabela 60) e confirmaram que para as variáveis respostas *Garbage Collector, CPU, Memória Física e Memória Virtual*, os resíduos estavam distribuídos de forma homogênea (teste de Breusch-Pagan, função *bpTest*). Já para o teste de distribuição normal (teste de Shapiro-Wilk, função *shapiro.test*), somente as variáveis respostas *Garbage Collector e CPU* os resíduos se aproximam de uma distribuição normal. Portanto, de acordo com Montgomery (2009), desvios de normalidade moderados podem ocorrer contanto que pontos discrepantes sejam investigados. Por fim, para o teste o teste de auto correlação negativa e positiva (teste de DurbinWatson, função durbinWatsonTest), a variável dependente *Heap* não pode-se confluir a dependência dos resíduos, as variáveis dependentes *Garbage Collector, CPU, Memória Física e Memória Virtual* não existe auto correlação dos resíduos.

Tabela 60: Testes formais de Breusch-Pagan, Shapiro-Wilk, DurbinWatson para o algoritmo *WordCount* após a transformação de Johnson para cada variável resposta

Variável dependente	**Breusch-Pagan valor-p**	**Shapiro-Wilk valor-p**	**DurbinWatson Dw**
Garbage Collector	0.4438	0.7487	2.334
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
CPU	0.4740	0.8577	1.741
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Memória física	0.2262	1.595e-05	1.778
	Aceita H_0	**Rejeita H_0**	**Aceita H_0**
Memória Virtual	0.1855	3.63e-08	2.3167
	Aceita H_0	**Rejeita H_0**	**Aceita H_0**
Heap	0.04277	3.019e-07	1.5741
	Rejeita H_0	**Rejeita H_0**	**Teste inconclusivo**

Como apresentado na Tabela 60, a hipótese de distribuição normal foi rejeitada pelo teste de Shapiro-Wilk para as variáveis respostas (*Memória Física, Memória Virtual e Heap*), porém a Tabela 61 mostra que os testes de Kolmogorov-Smirnov (Lilliefors, 1967) e Andereson-Darling (Razali, et al., 2011) confirmam que os resíduos aproximam de uma distribuição normal, aceitando a hipótese de Normalidade.

Tabela 61: Testes formais de homoscedasticidade para o algoritmo *WordCount* após a transformação de Johnson (H_0: variância dos erros são iguais)

Variável dependente	Shapiro-Wilk valor-p	Kolmogorov-Smirnov valor-p	Anderson-Darling valor-p
GARBAGE COLLECTOR	0.7487 **Aceita H_0**	0.9926 **Aceita H_0**	0.8951 **Aceita H_0**
CPU	0.8577 **Aceita H_0**	0.9994 **Aceita H_0**	0.9137 **Aceita H_0**
Memória física	1.595e-05 **Rejeita H_0**	0.4240 **Aceita H_0**	0.05357 **Aceita H_0**
Memória Virtual	3.63e-08 **Rejeita H_0**	0.1584 **Aceita H_0**	0.06143 **Aceita H_0**
Heap	3.019e-07 **Rejeita H_0**	0.1876 **Aceita H_0**	0.05012 **Aceita H_0**

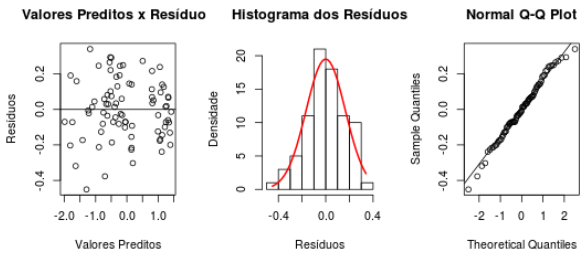
Também, como apresentado na Tabela 60, a hipótese de homoscedasticidade foi rejeitada pelo teste de Breusch-Pagan para as variáveis respostas (*Heap e Tempo*), porém a Tabela 62 mostra que os testes de *Goldfeld-Quandt* (Goldfeld, et al., 1965) e *Harrison-McCabe* (Harrison, et al., 1979) confirmam que os resíduos aproximam de uma distribuição normal, aceitando a hipótese de Homoscedasticidade.

Tabela 62: Testes formais de homoscedasticidade para o algoritmo *WordCount* após a transformação de Johnson (H_0: variância dos erros são iguais)

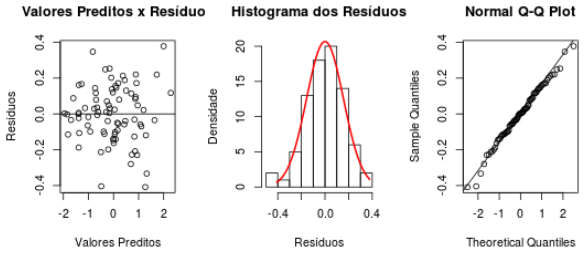
Variável dependente	Breusch-Pagan valor-p	Goldfeld-Quandt valor-p	Harrison-McCabe valor-p
GARBAGE COLLECTOR	0.4438 **Aceita H_0**	0.6299 **Aceita H_0**	0.714 **Aceita H_0**
CPU	0.4740 **Aceita H_0**	0.4023 **Aceita H_0**	0.254 **Aceita H_0**
Memória física	0.2262 **Aceita H_0**	0.4717 **Aceita H_0**	0.341 **Aceita H_0**
Memória Virtual	0.1855 **Aceita H_0**	0.084 **Aceita H_0**	2.2e-16 **Rejeita H_0**
Heap	0.04277 **Rejeita H_0**	0.1356 **Aceita H_0**	0.078 **Aceita H_0**

A Figura 74 apresenta a verificação do modelo através da análise dos resíduos para cada variável resposta. O gráfico da esquerda para direita representa a relação entre os resíduos pelos valores preditos e não apresenta estrutura ou padrão. O histograma dos resíduos que apresenta a distribuição da frequência dos resíduos apresentou aproximadamente simétrico em forma de sino. O gráfico à direita representa a probabilidade normal dos resíduos apresentando pontos próximos da reta.

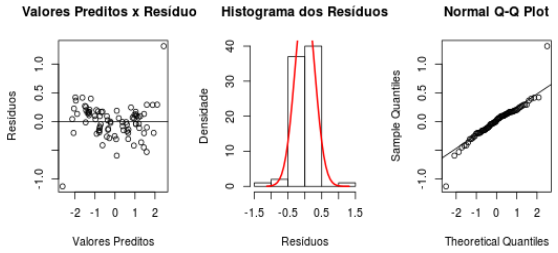
Figura 74: Verificação dos resíduos: gráfico resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem de dois do fatorial 3^4 do algoritmo *WordCount*, após a transformação de Johnson para cada variável resposta



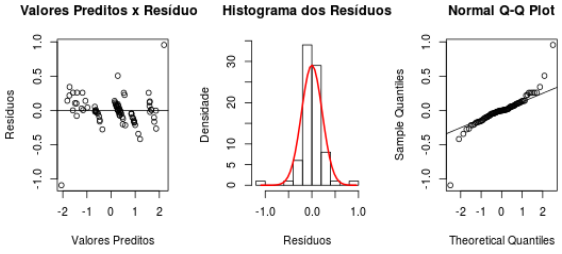
(a) *Garbage Collector*



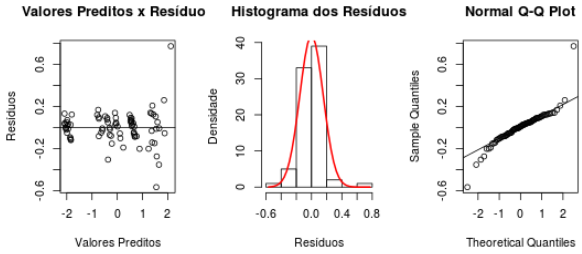
(b) CPU



(c) Memória Física



(d) Memória Virtual



(e) Heap

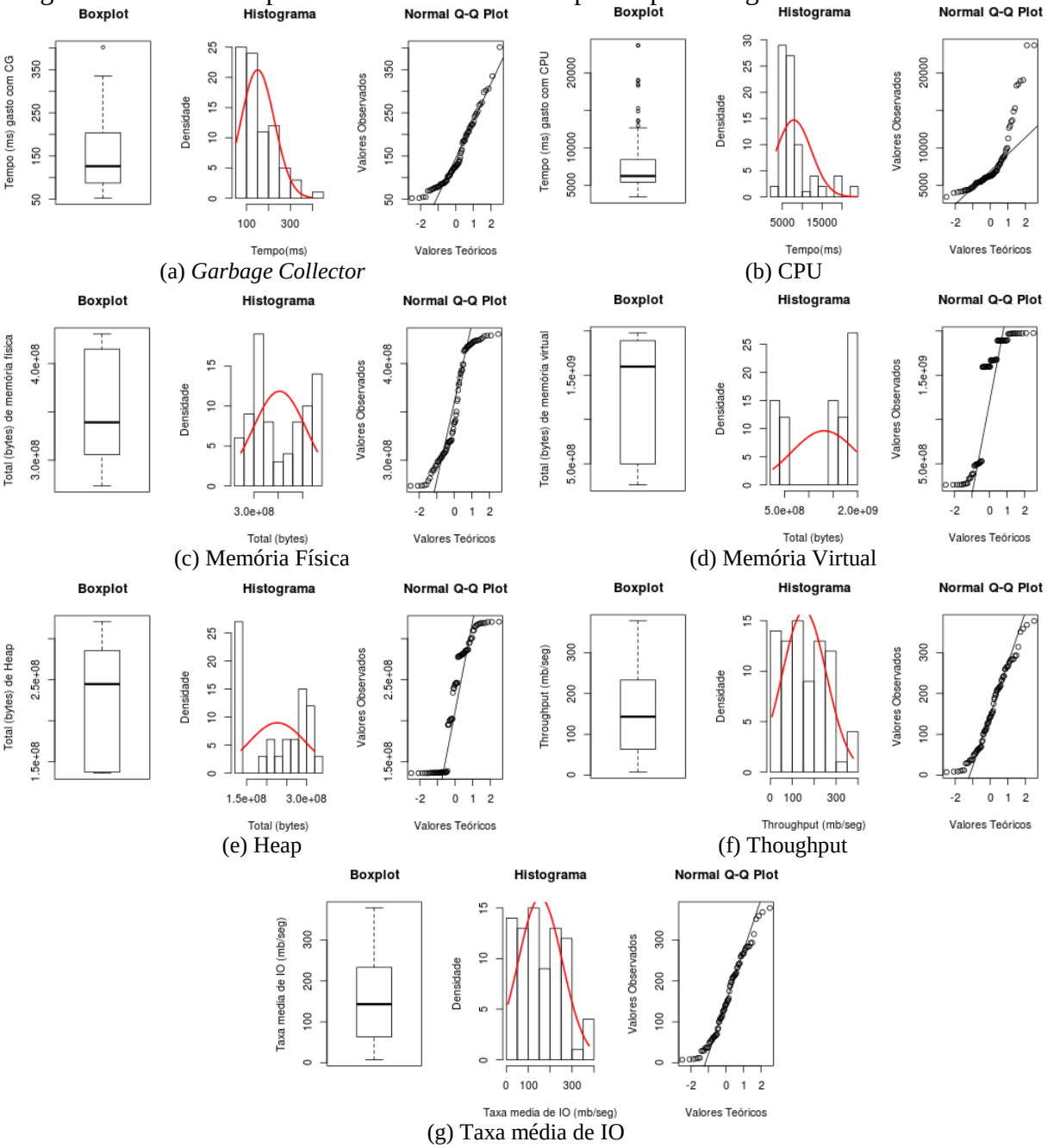
D.3 Algoritmo *TestDFSIO Read*

A Figura 75 apresenta da esquerda para a direita, informações exploratórias sobre as variáveis respostas (*Garbage Collector*, *CPU*, *memória física*, *memória virtual*, *throughput*, *taxa média de IO e heap*) respectivamente, encontradas durante a execução do algoritmo *TestDFSIO Read* nos ambientes virtuais analisados. O primeiro gráfico de cada variável resposta apresenta o diagrama de caixa com as respectivas respostas.

Pode-se verificar que para esse algoritmo o uso de *Garbage Collector* (Figura 75a) foi muito baixo, ou seja, 75% dos experimentos utilizaram menos que 200 milissegundos o *Garbage Collector*. O uso da CPU (Figura 75b) na maioria dos ensaios (75%) ficou abaixo de 10000 milissegundos, Para as variáveis respostas: memória física (Figura 75c), virtual (Figura 75d) e heap (Figura 75e) representou um grande uso durante a execução do algoritmo. Para o

Throughput (Figura 75f) e a taxa media de IO (Figura 75g), 75% dos experimentos a quantidade de megabytes por segundo foi inferior a 250 mb/seg.

Figura 75: Gráficos exploratórios das variáveis respostas para o algoritmo *TestDFSIO Read*



(a) *Garbage Collector* (b) CPU

(c) Memória Física (d) Memória Virtual

(e) Heap (f) Thoughput

(g) Taxa média de IO

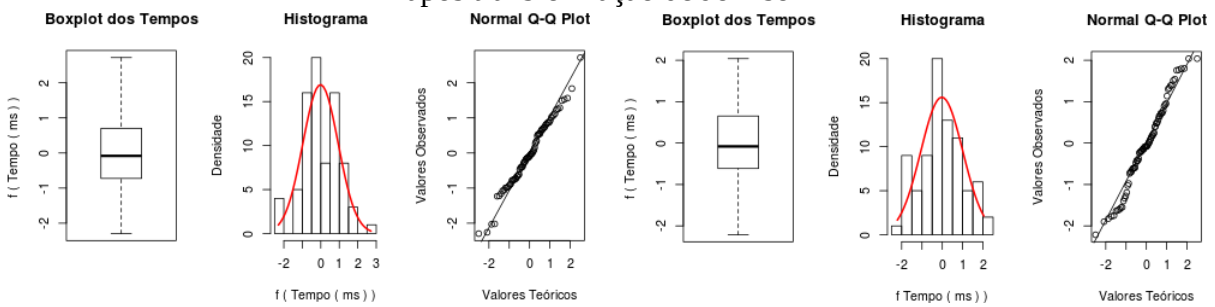
Os gráficos de normalidade da Figura 75 mostram que os dados coletados não se aproximam de uma distribuição normal o que caracteriza a necessidade de se aplicar uma transformação nos dados. Para cada conjunto de dados de cada variável resposta foi aplicada a transformação de Johnson (JOHNSON, 2002).

Para cada variável resposta o resultado da transformação de Johnson foi:

- *Garbage Collector*: família = SB, γ = 1.078898, λ = 406.4077, ε = 45.4989 e η = 0.8349106;
- CPU: família = SU, γ = -1.147514, λ = 850.0074, ε = 4840.354 e η = 0.8405214;
- Memória física: família = SB, γ = -0.195538, λ = 157757172, ε = 272870551 e η = 0.4090911;
- Memória virtual: família = SB, γ = -0.1643852, λ = 1716469916, ε = 259948545 e η = 0.2964719;
- Heap: família = SB, γ = 0.3763151, λ = 184358912, ε = 136252123 e η = 0.2292399;
- *Throughput*: família = SB, γ = 0.3872443, λ = 391.8339, ε = -4.365301 e η = 0.6411024;
- Taxa media de IO: família = SB, γ = 0.3872443, λ = 391.8339, ε = -4.3653 e η = 0.6411024;

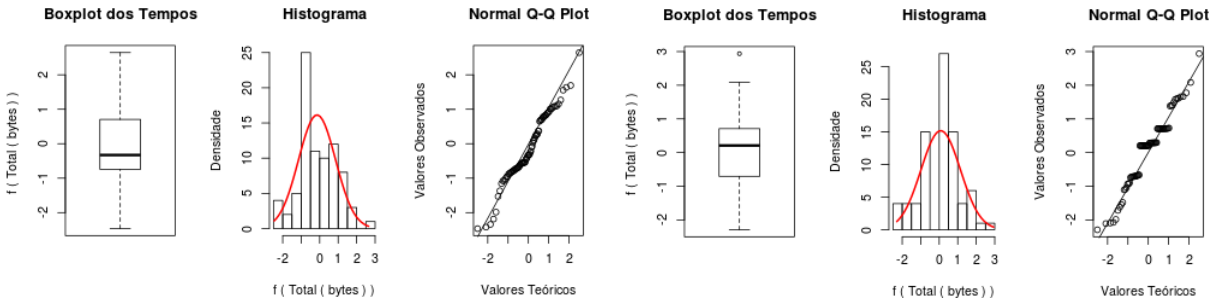
A Figura 76 mostra os mesmos gráficos, porém após a transformação de Johnson. Os diagramas de caixas e os histogramas apresentam ser mais simétricos em relação aos apresentados na Figura 75. E por fim, pode-se observar que todos os gráficos de normalidade os dados aproximam de uma distribuição normal.

