UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Avaliação da Validade Externa da Técnica de Análise Diferencial para Detecção de Envelhecimento de Software: Um Estudo Confirmatório com Replicação

Guilherme Otávio de Sena

Uberlândia, Minas Gerais.
2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Avaliação da Validade Externa da Técnica de Análise Diferencial para Detecção de Envelhecimento de Software: Um Estudo Confirmatório com Replicação

Guilherme Otávio de Sena

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Software
Orientador: Prof. Dr. Rivalino Matias Jr.

Uberlândia, Minas Gerais.
2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "Avaliação da Validade Externa da Técnica de Análise Diferencial para Detecção de Envelhecimento de Software: Um Estudo Confirmatório com Replicação" por Guilherme Otávio de Sena como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação.

Uberlândia, Agosto de 2017.

Orientador: _____

Prof. Dr. Rivalino Matias Jr.

Universidade Federal de Uberlândia (UFU) – Faculdade de Computação (FACOM)

Banca Examinadora: _____

Prof. Dr. Autran Macêdo

Universidade Federal de Uberlândia (UFU) – Faculdade de Computação (FACOM)

_____

Prof. Dr. Lúcio Borges Araujo

Universidade Federal de Uberlândia (UFU) – Faculdade de Matemática (FAMAT)

_____

Prof. Dr. Catello Di Martino

Nokia Bell Labs/EUA

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Agosto de 2017.

Autor:	**Guilherme Otávio de Sena**
Título:	Avaliação da Validade Externa da Técnica de Análise Diferencial para Detecção de Envelhecimento de Software: Um Estudo Confirmatório com Replicação
Faculdade:	Faculdade de Computação
Grau:	Mestrado

_____

Autor

DEDICATÓRIA

Aos meus pais Mário e Solange.

Aos meus amigos.

AGRADECIMENTOS

Ao meu orientador, Prof. Rivalino Matias Júnior,
pela sua paciência, incentivo e dedicação.

“Agradeço todas as dificuldades que enfrentei; não fosse por elas, eu não teria saído do lugar. As facilidades nos impedem de caminhar. Mesmo as críticas nos auxiliam muito.”

Francisco Cândido Xavier.

RESUMO

Sistemas de software que executam continuamente por um período de tempo podem sofrer de envelhecimento de software. Esse fenômeno está relacionado ao aumento da taxa de falha na medida em que o sistema executa. Recentemente, um estudo introduziu uma técnica para detecção do envelhecimento baseada em análise diferencial de software que, por meio de experimentos sob cargas sintéticas com foco em vazamento de memória, mostrou-se ser superior que outras abordagens utilizadas em SAR. A análise diferencial consegue distinguir entre o comportamento natural do comportamento do envelhecimento ao comparar, sob experimentos, duas versões do mesmo sistema: versão alvo (com envelhecimento) e versão base (sem envelhecimento). Esta pesquisa de mestrado avaliou a validade externa dessa abordagem para verificar se os resultados vistos anteriormente também se aplicavam frente a aplicações reais e cargas definidas através de um processo de caracterização de uso dessas aplicações. Para esta proposta, 4 aplicações reais amplamente conhecidas com *bugs* de vazamento de memória foram selecionadas. Os padrões de ativação de cada *bug* foram estudados a fim de incorporá-los na caracterização de representatividade dos cenários de cargas de trabalho utilizados. Posteriormente, para cada aplicação, replicações experimentais foram realizadas nas versões alvo e base considerando os cenários de carga de trabalho planejados. Em cada replicação, os indicadores RSS e HUS foram monitorados compondo, cada um, uma série temporal distinta. Em seguida, a fim de reduzir os efeitos de defasagem entre as séries, uma série temporal média foi estimada pelo método DTW para cada conjunto de replicações. Finalmente, as séries temporais médias de cada indicador foram processadas por meio da combinação de técnicas estatísticas de detecção de tendências e CEP, gerando gráficos de divergência para identificação de anomalias. Os gráficos de divergência permitem, de forma justa, comparar o desempenho na detecção do vazamento de cada combinação técnica/indicador. Os resultados mostraram que, diferentemente dos encontrados previamente com carga sintética, todas as combinações conseguiram detectar o vazamento de memória eficientemente, com nenhuma taxa de falso-negativos e com poucos falso-positivos. Além disso, as técnicas de detecção de tendências, em especial a de Hodrick Prescott (HP), foram melhores que as de CEP. Novamente, o indicador HUS mostrou-se superior ao RSS, determinando HP/HUS como a melhor combinação geral para detectar o vazamento de memória.

Palavras-chave: Envelhecimento de Software. Vazamento de Memória. Detecção de Anomalias. Análise Diferencial.

ABSTRACT

Software systems running continuously for a period of time often confront software aging. This phenomenon is related to the increase of the failure rate as the system executes. Recently, a study introduced a technique for aging detection based on differential software analysis that, through experiments under synthetic workloads with focus on memory leakage, proved superior to other approaches used in SAR. The differential analysis can distinguish between the natural behavior of aging behavior when comparing (under experiments) two versions of the same system: target version (with aging) and base version (without aging). This master's study evaluated the external validity of this approach to verify if the previous findings also applied to real applications and loads. For this purpose, 4 widely known real-world applications with memory leak bugs were selected. The activation patterns of the bugs were studieds in order to incorporate them into the representativeness characterization of the workload scenarios used. Subsequently, for each application, experimental replications were performed on the target and base versions considering the planned workload scenarios. In each replication, the RSS and HUS indicators were monitored, each composing a different time series. Then, in order to reduce the dissimilarity effects between the series, a mean time series was estimated by the DTW method for each set of replications. Finally, the mean time series of each indicator were processed through a combination of statistical techniques of trend detection and CEP, generating divergence charts for the anomalies identification. The divergence charts allow a fair comparison of the leak detection performance of each technique/indicator combination. The results showed that, unlike those previously findings with synthetic workloads, all combinations were able to detect memory leak efficiently, with no false-negatives and few false-positives rates. In addition, the trend detection techniques, in particular Hodrick Prescott (HP), were better than those of CEP. Again, the HUS indicator was superior to RSS, determining HP/HUS as the best overall combination to detect memory leakage.

Keywords: Software Aging. Memory Leak. Anomaly Detection. Differential Analysis.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

Lista de Algoritmos

LISTA DE ABREVIATURAS E SIGLAS

AR Error	Erro relacionado com o envelhecimento.
AR Failure	Falha relacionada com o envelhecimento.
AR Fault	Falta relacionada com o envelhecimento.
CEP	Controle Estatístico de Processos.
CS	Gráfico de Soma Cumulativa.
EW	Gráfico de Média Móvel Exponencialmente Ponderada.
HP	Filtro Hodrick-Prescott.
HSZ	Heap Usage Size.
HUS	Heap Usage.
ISP	Provedor de Serviço de Internet.
LR	Regressão Linear.
MA	Média Móvel.
MM	Mediana Móvel.
SAR	Envelhecimento e Rejuvenescimento de Software.
SH	Gráfico de Shewhart.
SO	Sistema Operacional.
RSS	Resident Set Size.
VM	Máquina Virtual.
VMM	Monitor de Máquina Virtual.
VSZ	Virtual Memory Size.
WS	Windows Size.

1. INTRODUÇÃO

1.1 Contextualização

Ao longo das últimas décadas é possível identificar grandes transformações tecnológicas que mudaram a forma com que os seres humanos realizam suas tarefas cotidianas. Tais transformações foram iniciadas num período conhecido como Era Digital ou Era da Informação que, atualmente, reflete na dependência de computadores e suas aplicações para realização das mais diversas tarefas, desde o simples uso de um cartão bancário quanto o complexo procedimento de enviar uma sonda a Marte. Portanto, é de extrema importância que tais aplicações realizem seus procedimentos com o máximo de precisão, pois a ocorrência de qualquer falha pode levar a consequências irreparáveis.

Uma área que se preocupa com a confiança no funcionamento de aplicações computacionais é conhecida como dependabilidade [Avižienis *et al*. 2004]. Portanto, a dependabilidade de um sistema computacional é a capacidade com que esse sistema fornece seus serviços de forma que seus usuários podem, justificadamente, confiar em seu funcionamento [Avižienis *et al*. 2004]. Os principais atributos da dependabilidade são: disponibilidade, confiabilidade, segurança, integridade e manutenibilidade. Dentre as várias ameaças à dependabilidade computacional [Avižienis *et al*. 2004], destacam-se as falhas causadas por envelhecimento de software.

Envelhecimento de Software é o nome dado ao fenômeno observado durante a execução de sistemas de software, onde a taxa de falha do sistema incrementa na medida em que o tempo de execução aumenta. Essas falhas são consequências do acúmulo de sucessivas ocorrências de erros causados pela ativação de faltas (defeitos ou bugs) de software, levando à deterioração do estado interno do sistema [Grottke *et al*. 2008]. Uma vez que essas faltas são ativadas durante a execução do sistema, então o acúmulo de erros pode provocar o deslocamento gradual do estado interno desse sistema de "correto" para um estado de "falha provável", onde problemas podem ocorrer, tal como a degradação da performance e/ou falhas no software [Huang *et al*. 1995].

O campo de pesquisa sobre Envelhecimento e Rejuvenescimento de Software (SAR) é relativamente novo se comparado a outras áreas de pesquisa. Este teve como início o ano de 1995 quando Huang e Kintala, pesquisadores do laboratório Bell Labs da AT&T, investigaram a degradação progressiva de um software enquanto executa, demonstrando uma preocupação com as aplicações de execução contínua que possuem alta correlação entre o aumento das taxas de falha por período de execução, como por exemplo, em sistemas do tipo servidor [Huang *et al*. 1995]. Huang e Kintala nomearam esse fenômeno como envelhecimento de processo, entretanto, todos os estudos posteriores a esse passaram a nomear tal

fenômeno como envelhecimento de software, completando 20 anos em 2015 [Matias 2015].

Existem diferentes tipos de efeitos causados pelo envelhecimento de software, como por exemplo: 1) vazamento de memória, de *threads*, de processos, de identificadores; 2) fragmentação de memória, de arquivos de banco de dados, de sistemas de arquivos; 3) erros numéricos de arredondamento; 4) corrupção de arquivos de banco de dados; etc. Dentre esses efeitos, os mais citados na literatura em SAR são os relacionados com a utilização de recursos do sistema operacional (SO), em especial a memória principal [Valentim *et al*. 2016].

Os efeitos relacionados com a memória podem se manifestar, principalmente, devido a problemas de fragmentação e/ou vazamento [Macedo *et al*. 2010]. A fragmentação de memória é uma consequência direta da execução dinâmica de um sistema, que se torna mais preocupante para aplicações que requerem grandes alocações de memória. O vazamento é causado pelo uso incorreto das rotinas de gerenciamento da alocação dinâmica de memória e pode ocorrer em diferentes níveis, como por exemplo, nível de processo, nível de núcleo (*kernel*) do sistema operacional, nível de monitor de máquina virtual, etc.

Para detectar os efeitos de envelhecimento em um sistema é necessário monitorá-los através dos indicadores de envelhecimento. Esses indicadores têm como propósito contribuir na detecção do envelhecimento [Grottke *et al*. 2008] e podem ser encontrados, com terminologias diferentes, na maioria dos sistemas operacionais. Indicadores frequentemente utilizados para detectar o vazamento de memória em estudos na literatura em SAR são o *resident set size* (RSS) e o *virtual memory size* (VSZ). Entretanto, outros têm ganhando espaço recentemente devido a sua eficiência: *heap size* (HSZ) e *heap usage* (HUS).

Atualmente na literatura existem várias técnicas propostas para mitigar os efeitos do envelhecimento de software. Algumas delas constituem o rejuvenescimento de software, apresentado pela primeira vez em [Huang *et al*. 1995]. Esse conjunto de técnicas empregado a um sistema propõe sua extensão de vida e já foi adotado em várias aplicações reais, como por exemplo, em gerenciadores de clusters de servidor [Castelli *et al*. 2001], em sistemas de telecomunicações [Avritzer e Weyuker 1997], em servidores *web* [Matias e Filho 2006] e vários outros.

1.2 Motivação

Conforme explicado anteriormente, o envelhecimento de software é, na maioria das vezes, causado por faltas ativadas durante a execução do sistema (conhecidas como faltas relacionadas com o envelhecimento – *AR Faults*). As *AR Faults* promovem o acúmulo de sucessivas ocorrências de erros (erros relacionados com o envelhecimento – *AR Errors*) e consequentemente levam o sistema para um estado onde falhas podem acontecer (falhas relacionadas com o envelhecimento – *AR Failures*). Essa relação entre faltas, erros e falhas voltadas para o envelhecimento de software pode ser vista mais detalhadamente em [Grottke *et al*. 2008].

Assim, considerando o ciclo de vida de construção de um software [Pressman e Maxim 2014], é benéfico localizar e corrigir *AR Faults* o quanto antes, uma vez que essas faltas, geralmente, são descobertas apenas quando o sistema já se

encontra no ambiente de produção e só podem ser corrigidas pelos usuários caso tenham livre acesso ao código-fonte e competência para tal [Trivedi *et al*. 2011].

Foi pensando nisso que o estudo realizado em [Matias *et al*. 2014] apresentou uma abordagem para detectar o envelhecimento de software ainda na fase de testes, executando testes de sistema – testes que avaliam o software em busca de falhas por meio de sua utilização. O ponto chave de tal abordagem é a aplicação de uma técnica conhecida como análise diferencial de software [Langner e Andrzejak 2013a] [Langner e Andrzejak 2013b]. O objetivo principal dessa técnica é comparar o comportamento de duas versões diferentes do mesmo software, a saber: versão estável e versão sob teste.

A abordagem proposta em [Matias *et al*. 2014] foi avaliada tendo como base a comparação entre duas versões do mesmo software, uma versão sob investigação de vazamento de memória, nomeada de versão alvo, e outra livre do problema, nomeada de versão base. Durantes os testes, indicadores de envelhecimento relacionados ao vazamento de memória foram monitorados e os dados coletados foram analisados por meio de diferentes técnicas estatísticas de detecção de tendências e de controle estatístico de processos (CEP). Os resultados mostraram uma grande efetividade (em curto intervalo de tempo) na detecção do vazamento de memória por meio da abordagem de análise diferencial de software, demonstrando o quanto tal abordagem pode ser eficiente ainda na fase de testes de software. Os resultados evidenciaram também que o indicador de envelhecimento HUS, em combinação com as técnicas de CEP, apresentou maior eficiência para a detecção de vazamento de memória na maioria dos cenários de cargas de trabalho avaliados.

Os resultados evidenciados em [Matias *et al*. 2014] tiveram como base uma variedade de cenários de cargas de trabalho sintéticas criadas por um algoritmo que emulava duas versões de um programa não real (utilizado para fins de estudo): a versão alvo e a versão base. Uma carga de trabalho sintética é geralmente formada por um ou mais programas parametrizados que simulam o comportamento de um sistema sob estudo e que, portanto, nem sempre se trata de um comportamento proveniente de um sistema real [Kiskis e Shin 1996].

Assim, uma vez que os resultados analisados em [Matias *et al*. 2014] não foram obtidos a partir de cenários de aplicações e cargas de trabalho reais, tem-se uma limitação em termos de validade externa do trabalho. Por meio da análise da validade externa de uma pesquisa, pesquisadores podem avaliar se os resultados da pesquisa podem ser mais ou menos generalizados para outros cenários além daqueles considerados na pesquisa em questão [Siegmund *et al*. 2015].

Neste contexto, a principal motivação desta dissertação de mestrado foi avaliar a validade externa da abordagem de análise diferencial aplicada ao envelhecimento de software, proposta em [Matias *et al*. 2014], por meio da realização de um conjunto de experimentos controlados tendo como base aplicações reais. Além disso, as cargas de trabalho foram definidas através de um processo de caracterização com objetivo de se aproximarem ao máximo do comportamento real de uso das aplicações reais consideradas.

Desse modo, os resultados desta pesquisa possibilitam uma melhor avaliação da eficácia do método de análise diferencial para uso na área de envelhecimento de software, oferecendo uma maior confiança em sua aplicação pela indústria de software.

1.3 Objetivos da Pesquisa

1.3.1 Geral:

Avaliar a combinação de diferentes técnicas estatísticas e indicadores de envelhecimento através da abordagem de análise diferencial de software voltada para detecção de envelhecimento de software, com foco em vazamento de memória, utilizando cargas de trabalho reais.

1.3.2 Específicos:

- Caracterizar o comportamento do envelhecimento de software em diferentes aplicações reais, com vistas para testes de sistemas baseados em análise diferencial de software.
- Analisar, de forma comparativa, se os resultados da abordagem de análise diferencial com carga de trabalho sintética se repetem em experimentos com cargas de trabalho reais.
- Incorporar o método *Dynamic Time Warping* ao protocolo de teste da análise diferencial, permitindo maior acuracidade nas análises com múltiplas replicações de séries temporais de indicadores de envelhecimento.

1.4 Desenvolvimento da Pesquisa

Para realizar um estudo confirmatório com replicação, avaliando a validade externa da abordagem apresentada em [Matias *et al*. 2014], esta pesquisa considerou a realização de algumas atividades, descritas de modo geral nesta Seção.

1.4.1 Revisão da Literatura

Uma revisão da literatura em Envelhecimento de Software com foco em vazamento de memória foi realizado nessa etapa. A busca se iniciou por uma criteriosa triagem de importantes trabalhos em SAR, centrada em vazamento de memória, desde o ano de 1995 (ano em que se iniciaram as pesquisas na área), tendo como fontes: livros, publicações em periódicos e anais de conferências. Além disso, pesquisas relacionadas com o trabalho de [Matias *et al*. 2014] também foram consideradas.

1.4.2 Material

Uma vez que esta pesquisa concentrou-se em avaliar a validade externa da abordagem apresentada em [Matias *et al*. 2014], frente a aplicações e cargas de trabalho reais, então os dados amostrais utilizados nesse trabalho foram obtidos pelo monitoramento de indicadores de envelhecimento em sistemas reais durante suas execuções. As seguintes aplicações reais foram utilizadas: 1) Squid – um *proxy cache* para *web* que suporta protocolos conhecidos (ex. HTTP, FTP, DNS) [Squid 2017a]; 2) Inkscape – um programa de manipulação de gráficos vetoriais [Inkscape 2017a]; 3) Lighttpd – um servidor *web* com suporte a diversas

funcionalidades para gerenciamento de aplicações para Internet [Lighttpd 2017a] e; 4) PostgreSQL – um sistema gerenciador de banco de dados [PostgreSQL 2017a].

A escolha das aplicações se deu por uma busca em fóruns oficiais de relatórios de *bugs*. Tais fóruns foram pesquisados em busca por *bugs* relacionados com vazamento de memória. Depois de encontrar um relato de *bug* de vazamento de memória para determinada aplicação, um teste prévio de confirmação desse vazamento foi realizado. Para cada caso positivo, outro teste buscando pela versão anterior mais recente em que o problema não se apresentava foi efetuado (versão base). Finalmente, depois de encontrada a versão base, a versão imediatamente posterior, apresentando o vazamento, foi selecionada como versão alvo. Portanto, as quatro aplicações escolhidas foram aquelas em que todos os critérios acima mencionados foram satisfeitos. Uma descrição mais detalhada sobre esse processo de seleção das aplicações é apresentada na Seção 4.3.

1.4.3 Métodos

Os métodos utilizados nessa pesquisa se estendem aos processos de caracterização e geração de carga de trabalho, coleta de dados, detecção do envelhecimento e análise dos dados.

Com relação ao processo de caracterização de carga de trabalho, para cada aplicação, os padrões de ativação de seu respectivo *bug* de vazamento foram estudados, buscando por trabalhos de pesquisa que apoiassem um planejamento experimental com cenários de cargas de trabalho que representassem o comportamento real de cada aplicação frente ao padrão de ativação sendo utilizado.

Para todas as aplicações, os cenários de cargas de trabalho foram planejados para considerar dois modos de operação (constante e variado), sendo que para carga constante três tipos de intensidade foram usados: 1) baixa; 2) normal e; 3) alta. Além disso, automatizações dos cenários de cargas de trabalho planejados com comportamento próximo do real foram implementadas. Essas automatizações foram usadas para gerar a carga automaticamente, baseada em cenários reais de uso. As aplicações estudadas foram analisadas com base em replicações para cada cenário de carga de trabalho, de modo a reduzir a influência de erros experimentais. Erros experimentais estão presentes em qualquer estudo experimental e o uso de replicações é uma forma de mitigar tais erros [Montgomery 2013].

Os dados considerados para análise são formados pela média dessas replicações em cada cenário e versão de teste (base e alvo). Assim, cada série temporal de um dado indicador de envelhecimento, obtida da média dessas replicações, foi avaliada por meio de diferentes técnicas de detecção de tendências e CEP. As técnicas empregadas em [Matias *et al*. 2014] foram mantidas nesta pesquisa, cobrindo quatro métodos estatísticos para análise de tendências: 1) Regressão Linear; 2) Média Móvel; 3) Mediana Móvel e; 4) Filtro Hodrick Prescott e os métodos CEP: Gráfico de Controle Shewhart, Média Móvel Exponencial Ponderada (EWMA) e Soma Cumulativa (CuSum).

Posteriormente, cada série temporal é processada pelo método da análise diferencial, o qual objetiva detectar o envelhecimento de software nos casos em que há divergência, estatisticamente significativa, entre o consumo de memória da

versão alvo em comparação com o consumo de memória da versão base de software sob investigação. Como resultado, tem-se um gráfico de divergências usado para indicar as alterações significativas no padrão do consumo de memória.

Em termos de coleta de dados, assim como em [Matias *et al*. 2014], os dados monitorados foram dos indicadores HUS e RSS. Para o monitoramento do RSS utilizou-se o sistema de arquivos /proc [Linux Filesystem Hierarchy 2017]. Para o HUS, todas as operações de alocação e desalocação dinâmica, realizadas em tempo de execução, foram interceptadas. Para isso, um *wrapper* de alocação dinâmica de memória chamado *DebugMalloc*, introduzido em [Costa e Matias 2015], foi utilizado. Tanto o RSS quanto o HUS foram amostrados a cada 5 segundos.

1.5 Estrutura do Documento

Os demais capítulos desta dissertação estão organizados da seguinte forma:

O Capítulo 2 aborda a fundamentação teórica usada como base para este trabalho, apresentando conceitos de envelhecimento de software com foco em vazamento de memória, os indicadores de envelhecimento utilizados e uma revisão da literatura correlata.

O Capítulo 3 trata sobre a abordagem de análise diferencial de software com foco em envelhecimento de software apresentada em [Matias *et al*. 2014], explicando sobre as técnicas de detecção de tendências e CEP que compõem a abordagem.

O Capítulo 4 apresenta as perguntas e hipóteses de pesquisa e descreve o estudo experimental realizado, considerando os detalhes das aplicações utilizadas, o plano experimental para cada aplicação, as ferramentas de coleta de dados e os procedimentos de geração de carga de trabalho.

O Capítulo 5 descreve e discute os resultados do estudo experimental para cada aplicação e as conclusões baseadas nas perguntas e hipóteses de pesquisa.

Finalmente, o Capítulo 6 apresenta as considerações finais sobre a pesquisa, destacando os principais resultados e conclusões, as contribuições para literatura, as dificuldades encontradas, as limitações da pesquisa e os desafios para trabalhos futuros.

2. REVISÃO DA LITERATURA

2.1 Introdução

O objetivo deste Capítulo é abordar a fundamentação teórica que serviu como base para elaboração desta pesquisa. Os assuntos aqui apresentados são essenciais para a compreensão das atividades realizadas neste trabalho. Os principais aspectos relacionados ao fenômeno do Envelhecimento de Software e seus indicadores de envelhecimento são explicados na Seção 2.2. Na Seção 2.3 são abordados os conceitos relacionados ao vazamento de memória. A Seção 2.4 apresenta os trabalhos relacionados com esta pesquisa.

2.2 Envelhecimento de Software

O fenômeno observado durante a execução de muitos sistemas de software que tem como característica um incremento na taxa de falha na medida em que o tempo de execução aumenta é conhecido como Envelhecimento de Software [Grottke *et al*. 2008]. A origem dos estudos desse fenômeno se deu no ano de 1995, quando Huang e Kintala formalizaram em [Huang *et al*. 1995] o processo de degradação de um software durante sua execução, abordando a correlação entre o aumento das taxas de falha e o tempo de execução. Esse trabalho deu início a toda uma área de pesquisa chamada de Envelhecimento e Rejuvenescimento de Software que completou 20 anos em 2015 e se situa dentro da subárea de confiabilidade/dependabilidade de software.

Para compreender melhor o fenômeno de envelhecimento é preciso entender conceitos de dependabilidade. A dependabilidade aplicada a um sistema computacional é a capacidade com que esse sistema fornece um serviço no qual seus usuários podem, de forma justificada, confiar em seu funcionamento [Avižienis *et al*. 2004]. Falta, erro e falha de software são as três ameaças à dependabilidade computacional e por esse motivo sua relação é profundamente discutida em [Avižienis *et al*. 2004].

De acordo com [Avižienis *et al*. 2004], uma falha em um sistema é um desvio do serviço entregue pelo mesmo com relação à sua especificação. O serviço prestado por um sistema é classificado como correto quando o serviço entregue ao seu usuário condiz com a função do sistema, ou seja, o serviço prestado está de acordo com o que o sistema se destina a fazer. Portanto, um desvio pode representar a entrega incorreta de um serviço ou a sua não entrega. Além do mais, o sistema pode entrar em modo de degradação, em que o serviço é entregue de forma limitada. Dessa forma, uma falha é causada por, pelo menos, um (ou mais) estado(s) externo(s) do sistema que desvia(m) do estado de serviço correto. Esse desvio é chamado de erro.

Um erro pode se transformar em outros erros e essa transformação é conhecida como propagação de erros. O estado externo de um sistema é parte do estado que é perceptível na interface de serviço (onde a prestação de serviços ocorre), a parte restante é seu estado interno. É importante ressaltar que falhas estão diretamente relacionadas à percepção, realizada pelo usuário do sistema, de um erro no serviço fornecido. Portanto, apesar da percepção de um erro ocorrer no momento da entrega do serviço, caracterizando uma falha, o sistema já poderia estar com erros muito tempo antes. Além do mais, muitos erros não atingem o estado externo do sistema e, portanto, não caracterizam uma falha.

Cada erro é, por sua vez, causado pela ativação de uma falta (defeito ou *bug*). Uma falta que ainda não foi ativada é chamada de falta dormente. A ativação de uma falta causa um erro, o qual pode ou não alcançar a interface de serviço do sistema. Essas três ameaças à dependabilidade possuem uma relação causal denominada de cadeia fundamental da dependabilidade e é apresentada na Figura 2.1.

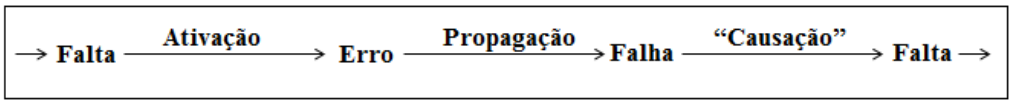


Figura 2.1. Cadeia fundamental da dependabilidade.
Traduzido de: [Grottke *et al*. 2008].

Como o Envelhecimento de Software é uma subárea da dependabilidade, então uma taxonomia de dependabilidade aplicada ao Envelhecimento de Software, definida em [Grottke *et al*. 2008], será usada neste trabalho. Assim, a relação entre faltas, erros e falhas que causam o fenômeno de envelhecimento é discutida como cadeia de ameaças para falhas relacionadas ao Envelhecimento de Software. Portanto faltas relacionadas com envelhecimento (*AR Faults*) estão dormentes até serem ativadas na presença de padrões de ativação específicos – denominados fatores de envelhecimento (*Aging Factors*). A ativação dessas faltas leva a erros relacionados com o envelhecimento (*AR Errors*), cujos efeitos acumulados com o tempo de execução causam a degradação do sistema/aplicação de software. Falhas dessa natureza são denominadas de falhas relacionadas com envelhecimento (*AR Failures*). Os efeitos acumulados pelas sucessivas ocorrências desses erros são chamados de efeitos de envelhecimento (*Aging Effects*). A Figura 2.2 apresenta a relação causal da cadeia de ameaças para falhas relacionadas ao Envelhecimento de Software.

Segundo [Grottke *et al*. 2008], um sistema não falha por envelhecimento, mas sim pela consequência de efeitos acumulados no decorrer do tempo de execução, podendo deslocar gradualmente de um estado interno saudável (sem presença de efeitos de envelhecimento) para um estado interno de falha provável (com presença de efeitos de envelhecimento). Em [Huang *et al*. 1995] é apresentado, pela primeira vez, um modelo de transição probabilístico de um sistema considerando quatro tipos diferentes de estados:

- Saudável: Não possui efeitos de envelhecimento acumulados o suficiente para causar uma falha no sistema.
- Falha provável: erros relacionados com o envelhecimento acumularam a ponto de o sistema falhar.

- Falha: Estado em que um sistema falha (falhas relacionadas com o envelhecimento).
- Rejuvenescimento: Estado em que o sistema está sendo rejuvenescido (os efeitos de envelhecimento acumulados são removidos).

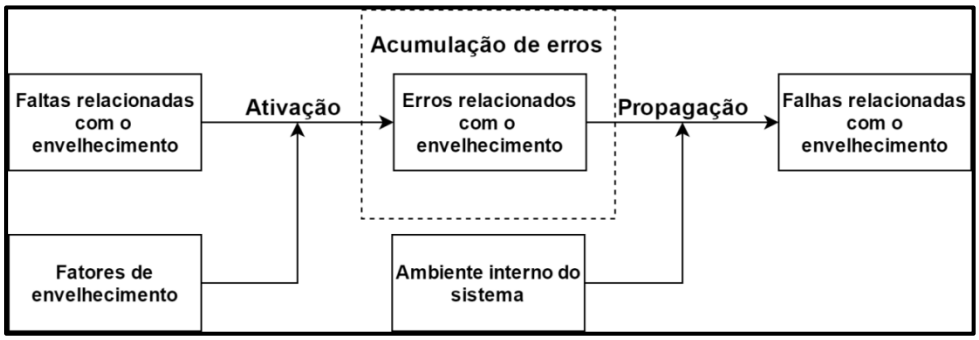


Figura 2.2. Cadeia de ameaças para falhas relacionadas com o Envelhecimento de Software.
Traduzido de: [Grottke *et al*. 2008].

Existem diversos tipos de efeitos de envelhecimento causados pelo fenômeno de Envelhecimento de Software. A taxonomia utilizada nessa pesquisa e definida em [Grottke *et al*. 2008] propõem classes de efeitos de envelhecimento baseado em suas características comuns:

- Classe 1 - vazamento de recursos: memória; *sockets*, descritor de arquivos; processos e *threads*.
- Classe 2 - fragmentação: memória; sistema de arquivos e arquivos do banco de dados.
- Classe 3 - acúmulo de erros numéricos: arredondamento.
- Classe 4 - acúmulo de dados corrompidos: sistema de arquivos e arquivos do banco de dados.

Ainda segundo [Grottke *et al*. 2008], tais efeitos podem ser classificados quanto ao seu nível de persistência:

- Efeitos voláteis: são removidos com a reinicialização do sistema, ou então, com a reinicialização de algum processo que esteja sofrendo de envelhecimento. Ex. vazamento de recursos do sistema operacional.
- Efeitos não voláteis: persistem mesmo após uma reinicialização. Ex. fragmentação do sistema de arquivos.

Na literatura em SAR existem várias técnicas propostas para mitigar os efeitos de envelhecimento. Tais técnicas são tradicionalmente conhecidas como rejuvenescimento de software, conceito apresentado pela primeira vez em [Huang *et al*. 1995], e compreende o conjunto de ações de tolerância a faltas que, quando aplicadas, têm por objetivo eliminar ou diminuir os efeitos do Envelhecimento de Software. A literatura traz diferentes relatos da aplicação de técnicas de rejuvenescimento em diferentes aplicações reais, tai como em sistemas de telecomunicações [Avritzer e Weyuker 1997], *clusters* de servidores [Castelli *et al*.

2001], em servidores *web* [Matias e Filho 2006] e vários outros [Valentim *et al*. 2016].

Para a aplicação de uma técnica de rejuvenescimento é necessário detectar os efeitos do envelhecimento. Essa detecção pode, em geral, ocorrer em duas das etapas do ciclo de vida do software [Pressman e Maxim 2014], a saber: fase de desenvolvimento e fase operacional. A abordagem avaliada nesta pesquisa tem por objetivo detectar os efeitos do envelhecimento, em especial o vazamento de memória, na fase de desenvolvimento, mas podendo ser também usada na fase operacional. O ideal, sem dúvida, é remover *AR Faults* na fase de desenvolvimento, contudo, técnicas para contra atacar seus efeitos na fase operacional do sistema passaram a ser investigadas na medida em que a detecção na fase de desenvolvimento nem sempre é viável ou possível.

2.3 Vazamento de Memória

Os efeitos de envelhecimento relacionados com a utilização da memória principal têm sido os mais citados na literatura [Valentim *et al*. 2016], como por exemplo, em [Shereshevsky *et al*. 2003], [Grottke *et al*. 2006], [Matias e Filho 2006] e [Matias *et al*. 2010a]. Tais problemas relacionados à memória podem se manifestar, principalmente, devido a problemas de fragmentação e/ou vazamento de memória [Macedo *et al*. 2010].

Apesar de estudos sobre a fragmentação de memória estarem ganhando mais espaço na literatura recente, como nos trabalhos de [Araujo et. al 2011]; [Matias *et al*. 2010b] e [Alonso et. al 2010], a maioria das pesquisas experimentais têm estudado problemas relacionados a vazamento de memória, como pode ser visto em um levantamento realizado por [Valentim *et al*. 2016]. O vazamento de memória é causado pelo uso incorreto de rotinas de gerenciamento de alocação dinâmica de memória e ocorre quando um processo de aplicação aloca dinamicamente blocos de memória e, por algum motivo, não os libera (integralmente ou parcialmente) ao sistema operacional, durante seu tempo de execução [Macedo *et al*. 2010].

Existem várias razões que podem levar a ocorrência do vazamento de memória. Em [Matias *et al*. 2012] é proposta uma classificação em dois tipos principais: 1) não liberação involuntária e; 2) não liberação voluntária. A não liberação involuntária de memória ocorre devido à incapacidade de um processo liberar um bloco de memória que foi previamente alocado. Esse fato geralmente ocorre pela perca da referência (endereço) de um bloco alocado, por exemplo, quando o valor do ponteiro que armazena o endereço de bloco alocado é sobrescrito, não permitindo a liberação do bloco e o transformando em um bloco órfão. Outro clássico exemplo é a utilização desequilibrada de *malloc* e *free* [GNU C Library 2017], por programadores, durante a execução de um programa, ou seja, não existe uma chamada *free* para cada chamada *malloc* correspondente, fazendo com que alguns blocos de memória não sejam liberados durante a execução de um programa [Matias *et al*. 2012].

Portanto, a não liberação involuntária é considerada como um *bug* no software, uma vez que é causada por faltas de software. Em contrapartida, a não liberação voluntária de memória está diretamente relacionada com o projeto de uma determinada aplicação, como por exemplo, uma aplicação que previamente aloca grandes quantidades de blocos de memória, evitando custos computacionais

de repetitivas operações de alocação e desalocação que poderiam prejudicar seu desempenho [Matias *et al*. 2012]. Em outros casos, a memória é esporadicamente aumentada devido a picos de carga trabalho. Tal efeito é geralmente tratado como um problema de explosão de memória (*blowout*), como pode ser visto em [Ferreira *et al*. 2011]. Assim, a não liberação voluntária de memória não é uma consequência direta de faltas (*bugs*) de software.

As operações de alocação de memória são uma das operações mais frequentes em programas de computador e são realizadas pelos alocadores de memória [Berger *et al*. 2000]. Um alocador tem como propósito realizar o gerenciamento da *heap* de um processo. A *heap* é a região do processo usada para mapear os blocos de memória alocados dinamicamente e, portanto, compõe uma parte da memória principal localizada na região de dados de um processo [Vahalia 1995] [Macedo *et al*. 2010].

Assim, basicamente, a estrutura geral de um alocador funciona da seguinte forma [Ferreira *et al*. 2011]: 1) o processo solicita memória via alguma rotina de alocação dinâmica de memória implementada por um alocador; 2) o alocador requisita ao sistema operacional uma área de *heap* que é criada e inicializada (podendo ser mais de uma dependendo da requisição); 3) a(s) área(s) de *heap* criada(s) e inicializada(s) é (são) estruturada(s) em várias fatias (*slices*) de memória livre. Dessa forma, quando uma nova requisição de memória feita pelo processo ao alocador excede a memória disponível na *heap*, esse procedimento se repete e uma nova *heap* é requisitada ao sistema operacional, sendo mesclada à *heap* anterior [Matias 2015].

Existem vários tipos de alocadores, tais como *Hoard*; *PTmalloc*; *TCMalloc*; *JEMalloc* e *TLSF* que, apesar de apresentarem similaridades em termos de estrutura de dados, se diferenciam na forma que gerenciam a *heap* pelas rotinas de gerenciamento de alocação dinâmica de memória, como pode ser visto em [Matias *et al*. 2011]. Alguns exemplos dessas rotinas implementadas como partes de um alocador são: *malloc*(); *realloc*() e *free*().

Considerando a arquitetura de um sistema operacional, segundo [Ferreira *et al*. 2011], podemos dividir os alocadores em dois níveis: 1) nível de *kernel* e; 2) nível de usuário. Em nível de *kernel*, o alocador serve o código do *kernel* do SO, que também requer alocações de memória dinâmica para suportar operações em áreas como sistema de arquivos, entrada e saída, gerenciamento de processos e etc. Alocadores desse nível são conhecidos como alocadores de memória do *kernel* (*Kernel-Level Memory Allocators* – KMA). Em nível de usuário, os alocadores servem as operações dentro dos processos que executam em espaço de usuário e geralmente são armazenadas em bibliotecas padrão vinculadas a esses processos. Dessa forma, um desenvolvedor de software pode escolher o alocador de memória que utilizará na construção de seu aplicativo. Alocadores desse nível são conhecidos como alocadores de memória de nível de usuário (*User-Level Memory Allocators* – UMA) [Ferreira *et al*. 2011].

Portanto, segundo [Matias *et al*. 2012], o vazamento de memória pode ocorrer tanto em nível de *kernel* quanto em *nível* de usuário. Em nível de *kernel* o vazamento afeta todo o sistema e não apenas um processo em específico, tornando os efeitos permanentes até que o SO seja reiniciado. Em contrapartida, no nível de usuário os efeitos causados pelo vazamento de memória terminam no momento em que o processo é encerrado, fazendo com que a memória alocada pelo processo

retorne ao sistema operacional. Assim, se um processo executa durante muito tempo, então pode ter a prestação de seu serviço degradado (ou até mesmo a impossibilidade de prestar o serviço) devido ao esgotamento de memória do sistema.

Conforme visto no Capítulo 1, para detectar os efeitos de envelhecimento em um sistema é necessário monitorá-los através dos indicadores de envelhecimento [Grottke *et al*. 2008]. Assim, os indicadores constituem o estado de um sistema, sob monitoramento, representados através de variáveis. Dessa forma, comparar os valores monitorados dessas variáveis com valores de comportamento padrão para um determinado sistema pode ajudar a revelar a presença de envelhecimento [Matias *et al*. 2012].

Existem diferentes níveis de indicadores que variam de acordo com o escopo a ser monitorado, por exemplo, processo de aplicação, sistema operacional, máquina virtual, etc. Uma vez que o envelhecimento pode acontecer em todos esses escopos, então é importante adotar indicadores apropriados para cada um, a fim de evitar uma detecção imprecisa. Os indicadores ainda podem ser classificados, de acordo com sua granularidade, em duas categorias [Grottke *et al*. 2008]:

- Indicadores de nível de sistema: oferecem informações globais sobre os subssistemas compartilhados do SO e geralmente são utilizados para avaliar os efeitos de envelhecimento no sistema como um todo. Ex. *free physical memory*, *used swap space*, *file table size* e *system load*.
- Indicadores específicos de aplicação: oferecem informações específicas sobre um processo de aplicação individual. Ex. *resident set size*, *JVM heap size*, *response time*, *heap usage* e *virtual memory size*.

Em ambas as categorias de indicadores é possível coletar dados em nível de usuário e em nível de *kernel*. A coleta em nível de usuário pode ser feita por programas que executam nesse nível, como pode ser visto em [Bao *et al*. 2005] e [Matias e Filho 2006]. A utilização desses programas pode ser limitada, uma vez que depende das informações que o SO exporta para o nível do usuário. Um exemplo clássico disso é o sistema de arquivos Linux/proc que armazena muitas informações desse tipo. Em contrapartida, é possível coletar dados do nível de *kernel* através de uma técnica conhecida como instrumentação do *kernel*, conforme visto em [Cotroneo *et al*. 2010] e [Matias *et al*. 2010b]. Tal técnica realiza uma a medição dentro do *kernel*, onde em teoria qualquer informação está disponível, tanto para indicadores de todo sistema quanto para específicos de aplicação. Tal instrumentação é poderosa, entretanto, requer alteração no código do *kernel*.

Devido a essa limitação de utilizar indicadores de nível de sistema para monitoramento em nível de usuário, diminuindo a eficácia na detecção do envelhecimento, é que muitos estudos na literatura, ao longo dos anos, passaram a preferir indicadores específicos de aplicação, como pode ser visto em [Avritzer e Weyuker 1997], [Macedo *et al*. 2010], [Matias *et al*. 2012], [Machida *et al*. 2013], [Matias *et al*. 2014] e [Matias *et al*. 2016].

A abordagem de análise diferencial de software, avaliada nesta pesquisa de mestrado, objetiva detectar o vazamento de memória em aplicações que executam em nível de usuário. Em [Matias *et al*. 2014] inicialmente foram considerados 66

indicadores de envelhecimento para vazamento de memória, entretanto, os autores descobriram os indicadores *resident set size* (RSS) e *heap usage* (HUS) foram os mais efetivos para tal propósito.

O RSS e o HUS são indicadores específicos de aplicação e usados para monitoramento dos efeitos de vazamento de memória em nível de usuário. Assim, o RSS significa a parte da memória ocupada por um processo que está mantido na memória principal (RAM) e considera várias partes do processo (texto, dados, pilha e *heap*) carregadas na memória principal. Se o valor total do espaço exigido por um processo exceder o RSS, então o resto é normalmente armazenado na área de *swap*. Um indicador que pode ser utilizado em combinação com o RSS, que identifica o quanto de memória virtual está ocupado pelo processo, incluindo na memória principal e na área de *swap* é *virtual memory size* (VSZ).

Já o HUS é outro indicador de envelhecimento a ser monitorado, capturando a memória usada da *heap* do processo, considerando apenas os blocos alocados. Alternativamente ao HUS, existe um indicador conhecido como *heap size* (HSZ) que captura o montante total utilizado da *heap* do processo, incluindo blocos alocados e não alocados. Ambos (HSZ e HUS) representam o que está na RAM e também o que está na área de *swap*.

Portanto, uma vez que o HUS captura apenas os blocos alocados e que outros indicadores, como os já citados, consideram montantes de memória de várias outras partes do processo, então é esperado que os efeitos do vazamento de memória irão refletir antes no HUS do que nos outros indicadores. Nesta pesquisa serão usados RSS e HUS nos estudos experimentais.

2.4 Trabalhos correlatos

Uma revisão da literatura, buscando principalmente por trabalhos relacionados ao vazamento de memória, foi realizada nessa etapa. Alguns desses trabalhos foram utilizados para explicar a fundamentação teórica nas Seções 2.2 e 2.3. Os critérios usados para seleção incluíram uma triagem de estudos, focando em vazamento de memória, desde 1995 até 2016. As fontes utilizadas nessa busca foram livros, publicações em revistas e anais de conferências científicas especializadas. Para obter tais fontes, foram utilizadas bibliotecas digitais como IEEE *Xplore Digital Library*, ACM *Digital Library* e a ferramenta de busca acadêmica *Google Scholar*. Além disso, os principais trabalhos relacionados com o estudo realizado em [Matias *et al*. 2014] também foram considerados.

Em [Matias *et al*. 2010c] é proposta e avaliada uma abordagem quantitativa que usa testes de vida acelerados (QALT) para reduzir o tempo de coleta de dados necessários para analisar sistemas que sofrem de envelhecimento de software. O método QALT foi originalmente idealizado para ser usado em sistemas de *hardware* e, portanto, necessitou de adaptação para ser aplicado em experimentos envolvendo envelhecimento de software. No trabalho os experimentos focaram na utilização do QALT para acelerar, em diferentes níveis, os efeitos do vazamento de memória em um servidor *web* Apache. Os resultados mostraram que o tempo para observar uma falha por envelhecimento foi significativamente reduzido em comparação ao estimado para a operação normal do sistema, permitindo uma avaliação mais detalhada dos tempos de falha do sistema avaliado se comparada a estudos em que não foi possível observar esse tipo de falha. Os autores concluíram que a principal contribuição do trabalho foi a de promover uma abordagem

sistematizada para acelerar a coleta de dados de falhas em sistemas que sofrem de envelhecimento de software.

Em [Carrozza *et al*. 2010] é apresentada outra abordagem para reduzir o tempo de coleta de dados necessários para detecção do envelhecimento. Para essa proposta, testes de sistemas com foco em vazamento de memória foram realizados na plataforma *middleware* industrial CARDAMOM [Cardamom 2017], colocando-a sob diferentes condições de estresse de cargas de trabalho para reduzir o tempo de coleta de dados. Essas condições de estresse se basearam em dois fatores: 1) o tempo entre consecutivas requisições e; 2) a quantidade de dados processados pelas requisições. Antes de executar os experimentos, testes preliminares foram efetuados para confirmar comportamentos relacionados a vazamento de memória no CARDAMON. Além disso, uma ferramenta foi desenvolvida para monitorar o vazamento, durante os experimentos, dentro do código fonte do *middleware*, buscando por *AR Faults*.

Os resultados mostraram que em um tempo experimental de 2 horas o consumo de memória aumentou de acordo com número de requisições. Além disso, quanto maior foi a quantidade de dados presentes nas requisições, maior foi o consumo de memória. A ferramenta criada para monitorar o vazamento dentro do código fonte do *middleware* mostrou que maior parte do consumo indiscreto de memória ocorreu devido ao serviço de *trace* do CARDAMON. Esses problemas foram relatados aos desenvolvedores da plataforma e ainda permanecem sem solução, levando os autores a concluir que pode ser devido à falta de competências adequadas para lidar com *bugs* de vazamento de memória durante o ciclo de vida do aplicativo. A ferramenta de localização de faltas de vazamento de memória no código fonte foi considerada uma das grandes contribuições desse trabalho. Ainda, os autores apontaram que as condições de estresse inseridas na carga de trabalho favoreceram um menor tempo de detecção do envelhecimento de software no CARDAMON.

Em [Machida *et al*. 2013] foi realizado um estudo para avaliar a efetividade da aplicação do teste de tendência *Mann-Kendall* na detecção do envelhecimento de software. Uma vez que muitos estudos na literatura em SAR consideram o aumento do uso de recursos do sistema como envelhecimento de software, então um dos principais objetivos desse trabalho foi mostrar que nem sempre uma tendência é causada por envelhecimento (ex. não liberação voluntária de memória – Seção 2.2), produzindo muitos falso-positivos. Experimentos foram realizados através de um gerador de carga de trabalho sintética, emulando o comportamento de um aplicativo de uso geral, vazando memória por meio de um *bug* introduzido nas funções de alocação e desalocação de memória. A carga de trabalho considerou diferentes taxas de vazamento (0%, 0,5% e 1,0%) e intensidades (baixa, média e alta) e os dados considerados para análise referem-se aos primeiros 5.000 segundos de cada experimento, onde foram monitorados, a cada 9 segundos, os indicadores RSS, HUS e FM (*free memory*).

Os resultados da aplicação do teste *Mann-Kendall* aos indicadores monitorados mostraram que o indicador FM não foi consistente em detectar o vazamento de memória gerando muitas taxas de falso-positivos e falso-negativos. O indicador HUS foi o mais robusto na detecção do vazamento de memória, produzindo o menor número de falsos alarmes. O indicador RSS mostrou boa precisão para intensidades alta e média, mas gerou falso-negativos sob intensidade baixa e taxa de vazamento de 0,5%. Como conclusão, os autores afirmaram que

testes de tendência não são por si só suficientes para detectar o envelhecimento e que um protocolo global precisaria ser seguido, respondendo perguntas tais como: qual a técnica de tendência mais apropriada para uma dada classe de indicadores? Como calcular o risco de falso-positivos e falso-negativos?

Entre os efeitos de envelhecimento mais citados em SAR está o vazamento de memória, conforme mostra os estudos realizados em [Cotroneo *et al*. 2011] e [Valentim 2016]. Em [Cotroneo *et al*. 2011] foi feito um levantamento considerando diversos artigos publicados em conferências, revistas e alguns importantes *workshops* relacionados com envelhecimento e rejuvenescimento de software. Com o objetivo de apresentar os aspectos que mais receberam atenção na literatura em 20 anos de pesquisa, alguns critérios de classificação foram adotados: 1) o tipo de análise conduzida; 2) o tipo de sistema estudado; 3) a dimensão dos indicadores analisados; 4) a abordagem de rejuvenescimento adotada ou proposta.