Figura 76: Gráficos exploratórios das variáveis respostas para o algoritmo *TestDFSIO Read* após transformação de Johnson



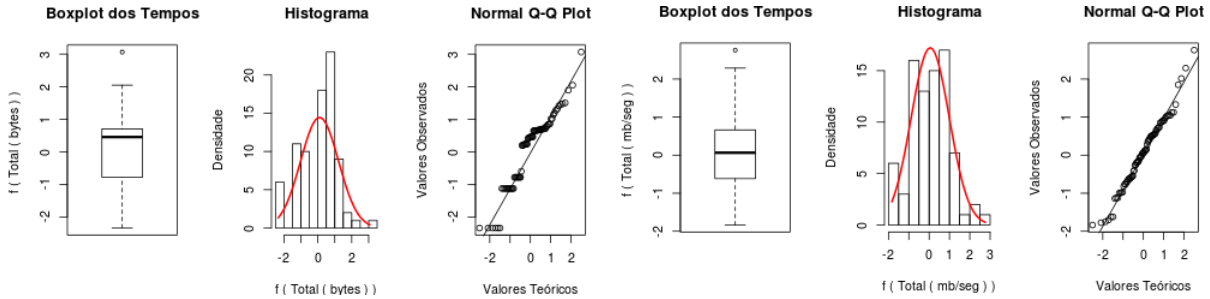
(a) *Garbage Collector*

(b) CPU

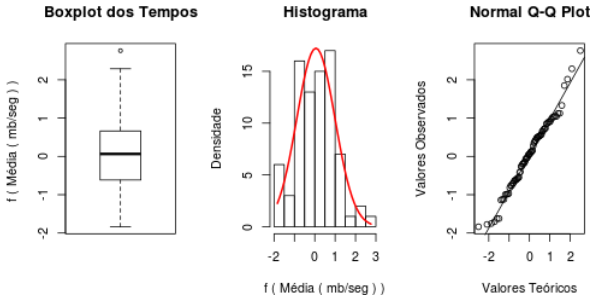


(c) Memória Física

(d) Memória Virtual



(e) Heap (f) Thoughput



(g) Taxa média de IO

A Figura 77 mostra as mesmas informações, porém agrupadas através da combinação do fator sistema operacional pelo fator máquina virtual. As análises baseadas em *boxplot* são limitadas, podendo tirar conclusões somente quando os agrupamentos são óbvios. A Tabela 63 mostra o resultado da aplicação do teste *t* para demais conclusões.

Pode-se então verificar que existem evidências estatísticas que a variável resposta *Garbage Collector* (Figura 77a) com a combinação VM:SO (nível –, nível +) em média possui o maior tempo de gasto em milissegundos do *Garbage Collector* (Tabela 63). Em relação a utilização de CPU (Figura 77b) a combinação KVM:Windows (nível –, nível +) e Xen:Windows (nível 0, nível +) em média utiliza em maior tempo a CPU.

A quantidade de memória física (Figura 77c) também a combinação KVM:Windows (nível –, nível +) em média utiliza a menor quantidade de memória física em *bytes*. Já em relação à utilização da memória virtual (Figura 77d) a máquina virtual não interfere no desempenho de um algoritmo. A maior utilização de memória virtual está com o sistema operacional CentOS e a menor utilização de memória virtual está com o sistema operacional Windows (Tabela 63). Em relação a variável resposta Heap (Figura 77e) mostrou que não existem evidências estatísticas que sejam diferentes (Tabela 63).

As variáveis respostas *Throughput* (Figura 77f) e taxa média de IO (Figura 77e) possui a mesma resposta, ou seja, a combinação KVM:Windows (nível –, nível +) e Xen:Windows

(nível 0, nível +) em média possuem uma vazão de megabytes por segundo inferior as demais configurações e também uma taxa média de IO inferior as demais configurações.

Figura 77: Diagrama de caixa (boxplot) das variáveis de respostas para o algoritmo *TestDFSIO Read*, após a transformação de Johnson: variáveis respostas agrupadas pela combinação do sistema operacional pela máquina virtual

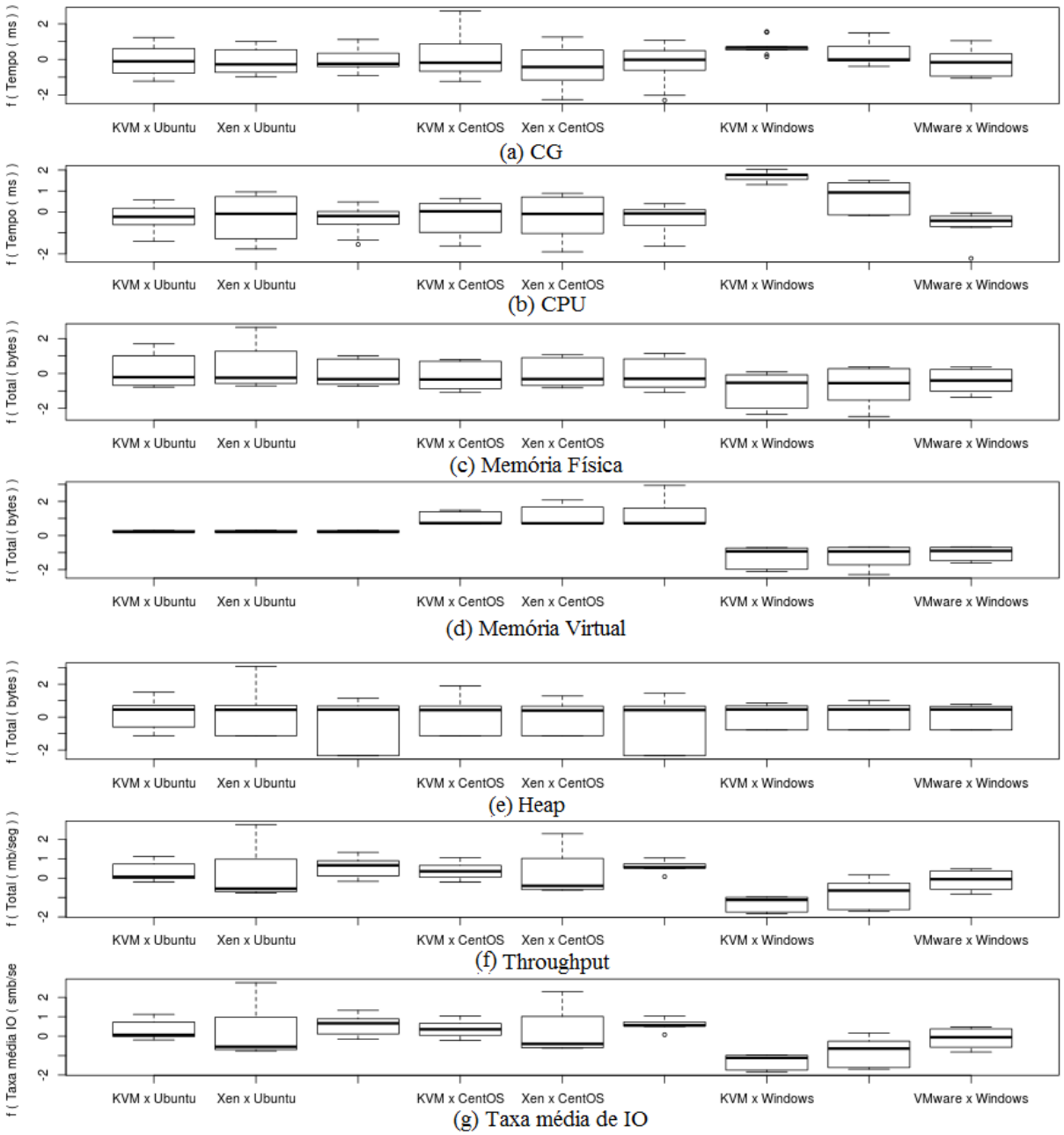


Tabela 63: Comparação entre os fatores após a transformação de Johnson: variáveis respostas agrupadas pela combinação do sistema operacional pela máquina virtual para o algoritmo *TestDFSIO Read*

Teste *t* aplicado para a variável dependente *Garbage Collector* (ms)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	-1	0	0
Xen x Ubuntu	0	--	0	0	0	0	-1	0	0
VMware x Ubuntu	0	0	--	0	0	0	-1	0	0
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	-1	0	0
VMware x CentOS	0	0	0	0	0	--	-1	0	0
KVM x Windows	+1	+1	+1	0	+1	+1	--	0	+1
Xen x Windows	0	0	0	0	0	0	0	--	0
VMware x Windows	0	0	0	0	0	0	-1	0	--
Teste *t* aplicado para a variável dependente CPU (ms)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	-1	-1	0
Xen x Ubuntu	0	--	0	0	0	0	-1	0	0
VMware x Ubuntu	0	0	--	0	0	0	-1	-1	0
KVM x CentOS	0	0	0	--	0	0	-1	-1	0
Xen x CentOS	0	0	0	0	--	0	-1	-1	0
VMware x CentOS	0	0	0	0	0	--	-1	-1	0
KVM x Windows	+1	+1	+1	+1	+1	+1	--	+1	+1
Xen x Windows	+1	0	+1	+1	+1	+1	-1	--	+1
VMware x Windows	0	0	0	0	0	0	-1	-1	--
Teste *t* aplicado para a variável dependente Memória Física (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	+1	0	0
Xen x Ubuntu	0	--	0	0	0	0	+1	+1	0
VMware x Ubuntu	0	0	--	0	0	0	+1	0	0
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	+1	0	0
VMware x CentOS	0	0	0	0	0	--	0	0	0
KVM x Windows	-1	-1	-1	0	-1	0	--	0	0
Xen x Windows	0	-1	0	0	0	0	0	--	0
VMware x Windows	0	0	0	0	0	0	0	0	--
Teste *t* aplicado para a variável dependente Memória Virtual (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	-1	-1	-1	+1	+1	+1
Xen x Ubuntu	0	--	0	-1	-1	-1	+1	+1	+1
VMware x Ubuntu	0	0	--	-1	-1	-1	+1	+1	+1
KVM x CentOS	+1	+1	+1	--	0	0	+1	+1	+1
Xen x CentOS	+1	+1	+1	+1	--	0	+1	+1	+1
VMware x CentOS	+1	+1	+1	+1	+1	--	+1	+1	+1
KVM x Windows	-1	-1	-1	-1	-1	-1	--	0	0
Xen x Windows	-1	-1	-1	-1	-1	-1	0	--	0
VMware x Windows	-1	-1	-1	-1	-1	-1	0	0	--
Teste *t* aplicado para a variável dependente Heap (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	0	0	0
Xen x Ubuntu	0	--	0	0	0	0	0	0	0

VMware x Ubuntu	0	0	--	0	0	0	0	0	0
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	0	0	0
VMware x CentOS	0	0	0	0	0	--	0	0	0
KVM x Windows	0	0	0	0	0	0	--	0	0
Xen x Windows	0	0	0	0	0	0	0	--	0
VMware x Windows	0	0	0	0	0	0	0	0	--
Teste *t* aplicado para a variável dependente Throughput (mb/seg)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	+1	+1	0
Xen x Ubuntu	0	--	0	0	0	0	+1	+1	0
VMware x Ubuntu	0	0	--	0	0	0	+1	+1	+1
KVM x CentOS	0	0	0	--	0	0	+1	+1	0
Xen x CentOS	0	0	0	0	--	0	+1	+1	0
VMware x CentOS	0	0	0	0	0	--	+1	+1	+1
KVM x Windows	-1	-1	-1	-1	-1	-1	--	0	-1
Xen x Windows	-1	-1	-1	-1	-1	-1	0	--	-1
VMware x Windows	0	0	-1	0	0	-1	+1	+1	--
Teste *t* aplicado para a variável dependente Taxa média IO (mb/seg)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	+1	+1	0
Xen x Ubuntu	0	--	0	0	0	0	+1	+1	0
VMware x Ubuntu	0	0	--	0	0	0	+1	+1	+1
KVM x CentOS	0	0	0	--	0	0	+1	+1	0
Xen x CentOS	0	0	0	0	--	0	+1	+1	0
VMware x CentOS	0	0	0	0	0	--	+1	+1	+1
KVM x Windows	-1	-1	-1	-1	-1	-1	--	0	-1
Xen x Windows	-1	-1	-1	-1	-1	-1	0	--	-1
VMware x Windows	0	0	-1	0	0	-1	+1	+1	--

A análise do fatorial 3^4 se iniciou com a projeção do gráfico de probabilidade normal dos efeitos estimados para a identificação dos efeitos significativos (Figura 78) para cada variável resposta. Pode-se verificar que em todos os gráficos de probabilidade normal os fatores SO e/ou VM foram estatisticamente significativos. Além disso, o fator Memoria foi bastante significativo em quase todos as variáveis respostas analisadas.

A Figura 79 mostra o gráfico de Pareto que apresenta os valores absolutos estimados de cada efeito em cada variável resposta para o algoritmo *TestDFSIO Read*. Pode-se verificar que na maioria dos gráficos (Figuras 79a, e 79c), a memória é o fator com maior efeito significativo. Idêntico a análise feita para os algoritmos *Pi* e *WordCount*.

Além disso, o fator memória também foi o mais significativo para as variáveis repostas *Throughput* e Taxa média de IO. Já os gráficos de Pareto para as variáveis respostas *Memória Virtual* (Figura 79d) *e Heap* (Figura 79e) mostraram que o fator VM não foi estatisticamente significativo diferentemente dos mesmos fatores analisados pelos algoritmos *Pi* e *WordCount.*

Figura 78: Gráfico da probabilidade normal dos efeitos estimados do fatorial 3^4 dos tempos para o algoritmo *TestDFSIO Read: Garbage Collector* (ms), CPU (ms), memória física (bytes), memória virtual (bytes), heap (bytes), throughput (mb/seg) e taxa média de IO (mb/seg), respectivamente, após a transformação de Johnson.

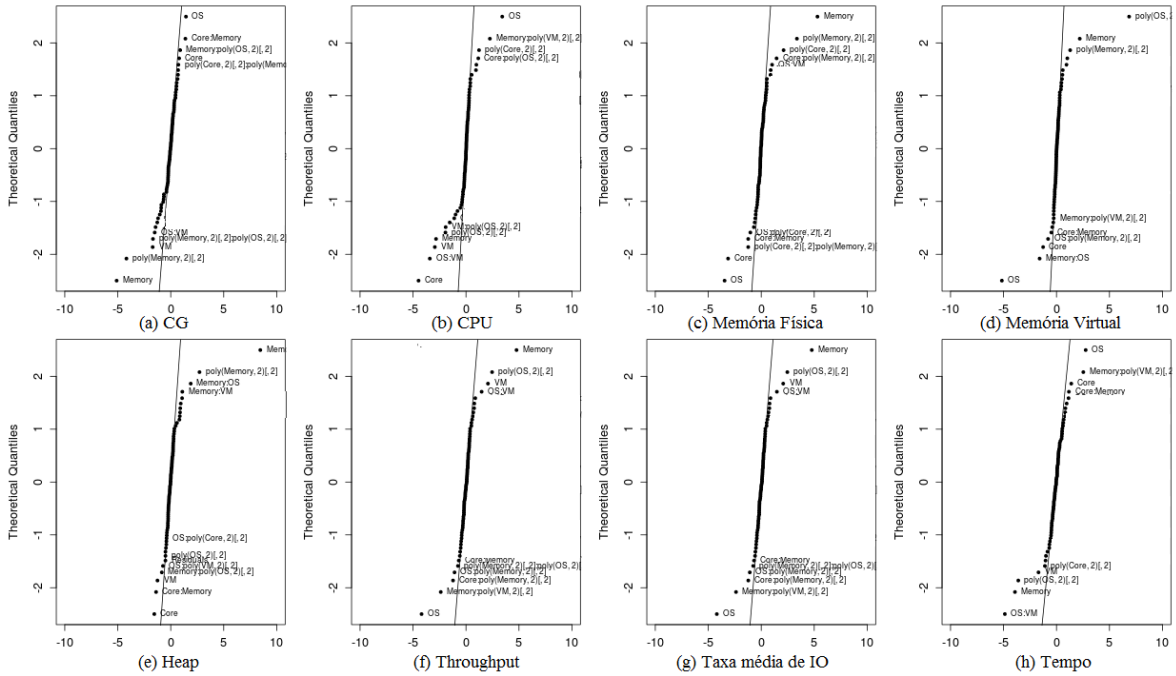
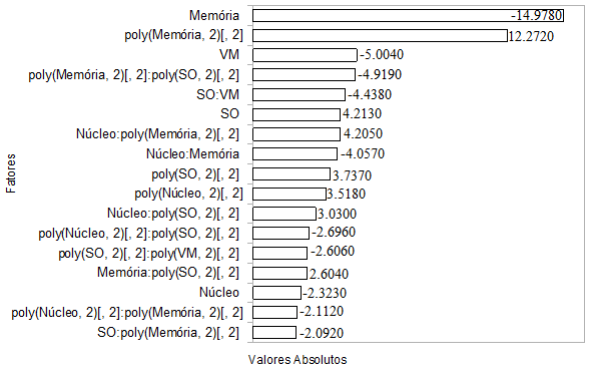
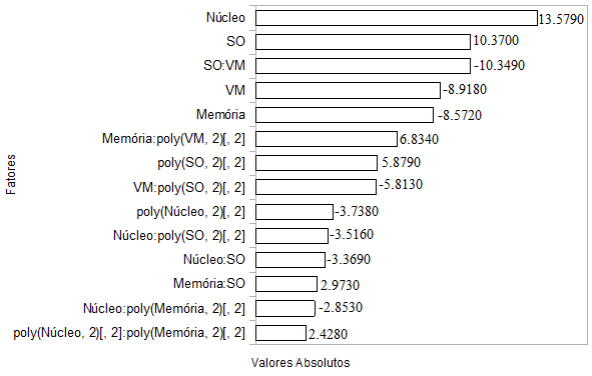


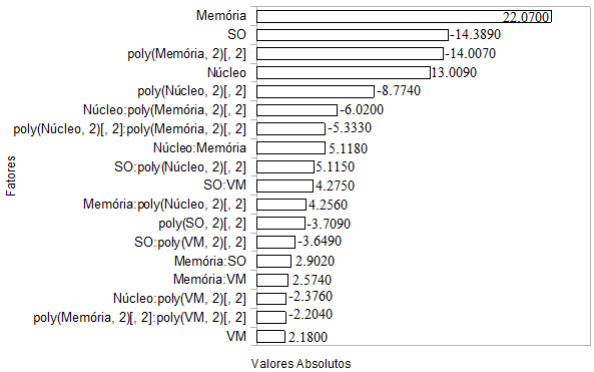
Figura 79: Gráfico de Pareto com os efeitos estimados em valor percentual absoluto para o tempo de execução para o algoritmo *TestDFISO Read*, após a transformação de Johnson.

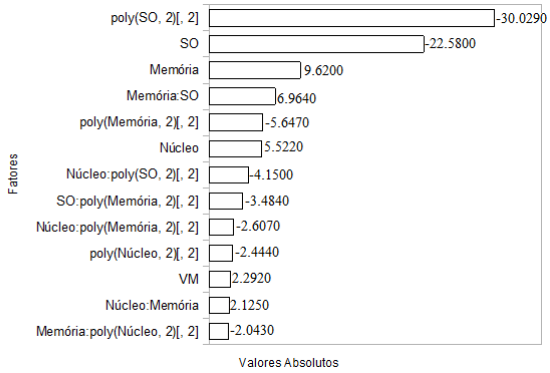


(a) *Garbage Collector*

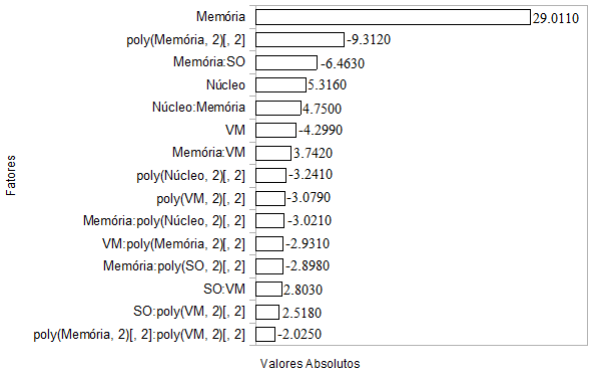


(b) CPU

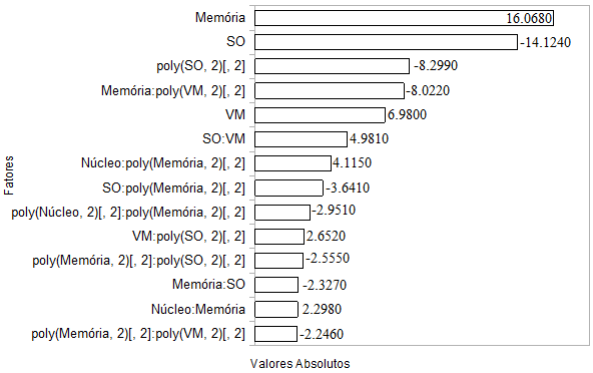




(c) Memória Física

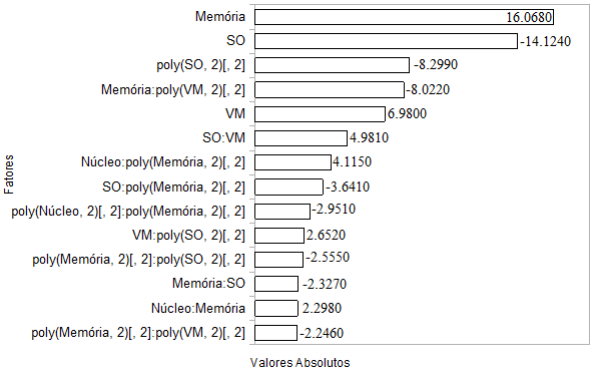


(d) Memória Virtual



(e) Heap

(f) Throughput



(g) Taxa média de IO

Com 5% de significância as Tabelas 64, 65, 66, 67, 68, 69 e 70 apresentam a Tabela ANOVA para as variáveis dependentes (*Garbage Collector*, *CPU*, *Memória Física*, *Memoria Virtual*, *Heap*, *Throughput e Taxa média IO*, respectivamente) para o algoritmo *TestDFSIO Read*.

Tabela 64: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Read* após a transformação Johnson das respostas para a variável dependente *Garbage Collector*

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	0.1268	-0.0141	0.0379	-0.3720	0.7116
Núcleo	0.7919	-0.1078	0.0464	-2.3230	0.0245
Memória	-5.1057	-0.6948	0.0464	-14.9780	< 2e-16
SO	1.4363	0.1955	0.0464	4.2130	0.0001
VM	-1.7059	-0.2321	0.0464	-5.0040	0.0000
poly(Núcleo, 2)[, 2]	-1.1993	1.1993	0.3409	3.5180	0.0010
poly(Memória, 2)[, 2]	-4.1833	4.1833	0.3409	12.2720	< 2e-16
poly(SO, 2)[, 2]	-1.2737	1.2737	0.3409	3.7370	0.0005
poly(VM, 2)[, 2]	-0.6419	0.6419	0.3409	1.8830	0.0658
Núcleo:Memória	1.3828	-0.2305	0.0568	-4.0570	0.0002
Núcleo:SO	0.5428	0.0905	0.0568	1.5920	0.1179
Núcleo:VM	-0.2241	-0.0374	0.0568	-0.6570	0.5140
Núcleo:poly(Memória, 2)[, 2]	-1.4333	1.7555	0.4175	4.2050	0.0001
Núcleo:poly(SO, 2)[, 2]	-1.0330	1.2652	0.4175	3.0300	0.0039
Núcleo:poly(VM, 2)[, 2]	0.1228	-0.1504	0.4175	-0.3600	0.7203

Memória:SO	-0.0633	0.0106	0.0568	0.1860	0.8535
Memória:VM	0.4527	0.0755	0.0568	1.3280	0.1905
Memória:poly(Núcleo, 2)[, 2]	-0.3185	0.3901	0.4175	0.9340	0.3548
Memória:poly(SO, 2)[, 2]	0.8877	1.0873	0.4175	2.6040	0.0122
Memória:poly(VM, 2)[, 2]	0.2174	0.2663	0.4175	0.6380	0.5266
SO:VM	-1.5129	-0.2521	0.0568	-4.4380	0.0001
SO:poly(Núcleo, 2)[, 2]	0.4418	-0.5411	0.4175	-1.2960	0.2011
SO:poly(Memória, 2)[, 2]	-0.7130	-0.8732	0.4175	-2.0920	0.0418
SO:poly(VM, 2)[, 2]	0.1626	-0.1992	0.4175	-0.4770	0.6355
VM:poly(Núcleo, 2)[, 2]	-0.0236	-0.0289	0.4175	-0.0690	0.9452
VM:poly(Memória, 2)[, 2]	-0.3951	0.4839	0.4175	1.1590	0.2522
VM:poly(SO, 2)[, 2]	0.1693	0.2073	0.4175	0.4970	0.6218
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.7199	-6.4790	3.0679	-2.1120	0.0399
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	-0.9191	-8.2722	3.0679	-2.6960	0.0096
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.1078	0.9702	3.0679	0.3160	0.7532
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-1.6770	-15.0927	3.0679	-4.9190	0.0000
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.1034	0.9309	3.0679	0.3030	0.7629
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.8883	-7.9950	3.0679	-2.6060	0.0122

Tabela 65: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Read* após a transformação Johnson das respostas para a variável dependente CPU

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	0.2147	-0.0239	0.0367	-0.6510	0.5183
Núcleo	-4.4792	0.6095	0.0449	13.5790	< 2e-16
Memória	-2.8275	-0.3848	0.0449	-8.5720	0.0000
SO	3.4208	0.4655	0.0449	10.3700	0.0000
VM	-2.9417	-0.4003	0.0449	-8.9180	0.0000
poly(Núcleo, 2)[, 2]	1.2331	-1.2332	0.3299	-3.7380	0.0005
poly(Memória, 2)[, 2]	0.3959	-0.3959	0.3299	-1.2000	0.2360
poly(SO, 2)[, 2]	-1.9393	1.9393	0.3299	5.8790	0.0000
poly(VM, 2)[, 2]	0.4068	-0.4068	0.3299	-1.2330	0.2235
Núcleo:Memória	-0.0942	0.0157	0.0550	0.2860	0.7764
Núcleo:SO	-1.1113	-0.1852	0.0550	-3.3690	0.0015
Núcleo:VM	0.1035	0.0173	0.0550	0.3140	0.7550
Núcleo:poly(Memória, 2)[, 2]	0.9410	-1.1524	0.4040	-2.8530	0.0064
Núcleo:poly(SO, 2)[, 2]	1.1598	-1.4205	0.4040	-3.5160	0.0010
Núcleo:poly(VM, 2)[, 2]	0.0463	-0.0567	0.4040	-0.1400	0.8890
Memória:SO	-0.9809	0.1635	0.0550	2.9730	0.0046
Memória:VM	0.3287	0.0548	0.0550	0.9970	0.3240
Memória:poly(Núcleo, 2)[, 2]	-0.0770	0.0943	0.4040	0.2330	0.8164
Memória:poly(SO, 2)[, 2]	0.2921	0.3578	0.4040	0.8850	0.3803
Memória:poly(VM, 2)[, 2]	2.2544	2.7611	0.4040	6.8340	0.0000
SO:VM	-3.4138	-0.5690	0.0550	-10.3490	0.0000
SO:poly(Núcleo, 2)[, 2]	-0.3513	0.4303	0.4040	1.0650	0.2922
SO:poly(Memória, 2)[, 2]	0.1895	0.2321	0.4040	0.5750	0.5683
SO:poly(VM, 2)[, 2]	0.0317	-0.0389	0.4040	-0.0960	0.9237
VM:poly(Núcleo, 2)[, 2]	0.2772	0.3395	0.4040	0.8400	0.4049
VM:poly(Memória, 2)[, 2]	-0.1871	0.2292	0.4040	0.5670	0.5732
VM:poly(SO, 2)[, 2]	-1.9176	-2.3486	0.4040	-5.8130	0.0000
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.8011	7.2098	2.9688	2.4280	0.0190
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.5298	4.7679	2.9688	1.6060	0.1148
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.2839	2.5553	2.9688	0.8610	0.3937
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-0.0760	-0.6840	2.9688	-0.2300	0.8188
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.0386	0.3472	2.9688	0.1170	0.9074
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.0604	-0.5432	2.9688	-0.1830	0.8556

Tabela 66: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Read* após a transformação Johnson das respostas para a variável dependente Memória Física

	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	1.2820	-0.1424	0.0267	-5.3330	0.0000
Núcleo	-3.1275	0.4256	0.0327	13.0090	< 2e-16
Memória	5.3056	0.7220	0.0327	22.0700	< 2e-16
SO	-3.4590	-0.4707	0.0327	-14.3890	< 2e-16
VM	0.5042	0.0686	0.0327	2.0970	0.0412
poly(Núcleo, 2)[, 2]	2.1093	-2.1093	0.2404	-8.7740	0.0000
poly(Memória, 2)[, 2]	3.3673	-3.3673	0.2404	-14.0070	< 2e-16
poly(SO, 2)[, 2]	0.8772	-0.8772	0.2404	-3.6490	0.0006
poly(VM, 2)[, 2]	0.4051	-0.4051	0.2404	-1.6850	0.0985
Núcleo:Memória	-1.2298	0.2050	0.0401	5.1150	0.0000
Núcleo:SO	-0.4399	-0.0733	0.0401	-1.8300	0.0735
Núcleo:VM	0.0094	0.0016	0.0401	0.0390	0.9691
Núcleo:poly(Memória, 2)[, 2]	1.4472	-1.7724	0.2944	-6.0200	0.0000
Núcleo:poly(SO, 2)[, 2]	0.1454	-0.1780	0.2944	-0.6050	0.5483
Núcleo:poly(VM, 2)[, 2]	0.5299	-0.6490	0.2944	-2.2040	0.0323
Memória:SO	-0.6188	0.1031	0.0401	2.5740	0.0132
Memória:VM	-0.5711	-0.0952	0.0401	-2.3760	0.0216
Memória:poly(Núcleo, 2)[, 2]	0.8916	-1.0920	0.2944	-3.7090	0.0005
Memória:poly(SO, 2)[, 2]	0.4327	0.5300	0.2944	1.8000	0.0781
Memória:poly(VM, 2)[, 2]	-0.2860	-0.3503	0.2944	-1.1900	0.2400
SO:VM	1.0231	0.1705	0.0401	4.2560	0.0001
SO:poly(Núcleo, 2)[, 2]	-1.0278	1.2588	0.2944	4.2750	0.0001
SO:poly(Memória, 2)[, 2]	0.0083	0.0102	0.2944	0.0350	0.9725
SO:poly(VM, 2)[, 2]	-0.6977	0.8546	0.2944	2.9020	0.0056
VM:poly(Núcleo, 2)[, 2]	0.2090	0.2559	0.2944	0.8690	0.3890
VM:poly(Memória, 2)[, 2]	-0.1906	0.2334	0.2944	0.7930	0.4319
VM:poly(SO, 2)[, 2]	0.0261	0.0320	0.2944	0.1090	0.9139
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-1.2304	11.0735	2.1636	5.1180	0.0000
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	-0.0058	-0.0522	2.1636	-0.0240	0.9809
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.1341	1.2073	2.1636	0.5580	0.5794
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-0.2878	-2.5901	2.1636	-1.1970	0.2371
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.5242	4.7177	2.1636	2.1800	0.0342
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	0.0380	0.3421	2.1636	0.1580	0.8750

Tabela 67: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Read* após a transformação Johnson das respostas para a variável dependente Memória Virtual

	Efeito	Coeficiente	E. Padrão	Valor t	Valor-p
(Interseção)	-0.7255	0.0806	0.0254	3.1780	0.0026
Núcleo	-1.2609	0.1716	0.0311	5.5220	0.0000
Memória	2.1965	0.2989	0.0311	9.6200	0.0000
SO	-5.1555	-0.7016	0.0311	-22.5800	< 2e-16
VM	0.5233	0.0712	0.0311	2.2920	0.0263
poly(Núcleo, 2)[, 2]	0.5581	-0.5581	0.2283	-2.4440	0.0182
poly(Memória, 2)[, 2]	1.2892	-1.2892	0.2283	-5.6470	0.0000
poly(SO, 2)[, 2]	6.8563	-6.8563	0.2283	-30.0290	< 2e-16
poly(VM, 2)[, 2]	-0.0436	0.0436	0.2283	0.1910	0.8495
Núcleo:Memória	-0.4853	0.0809	0.0381	2.1250	0.0387
Núcleo:SO	0.3115	0.0519	0.0381	1.3640	0.1788
Núcleo:VM	0.2505	0.0418	0.0381	1.0970	0.2781
Núcleo:poly(Memória, 2)[, 2]	0.5952	-0.7290	0.2796	-2.6070	0.0121
Núcleo:poly(SO, 2)[, 2]	0.9476	-1.1606	0.2796	-4.1500	0.0001

Núcleo:poly(VM, 2)[, 2]	0.0527	-0.0646	0.2796	-0.2310	0.8183
Memória:SO	-1.5901	0.2650	0.0381	6.9640	0.0000
Memória:VM	-0.1708	-0.0285	0.0381	-0.7480	0.4582
Memória:poly(Núcleo, 2)[, 2]	0.4666	-0.5714	0.2796	-2.0430	0.0465
Memória:poly(SO, 2)[, 2]	-0.0614	-0.0752	0.2796	-0.2690	0.7891
Memória:poly(VM, 2)[, 2]	-0.2631	-0.3222	0.2796	-1.1520	0.2550
SO:VM	0.2931	0.0488	0.0381	1.2840	0.2055
SO:poly(Núcleo, 2)[, 2]	0.0939	-0.1151	0.2796	-0.4110	0.6826
SO:poly(Memória, 2)[, 2]	-0.7954	-0.9742	0.2796	-3.4840	0.0011
SO:poly(VM, 2)[, 2]	-0.1293	0.1583	0.2796	0.5660	0.5739
VM:poly(Núcleo, 2)[, 2]	0.1023	0.1254	0.2796	0.4480	0.6560
VM:poly(Memória, 2)[, 2]	0.1552	-0.1901	0.2796	-0.6800	0.4999
VM:poly(SO, 2)[, 2]	-0.2373	-0.2906	0.2796	-1.0390	0.3039
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.0650	0.5852	2.0549	0.2850	0.7771
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.4532	4.0790	2.0549	1.9850	0.0529
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.2205	1.9842	2.0549	0.9660	0.3391
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.2896	2.6061	2.0549	1.2680	0.2108
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	0.0690	-0.6206	2.0549	-0.3020	0.7640
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	0.1605	1.4441	2.0549	0.7030	0.4856

Tabela 68: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Read* após a transformação Johnson das respostas para a variável dependente Heap