Os resultados mostraram que a maioria dos trabalhos conduzem análises baseadas em modelos – 41% contra 29% das análises baseadas em medidas. Os tipos de sistemas mais analisados são os não reais com 49%. Sistemas críticos são raramente avaliados (7%). Os indicadores mais considerados são os relativos à detecção do vazamento de memória (28%). Como conclusão, os autores afirmaram que a maioria das pesquisas na literatura são voltadas para análises baseadas em modelos que, geralmente, são validadas por modelos numéricos e/ou simulações. Além disso, a baixa porcentagem de experimentos considerando aplicações reais, principalmente sistemas críticos, é um fator preocupante. Com relação aos indicadores, os relativos à detecção do vazamento de memória são os que recebem mais atenção. Isso se deve ao fato de ser um problema comum em diversas linguagens e pela facilidade de detecção e medição.

Em [Valentim *et al*. 2016] também foi constatado que o vazamento de memória é o efeito de envelhecimento mais estudado na literatura. Nesse trabalho, uma abordagem sistemática de mapeamento foi utilizada, coletando 188 artigos de conferências e revistas internacionais recomendadas por experientes pesquisadores previamente entrevistados. Os resultados mostraram que 45,74% dos estudos focaram em envelhecimento de software – contra 54,25% em rejuvenescimento. Além disso, 46,81% dos trabalhos encontrados são do tipo analítico, seguido pelos estudos empíricos com 36,17%. A respeito das técnicas de modelagem, 65,42% das pesquisas encontradas aplicaram técnicas tais como: processos markovianos; redes de petri; análise de tendências; etc. A análise das classes de efeitos de envelhecimento mostrou que vazamento de memória foi o efeito mais recorrente com 62,90% dos estudos. Com relação aos indicadores, *free physical memory* (FM) e *swap space used* foram os mais utilizados nos primeiros anos de pesquisa, perdendo espaço com o passar dos anos para indicadores como RSS, HUS e JVMHS (*JVM Heap Size*). Finalmente, o sistema operacional mais prevalente nos estudos analisados foi o Linux/Unix com 85,07%. Em contrapartida, Windows foi foco apenas de 5,36% dos estudos.

Os autores concluíram que os resultados revelaram perspectivas promissoras, principalmente pelo fato de o número de estudos e novos autores estarem aumentando nos últimos anos. Além disso, apesar dos estudos do tipo analítico serem os mais frequentes, trabalhos que abrangem tanto tipo analítico quanto empírico têm se tornado uma tendência nos últimos anos. Ainda, como o

vazamento de memória tem sido efeito de envelhecimento mais estudado, então indicadores como RSS, HUS e JVMHS estão se tornando uma populares.

Em [Matias e Filho 2006] o vazamento de memória foi avaliado em um servidor *web* Apache onde uma política de rejuvenescimento foi proposta para minimizar o *dowtime* (tempo em que o sistema fica fora devido ao rejuvenescimento) a zero. Para isso, experimentos foram realizados no Apache considerando um gerador de carga de trabalho que englobou os seguintes fatores: 1) tamanho de página; 2) tipo de página e; 3) taxa de requisições. Todos esses fatores variaram em níveis baixo, médio e alto; e para defini-los foi realizado um trabalho de representatividade de cargas de trabalho, onde dados reais foram utilizados para configurar os valores dos fatores para cada nível. Uma combinação de 8 experimentos (cada um com duração de 8 horas), baseado nesses fatores e níveis, foi realizada e o indicador FM foi monitorado em cada um.

Os resultados dessa pesquisa mostraram que nas quatro primeiras combinações experimentais (cargas de trabalho mais leves) o Apache se manteve estável em termos de uso de memória. Porém, para as demais combinações houveram incrementos significativos de memória, confirmando um envelhecimento no processo do Apache para essas execuções. Assim, um agente de rejuvenescimento foi proposto para monitorar os processos do Apache e rejuvenescer quando os seus respectivos tamanhos, em termos de memória, excedessem determinado limite. Além disso, esse agente se aproveitou do sinal o SIGUSR1 – uma maneira agradável de reiniciar os processos escravos do Apache sem que transações sejam perdidas, não existindo *downtime* durante a execução.

Para avaliar o agente de rejuvenescimento foram usados dois cenários de carga de trabalho, um deles extraído de uma das 8 combinações usadas nos testes sem o agente (combinação com alto nível de estresse). Os resultados mostraram que o agente agiu conforme esperado para ambos os cenários, rejuvenescendo o Apache quando um determinado limite de memória foi alcançado. Além disso, a implementação do *zero dowtime* manteve constante o tempo de resposta do servidor que, diferentemente dos experimentos sem o agente, controlou o tamanho dos processos e não perdeu nenhuma requisição. Como conclusão, os autores afirmaram que a estratégia de *zero downtime* diminuiu o impacto na disponibilidade do sistema, evitando problemas detectados nos testes sem a execução do rejuvenescimento, tornando a aplicação do agente viável em casos de vazamento de memória no Apache.

Em [Araujo et. al 2011] também foi realizado uma análise dos efeitos do envelhecimento de software com foco em vazamento de memória e utilização da CPU. O ambiente computacional em nuvem Eucalyptus [Eucalyptus 2017], sofrendo de envelhecimento, foi selecionado para experimentos que exploraram consecutivos acessos a volumes de blocos remotos de armazenamento. Esses acessos são importantes na alocação flexível de máquinas virtuais (VMs). Um servidor *web* Apache também foi utilizado para, através do tempo de resposta, analisar o impacto do envelhecimento no Eucalyptus. Assim, o ambiente experimental foi composto de 6 máquinas (1 contendo componentes da arquitetura do Eucalyptus necessários para execução de VMs; 1 funcionando como cliente para monitorar o ambiente e realizar requisições aos componentes do Eucalyptus e as restantes hospedando 4 VMs cada – todas elas com o Apache). O tempo experimental foi de 100 horas.

A carga de trabalho foi configurada para acessar 50 volumes remotos de armazenamento atribuídos às VMs de modo a conectar-se e desconectar-se a eles repetidamente a cada 30 segundos, acelerando possíveis *AR Faults*. Além disso, solicitações HTTP foram emitidas, a cada minuto, da máquina cliente para os servidores Apache hospedados nas VMs, onde o tempo de resposta foi monitorado para medir o impacto dos efeitos do envelhecimento no Eucalyptus. Os resultados mostraram, através do monitoramento do RSS e da utilização da CPU, importantes sintomas de envelhecimento no Eucalyptus, especialmente no componente *Node Controller* (NC) – responsável pelas VMs e controle dos recursos físicos para mantê-las. Além disso, o tempo de resposta para requisições HTTP aumentou de acordo com o acúmulo dos efeitos do envelhecimento. Como conclusão, os autores afirmaram que o vazamento de memória presente no NC esteve diretamente relacionado à carga de trabalho que acessou blocos remotos de armazenamento das VMs.

Em [Mohan e Reddy 2015] foi estudada a correlação entre o vazamento de memória e a utilização de energia. Até então, nenhum outro estudo havia focado em estabelecer essa potencial correlação. Para esse propósito, uma abordagem baseada em medição, aplicando análise estatística de regressão a dados coletados foi utilizada. Um ambiente computacional composto de uma máquina hospedando 25 VMs, cada uma executando um servidor *web* Tomcat com vazamento de memória foi selecionado para o experimento. A carga de trabalho foi configurada para realizar requisições em intervalos de tempo constante (350 por segundo) e de tamanho estático (5kb) através um gerador de carga de trabalho, o Httperf. O tempo de resposta e uso da CPU de todas as 25 VMs, além do consumo de energia da máquina hospedeira foi monitorado em um período experimental de 180 horas.

Os resultados mostraram que a média da utilização de energia das 25 VMs, em relação ao tempo, apresentou um aumento de cerca de 0,344 *Watts*/hora. Como conclusão, os autores afirmaram que esse aumento no consumo de energia esteve diretamente ligado ao incremento do uso da CPU, que apresentou um crescimento médio de 0,437% por hora. Assim, uma alta correlação, evidenciada através de um coeficiente de determinação, confirmou que o servidor usa mais energia de acordo com o tempo de execução devido ao vazamento de memória.

Em [Macedo *et al*. 2010] é apresentado outro estudo sobre os mecanismos dos efeitos do envelhecimento, focando em apresentar a ocorrência interna do vazamento e fragmentação de memória em um conhecido alocador, o *ptmallocv2*. Para essa proposta, dois programas foram criados: *Mfrag* e *Mleak*. O primeiro consome toda a memória da *heap*, libera alguns blocos de forma não contígua e requisita um novo bloco. Como os blocos liberados não são grandes o bastante para acomodar o último pedido, o alocador emite um novo pedido ao sistema operacional, fragmentando a *heap*. O segundo intencionalmente causa vazamento de memória ao iterativamente usar um ponteiro que armazena uma referência para um bloco alocado e posteriormente receber um novo endereço, deixando órfão o bloco alocado anteriormente e saturando a *heap*. O *Mfrag* foi monitorado através do rastreamento das chamadas de sistema, já o *Mleak* foi monitorado pelo RSS.

Os resultados mostraram que, na execução do *Mfrag*, a cada vez que o programa requisita 8Kb, uma nova alocação é feita mesmo existindo 65Kb livres na *heap*, pois a *heap* está sofrendo de fragmentação de memória e prejudicando o desempenho do processo. Na execução do *Mleak* houve um aumento da memória

usado pelo processo de 300 Kb para 950 Kb. Os autores concluíram que dois problemas de envelhecimento surgem com a fragmentação de memória. O primeiro é o aumento do tamanho do processo (assim como em cenários de vazamento de memória). O segundo está relacionado à performance do processo. Cada solicitação para o SO vai exigir entrar em nível de *kernel* e depois voltar para nível de usuário, afetando significativamente o desempenho do processo.

Em [Matias *et al*. 2014] foi introduzida uma abordagem alternativa para mitigar os efeitos do envelhecimento, com foco em vazamento de memória, baseada em análise diferencial de software. O plano experimental consistiu em monitorar os indicadores RSS e HUS em um conjunto de experimentos, cada um durando 4 horas, contemplando vários cenários de cargas de trabalho. Um gerador de carga de trabalho sintética, emulando duas versões de um programa (uma com vazamento e outra sem), foi utilizado nos experimentos. Os dados dos indicadores foram combinados com técnicas estatísticas de detecção de tendências e de CEP para o cálculo dos valores de divergência. Esses valores de divergência foram usados para compor os gráficos de divergências, com o objetivo de detectar anomalias significativas e avaliar a melhor combinação técnica/indicador na detecção do vazamento de memória.

Os resultados mostraram que o indicador RSS apresentou muitos falso-positivos. Em contrapartida, o HUS apresentou os melhores resultados. No geral, a melhor combinação técnica/indicador para detecção do vazamento de memória foi as técnicas de CEP, fundamentalmente Soma Cumulativa, com o indicador HUS. Como conclusão, os autores afirmaram que a abordagem de análise diferencial de software foi efetiva na detecção do vazamento de memória, principalmente por conseguir diferenciar o comportamento natural de um sistema, sob carga de trabalho, do comportamento do envelhecimento. Além disso, como esperado, HUS foi mais robusto que RSS e as técnicas de CEP se mostraram melhor que técnicas de detecção de tendências frequentemente usadas na literatura.

Uma vez que [Matias *et al*. 2014] considerou uma variedade de cenários de cargas de trabalho sintéticas, então um estudo confirmatório e de replicação, avaliando essa abordagem frente a cenários de aplicações reais, foi realizado em [Matias *et al*. 2016]. Esse estudo selecionou o *proxy cache* Squid [Squid 2017a] com um *bug* de vazamento de memória confirmado, contendo uma versão alvo e uma versão base, para um conjunto experimental considerando dois cenários de cargas de trabalho definidas *adhoc*. Os mesmos indicadores, técnicas estatísticas e cálculo dos valores de divergência utilizados em [Matias *et al*. 2014] foram aplicados nesse estudo. Os resultados mostraram novamente que o indicador HUS foi mais efetivo que o RSS na detecção do vazamento de memória. Além disso, dos dois cenários de carga de trabalho considerados, em um as técnicas de CEP lideraram e em outro as técnicas de detecção de tendências. Os autores concluíram que o estudo contribuiu em diminuir limitações de validade externa da abordagem de análise diferencial, entretanto, mais experimentos frente a mais aplicações reais e a inclusão de cenários de cargas de trabalho definidos através de um estudo de representatividade se faziam necessários.

Diante dos trabalhos acima relacionados é importante ressaltar que, explorar falhas através de testes de curta duração pode não ser o método mais eficiente para expor todos os potenciais efeitos do envelhecimento que se apresentam em longo prazo, principalmente relacionado com vazamento de memória [Machida *et al*. 2013]. Além disso, outro fator preocupante são os números de falsos alarmes

cada vez mais frequentes em testes de pouca duração, principalmente por não combinar os melhores indicadores e/ou técnicas para detectar determinado efeito do envelhecimento [Machida *et al*. 2013].

Estudos discutidos anteriormente, como [Matias e Filho 2006]; [Carrozza *et al*. 2010]; [Araujo et. al 2011] e [Mohan e Reddy 2015], em geral, utilizaram uma metodologia que consitiu de processar dados de indicadores monitorados em técnicas de detecção de tendências para caracterizar o envelhecimento. O questionamento a se fazer sobre esses trabalhos é quanto ao risco do aparecimento de falsos alarmes, uma vez que os indicadores e técnicas utilizadas podem não ter sido as mais apropriadas para o efeito do envelhecimento analisado. Além disso, com exceção de [Matias e Filho 2006], todos os outros não realizaram nenhum estudo prévio de representatividade da carga trabalho, o que gera limitações de validade externa.

Assim, a abordagem de análise diferencial de software se torna interessante por conseguir distinguir, em uma aplicação sob carga de trabalho, o comportamento natural do comportamento do envelhecimento. Além disso, apresenta as melhores combinações técnica/indicador para detectar um determinado efeito do envelhecimento. Como os resultados de [Matias *et al*. 2014] e de [Matias *et al*. 2016] consideram cargas sintéticas definidas *adhoc*, onde inclusive o primeiro não utiliza aplicações reais, então esta pesquisa de mestrado tem como principal contribuição aplicar a abordagem de análise diferencial frente a aplicações reais considerando cargas definidas com comportamento o mais próximo ao uso real dessas aplicações.

Dessa forma, problemas de validade externa são diminuídos, permitindo uma generalização dos resultados a outros tipos de sistemas e, consequentemente, possibilitando uma melhor avaliação da eficácia da abordagem para uso na área de Envelhecimento de Software, oferecendo uma maior confiança em sua aplicação pela indústria de software.

3. Análise diferencial para Detecção do Envelhecimento de Software

3.1 Introdução

O objetivo deste Capítulo é descrever a técnica de análise diferencial de software voltada para detecção do envelhecimento e que é usada nesta pesquisa. Uma introdução ao funcionamento geral desta técnica, proposta por [Langner e Andrzejak 2013a] é realizada na Seção 3.2. A aplicação da técnica em uma abordagem para detecção de envelhecimento de software com foco em vazamento de memória, utilizando combinações de indicadores com técnicas estatísticas, conforme visto em [Matias *et al*. 2014], é descrita na Seção 3.3. As técnicas estatísticas utilizadas são divididas em dois tipos: detecção de tendências e de Controle Estatístico de Processos (CEP). Essas técnicas são descritas, respectivamente, nas Seções 3.4 e 3.5. Finalmente, a Seção 3.6 explica como as técnicas estatísticas são combinadas com os indicadores monitorados. Essa combinação é utilizada para gerar valores de divergência baseados na análise diferencial de software. Os valores, por sua vez, compõem os gráficos de divergências que são avaliados em busca de alterações significativas que caracterizem vazamentos de memória.

3.2 Funcionamento geral

A análise diferencial de software é uma técnica que visa explorar informações obtidas através de comparações comportamentais, em tempo de execução, de duas versões do mesmo software. Esta técnica foi introduzida por [Langner e Andrzejak 2013a], onde foi usada para uma detecção não automatizada de envelhecimento de software através da comparação visual de dados coletados de indicadores de envelhecimento em experimentos com duas versões de um sistema. Posteriormente, a mesma técnica foi utilizada em [Langner e Andrzejak 2013b] para detectar vazamentos de memória em programas Java.

Para entender melhor a técnica em questão, considere um software de uso comum que é desenvolvido e projetado para lançar atualizações caracterizando uma nova versão. Assim, a análise diferencial de software propõe comparar, sob testes de integração e desempenho, analisando indicadores de envelhecimento, a nova versão desse software (versão alvo) com sua versão imediatamente anterior, atualmente no ambiente operacional (versão base). Portanto, se a nova versão desse sistema contém em suas novas funcionalidades uma (ou mais) *AR Faults* que

não estão presentes na versão anterior, então anomalias poderão ser detectadas pela comparação dos dados de indicadores e os desenvolvedores constatarão que a nova versão contém envelhecimento de software, evitando que seja lançada no ambiente de produção. A Figura 3.1 apresenta o funcionamento geral da técnica de análise diferencial de software descrito acima.

A efetividade da análise diferencial de software é diretamente influenciada por dois fatores: 1) os indicadores de envelhecimento monitorados e; 2) a garantia de ativação da(s) *AR Fault*(*s*) caso ela(s) exista(m) na versão alvo – lembrando que não há certeza sobre a presença de envelhecimento nessa versão, por isso mesmo ela está sob investigação. Com relação aos indicadores, muitos deles são imprecisos e sofrem influências de vários ruídos, conforme mostrado por [Machida *et al*. 2013]. Além disso, outros podem não ser influenciados pelo efeito de envelhecimento que se deseja detectar. Portanto, para cada efeito é necessário escolher os indicadores mais adequados, ou seja, que serão mais efetivos na detecção e menos propensos a falsos alarmes. Recomenda-se buscar por estudos na literatura que apresentem comparações de qualidade de detecção para determinado efeito de envelhecimento. Em relação à garantia de ativação da *AR Fault*, é preciso que, durante os testes realizados na análise diferencial, a *AR Fault* presente na versão alvo seja ativada. Caso contrário, independente do indicador utilizado, o envelhecimento não será detectado. Portanto, a realização de testes que cubram grandes partes das alterações implementadas, na versão alvo, aumenta as chances da *AR Fault* ser ativada e, consequentemente, o envelhecimento ser detectado.

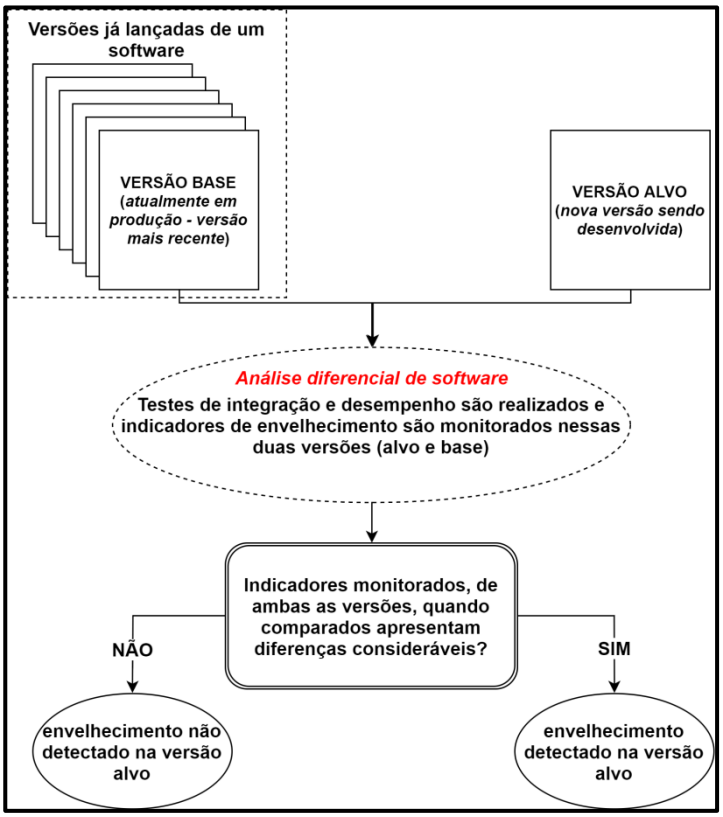


Figura 3.1. Funcionamento geral da técnica de análise diferencial de software.

3.3 Aplicação para detecção de vazamento de memória

A técnica de análise diferencial de software foi aplicada para detecção do envelhecimento de software, com foco em vazamento de memória, em [Matias *et al*. 2014]. Nesse trabalho, diferentemente de [Langner e Andrzejak 2013a], as comparações entre as versões alvo e base não foram realizadas visualmente (apenas observando os valores dos indicadores) e sim através do processamento de séries temporais de indicadores em técnicas estatísticas de detecção de tendências e CEP.

A utilização de indicadores de envelhecimento em combinação com técnicas estatísticas de detecção de tendências é uma metodologia frequentemente encontrada em trabalhos na literatura. Além disso, a maioria desses estudos não utilizam indicadores e/ou técnicas mais apropriadas para detectar determinado tipo de efeito de envelhecimento, apresentando muitos falso-positivos e falso-negativos [Machida *et al*. 2013]. Pensando em solucionar esses problemas é que a abordagem de [Matias *et al*. 2014] traz contribuições que vão além do trabalho de [Langner e Andrzejak 2013a], apresentando a melhor combinação técnica/indicador na detecção do vazamento de memória. Uma vez que utiliza como ponto central a análise diferencial de software, então problemas de falso-negativos e falso-positivos são diminuídos pela distinção entre o aumento natural de memória do comportamento real do vazamento.

Os resultados apresentados por [Matias *et al*. 2014] são baseados em experimentos realizados por meio de um gerador de carga de trabalho sintética, emulando o comportamento de um aplicativo de uso geral vazando memória. Assim, com a proposta de diminuir limitações de validade externa, essa pesquisa considerou as seguintes etapas metodológicas:

1. **Selecionar as aplicações reais:** as aplicações reais utilizadas contém vazamento de memória. Portanto, para encontrá-las foi preciso investigar fóruns oficiais de relatos de *bugs* de software. Assim, ao encontrar um relato de *bug* de vazamento para determinada aplicação, foi necessário realizar testes preeliminares para confirmar a sua existência. O objetivo dessa pesquisa foi avaliar, no mínimo, quatro (4) aplicações reais.
2. **Encontrar as versões base e alvo**: depois de confirmar o vazamento de memória em uma aplicação, foi preciso realizar novos testes preeliminares para encontrar a versão anterior mais recente em que o *bug* não se manifestasse – conhecida como versão base. Consequentemente, a versão imediatamente posterior à base, com o vazamento de memória, foi selecionada como versão alvo. Aplicações selecionadas onde uma versão base não foi encontrada nesses testes preeliminares, denotando que o vazamento sempre existiu desde a primeira versão, foram descartadas. Assim, no total 4 aplicações atenderam as etapas 1 e 2, são elas: Inkscape; Lighttpd; PostgreSQL e Squid. Embora a versão alvo seja uma versão sob investigação, não havendo certeza sobre a presença do envelhecimento, em [Matias *et al.* 2014], [Matias *et al.* 2016] e nesta pesquisa, a versão alvo utilizada continha envelhecimento confirmado pois o objetivo principal foi

demonstrar, através de experimentos controlados, a efetividade da análise diferencial na detecção do vazamento de memória.

3. **Realizar um estudo de representatividade de carga de trabalho para cada *bug* de vazamento:** as aplicações reais selecionadas e seus respectivos *bugs* de vazamento possuem padrões específicos de ativação (*Aging Factors*). Esses padrões são responsáveis por fazerem o *bug* se manifestar. Assim, cada padrão de ativação foi estudado para incorporá-lo na carga de trabalho utilizada para cada aplicação avaliada, resultando em um planejamento experimental com cenários de cargas de trabalho que representem o comportamento real de cada aplicação frente ao padrão de ativação sendo utilizado. É importante mencionar que [Matias *et al*. 2014] propôs vários cenários de cargas de trabalho incluindo modos (constante e variado) e intensidades (alta, normal e baixa). Com isso, tais cenários foram considerados para cada aplicação analisada. As fontes de busca para a caracterização foram obtidas por meio de bibliotecas digitais, tais como *IEEE Xplore Digital Library*; *ACM Digital Library* e a ferramenta de busca acadêmica *Google Scholar*.
4. **Realizar experimentos com as aplicações selecionadas, a fim de monitorar seus indicadores de envelhecimento:** Os cenários de cargas de trabalho definidos na etapa anterior foram exercidos com cada aplicação, nas versões alvo e base, e os indicadores de envelhecimento foram monitorados. Para cada aplicação, um experimento foi realizado com 10 replicações por cenário avaliado, em que cada replicação monitorou os indicadores *resident set size* (RSS) e *heap usage* (HUS). Portanto, para cada replicação tem-se duas séries temporais, uma por indicador. Cada replicação durou quatro (4) horas tal como em [Matias *et al*. 2014], o que demonstrou ser suficiente para observar efeitos de vazamento em uma análise diferencial. Os indicadores monitorados também são os mesmos usados em [Matias *et al*. 2014]. Assim, para cada aplicação, conforme mostra a Tabela 3.1, um total de 320 horas foram utilizadas para a realização de experimentos.