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	-0.9263	0.1029	0.0324	3.1790	0.0026
Núcleo	-1.5488	0.2108	0.0397	5.3160	0.0000
Memória	8.4523	1.1502	0.0397	29.0110	< 2e-16
SO	0.1474	0.0201	0.0397	0.5060	0.6152
VM	-1.2525	-0.1704	0.0397	-4.2990	0.0001
poly(Núcleo, 2)[, 2]	0.9442	-0.9442	0.2914	-3.2410	0.0022
poly(Memória, 2)[, 2]	2.7130	-2.7130	0.2914	-9.3120	0.0000
poly(SO, 2)[, 2]	-0.4986	0.4986	0.2914	1.7110	0.0935
poly(VM, 2)[, 2]	0.8972	-0.8972	0.2914	-3.0790	0.0034
Núcleo:Memória	-1.3839	0.2307	0.0486	4.7500	0.0000
Núcleo:SO	-0.2464	-0.0411	0.0486	-0.8460	0.4019
Núcleo:VM	-0.0866	-0.0144	0.0486	-0.2970	0.7675
Núcleo:poly(Memória, 2)[, 2]	0.0984	-0.1205	0.3568	-0.3380	0.7371
Núcleo:poly(SO, 2)[, 2]	0.2684	-0.3288	0.3568	-0.9210	0.3615
Núcleo:poly(VM, 2)[, 2]	0.1928	-0.2361	0.3568	-0.6620	0.5114
Memória:SO	1.8831	-0.3139	0.0486	-6.4630	0.0000
Memória:VM	1.0903	0.1817	0.0486	3.7420	0.0005
Memória:poly(Núcleo, 2)[, 2]	0.8803	-1.0781	0.3568	-3.0210	0.0040
Memória:poly(SO, 2)[, 2]	-0.8443	-1.0341	0.3568	-2.8980	0.0056
Memória:poly(VM, 2)[, 2]	0.2280	0.2793	0.3568	0.7830	0.4377
SO:VM	0.8166	0.1361	0.0486	2.8030	0.0073
SO:poly(Núcleo, 2)[, 2]	-0.3915	0.4796	0.3568	1.3440	0.1853
SO:poly(Memória, 2)[, 2]	0.1220	0.1494	0.3568	0.4190	0.6773
SO:poly(VM, 2)[, 2]	-0.7336	0.8985	0.3568	2.5180	0.0152
VM:poly(Núcleo, 2)[, 2]	-0.0948	-0.1161	0.3568	-0.3250	0.7463
VM:poly(Memória, 2)[, 2]	0.8540	-1.0460	0.3568	-2.9310	0.0052
VM:poly(SO, 2)[, 2]	0.2687	0.3291	0.3568	0.9220	0.3611
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.0133	-0.1197	2.6222	-0.0460	0.9638
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	-0.0983	-0.8845	2.6222	-0.3370	0.7373
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.3281	2.9527	2.6222	1.1260	0.2657
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.2587	2.3285	2.6222	0.8880	0.3790
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	0.5900	-5.3103	2.6222	-2.0250	0.0484
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.2240	-2.0164	2.6222	-0.7690	0.4457

Tabela 69: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Read* após a transformação Johnson das respostas para a variável dependente *Throughput*

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	-0.4912	0.0546	0.0330	1.6550	0.1045
Núcleo	-0.1920	0.0261	0.0404	0.6470	0.5209
Memória	4.7694	0.6490	0.0404	16.0680	< 2e-16
SO	-4.1924	-0.5705	0.0404	-14.1240	< 2e-16
VM	2.0720	0.2820	0.0404	6.9800	0.0000
poly(Núcleo, 2)[, 2]	0.2052	-0.2052	0.2968	-0.6910	0.4927
poly(Memória, 2)[, 2]	-0.5524	0.5524	0.2968	1.8610	0.0689
poly(SO, 2)[, 2]	2.4634	-2.4634	0.2968	-8.2990	0.0000
poly(VM, 2)[, 2]	-0.4046	0.4046	0.2968	1.3630	0.1792
Núcleo:Memória	-0.6821	0.1137	0.0495	2.2980	0.0260
Núcleo:SO	0.0602	0.0100	0.0495	0.2030	0.8401
Núcleo:VM	0.3667	0.0611	0.0495	1.2350	0.2227
Núcleo:poly(Memória, 2)[, 2]	-1.2215	1.4960	0.3635	4.1150	0.0002
Núcleo:poly(SO, 2)[, 2]	-0.2433	0.2979	0.3635	0.8200	0.4165
Núcleo:poly(VM, 2)[, 2]	-0.1645	0.2014	0.3635	0.5540	0.5821
Memória:SO	0.6906	-0.1151	0.0495	-2.3270	0.0243
Memória:VM	0.1087	0.0181	0.0495	0.3660	0.7158
Memória:poly(Núcleo, 2)[, 2]	0.3073	-0.3764	0.3635	-1.0350	0.3057
Memória:poly(SO, 2)[, 2]	0.3335	0.4084	0.3635	1.1230	0.2669
Memória:poly(VM, 2)[, 2]	-2.3812	-2.9164	0.3635	-8.0220	0.0000
SO:VM	1.4784	0.2464	0.0495	4.9810	0.0000
SO:poly(Núcleo, 2)[, 2]	0.0760	-0.0931	0.3635	-0.2560	0.7989
SO:poly(Memória, 2)[, 2]	-1.0808	-1.3238	0.3635	-3.6410	0.0007
SO:poly(VM, 2)[, 2]	-0.0022	0.0028	0.3635	0.0080	0.9940
VM:poly(Núcleo, 2)[, 2]	-0.2781	-0.3406	0.3635	-0.9370	0.3535
VM:poly(Memória, 2)[, 2]	0.5131	-0.6284	0.3635	-1.7280	0.0903
VM:poly(SO, 2)[, 2]	0.7871	0.9640	0.3635	2.6520	0.0108
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.8758	-7.8824	2.6715	-2.9510	0.0049
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.1686	1.5175	2.6715	0.5680	0.5727
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	-0.4178	-3.7603	2.6715	-1.4080	0.1657
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-0.7585	-6.8268	2.6715	-2.5550	0.0138
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	0.6668	-6.0015	2.6715	-2.2460	0.0293
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.0166	-0.1491	2.6715	-0.0560	0.9557

Tabela 70: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Read* após a transformação Johnson das respostas para a variável dependente Taxa média de IO

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	-0.4912	0.0546	0.0330	1.6550	0.1045
Núcleo	-0.1920	0.0261	0.0404	0.6470	0.5209
Memória	4.7694	0.6490	0.0404	16.0680	< 2e-16
SO	-4.1924	-0.5705	0.0404	-14.1240	< 2e-16
VM	2.0720	0.2820	0.0404	6.9800	0.0000
poly(Núcleo, 2)[, 2]	0.2052	-0.2052	0.2968	-0.6910	0.4927
poly(Memória, 2)[, 2]	-0.5524	0.5524	0.2968	1.8610	0.0689
poly(SO, 2)[, 2]	2.4634	-2.4634	0.2968	-8.2990	0.0000
poly(VM, 2)[, 2]	-0.4046	0.4046	0.2968	1.3630	0.1792
Núcleo:Memória	-0.6821	0.1137	0.0495	2.2980	0.0260
Núcleo:SO	0.0602	0.0100	0.0495	0.2030	0.8401
Núcleo:VM	0.3667	0.0611	0.0495	1.2350	0.2227
Núcleo:poly(Memória, 2)[, 2]	-1.2215	1.4960	0.3635	4.1150	0.0002
Núcleo:poly(SO, 2)[, 2]	-0.2433	0.2979	0.3635	0.8200	0.4165

Núcleo:poly(VM, 2)[, 2]	-0.1645	0.2014	0.3635	0.5540	0.5821
Memória:SO	0.6906	-0.1151	0.0495	-2.3270	0.0243
Memória:VM	0.1087	0.0181	0.0495	0.3660	0.7158
Memória:poly(Núcleo, 2)[, 2]	0.3073	-0.3764	0.3635	-1.0350	0.3057
Memória:poly(SO, 2)[, 2]	0.3335	0.4084	0.3635	1.1230	0.2669
Memória:poly(VM, 2)[, 2]	-2.3812	-2.9164	0.3635	-8.0220	0.0000
SO:VM	1.4784	0.2464	0.0495	4.9810	0.0000
SO:poly(Núcleo, 2)[, 2]	0.0760	-0.0931	0.3635	-0.2560	0.7989
SO:poly(Memória, 2)[, 2]	-1.0808	-1.3238	0.3635	-3.6410	0.0007
SO:poly(VM, 2)[, 2]	-0.0022	0.0028	0.3635	0.0080	0.9940
VM:poly(Núcleo, 2)[, 2]	-0.2781	-0.3406	0.3635	-0.9370	0.3535
VM:poly(Memória, 2)[, 2]	0.5131	-0.6284	0.3635	-1.7280	0.0903
VM:poly(SO, 2)[, 2]	0.7871	0.9640	0.3635	2.6520	0.0108
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.8758	-7.8824	2.6715	-2.9510	0.0049
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.1686	1.5175	2.6715	0.5680	0.5727
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	-0.4178	-3.7603	2.6715	-1.4080	0.1657
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-0.7585	-6.8268	2.6715	-2.5550	0.0138
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	0.6668	-6.0015	2.6715	-2.2460	0.0293
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.0166	-0.1491	2.6715	-0.0560	0.9557

Os coeficientes de determinação R^2 com 48 graus de liberdade, apresentado na Tabela 71.

Tabela 71: Coeficientes de determinação para o algoritmo *TestDFSIO Read* após a transformação de Johnson para cada variável resposta

Variáveis respostas	**R^2**
Garbage Collector	0.9235
CPU	0.9390
Memória Física	0.9654
Memória Virtual	0.9723
Heap	0.9592
Throughput	0.9399
Taxa média de IO	0.9399

Testes formais foram realizados (Tabela 72) e confirmaram que para todas as variáveis respostas, os resíduos estavam distribuídos de forma homogênea (teste de Breusch-Pagan, função *bpTest*). Já para o teste de distribuição normal (teste de Shapiro-Wilk, função *shapiro.test*), somente as variáveis respostas *Memória Física, Throughput e Taxa média de IO* os resíduos se aproximam de uma distribuição normal. Portanto, de acordo com Montgomery (2009), desvios de normalidade moderados podem ocorrer contanto que pontos discrepantes sejam investigados. Por fim, para o teste o teste de auto correlação negativa e positiva (teste de DurbinWatson, função durbinWatsonTest), a variável dependente *Heap* não pode-se confluir a dependência dos resíduos, as demais variáveis respostas não existe auto correlação dos resíduos.

Tabela 72: Testes formais de Breusch-Pagan, Shapiro-Wilk, DurbinWatson para o algoritmo *TestDFSIO Read* após a transformação de Johnson para cada variável resposta

Variável dependente	Breusch-Pagan valor-p	Shapiro-Wilk valor-p	DurbinWatson Dw
Garbage Collector	0.1391	0.02736	2.172
	Aceita H_0	**Rejeita H_0**	**Aceita H_0**
CPU	0.1881	4.547e-03	1.791
	Aceita H_0	**Rejeita H_0**	**Aceita H_0**
Memória física	0.3964	0.0549	2.013
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Memória Virtual	0.3373	8.231e-04	1.962
	Aceita H_0	**Rejeita H_0**	**Aceita H_0**
Heap	0.3165	1.346e-03	1.537
	Aceita H_0	**Rejeita H_0**	**Teste inconclusivo**
Throughput	0.1069	0.2437	1.792
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Taxa média de IO	0.1069	0.2437	1.792
	Aceita H_0	**Aceita H_0**	**Aceita H_0**

Como apresentado na Tabela 72, a hipótese de distribuição normal foi rejeitada pelo teste de Shapiro-Wilk para as variáveis respostas (*Garbage Collector, CPU, Memória Virtual e Heap*), porém a Tabela 73 mostra que os testes de Kolmogorov-Smirnov (Lilliefors, 1967) e Andereson-Darling (Razali, et al., 2011) confirmam que os resíduos aproximam de uma distribuição normal, aceitando a hipótese de Normalidade.

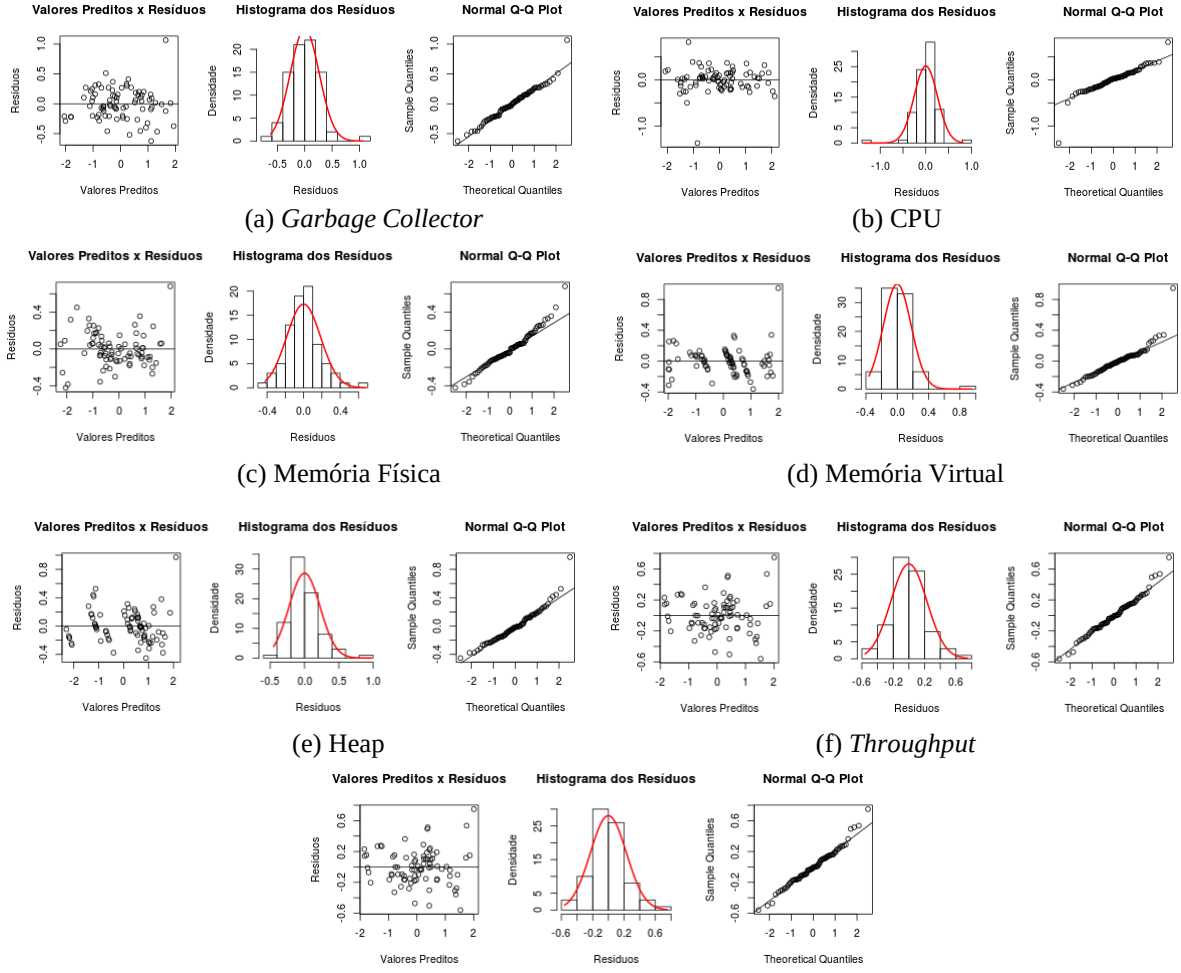
Tabela 73: Testes formais de homoscedasticidade para o algoritmo *TestDFSIO Read* após a transformação de Johnson (H_0: variância dos erros são iguais)

Variável dependente	Shapiro-Wilk valor-p	Kolmogorov-Smirnov valor-p	Anderson-Darling valor-p
GARBAGE COLLECTOR	0.02736	0.9495	0.4637
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
CPU	4.547e-03	0.3669	0.0534
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Memória física	0.0549	0.2541	0.0766
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Memória Virtual	8.231e-04	0.0723	0.0639
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Heap	1.346e-03	0.4294	0.6629
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Throughput	0.2437	0.8217	0.2948
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Taxa média de IO	0.2437	0.8217	0.2948
	Aceita H_0	**Aceita H_0**	**Aceita H_0**

A Figura 80 apresenta a verificação do modelo através da análise dos resíduos para cada variável resposta. O gráfico da esquerda para direita representa a relação entre os resíduos pelos valores preditos e não apresenta estrutura ou padrão. O histograma dos resíduos que apresenta

a distribuição da frequência dos resíduos apresentou aproximadamente simétrico em forma de sino. O gráfico à direita representa a probabilidade normal dos resíduos apresentando pontos próximos da reta.