Tabela 3.1. Visão geral do tempo experimental para cada cenário de carga de trabalho.

Cenário de carga	Versão alvo		Versão base	
	Replicações	Tempo total	Replicações	Tempo total
Constante e alta	10x 4 horas	40 horas	10x 4 horas	40 horas
Constante e normal	10x 4 horas	40 horas	10x 4 horas	40 horas
Constante e baixa	10x 4 horas	40 horas	10x 4 horas	40 horas
Variada	10x 4 horas	40 horas	10x 4 horas	40 horas
Total	40x 4 horas	160 horas	40x 4 horas	160 horas

5. **Estimar as séries temporais médias:** Dado que cada replicação produziu 10 séries temporais por indicador, faz-se necessário estimar a série temporal média para cada conjunto de 10 séries temporais analisado. Essa série temporal média foi estimada pelo método *Dynamic Time Warping* (DTW) [Sakoe e Chiba 1971], a fim de reduzir os efeitos de defasagem que

normalmente existem entre as múltiplas séries do mesmo conjunto analisado.

6. **Análise das séries temporais médias:** as séries temporais médias dos indicadores obtidas no passo anterior foram utilizadas para o processamento nas técnicas estatísticas de detecção de tendências (Regressão Linear, Média Móvel, Mediana Móvel e Filtro Hodrick Prescott) e técnicas de CEP (Gráfico de Controle Shewhart, Média Móvel Exponencial Ponderada e Soma Cumulativa). Para realizar comparações de forma justa, essas técnicas estatísticas são utilizadas para compor o cálculo dos valores de divergência. Esses valores são uma diferença normalizada entre as versões alvo e base e são explicados detalhadamente na Seção 3.6.
7. **Analisar os gráficos de divergência:** os valores de divergência calculados foram plotados em gráficos de divergência para facilitar a análise e apresentar a melhor combinação técnica/indicador para cada cenário de carga e aplicação.

A Figura 3.2 apresenta uma visão geral dos passos adotados para a avaliação da validade externa da análise diferencial de software aplicada na detecção do vazamento de memória.

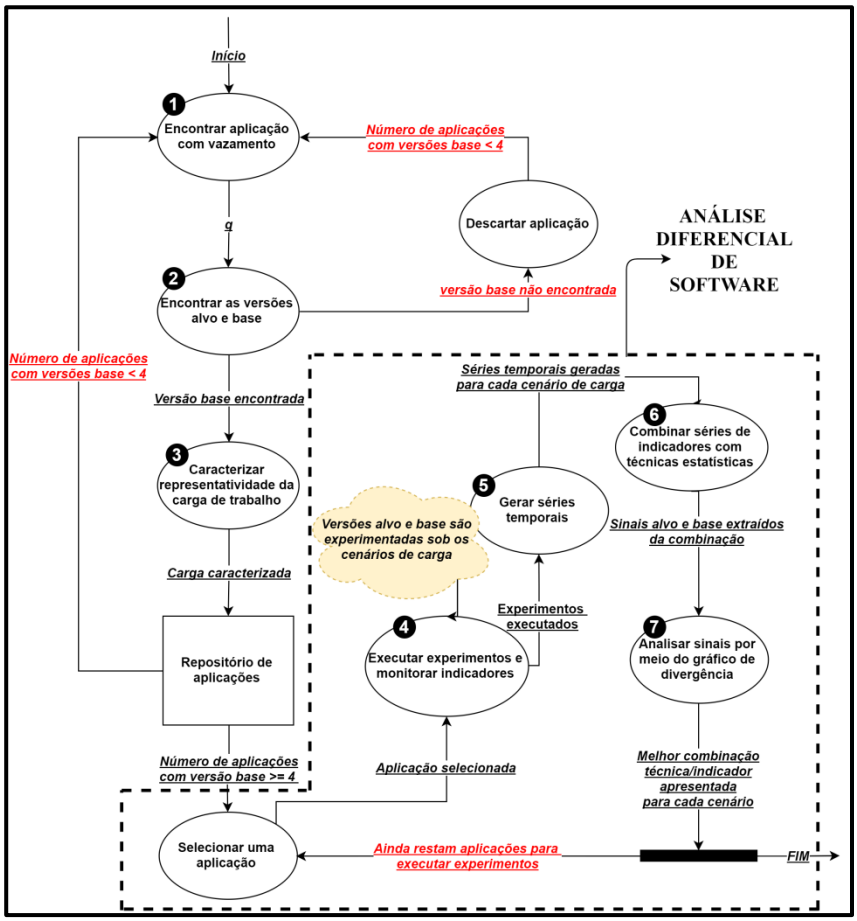


Figura 3.2. Visão geral das etapas de avaliação da validade externa da técnica de análise diferencial aplicada ao vazamento de memória.

3.4 Técnicas estatísticas de detecção de tendências

As técnicas de detecção de tendências têm como princípio a identificação de padrões através do processamento de séries temporais [Benattar e Trébuchet 2011]. Uma série temporal é qualquer conjunto de observações ordenadas e espaçadas igualmente no tempo [Gujarati 2000] [Morettin e Toloi 2006]. Além de serem analisadas em termos de tendências, também podem ser investigas objetivando revelar componentes de sazonalidade e outras características como valores extremos (ruídos) e auto correlação [Morettin e Toloi 2006].

Na literatura em SAR é comum encontrar trabalhos que utilizam técnicas de detecção de tendências para compreender melhor o fenômeno do envelhecimento de software e estimar o tempo em que um determinado recurso irá esgotar (ex. esgotamento da memória) e/ou o sistema irá degradar [Machida *et al*. 2013]. Esses trabalhos são classificados como estudos baseados em medições, uma vez que realizam predição baseada na análise de tendência dos indicadores monitorados.

Recentemente, o estudo [Valentim *et al*. 2016] mostrou que as técnicas de detecção de tendências estão presentes em uma grande parte dos trabalhos na literatura, prevalecendo em cerca de 38%; seguidas por modelos estocásticos como processos markovianos e redes de petri, que somam, respectivamente, 36% e 25%. Alguns exemplos de técnicas utilizadas em estudos na literatura para detectar tendências são: *Sen's slope* [Li *et al*. 2002] [Bovenzi *et al*. 2012], Teste de *Mann-Kendal* [Bovenzi *et al*. 2012], Regressão Linear [Mohan e Reddy 2015], Média Móvel Autoregressiva Integrada (ARIMA) [Magalhães e Silva 2010], *Holt-Winters* [Magalhães e Silva 2010], dentre outras.

Dado que esta pesquisa buscou avaliar a validade externa da abordagem apresentada em [Matias *et al*. 2014], então a fim de comparar os dois trabalhos, foram usadas as mesmas técnicas de detecção de tendências, a saber: Regressão Linear, Média Móvel, Mediana Móvel, e Filtro Hodrick-Prescott.

3.4.1 Média móvel

O termo média móvel é usado para descrever o procedimento onde a média é calculada se movendo através de uma série de valores. Por exemplo, considerando x_3 a terceira observação de uma série temporal de tamanho n, a média móvel irá incluir em seu cálculo algumas observações anteriores a x_3 (ex. x_2 e x_1) e sua próxima observação (x_4) [Hyndman 2014]. Uma característica útil dessa técnica é sua utilização para suavizar flutuações de curto prazo e destacar tendências ou ciclos de longo prazo (os parâmetros da média móvel são ajustados para determinar tais prazos) [Montgomery *et al*. 2008]. Suavizar uma série temporal significa criar uma função aproximada que capture padrões importantes nos dados, deixando de fora ruídos.

Existem vários tipos de médias móveis que podem ser utilizadas na análise de séries temporais. A escolha de qual tipo varia de acordo o peso que se deseja dar as observações (mais recentes / menos recentes). Os três tipos de média mais comuns são: 1) média simples; 2) média ponderada e; 3) média exponencial [Montgomery *et al*. 2008]. Nesta pesquisa de mestrado, assim como em [Matias *et al*. 2014], utilizou-se a média móvel simples, que atribui igual importância a todas

as observações na média [Hyndman 2014]. Portanto, o cálculo da média móvel utilizada neste trabalho segue o disposto na Equação 1:

$$MA_t = \frac{y_t + y_{t-1} + \cdots + y_{t-n+1}}{n} \quad (1)$$

Onde:

- MA_t*:* é a média móvel no período t.
- n*:* é o comprimento da média móvel.
- y_t*:* é uma observação no período t.

Assim, à medida que cada nova observação fica disponível, é adicionada à soma a partir da qual a média móvel é calculada e a observação mais antiga é descartada [Montgomery *et al*. 2008].

3.4.2 Mediana Móvel

A técnica estatística baseada na mediana móvel é uma excelente alternativa à média móvel, principalmente por ser mais eficaz na suavização de dados quando a série temporal está contaminada de ruídos [Montgomery *et al*. 2008]. O princípio de aplicação dessa técnica é o mesmo da média móvel, entretanto, ao invés de aplicar média, aplica-se a mediana aos dados, conforme representado pela Equação 2:

$$MM_t = \text{med}(y_t + y_{t-1} + \cdots + y_{t-n+1}) \quad (2)$$

Onde:

- MM_t: é a mediana móvel no período t.
- n: é o comprimento da mediana móvel.
- y_t: é uma observação no período t.

Lembrando que para um número ímpar de elementos n calculamos a mediana considerando o elemento $\frac{n+1}{2}$. Já para um número par de elementos n, a mediana é definida como a média dos dois elementos do meio: $\frac{n}{2}$ e $\frac{n}{2} + 1$ [Farias e Laurencel 2006] [Weisstein 2017].

3.4.3 Regressão Linear

A análise baseada em regressão é uma técnica utilizada para investigar e modelar o relacionamento entre variáveis. Sua aplicação é conhecida em todas as áreas de estudo, sendo talvez a técnica estatística mais usada [Montgomery *et al*. 2012]. Assim, idealizando que exista uma forte relação entre duas variáveis *y* e *x*, a equação de uma reta que representa a relação dessas variáveis é dada por:

$$\text{y} = \beta_0 + \beta_1 x \quad (3)$$

Onde:

- β_0 *é a constante de regressão*: constante que representa o intercepto da reta com o eixo y.

- β_1 *é o coeficiente de regressão*: constante que representa a inclinação da reta (variação de *y* em função da variação de uma unidade de *x*).
- x: representa o fator explicativo da equação.

Entretanto, nem sempre os valores resultantes da reta ($\beta_0 + \beta_1 x$) serão os mesmos dos valores observados de *y*. Essa diferença pode significar que: 1) as variações de *y* não são perfeitamente explicadas pelas variações de *x*; 2) existem outras variáveis das quais *y* depende ou; 3) os valores de *x* e *y* são obtidos de uma amostra que apresenta distorções em relação à realidade. Essa diferença é chamada de erro ou desvio (ε). Assim, é conveniente considerar que ε é uma variável aleatória que inclui todos os fatores residuais e os possíveis erros de medição [Montgomery *et al*. 2012]. Dessa forma, uma forma mais apropriada está representada pela Equação 4:

$$\mathrm{y} = \beta_0 + \beta_1 x + \varepsilon \tag{4}$$

Tal modelo é conhecido como regressão linear, onde, habitualmente *x* é conhecido como variável independente (explicativa) e *y* de variável dependente (resposta) [Montgomery *et al*. 2012]. Quando existe mais de uma variável explicativa, o processo é chamado de regressão linear múltipla [Freedman 2009]. Uma forma muito utilizada para diminuir o erro ε é a aplicação do Método dos Mínimos Quadrados (MMQ). Tal método tem por objetivo encontrar o melhor ajuste para um conjunto de dados tentando minimizar a soma dos quadrados da diferença entre o valor estimado e os dados observados [Montgomery *et al*. 2012]. Assim, a soma de quadrados de distâncias entre os pontos é minimizada, garantindo uma relação funcional entre *x* e *y*, com o mínimo de erro possível.

Ao analisar uma série temporal usando um modelo estatístico como o de regressão linear, é conveniente supor que os parâmetros do modelo são constantes ao longo do tempo. Entretanto, muitas vezes os valores desses parâmetros podem mudar consideravelmente, não sendo razoável supor que tais parâmetros sejam constantes. Uma forma muito comum de avaliar a constância dos parâmetros de um modelo é os calcular usando uma janela de rolamento de tamanho fixo. Tal técnica é conhecida como análise contínua (*rolling analysis*) e permite capturar mudanças nesses parâmetros durante algum ponto da série temporal analisada [Zivot e Wang 2006]. Essa pesquisa de mestrado utilizou a regressão linear em combinação com a análise contínua: *rolling linear regression.* Portanto, para uma dada janela de rolamento de tamanho $n < T$, a *rolling linear regression* pode ser expressa pela Equação 5:

$$y_t(\mathrm{n}) = \beta_{0t}(n) + \beta_1 x_t(n) + \varepsilon_t(n), \qquad t = n, \ldots, T \tag{5}$$

Assim, as n observações em $y_t(\mathrm{n})$ e $x_t(n)$ são os n valores mais recentes dos tempos $t - n + 1$ até t [Zivot e Wang 2006].

3.4.4 Filtro Hodrick-Prescott

A técnica estatística *Hodrick-Prescott* (HP) é uma técnica que tem por objetivo remover o componente cíclico e obter uma representação de curva suavizada em uma série temporal de dados que é mais sensível a flutuações de

longo prazo do que de curto prazo, sendo universalmente usada na macroeconomia [Hodrick e Prescott 1980]. Portanto, uma dada série temporal y_t, suavizada pela aplicação da técnica HP, é a soma de um componente de crescimento g_t e de um componente cíclico c_t:

$$y_t = g_t + c_t, \ para\ t = 1, \dots, T \tag{6}$$

Onde:

- g_t: é a soma dos quadrados de sua segunda diferença.
- c_t : representa os desvios de g_t.

Assim, a técnica *Hodrick-Prescott* (HP) pode ser definida pela Equação 7:

$$Min_{\{g_t\}_{t=-1}^{T}} \left\{ \sum_{t=1}^{T} c_t^2 + \lambda \sum_{t=2}^{T-1} [(g_t - g_{t-1})(g_{t-1} - g_{t-2})]^2 \right\} \tag{7}$$

Onde:

- $c_t^2 = y_t + g_t$.
- T: denota o tamanho da amostra.
- λ: é o parâmetro (não negativo) de ajuste da suavização, quanto maior for λ maior será a suavização da série. Hodrick e Prescott recomendam que quanto maior a série de dados, maior deve ser o ajuste de suavização. Assim, baseado nas séries temporais analisadas nesta pesquisa, assim como em [Matias *et al.* 2014], utilizou-se se λ = 5.000.
- $y = (y_1, \dots, y_t)$: é a série de dados a ser suavizada.

3.5 Técnicas de controle estatístico de processos (CEP)

As técnicas de controle estatístico de processos (CEP) foram incialmente criadas com a intenção de contribuir na ampliação da qualidade de processos industriais, tais como manufatura, químicos, etc. Para alcançar esse objetivo, as técnicas de CEP trabalham em cima das características da qualidade dos processos analisados.

O controle da qualidade por meio do CEP iniciou-se em meados dos anos 20, nos Estados Unidos, por meio de gráficos de controle criados por Walter A. Shewhart – físico e engenheiro da empresa de telefonia *Bell Telephone Laboratories* [Montgomery 2009]. Shewhart apresentou como ideia central que todo processo produtivo sempre possuirá alguma variabilidade, mesmo que este processo gere produtos da melhor qualidade. Tal variabilidade pode se apresentar de duas formas em um processo: 1) inerente ao processo, ou seja, sempre está presente e nunca pode ser eliminada, conhecida como causas aleatórias e; 2) não inerente ao processo, não é natural ao processo, altera a distribuição de variáveis aleatórias sob observação, prejudicando e comprometendo a qualidade do processo, conhecida como causas especiais [Montgomery 2009] [Costa *et al.* 2005].

Pode se observar que o principal objetivo do CEP é verificar a presença de causas especiais através de um sistema de inspeção por amostragem. Dessa forma,

um processo fora de controle estatístico apresenta causas aleatórias e especiais e a sua estabilidade pode ser alcançada, através do CEP, quando apenas causas aleatórias, inerentes ao próprio processo, estão presentes, tornando-o sob controle estatístico.

Na literatura em SAR, o primeiro uso do CEP para detectar efeitos de envelhecimento de software foi em [Matias *et al*. 2014]. A ideia central de sua utilização é semelhante ao que é feito com as técnicas de detecção de tendências, ou seja, séries temporais de indicadores monitorados são processadas por técnicas de CEP objetivando identificar se o processo na versão alvo está fora de controle (desvia-se significativamente do padrão observado para a versão base).

3.5.1 Gráficos de Controle

Os gráficos de controle foram criados por Shewhart como uma técnica simples para distinguir entre causas aleatórias e causas especiais. Os gráficos de controle analisam dados provenientes de amostragem e ajudam a identificar elementos que não são inerentes a um processo. Um exemplo de gráfico de controle padrão, mostrando a característica da qualidade de um processo versus a quantidade de amostras no tempo, é mostrado na Figura 3.3.

O gráfico contém uma linha central (*LC*), representando o valor médio da característica da qualidade correspondente ao processo monitorado, e outras duas linhas horizontais, conhecidas como limite inferior de controle (*LIC*) e limite superior de controle (*LSC*), determinando os limites para a variabilidade do processo. Quando a variabilidade do processo está dentro dos limites do gráfico, então o processo é considerado sob controle estatístico e nenhuma ação é necessária, caso contrário, está fora de controle e investigações e ações corretivas são necessárias para remover causas especiais responsáveis por esse comportamento [Costa *et al.* 2005] [Montgomery 2009] [Carvalho 2012]. Apesar de existirem muitas variações na forma de usar os gráficos de controle, os gráficos de Shewhart são um dos principais métodos utilizados no CEP.

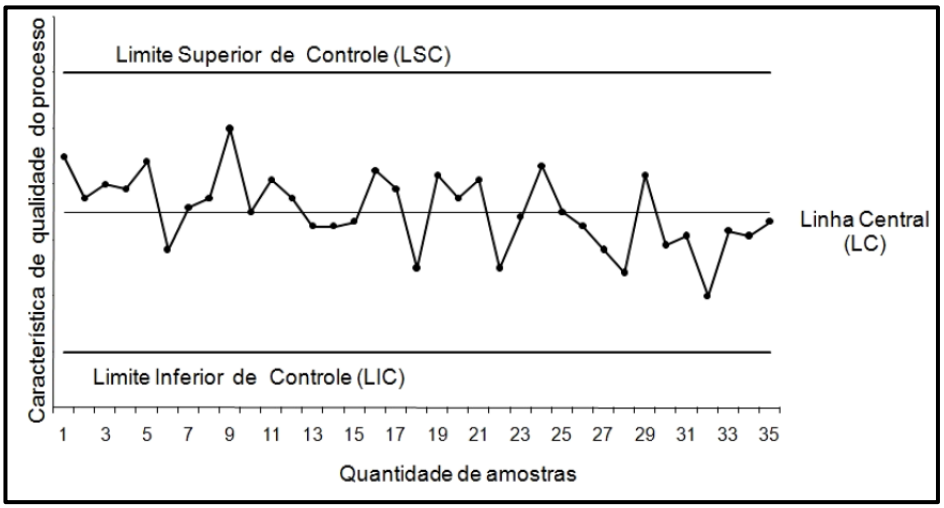


Figura 3.3. Exemplo de gráfico de controle padrão.
Traduzido de: [Montgomery 2009].

3.5.2 Gráficos de Controle de Shewhart

Os gráficos de Controle de Shewhart são amplamente utilizados e possuem várias formas. Nesta pesquisa, assim como em [Matias *et al.* 2014], utilizou-se o gráfico de controle para observações individuais, uma vez que as séries coletadas são compostas de valores individuais que representam um resíduo de monitoramento. Além disso, esses gráficos são indicados para se detectar grandes mudanças no processo.

3.5.2.1 Gráficos de Controle de Shewhart para observações individuais

Sua utilização mais comum é na detecção de mudanças grandes na média do processo [Costa 2005] [Montgomery 2009] e sua linha central e limites são definidas por:

$$LSC = \bar{x} + 3\frac{\overline{MR}}{d2} \quad (8)$$

$$LC = \bar{x} \quad (9)$$

$$LIC = \bar{x} - 3\frac{\overline{MR}}{d2} \quad (10)$$

Onde:

- $\overline{d_2}$: é uma constante tabelada que depende do valor da amostra, definida nesta pesquisa como 1,128.
- MR: é a media das amplitudes móveis MR_i calculada quando o processo estava sob controle, definida por:

$$M\,R = \frac{1}{m}\sum_{t=1}^{m} MR_i, \text{ com } M\,R_i = |f_i - f_{i-1}| \quad (11)$$

Existem também características da qualidade que não podem ser mensuráveis. Por exemplo, suponhamos que estamos examinando um recipiente e classificando-o em duas categorias: conformidade e não conformidade [Carvalho 2012]. Se o recipiente satisfaz os requisitos de uma ou mais características da qualidade, então está em conformidade. Tais características que não podem ser mensuráveis são conhecidas como de atributos [Montgomery 2009].

Talvez a vantagem mais importante de se utilizar gráficos de controle para variáveis é que geralmente fornecem muito mais informações necessárias para a indicação de problemas eminentes e tomada de decisões para medidas corretivas, antes que quaisquer defeitos sejam realmente produzidos [Montgomery 2009]. Nesta pesquisa de mestrado nos concentraremos nos gráficos de controle de Shewhart para características que são mensuráveis (variáveis). Uma descrição mais detalhada sobre toda a família de gráficos de Shewhart pode ser encontrada em [Montgomery 2009].

3.5.3 Gráficos de Controle de CUSUM

A desvantagem dos gráficos de controle de Shewhart é não acumular informações de observações passadas, se tornando insensíveis a pequenos

deslocamentos do processo. Assim, os gráficos de Shewhart tornam-se menos úteis em monitorar problemas quando o processo tende a operar em controle e causas especiais não resultam, tipicamente, em grandes transtornos ou distúrbios no processo [Montgomery 2009].

Uma alternativa em relação ao gráfico de controle de Shewhart e que acumula informações na estatística analisada são os Gráficos de Controle de Soma Acumulada (*Cumulative Sum Control Charts* – CUSUM). Esses gráficos podem detectar pequenas mudanças na distribuição da característica da qualidade monitorada e, ainda assim, manter um controle seguro do processo [Woodall 1985]. Além disso, o gráfico CUSUM incorpora diretamente toda a informação na sequência dos valores da amostra, plotando as somas cumulativas dos desvios dos valores da amostra de um valor alvo [Montgomery 2009].

O procedimento para utilização do gráfico CUSUM se baseia em coletar sucessivas amostras de tamanho n na qual é obtida a estatística da Soma Acumulada. Tal técnica pode ser aplicada tanto na construção do gráfico para observações individuais quanto para observações amostrais das médias de subgrupos racionais, permitindo ao analista dar mais ênfase em mantê-lo centrado no seu valor alvo [Lucas 1985]. Para as observações amostrais de tamanho ($n > 1$), o CUSUM é a soma acumulada dos desvios da média amostral com relação ao valor alvo. Para as observações individuais, a estatística utilizada é a soma acumulada dos desvios de cada valor individual com relação à medida dada pela hipótese sendo testada. O gráfico CUSUM é apresentado por [Montgomery 2009] como sendo aplicado à média e a variabilidade do processo, sendo possível projetar procedimentos CUSUM para outras variáveis estatísticas, tais como a amplitude e desvio padrão de subgrupos.

Para aplicar a técnica de soma acumulada o analista deve conhecer o valor alvo μ_0, que é o valor desejado para a média do processo. O procedimento se inicia com o cálculo dos desvios do valor alvo, projetando a diferença entre o valor desejado (média amostral) e o valor alvo μ_0. Através desse desvio a soma acumulada é iniciada. Assim, a soma acumulada C_i para o i-ésimo período, é a soma de todos os desvios do valor alvo até o período i [Montgomery 2009], dada pela seguinte equação:

$$C_i = \sum_{j=1}^{i} (X_j - \mu_0) \tag{12}$$

Onde:

- C_i: é a soma cumulativa da j-ésima amostra.
- X_j: é a j-ésima observação de um gráfico CUSUM.

Através da Equação 12 é possível realizar a plotagem CUSUM da amostra. Entretanto, para tornar tal plotagem em um gráfico de controle é necessário inserir limites de controle estatísticos [Montgomery 2009]. Uma das maneiras para representar os CUSUMS em gráficos de controle é o CUSUM Tabular.

3.5.3.1 CUSUM Tabular

O CUSUM Tabular utiliza o algoritmo de soma acumulada para calcular as somas unilaterais que, através do gráfico, são comparadas com um intervalo de

decisão *H*. Caso o valor da soma seja maior que este intervalo, então o processo está fora de controle. Tal procedimento é usado no monitoramento da média de um processo cuja estatística de controle são observações individuais ou médias de subgrupos racionais. Existem várias formas de construir um gráfico de controle CUSUM Tabular, utilizaremos a metodologia apresentada em [Montgomery 2009].