Figura 80: Verificação dos resíduos: gráfico resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem de dois do fatorial 3^4 do algoritmo *TestDFSIO Read*, após a transformação de Johnson para cada variável resposta



(a) *Garbage Collector* (b) CPU

(c) Memória Física (d) Memória Virtual

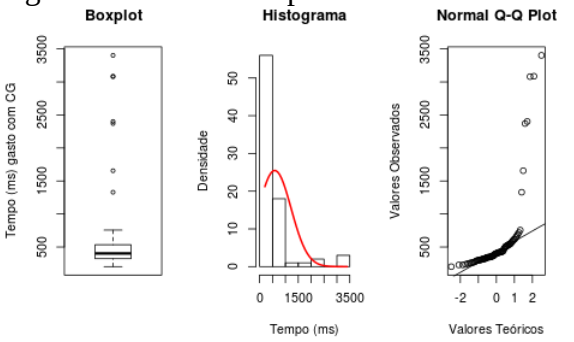
(e) Heap (f) *Throughput*

(g) Taxa média de IO

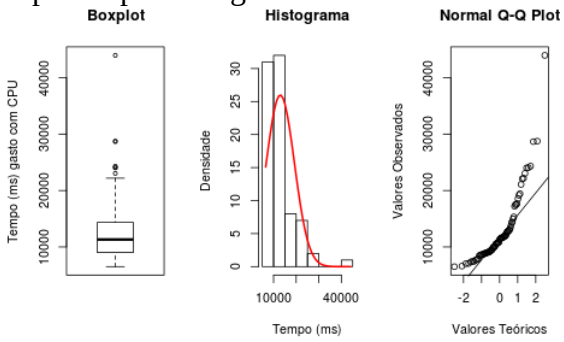
D.4 Algoritmo *TestDFSIO Write*

A Figura 81 apresenta da esquerda para a direita, informações exploratórias sobre as variáveis respostas (*Garbage Collector*, *CPU*, *memória física*, *memória virtual*, *throughput*, *taxa média de IO e heap e o tempo*) respectivamente, encontradas durante a execução do algoritmo *TestDFSIO Write* nos ambientes virtuais analisados. O primeiro gráfico de cada variável resposta apresenta o diagrama de caixa com as respectivas respostas.

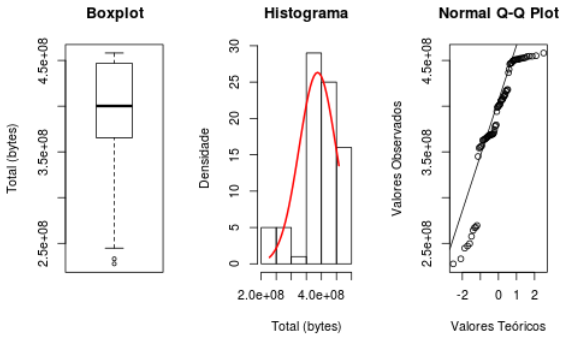
Figura 81: Gráficos exploratórios das variáveis respostas para o algoritmo *TestDFSIO Write*



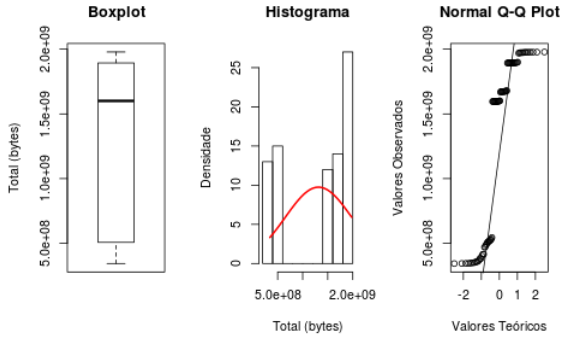
(a) *Garbage Collector*



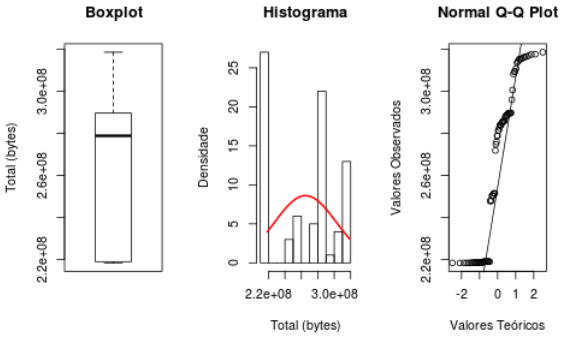
(b) CPU



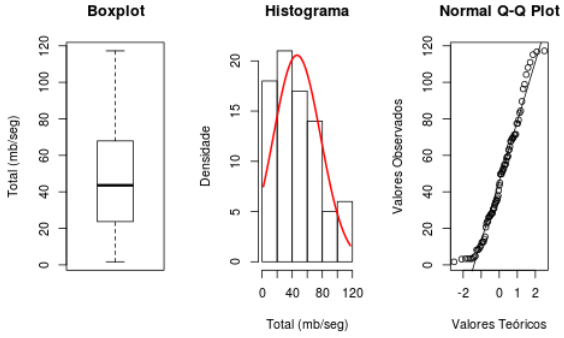
(c) Memória Física



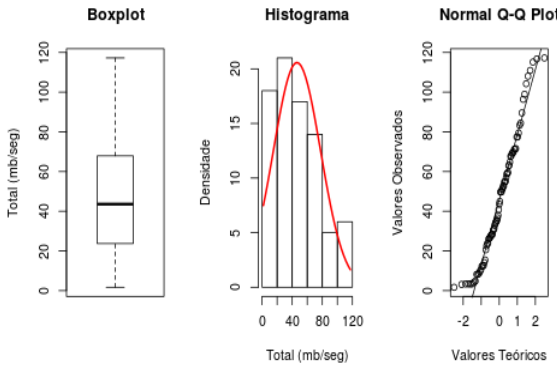
(d) Memória Virtual



(e) Heap



(f) Thoughput



(g) Taxa média de IO

Pode-se verificar que para esse algoritmo o uso de *Garbage Collector* (Figura 81a) foi muito baixo, ou seja, 75% dos experimentos utilizaram menos que 500 milissegundos o *Garbage Collector*. O uso da CPU (Figura 81b) na maioria dos ensaios (75%) ficou abaixo de 20000 milissegundos. Para as variáveis respostas: memória física (Figura 81c), virtual (Figura 81d) e heap (Figura 81e) representou um grande uso durante a execução do algoritmo. Para o *Throughput* (Figura 81f) e a taxa media de IO (Figura 81g), 75% dos experimentos a quantidade de megabytes por segundo foi abaixo de 80 mb/seg.

Os gráficos de normalidade da Figura 81 mostram que os dados coletados não se aproximam de uma distribuição normal (na sua maioria são assimétricos a direita) o que caracteriza a necessidade de se aplicar uma transformação nos dados. Para cada conjunto de dados de cada variável resposta foi aplicada então a transformação de Johnson (JOHNSON, 2002).

Para cada variável resposta o resultado da transformação de Johnson foi:

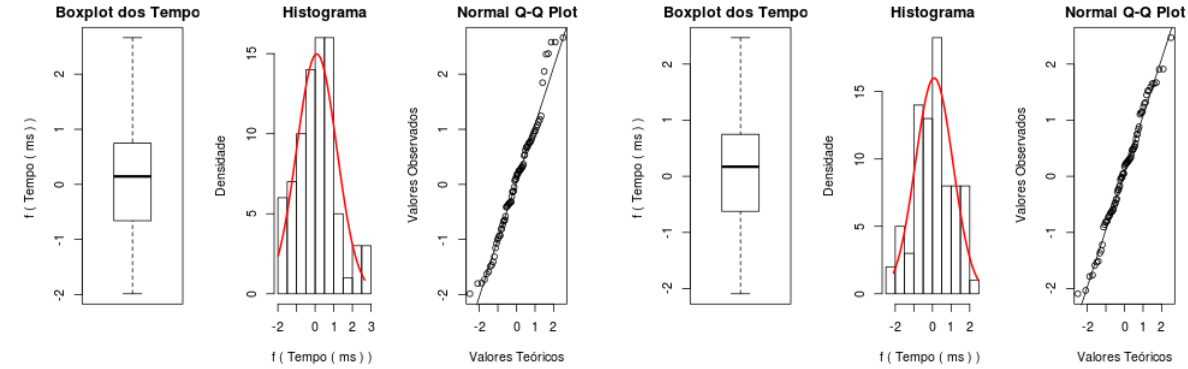
- *Garbage Collector*: família = SU, γ = -0.8896307, λ = 49.90694, ε = 306.9927 e η = 0.7367809;
- CPU: família = SU, γ = -1.476522, λ = 1615.93, ε = 7522.733 e η = 1.036158;
- Memória física: família = SB, γ = -1.169519, λ = 238597400, ε = 219756077 e η = 0.6564357;
- Memória virtual: família = SB, γ = -0.07011516, λ = 1635136135, ε = 343640174e η = 0.2678196;
- Heap: família = SB, γ = -0.1927686, λ = 131292154, ε = 191035216e η = 0.5103681;
- *Throughput*: família = SB, γ = 0. 7651067, λ = 183.445, ε = -19.41454 e η = 1.137075;
- Taxa media de IO: família = SB, γ = 0. 7651067, λ = 183.445, ε = -19.41454 e η = 1.137075;

A Figura 82 mostra os mesmos gráficos, porém após a transformação de Johnson. Os diagramas de caixas e os histogramas apresentam ser mais simétricos em relação aos apresentados na Figura 81. Por fim, pode-se observar que todos os gráficos de normalidade os dados aproximam de uma distribuição normal.

A Figura 83 mostra as mesmas informações, porém agrupadas através da combinação do fator sistema operacional pelo fator máquina virtual. As análises baseadas em *boxplot* são

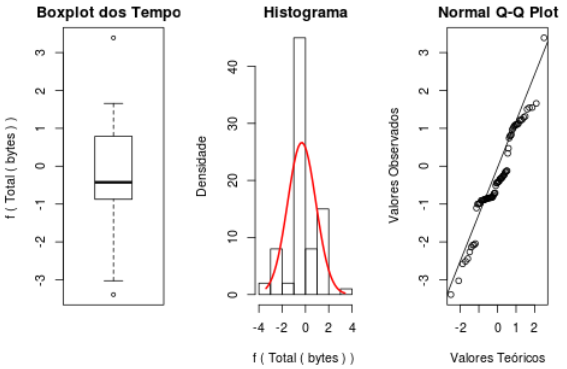
limitadas, podendo tirar conclusões somente quando os agrupamentos são óbvios. A Tabela 74 mostra o resultado da aplicação do teste *t* para demais conclusões.

Figura 82: Gráficos exploratórios das variáveis respostas para o algoritmo *TestDFSIO Write* após transformação de Johnson

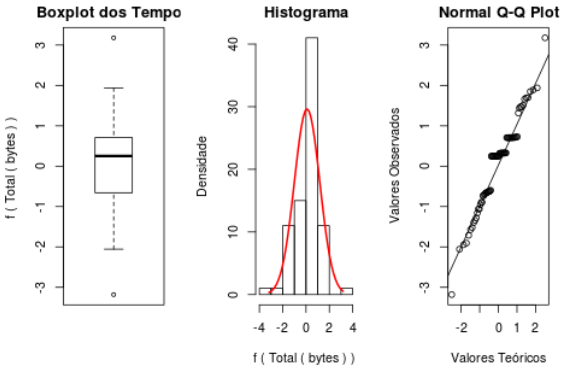


(a) GARBAGE COLLECTOR

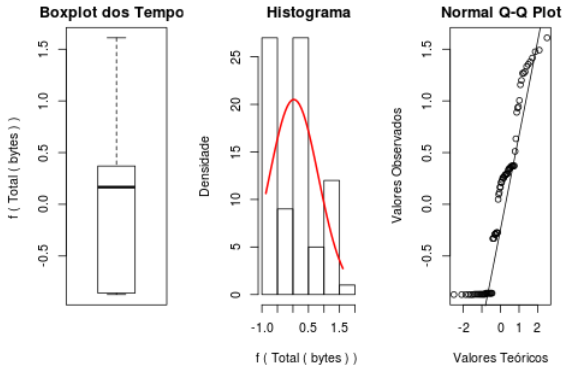
(b) CPU



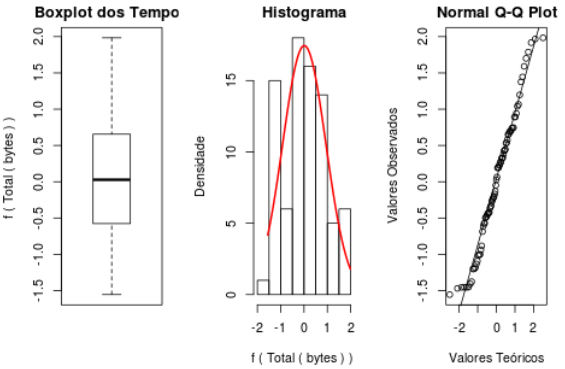
(c) Memória Física



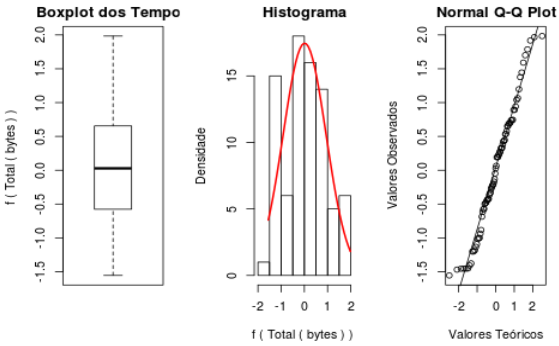
(d) Memória Virtual



(e) Heap



(f) Thoughput



(g) Taxa média de IO

Pode-se então verificar que existem evidências estatísticas que a variável resposta *Garbage Collector* (Figura 83a) com a combinação VM:SO (nível –, nível +) em média possui o maior tempo de gasto em milissegundos do *Garbage Collector* (Tabela 74). Em relação à utilização de CPU (Figura 83b) a combinação KVM:Windows (nível –, nível +) em média utiliza em maior tempo a CPU.

A quantidade de memória física (Figura 83c) também a combinação KVM:Windows (nível –, nível +) em média utiliza a menor quantidade de memória física em *bytes*. E também as outras combinações que o sistema operacional Windows está presente também utiliza a menor quantidade de memória física que os demais sistemas operacionais. Já em relação à utilização da memória virtual (Figura 83d) a máquina virtual não interfere substancialmente no desempenho de um algoritmo. A maior utilização de memória virtual está com o sistema operacional CentOS juntamente com a máquina virtual VMware. A menor utilização de memória virtual está com o sistema operacional Windows (Tabela 74).

Em relação a variável resposta *Heap* (Figura 83e) mostrou que não existem evidências estatísticas que as respostas sejam diferentes (Tabelas 74).

As variáveis respostas *Throughput* (Figura 83f) e taxa média de IO (Figura 83e) com base no teste *t* possui a mesma resposta, ou seja, a combinação KVM:Ubuntu (nível –, nível –) e KVM:Windows (nível –, nível +) em média possuem uma vazão de megabytes por segundo inferior as demais configurações e também uma taxa média de IO inferior às demais configurações. A maior vazão e a maior taxa média de IO está com a combinação VMware:Ubuntu (nível +, nível –) e VMware:CentOS (nível +, nível 0).