Para detectar mudanças positivas e negativas, o CUSUM Tabular utiliza duas estatísticas unilaterais: C_i^+ (plano superior) e C_i^- (plano inferior). Tais planos são caracterizados por um único parâmetro denominado intervalo de decisão ou limite de controle, representado por: $H = \pm h$ [Page 1954]. A estatística C_i^+é a soma acumulada dos desvios positivos, ou seja, desvios acumulados de μ_0 que estão acima do valor alvo, enquanto C_i^- é a soma acumulada dos desvios negativos, ou seja, desvio acumulado de μ_0 que está abaixo do valor alvo [Carvalho 2012]. As estatísticas unilaterais (C_i^+ e C_i^-) são denominadas CUSUM superior e inferior e são calculadas conforme as seguintes equações:

$$C_i^+ = \max[0, x_i - (\mu_0 + K) + C_{i-1}^+] \quad (13)$$
$$C_i^- = \max[0, (\mu_0 + K) - x_i + C_{i-1}^-] \quad (14)$$

Onde:

- Os valores iniciais padrão são zero para $C_0^+ = C_0^- = 0$.
- x_i: é a i-ésima observação controlada.
- μ_0: é a estimativa da média do processo.
- *K:* é um valor tabelado [Montgomery 2009], também conhecido como valor de tolerância, e é escolhido como a metade do valor que se tem interesse em detectar rapidamente (entre o valor pretendido μ_0 e o valor da média fora de controle μ_1).

Uma vez que a mudança é expressa em unidades de desvios padrão, quando $\mu_1 = \mu_0$, então *K* representa a metade da magnitude desta mudança:

$$K = \frac{\delta}{2}\frac{\sigma}{\sqrt{n}} = \frac{|\mu_1 - \mu_0|}{2} \quad (15)$$

Onde:

- δ: é o tamanho da mudança que se deseja detectar em unidades de desvios padrão.
- σ: é o desvio padrão.
- μ_0: é o valor pretendido e.
- μ_1: é o valor da média fora de controle.

A magnitude de variação que buscamos detectar com o gráfico CUSUM está diretamente relacionada com *o* fator de sensibilidade *k*. Quanto menor este fator, menor será a variação que o gráfico será capaz de detectar e, consequentemente, maior será a sensibilidade do gráfico. A escolha de um valor para *k* nem sempre é uma tarefa fácil e é importante selecionar valores razoáveis para *k* e para os limites de controle com o objetivo de obter um gráfico CUSUM adequado. Segundo [Montgomery 2009], o melhor modelo matemático para selecionar estes valores é através das equações abaixo:

$$K = \mathrm{k}\,\frac{\sigma}{\sqrt{n}} \tag{16}$$

$$H = \mathrm{h}\,\frac{\sigma}{\sqrt{n}}\ (\mathrm{LSC}) \tag{17}$$

$$H = -\mathrm{h}\,\frac{\sigma}{\sqrt{n}}\ (\mathrm{LIC}) \tag{18}$$

Onde:

- k: é uma constante (frequentemente usa-se k = 0,5).
- h: é uma constante (frequentemente usa-se h = 4 ou h = 5).
- σ: é o desvio padrão dos dados.

Através do algoritmo de soma acumulada são obtidos os valores dos desvios C_i^+ e C_i^-. Tais valores são incluídos numa tabela e acumulados sucessivamente. A soma destes desvios é comparada com um intervalo de decisão H. Portanto, o processo estará sob controle se os valores plotados de C_i^+ e C_i^- estiverem no intervalo de decisão $H = \pm h\,\frac{\sigma}{\sqrt{n}}$. Tal intervalo de decisão tem a mesma função dos limites de controle de Shewhart, onde um ponto é considerado fora de controle se estiver fora destes parâmetros. Nesta pesquisa, as Equações 13, 14, 17 e 18 foram usadas para o cálculo da Soma Cumulativa. Os valores das constantes foram definidos com $h = 4{,}77$ e $k = 0{,}5$. A escolha desses valores é baseado numa tabela de recomendações para essas duas constantes baseado em uma probabilidade de 0,27% de o gráfico apresentar um falso-positivo [Montgomery 2009].

3.5.4 Gráficos de Controle EWMA

Conforme mencionado anteriormente, a grande desvantagem dos gráficos de Shewhart é que não consideram informações relativas ao passado do processo, tornando estes gráficos poucos sensíveis em detectar pequenos deslocamentos na média do processo. O Gráfico de Controle de Média Móvel Exponencialmente Ponderada (*Exponential Weighted Moving Average* – EWMA) foi proposto para suprir esta deficiência, revelando rapidamente deslocamentos pequenos e moderados através da incorporação de informações de todas as observações anteriores até a observação atual do processo [Montgomery 2009] [Souza *et al*. 2008] [Maravelakis & Castagliola 2009] [Carvalho 2012]. Essas informações são ponderadas possibilitando atribuir aos valores passados um determinado grau de importância. A média móvel exponencial ponderada calcula a média ponderada, de todas as médias anteriores da amostra, através de um processo que é atualizado recursivamente, conforme mostra a equação a seguir:

$$Z_i = \lambda x_i + (1 - \lambda) Z_{i-1} \tag{19}$$

Onde:

- $0 < \lambda \leq 1$: é uma constante, denominado de fator de ponderação.
- x_i: corresponde ao valor da i-ésima observação.
- Z_i: é uma média ponderada de todas as i-ésimas amostras anteriores e logo.
- Z_0: é igual a média do processo utilizado para determinar o valor inicial da estatística EWMA.

Assim, dados prévios são considerados para calcular o valor de Z_i, que é a média móvel de todas as amostras anteriores (ou valores individuais). Dessa forma, o EWMA é insensível à suposição de normalidade, visto que levam em conta todas as observações (não somente a mais recente) e é apropriado para ser adotado sempre que as observações individuais são analisadas [Montgomery 2009].

O fator de ponderação λ determina a taxa em que os dados mais antigos participam do cálculo estatístico do EWMA. Um valor λ = 1 implica que somente a medida mais recente influência o EWMA. Assim, um valor grande para λ (perto de 1) dá mais peso aos dados recentes e menos peso aos dados mais antigos. Da mesma forma, um pequeno valor de λ (perto de 0) dá mais peso aos dados mais antigos. O valor recomendado para λ é geralmente definido como entre 0,2 e 0,3 e, embora seja uma escolha arbitrária, existem tabelas que ajudam a selecionar o valor de λ [Lucas e Saccucci 1990]. Portanto, sendo as observações de x_i variáveis aleatórias independentes com variância dada por σ^2, então a variância de Z_i é calculada através da Equação 20 [Wasserman 1995]:

$$\sigma_{Z_i}^2 = \sigma^2 \left(\frac{\lambda}{2-\lambda}\right)\left[1-(1-\lambda)^{2i}\right] \tag{20}$$

Com base nesta equação, torna-se possível criar os limites de controle para o gráfico EWMA. A linha central representa o valor alvo μ_0 que se deseja alcançar, podendo ser substituído pela média do processo. Os limites superior e inferior são calculados baseados em L unidades de desvio padrão da média do processo. O fator de ponderação λ também é usado para construção dos limites, pois esse valor está presente no cálculo da variância de Z_i. Assim, a linha central e os limites de controle são calculados através das seguintes equações:

$$LSC = \mu_0 + L_{\sigma_{zi}} = \left(\mu_0 + L_0\sqrt{\frac{\lambda}{(2-\lambda)}\left[1-(1-\lambda)^{2i}\right]}\right)^{0,5} \tag{21}$$

$$LC = \mu_0 \tag{22}$$

$$LIC = \mu_0 - L_{\sigma_{zi}} = \left(\mu_0 - L_0\sqrt{\frac{\lambda}{(2-\lambda)}\left[1-(1-\lambda)^{2i}\right]}\right)^{0,5} \tag{23}$$

Onde:

- μ_0: valor alvo que se deseja alcançar.
- L: corresponde a largura dos limites de controle em número de desvios padrão.
- σ: desvio padrão de Z_i.
- λ: fator de ponderação, devendo estar compreendida entre o intervalo de $0 < \lambda \leq 1$.

Nesta pesquisa, as Equações 19, 21 e 23 foram utilizadas para o cálculo da Média Móvel Exponencialmente Ponderada e os seguintes valores foram atribuídos às constantes: $\sigma = 3$, $L = 2{,}701$ e $\lambda = 0{,}5$, baseado nas recomendações de [Lucas e Saccucci 1990].

3.6 Detecção de alterações

Conforme apresentado na Seção 3.3, a etapa metodológica 6 da abordagem de análise diferencial de software é dedicada para a detecção de anormalidades nas séries temporais dos indicadores monitorados. Essa etapa é crucial para detectar vazamentos de memória e apresentar a melhor combinação técnica/indicador. A detecção de vazamentos é feita via localização de anomalias que dependem, fundamentalmente, do cálculo dos valores de divergência. Esse cálculo é composto pela combinação das séries temporais médias de cada indicador com as técnicas estatísticas consideradas, compreendendo sinais suavizados. Assim, as séries dos indicadores monitorados na versão base, quando processadas, compõem os sinais base e as séries da versão alvo compreendem os sinais alvo. Portanto, cálculo dos valores de divergência torna-se uma diferença normalizada entre os sinais base e alvo aplicados à mesma taxa de amostragem e esta Seção destina-se a explicar seu funcionamento.

3.6.1 Valores de divergência

Os valores de divergência são obtidos pelo cálculo de três séries temporais derivadas dos sinais alvo e base: 1) sinal alvo $\{f_t\}$; 2) limite inferior $\{L_t\}$ e; 3) limite superior $\{U_t\}$. Esses limites são usados como controle da amplitude em que o sinal alvo f_t pode variar e ser considerado sob controle. Portanto, para um determinado tempo fixo t, o valor de divergência calculado para os elementos correspondentes das três séries é dado por:

$$\left(\frac{\mathrm{f_t} - \mathrm{U_t}}{\mathrm{U_t} - \mathrm{L_t}}\right)^{+} \tag{24}$$

Onde:

- X^{+}: é X se X ≥ 0 ou 0 caso contrário. Assim, se f_t está dentro ou abaixo do limite de intervalo (L_t e U_t), então é obtido 0. Entretanto, se f_t estiver acima do limite, então é obtido a distância de f_t do limite superior U_t. A última propriedade da equação é a normalização, que permite uniformidade de limites para todas as técnicas.

A computação dessas três séries varia de acordo com cada técnica estatística adotada. Com relação às técnicas de detecção de tendências (Seção 3.4), o sinal alvo f_t é calculado pela aplicação da respectiva função de suavização na série temporal derivada do indicador monitorado. Por exemplo, processamento da média móvel da série temporal para o cenário de carga constante e alta do indicador RSS na versão alvo. Os limites L_t e U_t são obtidos por meio do cálculo do rolamento (*rolling*) do desvio padrão na série temporal original da versão base. Por exemplo, aplicação do rolamento do desvio padrão na série temporal original para o cenário de carga constante e alta do indicador RSS na versão base. A série original é chamada assim, pois ainda não foi processada por nenhuma técnica estatística. Posteriormente, o resultado desse cálculo é adicionado (para U_t) ou subtraído (para L_t) ao respectivo sinal base suavizado, pelas técnicas de tendências, na série temporal original. O sinal base suavizado corresponde ao processamento da série temporal original.

Entretanto, o processo de geração dessas três séries, para o cálculo dos valores de divergência, é mais complexo para as técnicas de CEP (Seção 3.5). Para CUSUM, o sinal alvo f_t é calculado pela aplicação da soma acumulada (ver Seção 3.5.3, Equações 13 e 14) nas séries temporais dos indicadores monitorados na versão alvo. Antes, tais séries precisam passar pela normalização *Z*, com média e desvio padrão obtidos do respectivo sinal base. Os limites de controle são constantes: $[-4{,}77\sigma,\ 4{,}77\sigma]$. Para Shewhart, a série f_t é o próprio sinal alvo (ver Seção 3.5.2, Equação 9) e os limites são obtidos do sinal base, adicionando ou subtraindo a média do sinal base ao termo $3MR/d_2$, onde d_2 é uma constante em Shewhart e MR é a media das amplitudes móveis MR_i (ver Seção 3.5.2, Equações 8, 10 e 11).

Para EWMA, o sinal alvo f_t é obtido pela aplicação da média móvel exponencial na série original da versão alvo (ver Seção 3.5.4, Equação 19) e os limites superior e inferior obtidos, respectivamente, pelas Equações 21 e 23 (ver Seção 3.5.4). Finalmente, as seguintes constantes utilizadas nas técnicas de CEP por [Matias *et al*. 2014] foram usadas: 1) Shewhart: (3σ e $d_2 = 1{,}128$); 2) CUSUM: ($k = 0{,}5$ e $h = 4{,}77$) e; 3) EW ($L = 2{,}701$ e $\lambda = 0{,}5$). Portanto, uma vez calculado os valores de divergência, é possível compor os gráficos de divergências para detecção de anomalias e propor de melhor combinação técnica/indicador.

3.6.2 Gráficos de divergência

Os valores de divergência calculados para cada combinação técnica/indicador são plotados nos gráficos de divergência. Esses gráficos são analisados com objetivo de facilitar a detecção de alterações comportamentais que caracterizem vazamento de memória. Cada alteração detectada é caracterizada como um evento de divergência. Para que um evento de divergência aconteça é preciso que: 1) um atual valor de divergência f_i esteja abaixo de um dado limite α e; 2) os cinco valores de divergências consecutivos (f_{i+5}) estejam acima (ou igual) a esse limite α. Assim, um evento de divergência acontece se, e somente se, em uma sequência de seis valores, o primeiro esteja abaixo do limite e todos os outros acima (ou igual) a ele. O último valor de divergência f_{i+5} é marcado como um evento de divergência.

Em [Matias *et al* 2014] a justificativa da escolha desses cinco valores consecutivos vem da teoria do CEP, baseado na probabilidade do surgimento de falsos alarmes dado um intervalo de limites de controle [Wetherill e Brown 1991]. Além disso, análise preliminares realizadas em [Matias *et al*. 2014] mostraram que a utilização de valores consecutivos maiores (ex. 10) não afetou significantemente os resultados. Outro ponto importante é que uma vez que os valores de divergência são obtidos por meio de uma normalização, então um limite α comum a todas as combinações técnica/indicador pode ser utilizado. Resultados preliminares em [Matias *et al*. 2014] revelaram que um limite $\alpha = 0{,}5$ foi considerado adequado por apresentar um bom equilíbrio entre a latência da detecção e a robustez.

A comparação da melhor combinação técnica/indicador em termos de precisão é realizada pela análise dos eventos de divergência. Para isso, em [Matias *et al*. 2014] foram definidos os seguintes conceitos:

- *DivFirstTime*: tempo do primeiro evento de divergência.

- *DivLastTime*: tempo do último evento de divergência.
- *DivNumEvents*: número de eventos de divergência após o primeiro evento (*DivFirstTime*).

A quantidade de eventos de divergência após o primeiro (*DivNumEvents*) está diretamente ligada a análise da melhor combinação técnica/indicador. Assim *DivNumEvents* determina o número de falsos alarmes. Esses falsos alarmes, em um sistema sofrendo de envelhecimento, tendem a desaparecer depois de certo período de tempo de execução [Matias *et al*. 2014]. Portanto, uma combinação técnica/indicador ideal é aquela onde não existam falsos alarmes, ou seja, *DivNumEvents* = 0 e *DivFirstTime* = *DivLastTime*. Em outras palavras, quando um efeito do envelhecimento é detectado pele primeira vez no gráfico de controle (*DivFirstTime*), então a série dos valores de divergência se mantém, até o fim, acima do limite α e nenhum falso alarme é gerado. Em contrapartida, quanto mais *DivNumEvents* existirem e/ou maior for a distância entre *DivFirstTime* e *DivLastTime*, então menos eficiente será a combinação técnica/indicador. A Figura 3.4 mostra um exemplo de um gráfico de divergência. Nela podemos observar que ambos os eventos marcados como "5" são eventos de divergência, denotando que há um evento de divergência após o primeiro e, consequentemente, *DivNumEvents* = 1 e *DivFirstTime* <> *DivLastTime*.

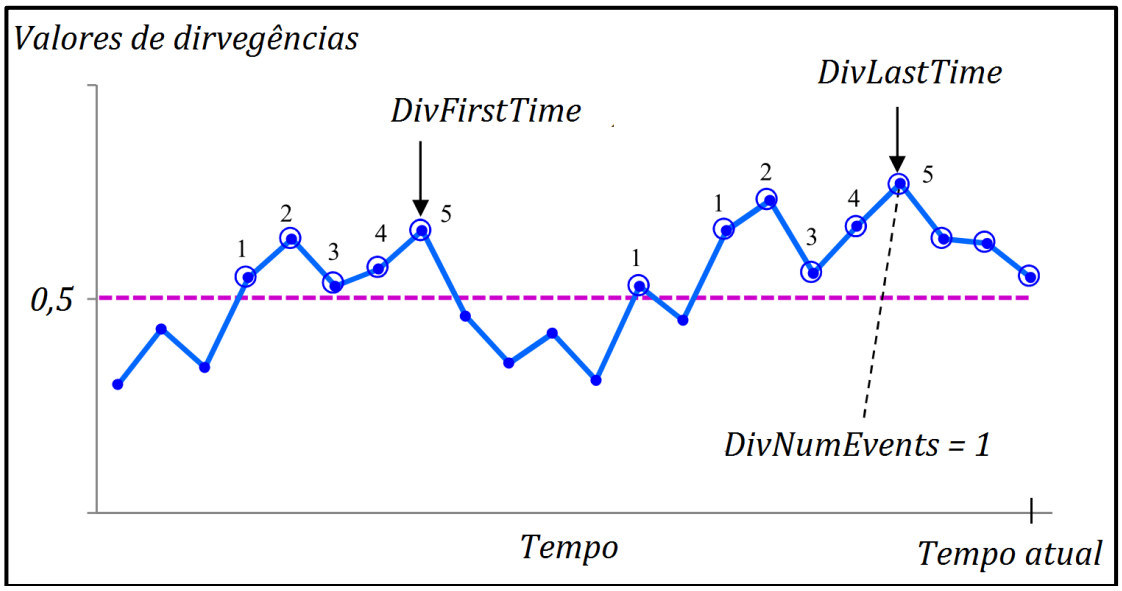


Figura 3.4. Exemplo de um gráfico de divergência e suas métricas.
Traduzido de: [Matias *et al*. 2014].

Portanto, os seguintes critérios, em ordem de classificação, para determinar a melhor combinação técnica/indicador foram adotados:

- **1º**: Menor tempo de detecção (menor *DivLastTime*).
  - Eventualmente uma combinação ideal, que não apresenta nenhum falso alarme (*DivNumEvents* = 0 e *DivFirstTime* = *DivLastTime*), frequentemente tem o menor tempo de detecção.
- **2º**: Combinações que apresentam o mesmo tempo de detecção (mesmo *DivLastTime*), a melhor será aquela que possuir o menor número de falsos alarmes (menor *DivNumEvents*).

- **3º:** Combinações que apresentam o mesmo tempo de detecção e o mesmo número de falsos alarmes, a melhor será aquela que possuir a menor distância entre o primeiro e o último evento de divergência (menor distância entre *DivFirstTime* e *DivLastTime*).

3.7 Dynamic Time Warping - DTW

Conforme explicado na Seção 3.3, séries temporais médias foram estimadas pelo método *Dynamic Time Warping* (DTW), a fim de reduzir os efeitos de defasagem entre as séries de cada replicação. Esta Seção destina-se a explicar os conceitos técnicos que envolvem a aplicação do método DTW nesta pesquisa de mestrado.

Nos últimos anos, a presença de séries temporais vem se tornando mais comum em várias áreas, em especial em mineração de dados. Muitas tarefas de mineração de dados, como classificação, clusterização, indexação, detecção de anomalias, descoberta de regras, etc., passaram a se tornar tópicos de grande interesse na comunidade [Petitjean *et al* 2011]. Uma rotina frequentemente utilizada nessas tarefas é o cálculo das medidas de distância. De modo geral, as medidas de distâncias podem ser definidas como medidas de similaridade e dissimilaridade. A primeira visa definir o grau de semelhança entre pontos e realiza o agrupamento de acordo com a coesão entre eles. Em contrapartida, a segunda calcula a diferença entre os atributos desses pontos [Frey e Dueck 2007].

A medida de similaridade mais utilizada em mineração de dados é a Distância Euclidiana (ED) [Euclidean distance 2017]. Essa medida visa encontrar a similaridade entre séries temporais comparando observações em um exato tempo *t*. Assim, considere $A = < a_1, \dots, a_t >$ e $B = < b_1, \dots, b_t >$ como duas séries temporais (ou sequências) e δ sendo a distância (ou coordenada) entre pontos dessas sequências. Dessa forma, ED é definida como a soma da raiz quadrada da diferença entre A e B em suas respectivas dimensões [Black 2004]:

$$D(A,B) = \sqrt{\delta(a_1,b_1)^2 + \cdots \delta(a_t,b_t)^2} \tag{25}$$

Entretanto, ED é limitada, pois seu cálculo não corresponde ao entendimento comum do que uma sequência realmente representa, não capturando semelhanças flexíveis [Petitjean *et al* 2011] e não detectando precisamente distorções no eixo temporal [Silva e Batista 2016]. Por exemplo, sejam $X = < 1, 2, 1, 1 >$ e $Y = < 1, 1, 2, 1 >$ duas sequências. Embora X e Y representem uma trajetória similar, elas são diferentes de acordo com a distância ED. Assim, muitas tarefas de mineração de dados requerem cálculo de similaridades mais flexíveis, onde uma determinada observação x_i, de uma série temporal no tempo *i*, pode ser associada a uma observação y_j, de outra série temporal no tempo *j* (sendo $i \neq j$).

Pensando em suprir essa necessidade é que [Sakoe e Chiba 1971] introduziram uma medida de similaridade alternativa a ED, aplicada também às séries temporais [Ratanamahatana e Keogh 2004], conhecida como *Dynamic Time Warping* (DTW). O DTW é baseado na distância Levenshtein (ou distância de edição) e busca capturar semelhanças flexíveis alinhando, da melhor forma, coordenadas entre duas sequências numéricas, sendo utilizado pela primeira vez para encontrar semelhanças no reconhecimento de fala [Sakoe e Chiba 1971]. Para

entender melhor o cálculo do alinhamento, considere duas sequências S e Q: $S = < s_1, s_2, s_3, \dots, s_i, \dots, s_n >$ e $Q = < q_1, q_2, q_3, \dots, q_j, \dots, q_m >$, onde n e m representam, respectivamente, o tamanho das séries S e Q e; i e j representam os índices das séries. O DTW relaciona essas duas sequências entortando (*warping*) o eixo de uma sob a outra. Como se trata de uma técnica de programação dinâmica, então o problema é divido em vários subproblemas, cada um contribuindo para o cálculo da distância cumulativa [Al-Naymat *et al*. 2009]. Assim, o melhor cálculo do alinhamento através das distâncias é definido recursivamente por:

$$D(i,j) = d(i,j) + min \begin{cases} D(i-1,j) \\ D(i-i,j-1) \\ D(i,j-1) \end{cases} \quad (26)$$

Primeiramente, o DTW cria uma matriz de distância local d com n vs m elementos. Esses elementos representam a Distância Euclidiana de cada dois pontos da série temporal. Posteriormente, com base na Equação 26, uma matriz de *warping* D é preenchida (as linhas de 1 a 7 do Algoritmo 1 mostram esse preenchimento). Com a matriz D preenchida, então o estágio final do DTW é encontrar o melhor caminho de *warping* W e a distância DTW. O caminho de *warping* é um conjunto de elementos adjacentes da matriz D que identificam o melhor mapeamento entre S e Q [Al-Naymat *et al*. 2009].

Algoritmo 1 – *DTW*: O algoritmo padrão para o *DTW*.
Entrada: S: sequência de tamanho n, Q: sequência de tamanho m
Saída: Distância DTW
01: **Inicialize** D(i,1) ←i$\boldsymbol{\delta}$ para cada i
02: **Inicialize** D(1,j) ←j$\boldsymbol{\delta}$ para cada j
03: **para cada** *i* onde 2 ≤ *i* ≤ *n* **faça**:
04: **para cada** *j* onde 2 ≤ *i* ≤ *m* **faça**:
05: Use a Equação 26 para calcular D(i,j)
06: fim para
07: fim para
08: **retorne** D(*n*,*m*)

Traduzido de: [Al-Naymat *et al*. 2009].