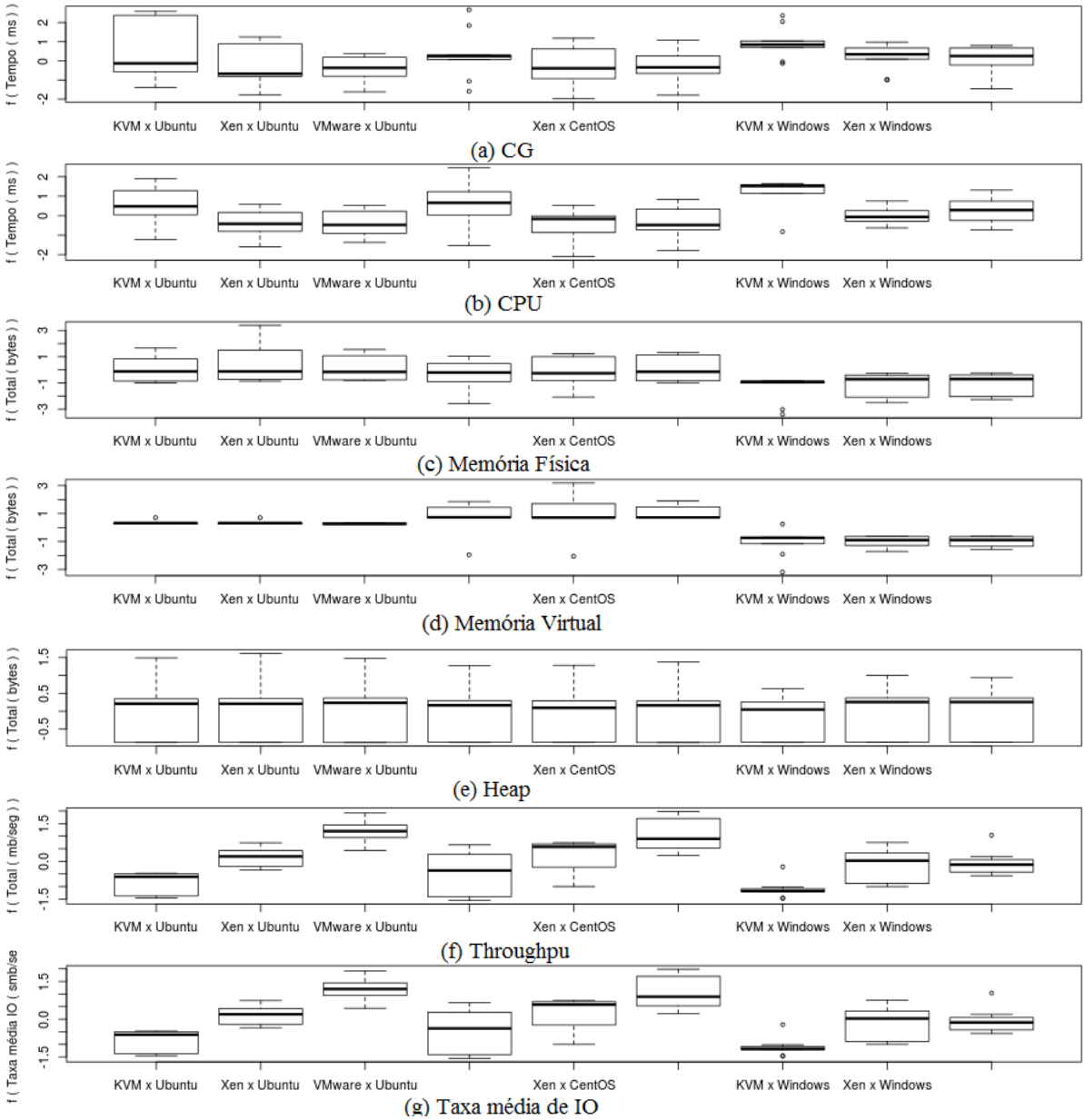
Tabela 74: Comparação entre os fatores após a transformação de Johnson: variáveis respostas agrupadas pela combinação do sistema operacional pela máquina virtual para o algoritmo *TestDFSIO Write*

Teste *t* aplicado para a variável dependente *Garbage Collector* (ms)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	0	0	0
Xen x Ubuntu	0	--	0	0	0	0	-1	0	0
VMware x Ubuntu	0	0	--	0	0	0	-1	0	0
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	-1	0	0
VMware x CentOS	0	0	0	0	0	--	-1	0	0
KVM x Windows	0	+1	+1	0	+1	+1	--	0	+1
Xen x Windows	0	0	0	0	0	0	0	--	0
VMware x Windows	0	0	0	0	0	0	-1	0	--
Teste *t* aplicado para a variável dependente CPU (ms)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	+1	+1	0	+1	0	0	0	0
Xen x Ubuntu	-1	--	0	-1	0	0	-1	0	0
VMware x Ubuntu	-1	0	--	-1	0	0	-1	0	0

KVM x CentOS	0	+1	+1	--	+1	0	0	0	0
Xen x CentOS	-1	0	0	-1	--	0	-1	0	-1
VMware x CentOS	0	0	0	0	0	--	-1	0	0
KVM x Windows	0	+1	+1	0	+1	+1	--	+1	+1
Xen x Windows	0	0	0	0	0	0	-1	--	0
VMware x Windows	0	0	0	0	+1	0	-1	0	--
Teste *t* aplicado para a variável dependente Memória Física (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	+1	+1	+1
Xen x Ubuntu	0	--	0	0	0	0	+1	+1	+1
VMware x Ubuntu	0	0	--	0	0	0	+1	+1	+1
KVM x CentOS	0	0	0	--	0	0	+1	0	0
Xen x CentOS	0	0	0	0	--	0	+1	+1	+1
VMware x CentOS	0	0	0	0	0	--	+1	+1	+1
KVM x Windows	-1	-1	-1	-1	-1	-1	--	0	0
Xen x Windows	-1	-1	-1	0	-1	-1	0	--	0
VMware x Windows	-1	-1	-1	0	-1	-1	0	0	--
Teste *t* aplicado para a variável dependente Memória Virtual (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	-1	+1	+1	+1
Xen x Ubuntu	0	--	0	0	0	-1	+1	+1	+1
VMware x Ubuntu	0	0	--	0	0	-1	+1	+1	+1
KVM x CentOS	0	0	0	--	0	0	+1	+1	+1
Xen x CentOS	0	0	0	0	--	0	+1	+1	+1
VMware x CentOS	+1	+1	+1	0	0	--	+1	+1	+1
KVM x Windows	-1	-1	-1	-1	-1	-1	--	0	0
Xen x Windows	-1	-1	-1	-1	-1	-1	0	--	0
VMware x Windows	-1	-1	-1	-1	-1	-1	0	0	--
Teste *t* aplicado para a variável dependente Heap (bytes)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	0	0	0	0	0	0	0	0
Xen x Ubuntu	0	--	0	0	0	0	0	0	0
VMware x Ubuntu	0	0	--	0	0	0	0	0	0
KVM x CentOS	0	0	0	--	0	0	0	0	0
Xen x CentOS	0	0	0	0	--	0	0	0	0
VMware x CentOS	0	0	0	0	0	--	0	0	0
KVM x Windows	0	0	0	0	0	0	--	0	0
Xen x Windows	0	0	0	0	0	0	0	--	0
VMware x Windows	0	0	0	0	0	0	0	0	--
Teste *t* aplicado para a variável dependente Throughput (mb/seg)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	-1	-1	0	-1	-1	0	-1	-1
Xen x Ubuntu	+1	--	-1	0	0	-1	+1	0	0
VMware x Ubuntu	+1	+1	--	+1	+1	0	+1	+1	+1
KVM x CentOS	0	0	-1	--	0	-1	0	0	0
Xen x CentOS	+1	0	-1	0	--	-1	+1	0	0
VMware x CentOS	+1	+1	0	+1	+1	--	+1	+1	+1
KVM x Windows	0	-1	-1	0	-1	-1	--	-1	-1
Xen x Windows	+1	0	-1	0	0	-1	+1	--	0
VMware x Windows	+1	0	-1	0	0	-1	+1	0	--
Teste *t* aplicado para a variável dependente Taxa média IO (mb/seg)									
Ambiente	**KVM x Ubuntu**	**Xen x Ubuntu**	**VMware x Ubuntu**	**KVM x CentOS**	**Xen x CentOS**	**VMware x CentOS**	**KVM x Windows**	**Xen x Windows**	**VMware x Windows**
KVM x Ubuntu	--	-1	-1	0	-1	-1	0	-1	-1
Xen x Ubuntu	+1	--	-1	0	0	-1	+1	0	0
VMware x Ubuntu	+1	+1	--	+1	+1	0	+1	+1	+1
KVM x CentOS	0	0	-1	--	0	-1	0	0	0
Xen x CentOS	+1	0	-1	0	--	-1	+1	0	0
VMware x CentOS	+1	+1	0	+1	+1	--	+1	+1	+1

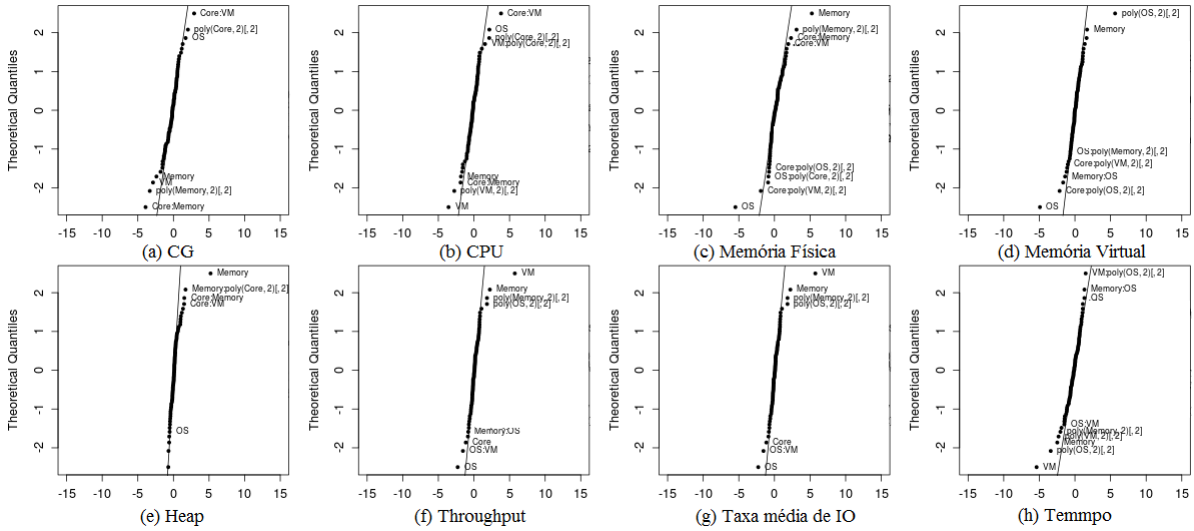
KVM x Windows	0	-1	-1	0	-1	-1	--	-1	-1
Xen x Windows	+1	0	-1	0	0	-1	+1	--	0
VMware x Windows	+1	0	-1	0	0	-1	+1	0	--

Figura 83: Diagrama de caixa (boxplot) das variáveis de respostas para o algoritmo *TestDFSIO Write*, após a transformação de Johnson: variáveis respostas agrupadas pela combinação do sistema operacional pela máquina virtual



A análise do fatorial 3^4 se iniciou com a projeção do gráfico de probabilidade normal dos efeitos estimados para a identificação dos efeitos significativos (Figura 84) para cada variável resposta. Pode-se verificar que em todos os gráficos de probabilidade normal os fatores SO e/ou VM foram estatisticamente significativos. Além disso, o fator Memoria foi bastante significativo em quase todos as variáveis respostas analisadas.

Figura 84: Gráfico da probabilidade normal dos efeitos estimados do fatorial 3^4 dos tempos para o algoritmo *TestDFSIO Write: Garbage Collector* (ms), CPU (ms), memória física (bytes), memória virtual (bytes), heap (bytes), throughput (mb/seg), taxa média de IO (mb/seg) e tempo (seg), respectivamente, após a transformação de Johnson.



As Tabelas 75, 76, 77, 78, 79, 80 e 81 apresentam a Tabela ANOVA para as variáveis dependentes (*Garbage Collector, CPU, Memória Física, Memoria Virtual, Heap, Throughput e Taxa média IO,* respectivamente) para o algoritmo *TestDFSIO Write.*

Tabela 75: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Write* após a transformação Johnson das respostas para a variável dependente *Garbage Collector*

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	-0.7626	0.0847	0.0802	1.0570	0.2957
Núcleo	0.5647	-0.0768	0.0982	-0.7830	0.4376
Memória	-2.3833	-0.3243	0.0982	-3.3040	0.0018
SO	1.7145	0.2333	0.0982	2.3770	0.0215
VM	-2.8603	-0.3892	0.0982	-3.9650	0.0002
poly(Núcleo, 2)[, 2]	2.0354	-2.0354	0.7213	-2.8220	0.0069
poly(Memória, 2)[, 2]	-3.3209	3.3209	0.7213	4.6040	0.0000
poly(SO, 2)[, 2]	-1.1108	1.1108	0.7213	1.5400	0.1302
poly(VM, 2)[, 2]	-1.5355	1.5355	0.7213	2.1290	0.0384
Núcleo:Memória	-3.9089	0.6515	0.1202	5.4190	0.0000
Núcleo:SO	0.5942	0.0990	0.1202	0.8240	0.4141
Núcleo:VM	2.9050	0.4842	0.1202	4.0270	0.0002

Núcleo:poly(Memória, 2)[, 2]	0.4010	-0.4912	0.8835	-0.5560	0.5808
Núcleo:poly(SO, 2)[, 2]	-1.1778	1.4426	0.8835	1.6330	0.1090
Núcleo:poly(VM, 2)[, 2]	-0.5247	0.6427	0.8835	0.7270	0.4705
Memória:SO	0.4182	-0.0697	0.1202	-0.5800	0.5648
Memória:VM	0.0223	0.0037	0.1202	0.0310	0.9755
Memória:poly(Núcleo, 2)[, 2]	-1.5472	1.8950	0.8835	2.1450	0.0370
Memória:poly(SO, 2)[, 2]	0.6684	0.8186	0.8835	0.9270	0.3588
Memória:poly(VM, 2)[, 2]	-0.1722	-0.2109	0.8835	-0.2390	0.8124
SO:VM	0.1332	0.0222	0.1202	0.1850	0.8543
SO:poly(Núcleo, 2)[, 2]	0.1279	-0.1566	0.8835	-0.1770	0.8600
SO:poly(Memória, 2)[, 2]	-1.1291	-1.3829	0.8835	-1.5650	0.1241
SO:poly(VM, 2)[, 2]	0.0423	-0.0518	0.8835	-0.0590	0.9535
VM:poly(Núcleo, 2)[, 2]	0.6393	0.7830	0.8835	0.8860	0.3799
VM:poly(Memória, 2)[, 2]	0.5530	-0.6772	0.8835	-0.7670	0.4471
VM:poly(SO, 2)[, 2]	-0.5279	-0.6466	0.8835	-0.7320	0.4678
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.6802	6.1218	6.4921	0.9430	0.3504
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.1487	1.3385	6.4921	0.2060	0.8375
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	-0.3704	-3.3335	6.4921	-0.5130	0.6100
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-0.4061	-3.6551	6.4921	-0.5630	0.5761
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.1590	1.4311	6.4921	0.2200	0.8265
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.0555	-0.4992	6.4921	-0.0770	0.9390

Tabela 76: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Write* após a transformação Johnson das respostas para a variável dependente CPU

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	-0.8083	0.0898	0.0671	1.3380	0.1873
Núcleo	-0.0969	0.0132	0.0822	0.1600	0.8733
Memória	-1.8222	-0.2480	0.0822	-3.0160	0.0041
SO	2.1633	0.2944	0.0822	3.5810	0.0008
VM	-3.5632	-0.4849	0.0822	-5.8980	0.0000
poly(Núcleo, 2)[, 2]	2.1525	-2.1525	0.6042	-3.5630	0.0008
poly(Memória, 2)[, 2]	-1.5552	1.5552	0.6042	2.5740	0.0132
poly(SO, 2)[, 2]	-1.2654	1.2654	0.6042	2.0940	0.0415
poly(VM, 2)[, 2]	-2.7578	2.7578	0.6042	4.5640	0.0000
Núcleo:Memória	-1.8734	0.3122	0.1007	3.1010	0.0032
Núcleo:SO	0.5974	0.0996	0.1007	0.9890	0.3277
Núcleo:VM	3.8086	0.6348	0.1007	6.3040	0.0000
Núcleo:poly(Memória, 2)[, 2]	-0.1753	0.2146	0.7400	0.2900	0.7730
Núcleo:poly(SO, 2)[, 2]	-0.2488	0.3047	0.7400	0.4120	0.6824
Núcleo:poly(VM, 2)[, 2]	-1.0259	1.2565	0.7400	1.6980	0.0960
Memória:SO	0.6744	-0.1124	0.1007	-1.1160	0.2699
Memória:VM	-0.2586	-0.0431	0.1007	-0.4280	0.6706
Memória:poly(Núcleo, 2)[, 2]	-0.8355	1.0233	0.7400	1.3830	0.1731
Memória:poly(SO, 2)[, 2]	-0.1034	-0.1266	0.7400	-0.1710	0.8648
Memória:poly(VM, 2)[, 2]	-0.7240	-0.8867	0.7400	-1.1980	0.2367
SO:VM	0.0083	0.0014	0.1007	0.0140	0.9890
SO:poly(Núcleo, 2)[, 2]	-1.0108	1.2380	0.7400	1.6730	0.1008
SO:poly(Memória, 2)[, 2]	-0.4790	-0.5866	0.7400	-0.7930	0.4318
SO:poly(VM, 2)[, 2]	-0.4523	0.5540	0.7400	0.7490	0.4577
VM:poly(Núcleo, 2)[, 2]	1.5576	1.9076	0.7400	2.5780	0.0131
VM:poly(Memória, 2)[, 2]	0.2309	-0.2828	0.7400	-0.3820	0.7040
VM:poly(SO, 2)[, 2]	0.1351	0.1655	0.7400	0.2240	0.8239
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.4156	3.7404	5.4376	0.6880	0.4948
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.7233	6.5101	5.4376	1.1970	0.2371
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.4630	4.1672	5.4376	0.7660	0.4472

poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-0.8406	-7.5656	5.4376	-1.3910	0.1705
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.1007	0.9062	5.4376	0.1670	0.8683
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.1129	-1.0158	5.4376	-0.1870	0.8526

Tabela 77: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Write* após a transformação Johnson das respostas para a variável dependente Memória Física

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	2.8587	-0.3176	0.0702	-4.5240	0.0000
Núcleo	0.9704	-0.1321	0.0860	-1.5360	0.1311
Memória	5.2545	0.7150	0.0860	8.3160	0.0000
SO	-5.4819	-0.7460	0.0860	-8.6760	0.0000
VM	1.1174	0.1521	0.0860	1.7680	0.0833
poly(Núcleo, 2)[, 2]	1.1328	-1.1328	0.6318	-1.7930	0.0793
poly(Memória, 2)[, 2]	3.0986	-3.0986	0.6318	-4.9040	0.0000
poly(SO, 2)[, 2]	1.6771	-1.6771	0.6318	-2.6540	0.0107
poly(VM, 2)[, 2]	0.7377	-0.7377	0.6318	-1.1680	0.2488
Núcleo:Memória	2.3268	-0.3878	0.1053	-3.6830	0.0006
Núcleo:SO	1.6083	0.2681	0.1053	2.5450	0.0142
Núcleo:VM	1.9422	0.3237	0.1053	3.0740	0.0035
Núcleo:poly(Memória, 2)[, 2]	0.4403	-0.5393	0.7739	-0.6970	0.4892
Núcleo:poly(SO, 2)[, 2]	-0.7496	0.9181	0.7739	1.1860	0.2413
Núcleo:poly(VM, 2)[, 2]	-1.9139	2.3441	0.7739	3.0290	0.0039
Memória:SO	-0.0675	0.0113	0.1053	0.1070	0.9154
Memória:VM	0.7840	0.1307	0.1053	1.2410	0.2207
Memória:poly(Núcleo, 2)[, 2]	1.7037	-2.0866	0.7739	-2.6960	0.0096
Memória:poly(SO, 2)[, 2]	1.1937	1.4620	0.7739	1.8890	0.0649
Memória:poly(VM, 2)[, 2]	-0.2043	-0.2502	0.7739	-0.3230	0.7479
SO:VM	0.4363	0.0727	0.1053	0.6910	0.4932
SO:poly(Núcleo, 2)[, 2]	-0.8369	1.0250	0.7739	1.3250	0.1916
SO:poly(Memória, 2)[, 2]	-0.5446	-0.6670	0.7739	-0.8620	0.3930
SO:poly(VM, 2)[, 2]	-0.4312	0.5282	0.7739	0.6830	0.4982
VM:poly(Núcleo, 2)[, 2]	-0.5566	-0.6817	0.7739	-0.8810	0.3827
VM:poly(Memória, 2)[, 2]	-0.0082	0.0100	0.7739	0.0130	0.9897
VM:poly(SO, 2)[, 2]	-0.4198	-0.5141	0.7739	-0.6640	0.5096
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.4286	3.8578	5.6866	0.6780	0.5008
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.8751	7.8759	5.6866	1.3850	0.1725
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.0560	0.5044	5.6866	0.0890	0.9297
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.1906	1.7158	5.6866	0.3020	0.7642
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.1796	1.6162	5.6866	0.2840	0.7775
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.4492	-4.0428	5.6866	-0.7110	0.4806

Tabela 78: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Write* após a transformação Johnson das respostas para a variável dependente Memória Virtual

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	-0.8186	0.0910	0.0712	1.2770	0.2077
Núcleo	0.6445	-0.0877	0.0872	-1.0050	0.3197
Memória	1.7079	0.2324	0.0872	2.6640	0.0105
SO	-4.9148	-0.6688	0.0872	-7.6670	0.0000
VM	0.4683	0.0637	0.0872	0.7310	0.4686
poly(Núcleo, 2)[, 2]	0.3928	-0.3928	0.6410	-0.6130	0.5429
poly(Memória, 2)[, 2]	1.2125	-1.2125	0.6410	-1.8920	0.0646
poly(SO, 2)[, 2]	5.6405	-5.6405	0.6410	-8.8000	0.0000
poly(VM, 2)[, 2]	0.1582	-0.1582	0.6410	-0.2470	0.8062

Núcleo:Memória	1.1495	-0.1916	0.1068	-1.7930	0.0792
Núcleo:SO	0.4637	0.0773	0.1068	0.7230	0.4730
Núcleo:VM	0.9711	0.1618	0.1068	1.5150	0.1364
Núcleo:poly(Memória, 2)[, 2]	0.2625	-0.3215	0.7851	-0.4100	0.6839
Núcleo:poly(SO, 2)[, 2]	-2.1555	2.6399	0.7851	3.3630	0.0015
Núcleo:poly(VM, 2)[, 2]	-0.9975	1.2217	0.7851	1.5560	0.1262
Memória:SO	-1.3863	0.2311	0.1068	2.1630	0.0356
Memória:VM	-0.0864	-0.0144	0.1068	-0.1350	0.8933
Memória:poly(Núcleo, 2)[, 2]	0.6758	-0.8277	0.7851	-1.0540	0.2970
Memória:poly(SO, 2)[, 2]	0.7748	0.9489	0.7851	1.2090	0.2327
Memória:poly(VM, 2)[, 2]	0.2000	0.2449	0.7851	0.3120	0.7564
SO:VM	0.2435	0.0406	0.1068	0.3800	0.7057
SO:poly(Núcleo, 2)[, 2]	-0.2093	0.2563	0.7851	0.3260	0.7455
SO:poly(Memória, 2)[, 2]	-0.6402	-0.7841	0.7851	-0.9990	0.3229
SO:poly(VM, 2)[, 2]	0.0121	-0.0148	0.7851	-0.0190	0.9850
VM:poly(Núcleo, 2)[, 2]	-0.2674	-0.3275	0.7851	-0.4170	0.6784
VM:poly(Memória, 2)[, 2]	-0.2972	0.3640	0.7851	0.4640	0.6450
VM:poly(SO, 2)[, 2]	-0.5071	-0.6210	0.7851	-0.7910	0.4328
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	-0.4471	4.0244	5.7690	0.6980	0.4888
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	1.0795	9.7159	5.7690	1.6840	0.0986
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.2098	1.8883	5.7690	0.3270	0.7449
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.7622	6.8600	5.7690	1.1890	0.2403
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.0709	0.6384	5.7690	0.1110	0.9123
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	0.0458	0.4125	5.7690	0.0720	0.9433

Tabela 79: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Write* após a transformação Johnson das respostas para a variável dependente Heap

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	-0.2674	0.0297	0.0486	0.6120	0.5437
Núcleo	0.1786	-0.0243	0.0595	-0.4090	0.6847
Memória	5.2154	0.7097	0.0595	11.9270	0.0000
SO	-0.5313	-0.0723	0.0595	-1.2150	0.2303
VM	0.1858	0.0253	0.0595	0.4250	0.6728
poly(Núcleo, 2)[, 2]	1.1874	-1.1874	0.4373	-2.7160	0.0092
poly(Memória, 2)[, 2]	1.0485	-1.0485	0.4373	-2.3980	0.0204
poly(SO, 2)[, 2]	0.0454	-0.0454	0.4373	-0.1040	0.9177
poly(VM, 2)[, 2]	0.0972	-0.0972	0.4373	-0.2220	0.8251
Núcleo:Memória	1.5274	-0.2546	0.0729	-3.4930	0.0010
Núcleo:SO	0.6701	0.1117	0.0729	1.5330	0.1320
Núcleo:VM	1.4997	0.2500	0.0729	3.4300	0.0013
Núcleo:poly(Memória, 2)[, 2]	0.7604	-0.9313	0.5355	-1.7390	0.0884
Núcleo:poly(SO, 2)[, 2]	-0.3244	0.3973	0.5355	0.7420	0.4618
Núcleo:poly(VM, 2)[, 2]	-0.4570	0.5597	0.5355	1.0450	0.3012
Memória:SO	0.3431	-0.0572	0.0729	-0.7850	0.4365
Memória:VM	0.9617	0.1603	0.0729	2.1990	0.0327
Memória:poly(Núcleo, 2)[, 2]	1.7033	-2.0861	0.5355	-3.8950	0.0003
Memória:poly(SO, 2)[, 2]	-0.0202	-0.0247	0.5355	-0.0460	0.9634
Memória:poly(VM, 2)[, 2]	0.0719	0.0880	0.5355	0.1640	0.8701
SO:VM	0.2260	0.0377	0.0729	0.5170	0.6076
SO:poly(Núcleo, 2)[, 2]	-0.2925	0.3583	0.5355	0.6690	0.5067
SO:poly(Memória, 2)[, 2]	-0.1144	-0.1401	0.5355	-0.2620	0.7947
SO:poly(VM, 2)[, 2]	0.1291	-0.1581	0.5355	-0.2950	0.7691
VM:poly(Núcleo, 2)[, 2]	-0.0512	-0.0627	0.5355	-0.1170	0.9072
VM:poly(Memória, 2)[, 2]	-0.4068	0.4983	0.5355	0.9300	0.3568
VM:poly(SO, 2)[, 2]	0.0719	0.0881	0.5355	0.1650	0.8700

poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.3284	-2.9557	3.9354	-0.7510	0.4563
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.0374	0.3367	3.9354	0.0860	0.9322
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.0811	0.7296	3.9354	0.1850	0.8537
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	-0.0745	-0.6703	3.9354	-0.1700	0.8655
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.0776	0.6983	3.9354	0.1770	0.8599
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.1401	-1.2607	3.9354	-0.3200	0.7501

Tabela 80: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Write* após a transformação Johnson das respostas para a variável dependente *Throughput*

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	-0.1062	0.0118	0.0475	0.2480	0.8050
Núcleo	-1.1400	0.1551	0.0582	2.6650	0.0105
Memória	2.2349	0.3041	0.0582	5.2250	0.0000
SO	-2.2762	-0.3097	0.0582	-5.3210	0.0000
VM	5.7413	0.7813	0.0582	13.4220	< 2e-16
poly(Núcleo, 2)[, 2]	0.4488	-0.4488	0.4278	-1.0490	0.2993
poly(Memória, 2)[, 2]	1.8352	-1.8352	0.4278	-4.2900	0.0001
poly(SO, 2)[, 2]	1.8307	-1.8307	0.4278	-4.2800	0.0001
poly(VM, 2)[, 2]	0.4893	-0.4893	0.4278	-1.1440	0.2583
Núcleo:Memória	0.7865	-0.1311	0.0713	-1.8390	0.0722
Núcleo:SO	0.2478	0.0413	0.0713	0.5790	0.5650
Núcleo:VM	0.6718	0.1120	0.0713	1.5710	0.1228
Núcleo:poly(Memória, 2)[, 2]	0.2896	-0.3547	0.5239	-0.6770	0.5017
Núcleo:poly(SO, 2)[, 2]	-0.3828	0.4688	0.5239	0.8950	0.3753
Núcleo:poly(VM, 2)[, 2]	-0.5789	0.7090	0.5239	1.3530	0.1823
Memória:SO	-0.7880	0.1313	0.0713	1.8420	0.0716
Memória:VM	0.1092	0.0182	0.0713	0.2550	0.7996
Memória:poly(Núcleo, 2)[, 2]	0.8528	-1.0445	0.5239	-1.9940	0.0519
Memória:poly(SO, 2)[, 2]	-0.1412	-0.1729	0.5239	-0.3300	0.7428
Memória:poly(VM, 2)[, 2]	0.1376	0.1686	0.5239	0.3220	0.7491
SO:VM	-1.5431	-0.2572	0.0713	-3.6080	0.0007
SO:poly(Núcleo, 2)[, 2]	-0.4299	0.5266	0.5239	1.0050	0.3199
SO:poly(Memória, 2)[, 2]	0.3805	0.4661	0.5239	0.8900	0.3781
SO:poly(VM, 2)[, 2]	0.8053	-0.9863	0.5239	-1.8830	0.0658
VM:poly(Núcleo, 2)[, 2]	-0.7065	-0.8653	0.5239	-1.6520	0.1051
VM:poly(Memória, 2)[, 2]	0.1633	-0.2000	0.5239	-0.3820	0.7043
VM:poly(SO, 2)[, 2]	-0.1160	-0.1421	0.5239	-0.2710	0.7874
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.0892	-0.8030	3.8497	-0.2090	0.8356
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.1319	1.1874	3.8497	0.3080	0.7591
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.1786	1.6074	3.8497	0.4180	0.6781
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.7368	6.6312	3.8497	1.7230	0.0914
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.2920	2.6277	3.8497	0.6830	0.4982
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.6550	-5.8947	3.8497	-1.5310	0.1323

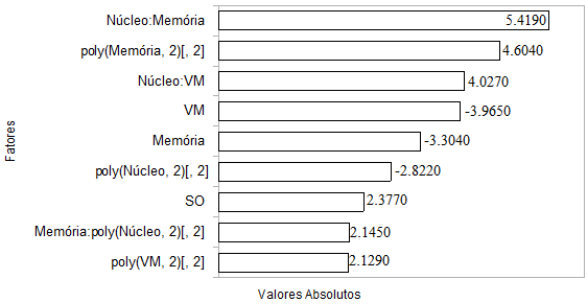
Tabela 81: Modelo para o fatorial 3^4 para o algoritmo *TestDFSIO Write* após a transformação Johnson das respostas para a variável dependente Taxa média de IO

	Efeito	**Coeficiente**	**E. Padrão**	**Valor t**	**Valor-p**
(Interseção)	-0.1062	0.0118	0.0475	0.2480	0.8050
Núcleo	-1.1400	0.1551	0.0582	2.6650	0.0105
Memória	2.2349	0.3041	0.0582	5.2250	0.0000
SO	-2.2762	-0.3097	0.0582	-5.3210	0.0000
VM	5.7413	0.7813	0.0582	13.4220	< 2e-16
poly(Núcleo, 2)[, 2]	0.4488	-0.4488	0.4278	-1.0490	0.2993

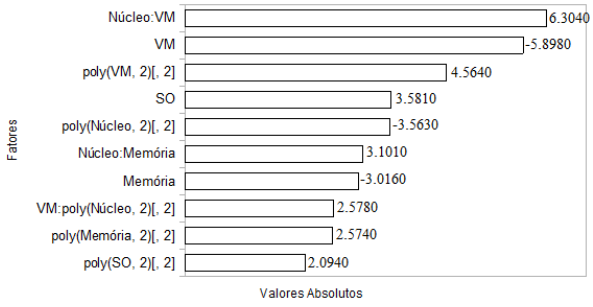
poly(Memória, 2)[, 2]	1.8352	-1.8352	0.4278	-4.2900	0.0001
poly(SO, 2)[, 2]	1.8307	-1.8307	0.4278	-4.2800	0.0001
poly(VM, 2)[, 2]	0.4893	-0.4893	0.4278	-1.1440	0.2583
Núcleo:Memória	0.7865	-0.1311	0.0713	-1.8390	0.0722
Núcleo:SO	0.2478	0.0413	0.0713	0.5790	0.5650
Núcleo:VM	0.6718	0.1120	0.0713	1.5710	0.1228
Núcleo:poly(Memória, 2)[, 2]	0.2896	-0.3547	0.5239	-0.6770	0.5017
Núcleo:poly(SO, 2)[, 2]	-0.3828	0.4688	0.5239	0.8950	0.3753
Núcleo:poly(VM, 2)[, 2]	-0.5789	0.7090	0.5239	1.3530	0.1823
Memória:SO	-0.7880	0.1313	0.0713	1.8420	0.0716
Memória:VM	0.1092	0.0182	0.0713	0.2550	0.7996
Memória:poly(Núcleo, 2)[, 2]	0.8528	-1.0445	0.5239	-1.9940	0.0519
Memória:poly(SO, 2)[, 2]	-0.1412	-0.1729	0.5239	-0.3300	0.7428
Memória:poly(VM, 2)[, 2]	0.1376	0.1686	0.5239	0.3220	0.7491
SO:VM	-1.5431	-0.2572	0.0713	-3.6080	0.0007
SO:poly(Núcleo, 2)[, 2]	-0.4299	0.5266	0.5239	1.0050	0.3199
SO:poly(Memória, 2)[, 2]	0.3805	0.4661	0.5239	0.8900	0.3781
SO:poly(VM, 2)[, 2]	0.8053	-0.9863	0.5239	-1.8830	0.0658
VM:poly(Núcleo, 2)[, 2]	-0.7065	-0.8653	0.5239	-1.6520	0.1051
VM:poly(Memória, 2)[, 2]	0.1633	-0.2000	0.5239	-0.3820	0.7043
VM:poly(SO, 2)[, 2]	-0.1160	-0.1421	0.5239	-0.2710	0.7874
poly(Núcleo, 2)[, 2]:poly(Memória, 2)[, 2]	0.0892	-0.8030	3.8497	-0.2090	0.8356
poly(Núcleo, 2)[, 2]:poly(SO, 2)[, 2]	0.1319	1.1874	3.8497	0.3080	0.7591
poly(Núcleo, 2)[, 2]:poly(VM, 2)[, 2]	0.1786	1.6074	3.8497	0.4180	0.6781
poly(Memória, 2)[, 2]:poly(SO, 2)[, 2]	0.7368	6.6312	3.8497	1.7230	0.0914
poly(Memória, 2)[, 2]:poly(VM, 2)[, 2]	-0.2920	2.6277	3.8497	0.6830	0.4982
poly(SO, 2)[, 2]:poly(VM, 2)[, 2]	-0.6550	-5.8947	3.8497	-1.5310	0.1323

A Figura 85 mostra o gráfico de Pareto que apresenta os valores absolutos estimados de cada efeito em cada variável resposta para o algoritmo *TestDFSIO Write*. Pode-se verificar que em todos os gráficos (Figura 85) a memória foi o efeito significativo. Já os gráficos de Pareto para a variável resposta *Memória Virtual* (Figura 85d) mostrou que o fator VM não foi estatisticamente significativo igual aos algoritmos *Pi*, *WordCount* e *TestDFSIO Read*.

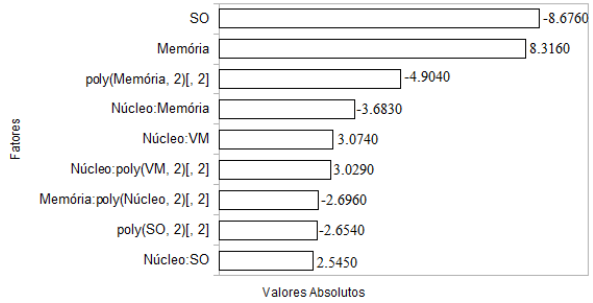
Figura 85: Gráfico de Pareto com os efeitos estimados em valor percentual absoluto para o tempo de execução para o algoritmo *Pi*: *Garbage Collector* (ms), CPU (ms), memória física (bytes), memória virtual (bytes), heap (bytes), throughput (mb/seg) e taxa média de IO (mb/seg), respectivamente para o algoritmo *TestDFSIO Write*, após a transformação de Johnson.