A Figura 3.5 mostra um exemplo de como duas sequências são entortadas (*warped*) e suas distâncias calculadas. As células circuladas representam o melhor caminho de *warping* encontrado. O número total de elementos no caminho W é dado por *K.* Sendo K um fator de normalização com os seguintes atributos:

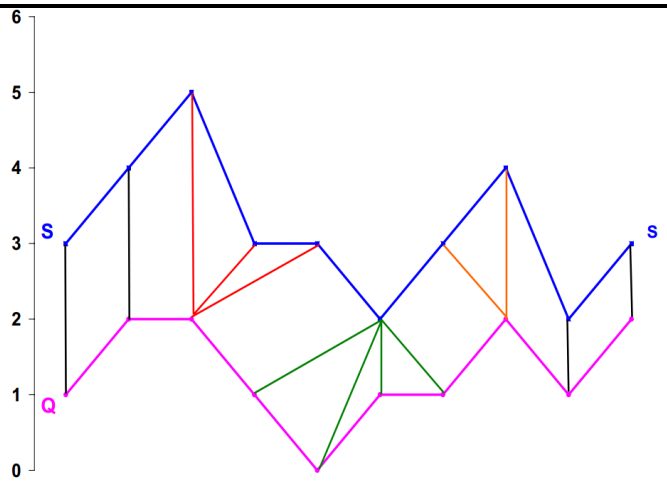
$$W = w_1, w_2, \dots, w_K$$
$$\max((|S|,|Q|) \leq K < (|S| + |Q|)$$

Assim, o melhor caminho de *warping* é calculado baseado na distância entre duas sequências por meio da seguinte equação:

$$DTW(S,Q) = min\left\{\frac{\sqrt{\sum_{k=1}^{K} W_k}}{K}\right. \quad (27)$$

Onde:

- *K*: é usado para normalizar diferentes caminhos de *warping* com diferentes tamanhos.



A) O alinhamento de medições para medir a distância DTW entre as duas sequências S e Q.

	Q									
D	1	2	2	1	0	1	1	2	1	2
3	**4**	5	6	10	19	23	27	28	32	33
4	13	**8**	9	15	26	28	32	31	37	36
5	29	17	**17**	25	40	42	44	40	47	45
3	33	18	**18**	21	30	34	38	39	43	44
3	37	19	**19**	22	30	34	38	39	43	44
2	38	19	19	**20**	**24**	**25**	**26**	26	27	27
3	42	20	20	23	29	28	26	**27**	30	28
4	51	24	24	29	39	37	37	**31**	36	32
2	52	24	24	25	29	30	31	31	**32**	32
3	56	25	25	28	34	33	34	32	35	**33**

B) A matriz de warping D produzida pelo DTW; as células destacadas correspondem ao melhor caminho de warping.

Figura 3.5. Ilustração do *DTW*.
Traduzido de: [Al-Naymat *et al*. 2009].

Conforme explicado no início dessa Seção, o DTW foi utilizado neste trabalho para gerar uma série temporal média, a partir das 10 séries temporais obtidas para cada cenário de carga e indicador (ver Seção 3.3, etapa 5). Entretanto, gerar uma série temporal média através do DTW não é uma tarefa trivial, principalmente devido à necessidade de o DTW conseguir realinhar sequências ao longo do tempo [Petitjean *et al* 2011]. Ao longo dos anos, várias tentativas de definir um método de média para o DTW foram realizadas e a maioria delas resultou em técnicas de emparelhamento (*pairwise*). Porém, a utilização de *pairwise* leva a uma noção imprecisa da média, uma vez que esses métodos não garantem que as diferentes ordens levarão ao mesmo resultado [Niennattrakul e Ratanamahatana 2007].

A Figura 3.6 mostra um exemplo da realização de média por *pairwise*, onde o DTW é aplicado a duas sequências e o resultado é utilizado para um novo DTW até

chegar-se a média. Pensando em solucionar esse problema é que em [Petitjean *et al* 2011] foi introduzido um método de otimização chamado *DTW Barycenter Averaging* (DBA), onde experimentos mostraram que o DBA é mais preciso que todos os métodos *pairwise* propostos anteriormente. Portanto, nesta pesquisa utilizou-se o método DBA para a obtenção da série temporal média por DTW.

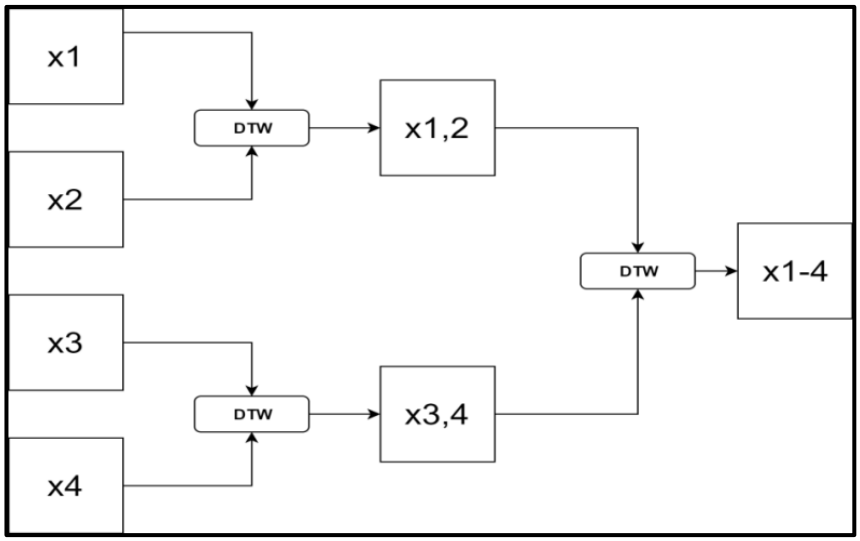


Figura 3.6. Exemplo da aplicação do método *pairwise*.
Fonte: [Niennattrakul e Ratanamahatana 2007].

3.7.1 DTW Barycenter Averaging (DBA)

O *DTW Barycenter Averaging* (*DBA*) consiste em uma estratégia heurística projetada como um método global para média do DTW. Sua ideia principal está em, iterativamente, minimizar a soma das distâncias ao quadrado (DTW) de uma sequência média extraída de um conjunto de sequências [Petitjean *et al* 2011]. Essa soma é constituída de distâncias individuais entre cada coordenada da sequência média e as coordenadas das sequências associadas a ela. Portanto, a contribuição de uma coordenada da sequência média para a soma total da distância quadrática é na realidade a soma das distâncias euclidianas entre a coordenada da sequência média e as coordenadas das sequências associadas a ela durante o cálculo do DTW [Petitjean *et al* 2011].

Dessa forma, uma única coordenada de uma das sequências pode influenciar em uma nova posição de várias coordenadas da sequência média. Em outras palavras, o princípio básico do DBA é calcular cada coordenada da sequência média como o centro da massa (*barycenter*) das coordenadas associadas ao conjunto de sequências. Quando todos os *barycenters* são calculados, então a sequência média é definida [Petitjean *et al* 2011]. Assim, para cada iteração o *DBA* trabalha em duas etapas:

- Calcular o DTW entre cada sequência individual e a sequência média temporária a ser refinada: encontrando associações entre coordenadas da sequência média e coordenadas do conjunto de sequências.
- Atualizar cada coordenada da sequência média: como o *barycenter* de coordenadas associadas a ela durante a primeira etapa.

Para entender melhor o DBA, consideremos: 1) $S = < s_1, \dots, s_N >$ como o conjunto de sequências em que se deseja extrair uma sequência média; 2) $C = < c_1, \dots, c_T >$ como a sequência média na iteração i e; 3) $C' = < c'_1, \dots, c'_T >$ a atualização de C na iteração $i + 1$. Cada coordenada da sequência média é definida como um espaço vetorial arbitrário E (normalmente um espaço euclidiano) [Petitjean *et al* 2011]:

$$\forall t \in [1, T], C_t \in E$$

Cada coordenada da sequência média é então ligada a uma ou mais coordenadas das sequências de S por uma função de associação (*assoc*). Assoc é definida durante o calculo do DTW entre C e cada sequência de S [Petitjean *et al* 2011], onde a t-ésima coordenada sequência média C'_t é definida como:

$$C'_t = barycenter(assoc(C_t))$$

onde,

$$barycenter\{X_1, \dots, X_2\} = \frac{X_1 + \cdots + X_2}{\alpha}$$

Assim, as associações criadas por DTW podem mudar sempre que um novo DTW é calculado entre a sequência média e todas as sequências de S. Como se torna impossível prever como essas associações vão mudar, então [Petitjean *et al* 2011] propôs realizar conversões iterativas. A Figura 3.7 mostra quatro iterações do DBA para um exemplo de quatro sequências – mais detalhes podem ser encontrados em [Petitjean *et al* 2011].

Portanto, uma vez que o DBA é uma abordagem global que pode extrair a média de várias sequências através do cálculo do DTW e que a atualização da sequência média entre duas iterações é independente da ordem em que são colocadas as sequências, então esse método foi selecionado para realizar o cálculo da média das séries temporais extraídas de cada cenário de carga de trabalho.

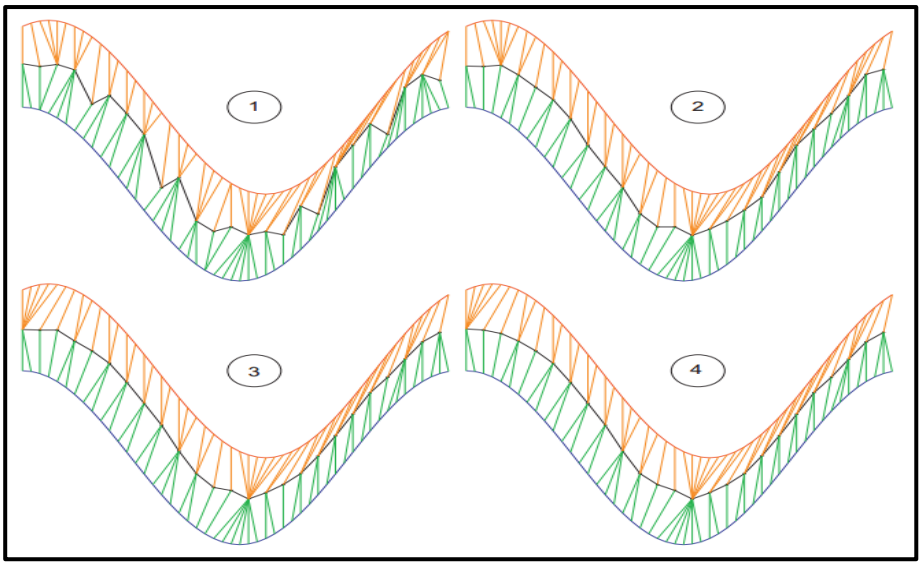


Figura 3.7. Funcionamento do DBA para estimar a média entre duas sequências.
Fonte: [Petitjean *et al* 2011].

3.7.2 Constante de delimitação Windows Size (WS)

Uma recente extensão criada para o DTW e que acelera significativamente seu cálculo é uma constante de delimitação chamada *Windows Size* (WS*)*. Essa constante tem por objetivo reduzir o número de caminhos a serem considerados durante o cálculo da matriz de *warping*, visto no início da Seção 3.7 [Keogh e Ratanamahatana 2002]. Além de acelerar a computação, o WS também permite evitar problemas estruturais da aplicação do *warping*, como por exemplo, uma seção relativamente pequena de uma sequência ser mapeada para uma seção muito maior de outra sequência. A Figura 3.8 ilustra como o WS pode limitar os caminhos considerados durante o cálculo de uma matriz *warping*. Dois tipos de WS utilizados na literatura são usados como exemplo: *Sakoe-Chiba Band* [Sakoe e Chiba 1978] e *Itakura Parallelogram* [Itakura 1975]. Portanto, com um valor de WS definido, o cálculo recursivo do DTW (*ver* Equação 38) irá respeitar o limite imposto pelo WS.

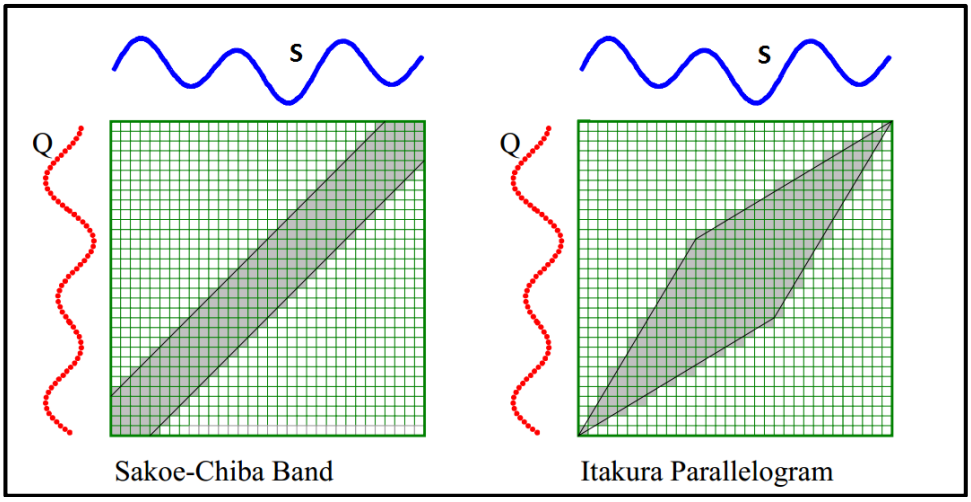


Figura 3.8. Exemplo de funcionamento do WS.
Fonte: [Ratanamahatana e Keogh 2004].

Alguns trabalhos têm utilizado um WS de 10% do tamanho das sequências a serem analisadas, essa abordagem é conhecida como *Sakoe-Chiba Band* [Aach e Church 2001] [Rabiner *et al*. 1978] [Sakoe e Chiba 1978]. Já outros trabalhos consideram que quanto maior o valor de WS melhor será a precisão do DTW [Zhu e Shasha 2003]. Também existem trabalhos em que nenhum valor de WS foi definido [Laaksonen *et al*. 1998] ou não foi especificado pelos autores [Kornfield *et al*. 2004]. Essa falta de consenso para atribuição de um valor para WS, segundo [Ratanamahatana e Keogh 2004], ocorre pela falta de estudos que demonstrem que o melhor valor para WS pode variar de acordo com as sequências a serem analisadas.

Por esse motivo que [Ratanamahatana e Keogh 2004] e [Mueen e Keogh 2016] realizaram estudos empíricos para demonstrar que o valor de WS tem participação efetiva na precisão do DTW. Em ambos os trabalhos, os resultados mostraram que grandes valores para WS não trazem maior acuracidade, pelo contrário, para vários tipos de sequências analisadas, valores ≤ 10 para WS apresentaram os melhores resultados. Assim, esses trabalhos sugerem a realização de um estudo prévio para definir o melhor valor de WS para cada caso de análise. Esse estudo pode ser realizado através da conhecida técnica de validação cruzada.

A validação cruzada é uma técnica que verifica como os resultados de uma análise estatística se generalizam para um conjunto de dados independentes. Geralmente é aplicada em modelos de previsão, onde se deseja estimar a precisão que um determinado modelo preditivo pode alcançar. Seu funcionamento consiste basicamente em considerar um conjunto de dados para o treinamento. O modelo em que se está analisando a precisão é calculado sob esses dados de treinamento e, posteriormente, os valores resultantes são usados para medir a precisão, comparando-os com outro conjunto de dados selecionados para validação (ou teste) – maiores detalhes sobre essa técnica podem ser vistos em [Cross-Validation 1997]. Uma forma de comparar a acuracidade de WS pela validação cruzada é analisando a taxa de erro quadrático médio [Statistics How To 2013].

Algoritmo 2 – Escolha do melhor WS.

```
Entrada: 10 séries temporais obtidas de um cenário de carga e indicador
Saída: Melhor WS
01: series: lista contendo as 10 séries de entrada
02: seriesTreinamento: lista utilizada para armazenar as séries de treinamento
03: serieValidacao: lista utilizada para armazenar a séries de validação
04: ws: lista de ws considerados
05: erros: lista de erros quadráticos para cada ws
06: para cada i em series faca
07:         seriesValidacao.Insere(series[i])
08:         para cada j em series faca
09:                 se i <> j faca
10:                         seriesTreinamento.Insere(series[j])
11:                 fim se
12:         fim para
13:                 para cada z em ws faca
14:                         serieDBA = DBA(seriesTreinamento, ws[z])
15:                         erros[z].Insere(validacaoCruzada(serieDBA, seriesValidacao))
16:                 fim para
17:         fim para
18: retorna melhorWS(erros)
```

Portanto, considerando todos os fatores mencionados anteriormente, nesta pesquisa foi realizado um estudo de sensibilidade para definir o melhor valor de WS, para cada cenário de carga de trabalho e indicador analisado. Esta análise seguiu a aplicação da validação cruzada. O Algoritmo 2 ilustra como essa análise foi realizada. A entrada consiste em 10 séries temporais extraídas de um cenário de carga de trabalho para um determinado indicador, por exemplo, carga constante e normal do indicador HUS.

As linhas de 01 a 05 são destinadas a variáveis utilizadas: 1) series: contendo as 10 séries da entrada; 2) seriesTreinamento: lista de séries de treinamento utilizadas para cada iteração da validação cruzada – cada posição dessa lista armazena uma série; 3) serieValidacao: contendo a série que será usada para validar o DBA em cada iteração; 4) ws: lista contendo os valores de *Windows Size* avaliados: {5; 10; 15; 20 e nenhum} e; 5) erros: lista contendo os erros quadráticos calculados, em cada iteração, para cada *Windows Size*; cada posição dessa lista armazena outra lista que contém o erro quadrático de cada iteração, por exemplo, a posição 1 contém uma lista de todos os erros quadráticos para ws = 5.

As linhas 06 a 18 são destinadas ao processo de validação cruzada. Assim, na linha 06 se inicia uma sequência de iterações pelas 10 séries, onde em cada iteração uma série é selecionada para validação e as nove restantes são usadas para o treinamento (linhas 08 a 12). Nas linhas 13 a 17 é realizado o cálculo do DBA (na linha 14), para cada valor de *Windows Size*, considerando as séries de treinamento (adicionadas na linha 10) e, posteriormente, a validação cruzada é realizada (na linha 15) levando em conta o resultado do DBA e a série de validação (adicionada na linha 07). O erro quadrático médio é armazenado na posição correspondente ao *Windows Size* avaliado na variável erros. Finalmente, o menor erro quadrático médio é calculado na linha 18 e o respectivo *Windows Size* é retornado.

4. ESTUDO EXPERIMENTAL

4.1 Introdução

O principal objetivo desta pesquisa é avaliar a validade externa da abordagem de análise diferencial de software aplicada ao vazamento de memória. Para tanto, faz-se necessário realizar experimentos com diferentes aplicações reais para verificar se os resultados se mantêm consistentes com aqueles obtidos nos trabalhos anteriores [Matias *et al.* 2014] e [Matias *et al.* 2016]. Para esse tipo de análise adotou-se a abordagem de estudo confirmatório com replicação, o qual tem como objetivo testar, confirmando ou não, por meio da replicação de experimentos, os resultados de uma pesquisa anterior. Assim, as perguntas e as hipóteses de pesquisa são apresentadas na Seção 4.2. Além disso, todo o estudo experimental, incorporando as aplicações estudadas e seus respectivos bugs de vazamento, a caracterização do cenários de cargas de trabalho, o projeto dos experimentos e o instrumental utilizado, são descritos na Seção 4.3.

4.2 Perguntas de Pesquisa e Hipóteses

Para avaliar a validade externa da abordagem de análise diferencial de software, as seguintes perguntas de pesquisa foram levantadas:

1. O desempenho das técnicas estatísticas e dos indicadores de envelhecimento, usados para detectar vazamento de memória com carga sintética *adhoc*, será o mesmo frente a aplicações reais e cargas definidas através de um processo de caracterização de uso dessas aplicações?
2. Qual o efeito do tipo de aplicação (ex. servidor *web* ou banco de dados) sobre os resultados das combinações técnica/indicador?

Assim, diante destas perguntas, as seguintes hipóteses foram consideradas:

1. O desempenho dos indicadores de envelhecimento será o mesmo visto com carga de trabalho sintética *adhoc* [Matias *et al*. 2014]. Isso porque se espera que os efeitos do vazamento reflitam antes no HUS que no RSS. A mesma expectativa também é prevista para as técnicas estatísticas, pois acredita-se que as técnicas de CEP vão manter os melhores desempenhos. Com relação ao número de falso-negativos e falso-positivos, crê-se que a utilização de cargas planejadas para apenas ativar os *bugs* de vazamento encontrados irá influenciar nessa perspectiva, apresentando um número menor de taxas do que com carga sintética *adhoc*, como a projetada em [Matias *et al*. 2014],

que considera também fatores que não estão relacionados à ativação de *bugs* de envelhecimento.

2. O efeito do tipo de aplicação não irá influenciar em termos de falso-negativos e falso-positivos nas combinações técnica/indicador e sim a utilização de cargas projetadas apenas para ativar os *bugs* de vazamento considerados.

4.3 Estudo experimental

A avaliação da validade externa focou na realização de experimentos controlados. Para isso, conforme descrito na Seção 3.3, várias etapas foram consideradas mantendo a mesma configuração experimental dos trabalhos anteriores, a fim de facilitar a comparação dos resultados.

Inicialmente, foi necessário encontrar aplicações reais com vazamento de memória (ver Seção 3.3, etapa 1). Com relação às duas versões consideradas na análise diferencial de software (alvo e base), é importante mencionar que a versão base refere-se àquela onde não há presença de envelhecimento. Em contrapartida, a alvo é àquela sob suspeita de envelhecimento e que está sob investigação. Assim, em uma situação real de uso da técnica de análise diferencial, em um ambiente de desenvolvimento, os desenvolvedores não têm certeza sobre o envelhecimento na versão alvo e por isso utilizariam a técnica para descobrir. Entretanto, como esta pesquisa realiza experimentos controlados para verificar se a abordagem de [Matias *et al.* 2014] é efetiva na detecção de vazamentos de memória em programas reais, então todas as aplicações utilizadas precisavam ter versões com vazamento de memória comprovado, fazendo com que fosse conhecido previamente que a versão alvo de cada aplicação continha envelhecimento. Na Seção 6.1, para diminuir ameaças de validade externa, é mencionado sobre experimentos realizados em [Matias *et al.* 2016] que mostram que a análise diferencial é efetiva em não apresentar falso-positivos quando a versão alvo não contém envelhecimento.

Para encontrar as aplicações reais foi preciso verificar fóruns oficiais de relatos de *bugs* de software. Como o efeito de envelhecimento investigado foi o vazamento de memória, então palavras-chaves relacionadas com esse efeito foram utilizadas no processo de busca (ex. vazamento de memória, aumento de memória, crescimento de memória, etc.). É importante salientar que como o envelhecimento de software e seus termos não são padronizados na indústria de software, então frequentemente muitos usuários reportam *bugs* utilizando diferentes termos. Assim, se a descrição de um *bug* se assemelhava a um vazamento de memória, então ele era selecionado para confirmação do problema através de experimentos preliminares.

Esses experimentos preliminares consistiram em colocar a aplicação sob teste com base nos padrões de ativação do respectivo *bug*, reportados nos relatórios encontrados, e monitorando os indicadores RSS e HUS. Se as informações obtidas desses indicadores demonstravam um aumento incremental da memória durante a carga de trabalho e depois que a carga era pausada a memória não diminuía, então os mesmos experimentos eram realizados em versões anteriores para encontrar um comportamento de memória diferente (ver Seção 3.3, etapa 2). Esse comportamento diferente poderia consistir de duas

maneiras: 1) não havia aumento durante a carga de trabalho ou; 2) havia aumento, as vezes menor, e depois que a carga era pausada, a memória decrementava. Esse último comportamento, frequentemente visto para o RSS, pode ser explicado pela presença de efeitos transitórios sobre esse indicador. Em caso positivo, essa versão com comportamento diferente do padrão de utilização de memória era selecionada como versão base e sua versão imediatamente posterior usada como versão alvo.

Houve casos em que *bugs* selecionados para esses testes de investigação não se confirmaram como vazamento de memória, onde na maioria das vezes o aumento de memória era consequência natural do padrão de uso da aplicação. Também, houve situações, para *bugs* com forte suspeita de vazamento, onde nenhuma versão anterior com comportamento diferente foi encontrada para ser usada como base. Assim, para ambos os casos, os *bugs* e suas respectivas aplicações foram descartadas. Finalmente, um total de 4 aplicações atenderam todos os requisitos das etapas 1 e 2 (ver Seção 4.3.1). Estas aplicações foram: Inkscape, Lighttpd, PostgreSQL, Squid.

4.3.1 Aplicações utilizadas

Cada uma das 4 aplicações selecionadas possui seu respectivo *bug* de vazamento de memória que é ativado frente a fatores específicos (ou uma combinação destes) exercidos na aplicação, os quais são denominados de fatores de envelhecimento (*Aging Factors*). Por exemplo, um valor de entrada de uma função que causa a execução de uma região de código onde se localiza o *bug*. Esse valor específico é um padrão de ativação para o *bug*, ou seja, um fator de envelhecimento, pois na sua presença ocorre uma ativação da falta de envelhecimento (*AR Fault*). Dessa forma, cada padrão de ativação foi estudado para planejar a carga de trabalho de acordo com o comportamento típico da aplicação nos casos onde ocorrem as ativações do *bug* de vazamento (ver Seção 3.3, etapa 3).