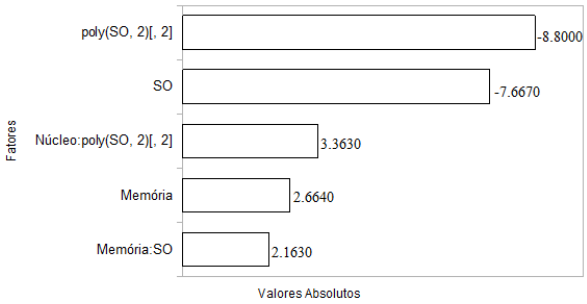
(a) *Garbage Collector*



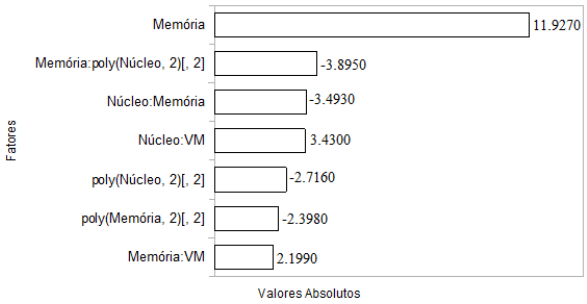
(b) CPU



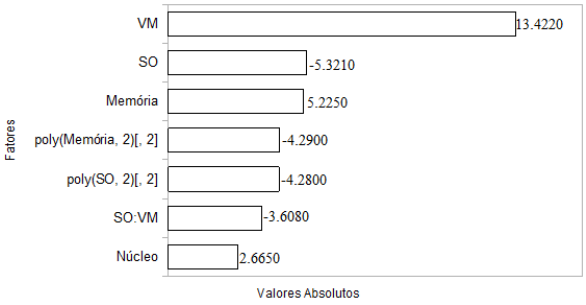
(c) Memória Física



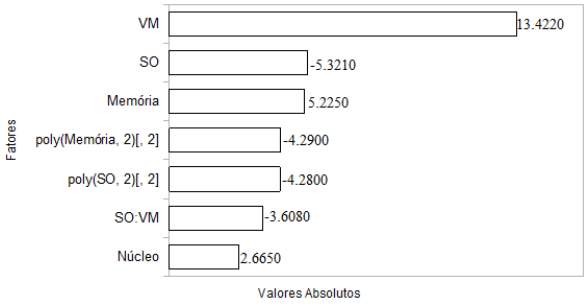
(d) Memória Virtual



(e) Heap



(f) Throughput



(g) Taxa média de IO

Os coeficientes de determinação R^2 com 48 graus de liberdade, apresentado na Tabela 82.

Tabela 82: Coeficientes de determinação para o algoritmo *TestDFSIO Write* após a transformação de Johnson para cada variável resposta

Variáveis respostas	**R^2**
Garbage Collector	0.7312
CPU	0.7846
Memória Física	0.8369
Memória Virtual	0.7925
Heap	0.8718
Throughput	0.8718
Taxa média de IO	0.8718

Testes formais foram realizados (Tabela 83) e confirmaram que para as variáveis respostas *Garbage Collector e Memória Física*, os resíduos estavam distribuídos de forma homogênea (teste de Breusch-Pagan, função *bpTest*). Já para o teste de distribuição normal

(teste de Shapiro-Wilk, função *shapiro.test*), somente a variável resposta *Memória Virtual* os resíduos não se aproximam de uma distribuição normal. Portanto, de acordo com Montgomery (2009), desvios de normalidade moderados podem ocorrer contanto que pontos discrepantes sejam investigados. Por fim, para o teste de auto correlação negativa e positiva (teste de DurbinWatson, função durbinWatsonTest), somente a variável resposta *Heap* não pode-se confluir a dependência dos resíduos, as demais variáveis respostas não existe auto correlação dos resíduos.

Como apresentado na Tabela 83, a hipótese de distribuição normal foi rejeitada pelo teste de Shapiro-Wilk para a variável resposta *Memória Virtual*, a Tabela 84 mostra que os testes de Kolmogorov-Smirnov (Lilliefors, 1967) e Andereson-Darling (Razali, et al., 2011) confirmam que os resíduos aproximam de uma distribuição normal, aceitando a hipótese de Normalidade.

Tabela 83: Testes formais de Breusch-Pagan, Shapiro-Wilk, DurbinWatson para o algoritmo *TestDFSIO Write* após a transformação de Johnson para cada variável resposta

Variável dependente	**Breusch-Pagan**	**Shapiro-Wilk**	**DurbinWatson**
	valor-p	**valor-p**	**Dw**
Garbage Collector	0.09976	0.671	2.627
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
CPU	**0.01395**	0.2686	2.303
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Memória física	0.05014	0.9452	1.919
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Memória Virtual	**0.005051**	8.301e-06	2.021
	Rejeita H_0	**Rejeita H_0**	**Aceita H_0**
Heap	**0.005674**	0.2572	1.670
	Rejeita H_0	**Aceita H_0**	**Teste inconclusivo**
Throughput	**0.006608**	0.6035	1.795
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Taxa média de IO	**0.006608**	0.6035	1.795
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Tempo	0.05334	0.999	1.909
	Aceita H_0	**Aceita H_0**	**Aceita H_0**

Por fim, também como apresentado na Tabela 83, a hipótese de homoscedasticidade foi rejeitada pelo teste de Breusch-Pagan para as variáveis respostas (*Heap e Tempo*), porém a Tabela 85 mostra que os testes de *Goldfeld-Quandt* (Goldfeld, et al., 1965) e *Harrison-McCabe* (Harrison, et al., 1979) confirmam que os resíduos aproximam de uma distribuição normal, aceitando a hipótese de Homoscedasticidade.

Tabela 84: Testes formais de homoscedasticidade para o algoritmo *TestDFSIO Write* após a transformação de Johnson (H_0: variância dos erros são iguais)

Variável dependente	Shapiro-Wilk valor-p	Kolmogorov-Smirnov valor-p	Anderson-Darling valor-p
Garbage Collector	0.671	0.7452	0.5489
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
CPU	0.2686	0.7935	0.4376
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Memória física	0.9452	0.8902	0.7272
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Memória Virtual	8.301e-06	0.09265	0.08063
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Heap	0.2572	0.4377	0.1147
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Throughput	0.6035	0.9956	0.8320
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Taxa média de IO	0.6035	0.9956	0.8320
	Aceita H_0	**Aceita H_0**	**Aceita H_0**

Tabela 85: Testes formais de homoscedasticidade para o algoritmo *TestDFSIO Write* após a transformação de Johnson (H_0: variância dos erros são iguais)

Variável dependente	Breusch-Pagan valor-p	Goldfeld-Quandt valor-p	Harrison-McCabe valor-p
GARBAGE	0.09976	0.5206	0.277
COLLECTOR	**Aceita H_0**	**Aceita H_0**	**Aceita H_0**
CPU	**0.01395**	0.8963	0.981
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Memória física	0.05014	0.4119	0.227
	Aceita H_0	**Aceita H_0**	**Aceita H_0**
Memória Virtual	**0.005051**	0.4446	0.147
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Heap	**0.005674**	0.05466	0.055
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Throughput	**0.006608**	0.6558	0.508
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**
Taxa média de IO	**0.006608**	0.6558	0.508
	Rejeita H_0	**Aceita H_0**	**Aceita H_0**

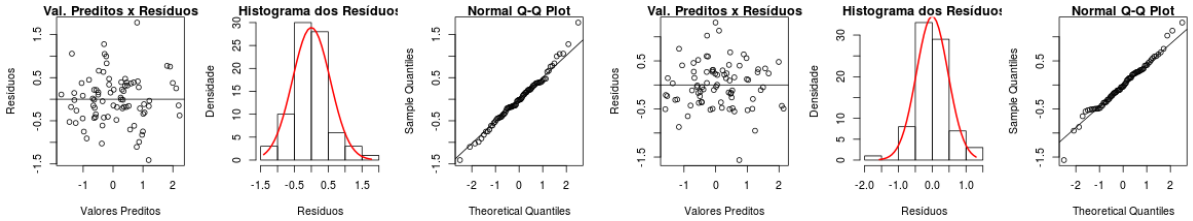
A Figura 86 apresenta a verificação do modelo através da análise dos resíduos para cada variável resposta. O gráfico da esquerda para direita representa a relação entre os resíduos pelos valores preditos e não apresenta estrutura ou padrão. O histograma dos resíduos que apresenta a distribuição da frequência dos resíduos apresentou aproximadamente simétrico em forma de sino. O gráfico à direita representa a probabilidade normal dos resíduos apresentando pontos próximos da reta.

D.5 Considerações finais

De acordo com os algoritmos analisados, para a variável resposta *Garbage Collector*, o fator sistema operacional não interferiu no tempo de uso do *Garbage Collector*, pois para todos

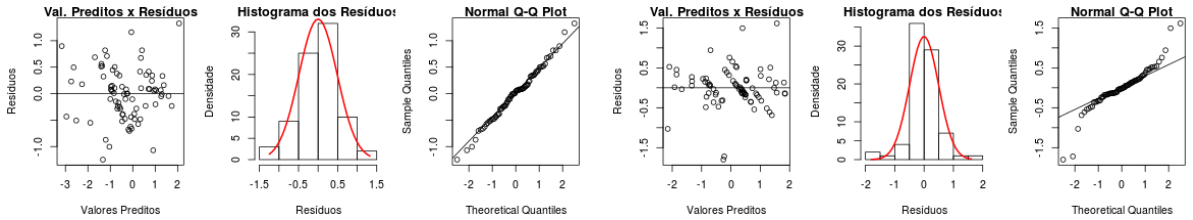
os algoritmos o resultado foi igual, ou seja, a um nível de significância de 5% não existiu evidências estatísticas que suportem afirmar que os tempos de uso do *Garbage Collector* sejam diferentes. Para o fator Memória no nível – (1GB) o tempo de utilização do *Garbage Collector* é superior aos demais níveis (nível 0 e nível +) para todos os algoritmos analisados. O nível + do fator memória utiliza menos tempo que os demais níveis (nível – e nível 0). Por fim, o fator núcleo, para o algoritmo *TestDFSIO*, não interferiu no tempo de uso de *Garbage Collector*.

Figura 86: Verificação dos resíduos: gráfico resíduos versus valores preditos e gráfico de probabilidade normal dos resíduos para o modelo de ordem de dois do fatorial 3^4 do algoritmo *TestDFSIO Write*, após a transformação de Johnson para cada variável resposta



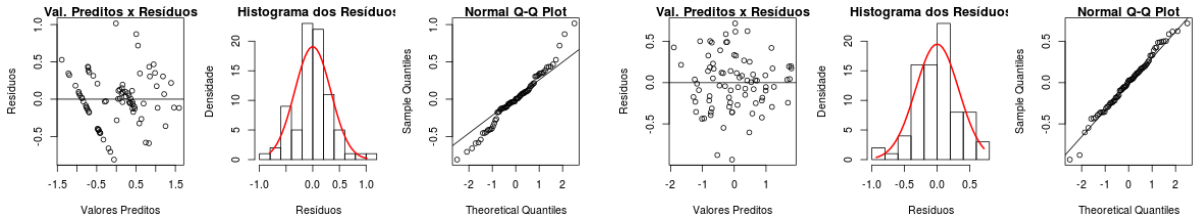
(a) *Garbage Collector*

(b) CPU



(c) Memória Física

(d) Memória Virtual



(e) Heap

(f) *Throughput*

(g) Taxa média de IO

Para a variável resposta *CPU*, o fator núcleo no nível – utiliza em média menos tempo a CPU que o nível + que por sua vez utiliza em média mais tempo a CPU que o nível –. Em média o sistema operacional que utiliza mais tempo o processamento de CPU é o sistema

operacional Windows, em relação às distribuições do Linux analisadas. E por fim, a máquina virtual que utiliza em média maior tempo o processamento de CPU é o KVM.

Para a variável resposta *Memória Física*, os fatores núcleo e memória no nível – utiliza, em média, menos quantidade de memória que os demais níveis (nível 0 e +). E por fim, o sistema operacional Windows utiliza, em média, menos quantidade de memória física que os demais sistemas operacionais.

Para a variável resposta *Memória Virtual*, os fatores núcleo e máquina virtual não interferiram na quantidade de *bytes* de memória virtual utilizada para resolver os problemas, pois para todos os algoritmos o resultado foi igual a 0, ou seja, a um nível de significância de 5% não existiu evidências estatísticas que suportem afirmar que sejam diferentes. Já o fator sistema operacional que utiliza, em média, mais quantidade de memória virtual para resolver um problema é o sistema operacional CentOS (nível +) e o sistema operacional Windows (nível +) utiliza, em média, menos quantidade de memória virtual.

Para a variável resposta *Heap*, na maioria dos casos os fatores núcleo, sistema operacional e máquina virtual não interferiram na quantidade de *bytes* de *heap* utilizada para resolver os problemas, pois para quase todos os algoritmos o resultado foi igual a 0, ou seja, a um nível de significância de 5% não existiu evidências estatísticas que suportem afirmar que sejam diferentes. Já o fator memória no nível + utiliza em média mais quantidade de heap para resolver um problema e o nível – em media utiliza menos quantidade em *bytes* de *heap* para resolver o problema.

Para as variáveis respostas *Throughput e Taxa média de IO*, o fator núcleo não interferiu na quantidade de megabytes por segundo, pois para quase todos os algoritmos o resultado da comparação foi igual a 0, ou seja, a um nível de significância de 5% não existiu evidências estatísticas que suportem afirmar que sejam diferentes. O fator memória no nível –, em média possui uma vazão e uma taxa de IO de megabytes por segundo menor que os demais níveis. Por fim, o fator sistema operacional Windows (nível +), em média, possui uma vazão e uma taxa de IO de *megabytes* por segundo menor que os demais níveis.

Além do teste de médias para os ambientes computacionais foi aplicado também o teste de variância (ANOVA), sobre os dados transformados pela Transformação de Johnson, para confirmar a significância dos efeitos e responder as 8 Indagações apresentadas na Seção 6.1. Sendo assim, levando em consideração a variável resposta *Garbage Collector*, pode-se verificar que os efeitos principais Memória, SO e VM foram estatisticamente significativos para a maiorias dos algoritmos analisados aceitando então as Indagações (I_1, I_2, I_3). Já as interações

entre núcleo:memória e memória:SO foram também na maioria dos casos significativos aceitando então as indagações (I_4, I_5, I_6). Pode-se verificar também que o efeito mais impactante nessa variável resposta foi o efeito da memória.

Para a variável resposta *CPU*, pode-se verificar que todos os efeitos principais foram estatisticamente significativos para a maiorias dos algoritmos analisados aceitando então as Indagações (I_0, I_1, I_2, I_3). Já as interações entre núcleo:SO, memória:VM e SO:VM foram também na maioria dos casos significativos, aceitando então as indagações (I_4, I_5, I_6, I_7). Novamente para a variável resposta CPU, o efeito mais significativo foi o núcleo seguido da memória, SO e VM.

Os efeitos mais significativos para a variável resposta *Memória Física* para a maiorias dos algoritmos analisados foram núcleo, memória, SO e VM, aceitando então as Indagações (I_0, I_1, I_2, I_3). Já as interações entre núcleo:memória e núcleo:SO foram também na maioria dos casos significativos, aceitando então as indagações (I_4, I_5, I_6). Pode-se verificar também que o efeito mais impactante nessa variável resposta foi o efeito da memória.

A análise de variância para todos os algoritmos da variável resposta *Memória Virtual* mostra que somente o efeito VM não pode ser considerado significativo, aceitando então as Indagações (I_0, I_1, I_2). Já as interações entre núcleo:memória, núcleo:SO e memória:SO foram também na maioria dos casos significativos, aceitando então as indagações (I_4, I_5, I_6). Pode-se verificar que no caso da *memória virtual* o efeito que mais impactou foi o sistema operacional seguido da memória.

Para a variável resposta *Heap*, pode-se verificar que os efeitos principais núcleo, memória e VM foram estatisticamente significativos para a maiorias dos algoritmos analisados aceitando então as Indagações (I_0, I_1, I_3). Já as interações entre núcleo:memória, memória:SO, memória:VM e SO:VM foram também na maioria dos casos significativos, aceitando então as indagações (I_4, I_5, I_6, I_7). Novamente para a variável resposta *Heap*, o efeito mais significativo foi o memória seguido do núcleo e da interação entre a memória e o SO e entre a memória e a VM.

Os efeitos mais significativos para as variáveis resposta *Throughput e Taxa média de IO* para a maiorias dos algoritmos analisados foram memória, SO e VM, aceitando então as Indagações (I_1, I_2, I_3). Já as interações entre SO:Memória e SO:VM foram também na maioria dos casos significativos, aceitando então as indagações (I_5, I_6, I_7). Neste casso, o efeito mais impacto na taxa de megabytes por segundo foi o efeito do sistema operacional, seguido pela memória e máquina virtual.