O estudo de planejamento da carga de trabalho considerou modos e intensidades que resultaram nos seguintes cenários de carga de trabalho: 1) carga constante e alta; 2) carga constante e normal; 3) carga constante e baixa e; 4) carga variada. Portanto esta Seção destina-se a descrever cada aplicação utilizada e seu respectivo padrão de ativação do *bug* considerado. Além disso, é apresentado o estudo de caracterização dos cenários de carga de trabalho realizado por aplicação.

4.3.1.1 Inkscape

O Inkscape [Inkscape 2017a] é um programa de edição de gráficos vetoriais, escrito principalmente em *C++* [Openhub 2017a], que oferece versões para Windows, Linux e MAC OS X. Foi lançado em 2003 e atualmente possui mais de 550.000 linhas de código fonte [Openhub 2017a]. O Inkspace proporciona suporte de importação e exportação para diversos formatos de arquivo e tem um grande conjunto de funcionalidades para manipulação de gráficos vetoriais, permitindo personalização através de extensões. Apesar de não ter todos os recursos de outros softwares de edição de imagens do mercado (ex. Adobe Illustrator, Corel Draw e Xara Xtreme), as versões mais recentes do Inkspace fornecem uma grande parte dos recursos básicos de edição de gráficos vetoriais. Provavelmente, os casos mais bem sucedidos de sua utilização são os das organizações Wikipedia e Openclipart,

que manipulam milhares de imagens todos os dias utilizando esse software [Inkscape 2017b].

4.3.1.1.1 O bug de vazamento

O *bug* de vazamento de memória encontrado para o Inkscape e utilizado nesta pesquisa refere-se a um problema relacionado à exibição, em caixa de pré-visualização (*preview*), de arquivos de gráficos vetoriais escaláveis (SVG). O vazamento foi reportado no fórum oficial de relatos de *bugs* do Inkscape para a versão 0.46 do software [Bugs Launchpad 2017] e possui como padrão de ativação os seguintes procedimentos:

1. Executar o Inkscape.
2. Clicar no menu Arquivo/Abrir ou Arquivo/Importar.
3. Ir até o diretório /usr/share/icons/gnome/scalable e clicar em qualquer pasta dentro desse diretório.
4. Dentro de qualquer uma das pastas clicadas existirão arquivos SVG que, quando colocados em pré-visualização (*preview*), ocasionarão vazamento de memória.

Uma representação do padrão de ativação do *bug* do Inkscape pode ser visto na Figura 4.1. Portanto, conforme explicado no início da Seção 4.1 e, consequentemente, nas etapas 1 e 2 da Seção 3.3, testes preliminares precisaram ser realizados para: 1) verificar a manifestação do vazamento de memória perante o padrão de ativação e; 2) encontrar a versão anterior mais recente livre do problema (versão base). Assim, a versão base encontrada foi a 0.45 e a versão alvo selecionada foi a imediatamente posterior, 0.46.

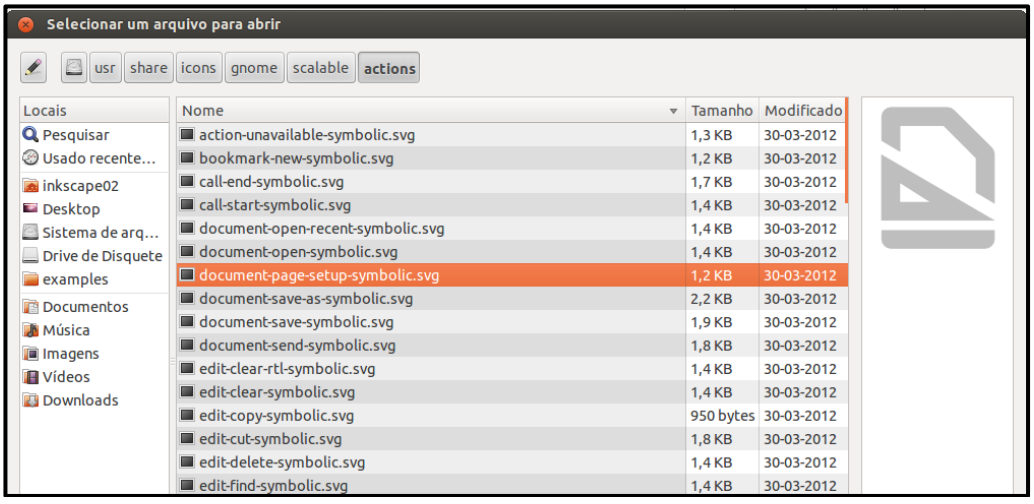


Figura 4.1. Inkscape: padrão de ativação – arquivos SVG sendo colocados em *preview*.

4.3.1.1.2 Caracterização da carga de trabalho

Uma vez que o vazamento está associado à funcionalidade de pré-visualização (*preview*), comum a softwares de edição de imagens, onde o usuário a utiliza para verificar se o arquivo que está selecionando é o que deseja abrir, então o estudo de caracterização de representatividade da carga de trabalho buscou focar em procurar trabalhos acadêmicos que utilizaram algum tipo de carga

direcionada a listar imagens em *preview* ou até mesmo a utilização de imagens como parte da carga.

Os resultados dessa procura não foram satisfatórios, uma vez que nenhum estudo que utilizou cargas relacionadas à listagem de imagens em *preview* e/ou usou algum editor de imagens para realizar experimentos relacionados a abrir imagens foi encontrado. Entretanto, alguns trabalhos usaram imagens para outros objetivos, como em [Arguel e Jamet 2009], que realizou um estudo para entender se a utilização de animações promove melhor aprendizado para um ser humano do que imagens estáticas, avaliando a percepção humana. Assim, considerando o propósito de uma funcionalidade de exibição em *preview* em um editor de imagens e o aspecto da percepção humana, pode-se concluir que ambos têm relação.

Portanto, para relacionar a percepção humana de imagens com a função de *preview* do Inkscape, um experimento para caracterização de carga de trabalho foi realizado. Esse experimento teve como inspiração um trabalho recente que concluiu que um ser humano leva em média 100 milissegundos para conseguir interpretar uma imagem [Liu *et al.* 2009]. Assim, o objetivo do experimento foi verificar o quanto um usuário do Inkscape leva em média para identificar um arquivo que deseja abrir no programa, através da função de *preview*.

Para realização desse experimento, foi utilizado um banco de 522 diferentes imagens extraídas de [Webartz 2017] e armazenadas em uma pasta dentro do sistema operacional de um computador com o Inkscape instalado. A ideia central do experimento foi escolher, aleatoriamente a cada rodada, uma das 522 imagens e, com o Inkscape aberto, fazer com que um usuário explorasse o padrão de ativação do *bug* caminhando até o diretório contendo as 522 imagens.

Dessa forma, o usuário precisou utilizar a função de *preview*, passando pelas imagens dentro da pasta, utilizando a seta do teclado, até chegar à imagem escolhida aleatoriamente. No final de cada rodada, o número de arquivos listados em *preview*, até encontrar a imagem desejada, e o tempo gasto para fazer essa operação foi considerado. Com isso, para cada rodada, foi estimado o tempo médio em que cada arquivo foi colocado sob *preview* por um usuário de um editor de imagens.

Com o intuito de diminuir os erros experimentais, dez (10) pessoas, de diferentes idades e níveis de afinidade com tecnologia, foram escolhidas para participarem do experimento. Além disso, para cada pessoa selecionada, uma replicação de dez (10) rodadas foi realizada. Portanto, foi calculada, para cada pessoa, uma média das 10 rodadas e, posteriormente, extraída uma média dessas médias de rodadas, concluindo que um arquivo fica listado em *preview*, em média, por 0,773 segundos. Portanto, devido à ausência de trabalhos na literatura relacionados ao padrão de ativação do *bug* estudado para o Inkscape, foram usados os resultados do experimento detalhado anteriormente para caracterizar a carga de trabalho para a ativação do *bug*. A média de 0,773/s foi utilizada para compor o cenário de carga constante e normal. Para carga constante e baixa foi adotado o critério de levar em conta, dentre as 10 pessoas selecionadas para o experimento de caracterização, aquela que teve um tempo de resposta maior na percepção de imagens, resultando em 1,019/s. Para carga constante e alta, o critério adotado foi justamente o contrário usado para carga baixa, chegando a um tempo de resposta de 0,497/s. A carga variada alternou entre os valores da carga alta e baixa. A

Tabela 4.1 apresenta uma visão geral dos cenários de cargas de trabalho definidos para o Inkscape.

Tabela 4.1. Inkscape: cenários de cargas de trabalho caracterizados.

Cenário de carga	**Caracterização**
Constante e alta	1 arquivo colocado em *preview* a cada 0,497s.
Constante e normal	1 arquivo colocado em *preview* a cada 0,773s.
Constante e baixa	1 arquivo colocado em *preview* a cada 1,019s.
Variada	1 arquivo colocado em *preview* entre 0,497s a 1,019s.

4.3.1.2 Lighttpd

O Lighttpd [Lighttpd 2017a] é um servidor *web*, predominantemente escrito em *C* [Openhub 2017b] e voltado para ambientes computacionais de alto desempenho [DreamHost 2012]. Tais características tornam o Lighttpd ideal para servidores submetidos a situações de sobrecarga. De código fonte aberto, oferece suporte a sistemas operacionais Windows e baseados em Unix [Lighttpd 2017a], e atualmente está com mais de 100.000 linhas de código fonte [Openhub 2017b].

Possui um grande conjunto de funcionalidades, assim como outros servidores *web* (ex. Apache), entre elas destacam-se: *FastCGI*, *SCGI*, *Auth*, *Output-Compressão* e *URL-Rewriting* [Lighttpd 2017a]. O Lighttpd é usado por muitos sites que gerenciam alto tráfego de dados, entre eles, destacam-se: o Bloglines – um agregador de notícias; e o Viator – um site para vendas de pacotes de passagens [WebTechster 2017]. Os casos mais bem sucedidos de uso são, provavelmente, o do Wikimedia – um projeto mantido pela Wikimedia Foundation com o objetivo de servir de repositório para imagens e outros tipo de multimídia livre; e do Youtube [Lighttpd 2017b].

4.3.1.2.1 O bug de vazamento

O *bug* de vazamento encontrado para o Lighttpd refere-se a um problema relacionado a instruções IF/ELIF da linguagem de *scripts Server Side Includes* (SSI). Para executar tais instruções em um arquivo, habilitado para SSI, é preciso que o nome do arquivo termine com uma extensão especial (por padrão utiliza-se .shtml, .stm ou .shtm). O SSI é suportado por uma série de servidores *web* bem conhecidos, como o Apache, LiteSpeed, nginx, lighttpd e IIS. Seu uso mais frequente é de incluir pedaços de códigos comuns em um site (ex. cabeçalho, rodapé e menu de navegação) [Gundavaram 1996]. Um exemplo de instrução SSI que mostra a data atual pode ser visto abaixo:

```
<!--#echo var="DATE_LOCAL" -->
```

O vazamento foi reportado no fórum oficial de relatos de *bugs* do Lighttpd para a versão 1.4.19 do software [Redmine Lighttpd 2017] e possui como padrão de ativação os seguintes procedimentos:

1. Ativar o modo SSI no Lighttpd
2. Criar uma página, de extensão shtml, contendo instruções IF/ELIF e armazená-la no diretório de hospedagens de páginas do Lighttpd.
3. Acessar, através de um cliente (por exemplo, um navegador), a página de extensão shtml criada – cada acesso acarretará vazamento de memória.

```
<html>
<head><title></title></head>
<body>

<!--#if expr="$SERVER_SOFTWARE = 'lighttpd/1.4.19'" -->lighttpd-1.4.19<!--#else -->other<!--#endif --><br>

</body>
</html>
```

Figura 4.2. Lighttpd: padrão de ativação - código contendo instruções SSI do tipo IF/ELIF.

Uma representação do padrão de ativação do *bug* do Lighttpd pode ser visto na Figura 4.2. Testes preliminares foram realizados para verificar a manifestação do vazamento de memória e encontrar as versões base e alvo. A versão base encontrada foi a 1.2.6 e, portanto, a versão imediatamente posterior (1.2.7) seria a versão alvo. Entretanto, a versão 1.2.7 não está presente no repositório de versões considerado [Fedora Project 2015]. Assim, foi selecionada como versão alvo àquela com a menor distância da versão base (1.2.6) presente nesse repositório, resultando na versão 1.3.11.

4.3.1.2.2 Caracterização da carga de trabalho

Uma vez que o vazamento está ligado a acessar páginas hospedadas no Lighttpd contendo instruções SSI do tipo IF/ELIF, então o estudo de caracterização de representatividade da carga de trabalho procurou focar na pesquisa de trabalhos acadêmicos que fizeram uma caracterização de cargas em servidores *web*. Essa pesquisa também buscou encontrar repositórios de dados que tivessem informações estatísticas relacionadas a acessos recebidos por páginas hospedadas em um servidor *web*. Uma visão geral dos trabalhos acadêmicos encontrados é apresentada na Tabela 4.2.

É preciso ressaltar que, após os anos 2000, páginas geradas dinamicamente passaram a ganhar cada vez mais espaço, diminuindo a participação de páginas HTML na análise de *logs* de servidores *web* e dando espaço para páginas de outras extensões (ex. php). Assim, muitos estudos de análise de *logs* nesse período passaram a considerar taxas de acesso a páginas *web* e não puramente a páginas HTML e por isso na Tabela 4.2 há essa distinção.

O trabalho realizado por [Eldin *et al.* 2014] foi incluído na Tabela 4.2 por um motivo especial, pois objetivou a compreensão precisa de cargas de trabalho como chave para o gerenciamento eficiente de recursos em nuvem . Para tal, uma análise da carga de trabalho recebida pelo Wikipedia foi realizada através de informações fornecidas por uma API de dados estatísticos de acessos referentes às paginas do Wikipedia [Wikitech 2017]. Portanto, considerando que: 1) tal API está acessível para coleta de dados; 2) seus dados estão atualizados e; 3) a Wikipedia possui considerável representatividade por ser um dos sites mais acessados do mundo; então essa API foi escolhida, em detrimento dos trabalhos anteriormente mencionados na tabela, para a realização da caracterização da carga de trabalho para o Lighttpd.

Assim, através dessa API foi possível obter as 1.000 páginas mais acessadas em cada dia do mês de Dezembro/2016 (data mais recente à época em que a caracterização foi realizada). Desconsiderando a pagina principal e a pagina de busca foi encontrado que, no mês de Dezembro/2016, uma página do Wikipédia recebeu em média 25.465 requisições/dia.

Tabela 4.2. Lighttpd: visão geral dos trabalhos encontrados no processo de caracterização.

Trabalho	Objetivo	Resultados (taxa média de requisições)
[Arlitt e Williamson 1996] e [Arlitt e Williamson 1997]	Análise do tráfego de 6 ambientes computacionais (3 universidades, 2 organizações e 1 ISP).	36.257 requisições/dia a paginas HTML.
[Arlitt *et al.* 1999]	Análise do tráfego de um servidor *proxy web* hospedado em um importante provedor de internet.	157.761 requisições/dia a paginas HTML.
[Williams *et al.* 2005]	Revisar os resultados de [Arlitt e Williaamson 1996] e [Arlitt e Williamson 1997] através de uma nova análise do tráfego das 3 universidades consideradas anteriormente.	17.632 requisições/dia a paginas HTML.
[Bent *et al.*2006]	Análise do tráfego de 3.000 sites comerciais hospedados em um ISP.	1.058.924 requisições/dia a paginas HTML.
[Mahanti *et al.* 2009]	Análise do tráfego de um servidor *web* usado em uma importante conferência.	4.385 requisições/dia a paginas *web*.
[Sisodia e Verma 2012]	Análise do tráfego de um servidor *web* do Instituto Nacional de Tecnlogia de Raipur (NITR).	1.525 requisições/dia a paginas *web*.
[Sharma e Gupta 2013]	Análise do tráfego de um servidor *web* de um importante instituto educacional.	39 requisições/dia a paginas *web*.
[Goel e Jha 2013]	Análise do tráfego de um servidor *web* de um importante site de astrologia.	8 requisições/dia a paginas *web*.
[Liu e Williamson 2016]	Análise do tráfego de um servidor *web* da Universidade de Calgary.	1.479 requisições/dia a paginas *web*.
[Eldin *et al.* 2014]	Análise da carga de trabalho recebida pelo Wikipedia.	N/A.

Portanto, considerando a existência de um servidor Lighttpd, habilitado para SSI, contendo uma página hospedada com instruções SSI do tipo IF/ELIF responsáveis pelo *bug* de vazamento de memória, então a taxa média de 25.465 requisições/dia foi usada para configurar o cenário de carga constante e normal, denotando a quantidade média de requisições recebidas por essa página diariamente. Para carga constante e baixa, foi adotado o critério de levar em conta a média das páginas menos acessadas durante o período considerado (páginas com média diária de acesso inferior a 25.465 requisições/dia), resultando em uma taxa de 9.144 requisições/dia). Para carga constante e alta, o critério adotado foi justamente o contrário usado para carga baixa, chegando a uma taxa média de 73.616 requisições/dia. A carga variada alternou entre as taxas das cargas alta e baixa. A Tabela 4.3 apresenta uma visão geral dos cenários de cargas de trabalho definidos para o Inkscape.

Tabela 4.3. Lighttpd: cenários de cargas de trabalho caracterizados.

Cenário de carga	Caracterização
Constante e alta	1 acesso recebido a cada aprox. 1,17s → Baseado na taxa média de 73.616/dia.
Constante e normal	1 acesso recebido a cada aprox. 3,39s → Baseado na taxa média de 25.465/dia.
Constante e baixa	1 acesso recebido a cada aprox. 9,45s → Baseado na taxa média de 9.144/dia.
Variada	1 acesso recebido entre aprox. 1,17s a 9,45s → Baseado na taxa média de 73.616/dia a 9.144/ dia.

4.3.1.3 PostgreSQL

PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional (SGDBOR), predominantemente escrito em *C* [Openhub 2017c] e oferece suporte para a maioria dos sistemas operacionais disponíveis (incluindo Windows, Linux e MAC OS X) [PostgreSQL 2017a]. Surgiu no ano de 1986 como um projeto de continuação de seu predecessor Ingres, e teve como objetivo principal agregar o menor recurso necessário para suportar tipos de dados extensíveis com índices e conceitos objeto-relacional. Além disso, possui mais de 905.000 linhas de código fonte e oferece diversas funcionalidades tais como *views*, *triggers*, chaves estrangeiras, *stored procedures*, processamento transacional, etc. Atualmente o PostgreSQL é o banco de dados padrão do macOS e é usado por várias organizações em todo o mundo(ex. *CISCO*, *NTT Data*, *Fujitsu*) [PostgreSQL 2017b].

4.3.1.3.1 O bug de vazamento

O *bug* de vazamento de memória encontrado para o PostgreSQL e estudado nesta pesquisa refere-se a um problema relacionado a uma função criada dentro de uma base de dados, escrita em *Perl,* que recebe dados do tipo texto como entrada e processa esses dados de forma específica antes de retornar o resultado. Esse tipo de função pode ser muito útil quando se deseja realizar processamentos de dados originados de uma ou mais tabelas, ainda dentro do banco de dados, antes de retornar para a aplicação cliente. O vazamento foi reportado no fórum oficial de relatos de *bugs* do PostgreSQL para a versão 9.2.6 do software [PostgreSQL 2017c] e possui como padrão de ativação os seguintes procedimentos:

1. Criar uma base de dados no PostgreSQL.
2. Criar, dentro da base de dados, uma ou mais tabelas e realizar seu(s) preenchimento(s).
3. Criar uma função na base de dados, escrita em *Perl*, similar a reportada em [PostgreSQL 2017b].
4. Chamar essa função passando como entrada o retorno de uma consulta realizada na(s) tabela(s) criada(s) no 2º passo. Em cada chamada da função, os dados serão processados e o vazamento de memória ocorrerá.

```
create function perl_test(IN data text, OUT v1 text, OUT v2 integer, OUT v3
integer )
  returns record as
$BODY$

use strict;
use warnings;

my $res->{'v1'} = 'content';

return $res;

$BODY$
  language plperlu volatile strict;
```

Figura 4.3. PostgreSQL: padrão de ativação – função escrita em Perl.

Uma representação do padrão de ativação do *bug* do Inkscape pode ser visto na Figura 4.3. A confirmação do vazamento de memória foi realizada e as versões

base (9.1.24) e alvo (9.2.0) foram encontradas através de testes preliminares. É importante ressaltar uma particularidade para esse bug: a versão base não oferece suporte a dados do tipo *json* (um dos tipos de dados retornados na função *Perl* reportada em [PostgreSQL 2017b]). Entretanto, tal particularidade não influencia na manifestação do vazamento, uma vez que o problema está relacionado à linguagem da função: *Perl*. A mesma função escrita em *PL/Python* não ocasiona o vazamento. Assim, o retorno do tipo *json* foi removido na função criada para a realização dos experimentos.

4.3.1.3.2 Caracterização da carga de trabalho

Uma vez que o vazamento está ligado a uma função escrita em *Perl* que processa dados do tipo de texto, como os de uma instrução *Select*; então o estudo de caracterização da carga de trabalho procurou focar em pesquisar trabalhos acadêmicos que utilizaram algum tipo de carga de trabalho direcionado a uma quantidade de instruções SQL realizadas em um período de tempo. Por exemplo, um perfil de carga de trabalho contemplando uma quantidade de *Selects*; *Updates*; *Inserts*; *Deletes*; etc.. Esta pesquisa também buscou encontrar repositórios de dados que tivessem informações estatísticas relacionadas a instruções SQL executadas em algum banco de dados. Uma visão geral dos trabalhos encontrados é apresentada na Tabela 4.4.

Tabela 4.4. PostgreSQL: visão geral dos trabalhos encontrados no processo de caracterização.

Trabalho	Objetivo	Resultados (taxa média de requisições)
[Yu *et al.* 1992]	Estudar a estrutura e complexidade de instruções SQL; o comportamento e composição das consultas; as transações e as relações utilizadas em uma base de dados.	1) *Select*: 72.804/hora; 2) *Update*: 150/hora; 3) *Insert*: 6/hora e; 4) Outras: 1.936/hora.
[Armstrong *et al.* 2013]	Fornecer um *benchmark* baseado em bancos de dados que armazenam gráficos sociais do Facebook, voltado para serviços *web* de grande escala (*LinkBench*).	1) *Select*: 1,68 milhões/hora; 2) *Update*: 374 mil/hora; 3) *Insert*: 281 mil/hora e; 4) Delete: 97 mil/hora.
[Bronson *et al.* 2013]	Analisar uma amostra aleatória de 6,5 milhões de solicitações de uma base de dados que armazena gráficos sociais do Facebook.	1) *Select*: 6.790/hora; 2) *Update*: 2,92/hora; 3) *Insert*: 9,34/hora e; 4) Delete: 1,39/hora.
[Yao *et al.* 2005]	Avaliação de uma abordagem que agrupa seções de usuário, em um banco de dados, para realizar operações SQL. Para isso, analisou *logs* de uma base dados real.	1) *Select*: 7.583; 2) *Update*: 572 e; 3) *Insert*: 213. ** Os dados coletados da base de dados real não são apresentados por período.

É importante mencionar que muitos outros trabalhos que examinaram o desempenho de bases de dados, aplicando correlações entre transações executadas e recursos do sistema operacional, foram encontrados [Lo *et al.* 1992]; [Elnaffar *et al.* 2002]; [Hankins *et al.* 2003]; [Shao *et al.* 2005] e [Ahmad *et al.* 2009]. Entretanto, esses trabalhos não foram incluídos na Tabela 4.4 por não apresentaram resultados em nível de classificação de instruções SQL tornando impossível sua utilização na caracterização da carga de trabalho.

Também é preciso dizer que relevantes *benchmarks* (ex. [TPC 2017a], [TPC 2017b e [TPC 2017c]]) foram investigados, entretanto todos eles também foram descartados porque suas informações não se assemelham ao perfil necessário para a caracterização do PostrgreSQL.

Devido a falta de maiores opções para caracterizar o perfil de carga de trabalho para o PostgreSQL, principalmente por [Yu *et al.* 1992] ser muito antigo e [Armstrong *et al.* 2013] e [Bronson *et al.* 2013] estarem distante da realidade pretendida, pois avaliam informações de milhares de operações SQL realizadas pelo Facebook, então [Yao *et al.* 2005] foi considerado uma boa opção para caracterização. É relevante mencionar que [Yao *et al.* 2005] foi suficientemente importante a ponto de ter sido mencionado posteriormente em [Kamra *et al.* 2008] e [Shabtai *et al.* 2012] (esse último uma publicação em um importante periódico na área de banco de dados).

Os dados de [Yao *et al.* 2005] fazem parte de um conjunto real de dados coletado de um servidor de banco de dados de uma clínica de fisioterapia privada localizada em Toronto. Essa clínica possui cinco (5) filiais em toda cidade e, a cada dia, nove aplicativos realizam as mais diversas tarefas através de conexões com o banco de dados (ex. verificação de pacientes, agendamento de atendimento, exibição de horários e procedimentos de tratamentos, venda de produtos, etc.).

Entretanto, o período de rastreamento desse conjunto real não foi informado em [Yao *et al.* 2005] e, para resolver esse problema, utilizou-se o critério de distribuir as instruções do conjunto real (7.583 *Selects*, 213 *Inserts* e 572 *Updates*) em um período de 3h30min para caracterizar o cenário de carga constante e normal (3h30min é o tempo experimental sob carga de trabalho – ver Seção 3.3, etapa 4). Devido a melhores critérios para definir os outros cenários de carga, adotou-se dividir e multiplicar por 2, respectivamente, os valores médios de cada instrução, em 3h30min, para os cenários de carga constante e baixa e carga constante e alta. A Tabela 4.5 apresenta uma visão geral dos cenários de cargas de trabalho definidos para o PostgreSQL.

Tabela 4.5. PostgreSQL: cenários de cargas de trabalho caracterizados.

Cenário de carga	**Caracterização**
Constante e alta	a) 1 *Select* a cada 0,83s; b) 1 *Update* a cada 11,01s e; c) 1 *Insert* a cada 29,58s.
Constante e normal	a) 1 *Select* a cada 1,66s; b) 1 *Update* a cada 22,0s e; c) 1 *Insert* a cada 59,15s.
Constante e baixa	a) 1 *Select* a cada 3,32s; b) 1 *Update* a cada 44,06s e; c) 1 *Insert* a cada 118,30s.
Variada	a) 1 *Select* entre a cada 0,83s e 3,32s; b) 1 *Update* entre a cada 11,01s e 44,06s; c) 1 *Insert* entre a cada 29,58s e 118,30s.

4.3.1.4 Squid

O Squid é um *proxy* de *cache* para *web*, predominantemente escrito em *C++* [15], que suporta protocolos conhecidos (ex. HTTP, FTP, DNS e outros), além de oferecer suporte à maioria dos sistemas operacionais disponíveis (incluindo Windows, Linux e MAC OS X) [Squid 2017a]. É um projeto de software maduro e bem estabelecido, com mais de 24 anos de existência e mais de 810.000 linhas de código fonte em sua versão atual [Openhub 2017d]. Começou a ser desenvolvido no início dos anos 90 como parte do projeto *Harvest*, financiado pela Agência de Projetos de Pesquisa Avançada de Defesa – *Defense Advanced Research Projects*

Agency (DARPA) [DARPA 2017] e hospedado na Universidade do Colorado em Boulder.

Os principais benefícios de um sistema *proxy/cache* são relativos às áreas de segurança (ex. filtragem de conteúdo), desempenho (ex. armazenamento de conteúdo em *cache*) e disponibilidade (ex. balanceamento de carga). Assim, O Squid pode ser usado de várias formas, sendo a mais comum, a de desempenhar um papel de camada intermediária entre clientes e servidores para diferentes protocolos, reduzindo a largura da banda, melhorando tempos de resposta ao armazenar conteúdo em *cache* e reutilizando páginas *web* frequentemente acessadas [Matias *et al.* 2016]. Além disso, é um dos aceleradores de conteúdo mais antigos [Squid 2017a]. Por essas razões que o Squid é usado por centenas de provedores de internet e sites em todo mundo para aumentar a entrega de conteúdo. A organização Wikipedia é, provavelmente, um dos casos de maior sucesso de utilização do Squid no mundo [Wikimedia Blog 2014], processando aproximadamente 75% de todo tráfego recebido pela organização. Muitos serviços de incorporação a produtos de aparelhos de redes, ampliando o campo de atuação, também são oferecidos pelo Squid [Squid 2017b].

4.3.1.4.1 O bug de vazamento

O *bug* de vazamento de memória encontrado para o Squid e estudado nesta pesquisa refere-se a um problema relacionado ao processamento de listagens de diretórios FTP. Assim, toda vez que o Squid recebe uma solicitação de um cliente (ex. um navegador *web*) para listar diretórios de um servidor FTP, ele processa a solicitação e, ao analisar os resultados recebidos do servidor FTP, via listagem de diretórios, ocorre o vazamento. O vazamento foi reportado no fórum oficial de relatos de *bugs* do Squid para a versão 3.4.6 do software [Bugs Squid 2015] e possui como padrão de ativação os seguintes fatores:

1. Executar o Squid em uma máquina realizando papel de servidor.
2. Em uma máquina cliente, configurar o servidor Squid para filtrar todas as requisições *web* realizadas por esse cliente – recomenda-se utilizar um navegador *web*.
3. Na máquina cliente, realizar uma requisição FTP através de uma URL FTP que liste diretórios de um servidor FTP. Cada listagem ocasionará vazamento de memória.

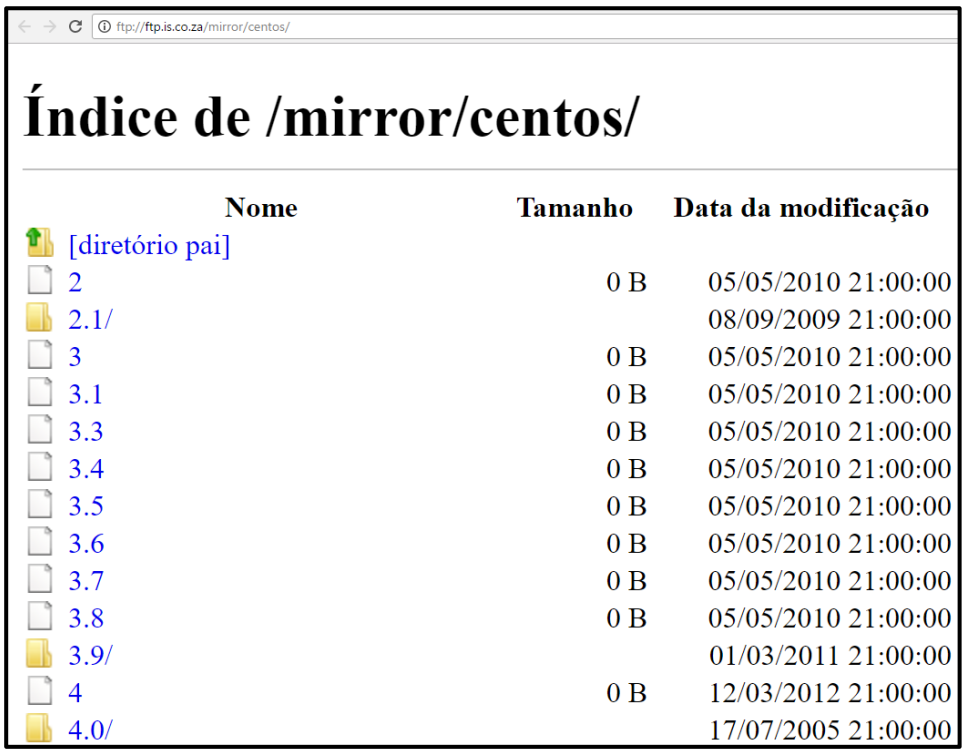


Figura 4.4. Squid: padrão de ativação - listagem de um diretório FTP.

Uma representação do padrão de ativação do *bug* do Inkscape pode ser visto na Figura 4.4. Testes preliminares confirmaram o vazamento de memória e encontraram as versões base (3.1.23) e alvo (3.2.1).

4.3.1.4.2 Caracterização da carga de trabalho

Uma vez que o vazamento está ligado listagens de diretórios FTP, então o estudo de caracterização de representatividade da carga de trabalho procurou focar em pesquisar trabalhos acadêmicos que fizeram uma caracterização através de requisições por protocolos (ex. HTTP, FTP, etc.). Tendo em mente uma quantidade de requisições FTP recebidas por um servidor *web* em seu ambiente de produção, é possível caracterizar uma carga de trabalho, referente ao padrão de ativação do *bug* do Squid, que reflita um comportamento real da carga recebida. Uma visão geral dos trabalhos acadêmicos encontrados pode ser vista na Tabela 4.6. É importante dizer que alguns estudos apresentam resultados em termos de fluxo e outros em termos de requisições e, por isso, essa diferenciação está presente na tabela abaixo.

Diante todos os trabalhos acadêmicos presentes na Tabela 4.6, o único a apresentar resultados mais concretos relativos a requisições FTP foi o realizado em [Arlitt *et al.* 1999]. Todos os outros, com exceção de [Claffy *et al.* 1998], ou apresentaram apenas em porcentagem ou não evidenciaram tais números – em [Claffy *et al.* 1998] os resultados são apresentados como fluxo de dados. Além disso, todos esses estudos são relativamente antigos e, portanto, considerou-se a hipótese da utilização de repositórios de dados estatísticos, mais atuais, de servidores *web* para caracterizar o perfil de carga de trabalho para o *bug* do Squid estudado.

Tabela 4.6. Squid: visão geral dos trabalhos encontrados no processo de caracterização.

Trabalho	Objetivo	Resultados (taxa média de requisições)
[Thompson *et al*. 1997]	Apresentar padrões e características do tráfego na internet através de dados coletados de 2 *backbones* comerciais da *MCI Communications Corp*.	Fluxos FTP → 1% de todos os fluxos analisados. **Resultados apresentados em %.
[Claffy *et al.* 1998]	Nova análise do tráfego dos backbones avaliados em [Thompson *et al*. 1997].	123.702 fluxos FTP/dia.
[Arlitt *et al.* 1999]	Caracterizar a carga de trabalho de um servidor *proxy web* em um importante ISP.	2.306 requisições FTP/dia.
[Yuksel *et al.* 2000]	Analisar o tráfego de um *bakcbone* da Fundação Nacional de Ciência (NSF) dos EUA.	Requisições FTP → 14% de todos as requisições analisadas. **Resultados apresentados em %.
[Erman *et al.* 2006]	Análise o tráfego de internet através de dados obtidos de 2 universidades.	Requisições FTP → 1,3% de todos as requisições analisadas. **Resultados apresentados em %.

Assim, 3 repositórios referentes a espelhos (*mirrors*) FTP: 1) NLUUG [NLUUG/SURFnet FTP Server Usage 2017]; 2) Universidade I-Shou [I-Shou University FTP Server Usage 2005] e; 3) JAIST [JAIST FTP Server Usage 2017]), contendo estatísticas de acesso, foram encontrados. Depois de analisar as informações desses 3 repositórios chegou-se a conclusão de que o mais adequado para a caracterização da carga de trabalho foram os dados do JAIST. Isso porque estão consideravelmente atualizados e disponibilizam informações mais adequadas para a caracterização.

O JAIST refere-se ao Instituto Avançado de Ciência e Tecnologia Japonês que foi fundado em 1990, como a primeira escola nacional independente de pós-graduação do Japão, com o objetivo de realizar pesquisas no mais alto nível em ciência e tecnologia avançada. Por ser um modelo de excelência em pós-graduação, o JAIST foi incorporado como uma Corporação Universitária Nacional, no Japão, em 2004 [JAIST 2017a]. Com a finalidade de promover a pesquisa em nível mundial e treinar os alunos para contribuir com a sociedade internacional, o JAIST trabalha ativamente em intercâmbios acadêmicos de professores e estudantes, além de realizar pesquisas colaborativas com mais de 119 institutos de pesquisas espalhados por 31 países (no Brasil está em parceria com a Universidade de São Paulo – USP). Atualmente conta com mais de 904 estudantes/pesquisadores realizando mais de 140 projetos de pesquisas anuais [JAIST 2017a]. O JAIST possui um servidor com 64 TB de armazenamento e é certificado como um servidor *mirror* oficial para muitos projetos [JAIST 2017b].

Assim, os dados encontrados em [JAIST FTP Server Usage 2017] são referentes ao período de Fevereiro de 2013 a Fevereiro de 2014 onde uma taxa média de 1729 visitas/dia recebidas ao servidor FTP foi encontrada. Portanto, considerando que, a cada visita recebida, um diretório FTP é listado, então essa taxa média foi incorporada à caracterização do cenário de carga constante e normal. Para carga constante e baixa, foi adotado o critério de levar em conta a média dos meses que obtiveram um número de visitas inferior ao número médio de 1.729 visitas/dia, resultando em 1.233 visitas/dia. Para carga constante e alta, o critério adotado foi justamente o contrário, chegando-se ao valor de 2.155

visitas/dia. A Tabela 4.7 apresenta uma visão geral dos cenários de cargas de trabalho definidos para o Squid.

Tabela 4.7. Squid: cenários de cargas de trabalho caracterizados.

Cenário de carga	Caracterização
Constante e alta	1 diretório FTP listado a cada aprox. 40,09s → valor obtido da taxa média de 2.155 visitas/dia.
Constante e normal	1 diretório FTP listado a cada aprox. 49,98s → valor obtido da taxa média de 1.729 visitas/dia.
Constante e baixa	1 diretório FTP listado a cada aprox. 70,07s → valor obtido da taxa média de 1.233 visitas/dia.
Variada	1 diretório FTP listado entre a cada aprox. 40,09s e 70,07s → valor obtido entre as taxas média de 2.155/dia e 1.233/dia.

4.3.2 Projeto dos Experimentos

Com as aplicações e suas respectivas caracterizações dos cenários de carga de trabalho definidos, o próximo passo é o de realizar os experimentos. Para cada aplicação foi realizado um conjunto experimental composto de 10 replicações, onde em cada replicação os indicadores de envelhecimento *resident set size* (RSS) e *heap usage* (HUS) foram monitorados gerando, cada um, uma série temporal distinta. A Tabela 4.8 apresenta uma visão geral da quantidade de séries temporais obtidas dos experimentos em cada versão (alvo e base) e cenário de carga de trabalho de uma aplicação. Assim, para cada cenário e versão, 20 séries temporais (10 para HUS e 10 para RSS) são alcançadas. Posteriormente, cada conjunto de 10 séries foi processado pelo método *Dynamic Time Warping* (DTW) a fim gerar uma série temporal média e reduzir os efeitos de defasagem entre as replicações.

Tabela 4.8. Visão geral da quantidade de séries temporais obtidas dos experimentos por versão e cenário de carga em uma aplicação.

Cenário de carga	Versão alvo		Versão Base	
	Quantidade de séries temporais		Quantidade de séries temporais	
	HUS	RSS	HUS	RSS
Constante e alta	10	10	10	10
Constante e normal	10	10	10	10
Constante e baixa	10	10	10	10
Variada	10	10	10	10
Série média (DTW)	1 por cenário	1 por cenário	1 por cenário	1 por cenário

Cada replicação teve a duração de 4 horas, sendo 3h 30min processando a carga de trabalho e os últimos 30 minutos sem carga. Esse último período de 30 minutos foi utilizado para observar se a memória foi temporariamente alocada como consequência da carga de trabalho processada ou se realmente representou um vazamento de memória. Assim, é importante dizer que em cada replicação os indicadores RSS e HUS foram monitorados a cada 5 segundos, gerando sempre séries temporais com 2.880 observações para cada um (RSS e HUS). Entretanto, as observações dessas séries consideradas no processamento do DTW se referem as primeiras 3h 30min sob carga de trabalho, ou seja, as primeiras 2.520 observações.

A estratégia de replicação foi usada para evitar erros experimentais. Erros experimentais estão presentes em qualquer estudo experimental e o uso de replicações é uma forma de trabalhar a validade interna em estudos empíricos de engenharia de software [Siegmund *et al.* 2015]. Como a abordagem analisada nesta pesquisa tem como ideia principal a análise diferencial de software, então, em cada

cenário de carga, o conjunto de dez replicações experimentais foi aplicado tanto para a versão base quanto para a versão alvo.

O tempo experimental de quatro horas foi definido e justificado em [Matias *et al.* 2014], por ser suficiente para observar efeitos de vazamento em uma análise diferencial. Portanto, cada cenário de carga passou por 80 horas experimentais – 40 na versão base e 40 na versão alvo. Uma vez que foram considerados 4 cenários de carga, então um total de 320 horas experimentais foi utilizado para cada aplicação (ver Tabela 3.1).

O Algoritmo 3 apresenta uma visão geral desse projeto experimental, onde recebe como entrada uma das aplicações utilizadas para os experimentos (Seção 4.3.1) e realiza os mesmos conjuntos de replicações, para cada cenário de carga de trabalho, nas versões alvo e base (linhas 04 e 05). As linhas de 06 a 30 são responsáveis pela função que aplica o conjunto de experimentos, onde uma lista contendo os cenários: 1) constante e alto, 2) constante e normal, 3) constante e baixo, e 4) variado, é carregada e colocada sob iterações (linha 08). A cada iteração o conjunto de dez replicações é realizado em cada cenário (linhas de 09 a 28), onde 3h30 min são destinados à carga de trabalho (linhas de 18 a 21) e os últimos 30 min restantes sem carga (linhas de 23 a 26) – durante a cada replicação, os indicadores HUS e RSS são coletados a cada 5 segundos (linhas 16 e 27).

Algoritmo 3 – Visão geral do projeto experimental.

```
Entrada: aplicação selecionada para os conjuntos experimentais
01: versaoAlvo:= versão da aplicação com vazamento
02: versaoBase:= versão da aplicação sem vazamento
03: //realização dos conjuntos de experimentos em ambas as versões
04: conjuntoExperimentos(versaoAlvo)
05: conjuntoExperimentos(versaoBase)
06: funcao conjuntoExperimentos(versaoApp)
07:   cenariosCarga := lista contemplando todos os cenários de carga de trabalho
08:   para cada c em cenariosCarga faca
09:     para cada i:=1 ate 10 faca
10:       //cada experimento durou 4 horas
11:       //3h:30min sob carga de trabalho
12:       //0h:30min sem carga de trabalho
13:       tempoAtual:= tempo atual
14:         tempoFinalizarCarga:= tempo em que a carga será finalizada
15:         tempoFinalizarExperimento:= tempo em que o experimento será finalizado
16:         começa a coletar RSS e HUS a cada 5 segundos na versaoApp
17:         //3h:30min sob carga de trabalho
18:         enquanto tempoAtual <= tempoFinalizarCarga faca
19:           realiza o cenário de carga de trabalho c na versaoApp
20:           tempoAtual:= recebe o tempo atual
21:         fim enquanto
22:         //0h:30min sem carga de trabalho
23:         enquanto tempoAtual <= tempoFinalizarExperimento faca
24:           não faz nada, apenas observa o comportamento na versaoApp
25:           tempoAtual:= recebe o tempo atual
26:         fim enquanto
27:         para de coletar RSS e HUS a cada 5 segundos na versaoApp
28:         fim para
29:   fim para
30: fim funcao
```

Os cenários de cargas de trabalho planejados com comportamento o mais próximo do real foram automatizados no processo de geração da carga de trabalho. Essas automatizações garantiram que, para um dado cenário de carga, todas as replicações experimentais fossem exatamente iguais, evitando influências experimentais e possíveis erros humanos:

- **Inkscape:** foi criado um *script*, escrito em *Phython*, responsável por colocar arquivos de extensão SVG em *preview* durante determinado intervalo de tempo, ao qual será chamado de Script-Inkscape.
- **Lighttpd:** foi criada uma página *web* de extensão SHTML, com instruções SSI do tipo IF/ELIF, hospedada no servidor Lighttpd, contendo um código *Javascript* responsável por recarregar a página em intervalos de tempo, ao qual será chamada de Página-Lighttpd.
- **PostgreSQL:** foi criado um *script*, escrito em *Python*, responsável por processar a carga de instruções SQL (*Selects*; *Updates* e *Inserts*) em intervalos de tempo numa base de dados, onde os dados retornados de cada *Select* realizado foram colocados como valor de entrada da função *Perl*. Esse *script* será nomeado como Script-PostgreSQL.
- **Squid:** foi criado um página *web* de extensão HTML contendo um código *Javascript* responsável por realizar listagens de diretórios FTP, em intervalos de tempo, em um navegador *web* presente em uma máquina desempenhando papel de cliente e configurado para ter suas requisições filtradas por um servidor com o Squid instalado. Para essa proposta o código acessou uma *lista* de 100 *URLs FTP* contendo diferentes diretórios de servidor*es* FTP reais na internet. A mesma lista foi utilizada em todos os experimentos. Essa página será chamada de Página-Squid.

Os intervalos de tempo de cada uma dessas automatizações foram controlados, de acordo com cada cenário de carga de trabalho definido na caracterização. Tal processo será explicado na Seção seguinte, que é destinada a parte instrumental e por isso cada automatização recebeu uma nomenclatura, de forma a facilitar a explicação.

4.3.3 Instrumental

A bancada de testes utilizada para os experimentos consistiu de um computador hospedando VMs para a realização dos testes de análise diferencial em cada aplicação. As VMs foram criadas de acordo com a necessidade de cada experimento e informações sobre elas podem ser vistas na Tabela 4.9.

Tabela 4.9. Visão geral da bancada de testes utilizada nos experimentos.

Máquinas	Configuração	Objetivo
Hospedeira	Intel Core i7-4510U CPU @ 2.60GHz, 8GB de RAM, 500 GB de disco, Windows 10.	Hospedar VMs para realização dos testes de análise diferencial.
VM1-Inkscape	4 CPU Cores, 3.5 GB RAM, 50 GB disco, Linux 3.11.0-15-generic x86_64.	Comportar ambas as versões (alvo e base) do Inkscape executando o Script-Inkscape para explorar seu bug de vazamento.
VM2-Lighttpd-1	4 CPU Cores, 2.0 GB RAM, 50 GB disco, Linux 3.11.0-15-generic x86_64.	Comportar ambas as versões (alvo e base) do Lighttpd e hospedar a Página-Lighttpd para receber requisições de uma máquina desempenhando papel de cliente (VM3-Lighttpd-2).
VM3-Lighttpd-2	4 CPU Cores, 1.0 GB RAM, 50 GB disco, Linux 3.11.0-15-generic x86_64.	Desempenhar papel de cliente e realizar requisições à Página-Lighttpd hospedada na VM2-Lighttpd-1.
VM4-PostgreSQL-1	4 CPU Cores, 3.5 GB RAM, 50 GB disco, Linux 3.11.0-15-generic x86_64.	Comportar ambas as versões (alvo e base) do PostgreSQL com uma base de dados recebendo requisições de instruções SQL do Script-PostgreSQL, desempenhando papel de cliente e executado também nessa máquina.
VM5-Squid-1	4 CPU Cores, 2.0 GB RAM, 50 GB disco, Linux 4.4.0-28-generic x86_64.	Comportar ambas as versões (alvo e base) do Squid filtrando todas as requisições *web* de uma máquina desempenhando papel de cliente (VM6-Squid-2).
VM6-Squid-2	4 CPU Cores, 1.0 GB RAM, 50 GB disco, Linux 2.6.32-431.el6.x86_64 x86_64.	Desempenhar papel de cliente comportando a Página-Squid para realizar listagens de diretórios FTP, onde cada listagem é filtrada pelo Squid instalado na VM5-Squid-1.

Para criação dessas VMs foi utilizado o VMWare Workstation 11.1.0. Houve casos em que se optou por criar mais de uma VM por aplicação estudada, como para o Lighttpd e para o Squid, onde as cargas de trabalho foram executadas sob VMs desempenhando papel de cliente. Em contrapartida, para outros casos, a mesma VM desempenhando papel de servidor também realizou o de cliente (PostgreSQL). O Inkscape, se tratando de uma aplicação do tipo *desktop*, não precisou de outra VM para executar a carga de trabalho. Para explorar as automatizações (Script-Inkscape, Página-Lighttpd, Script-PostgreSQL e Página-Squid) os seguintes processos de geração de carga de trabalho foram utilizados:

1) Script-Inkscape: esse *script* foi composto de uma API do *Phython* para Interface Gráfica de Usuário – *Graphical User Interface* (*GUI*) conhecida como PyAutoGUI [PyAutoGUI 2014]. O objetivo dessa API é fornecer uma multiplataforma para automação de testes GUI, que caso contrário, seriam realizados por seres humanos. Assim, uma vez que o padrão de ativação do *bug* do Inkscape necessita de arquivos SVG colocados em preview (ver Seção 4.3.1.1), então ao invés de durante cada replicação de cada conjunto de replicações experimentais (ver Seção 4.3.2) um ser humano precisar ficar durante 3h30min colocando arquivos em *preview*, o PyAutoGUI automatizou esse processo. Assim indesejáveis influências experimentais ocasionadas por seres humanos são evitadas, o que seria impossível de não acontecer caso o processo fosse feito por humanos, visto que se esse um precisaria ficar cerca de 320 horas realizando essas ações.

Portanto, o Script-Inkscape, incorporando o PyAutoGUI, foi executado juntamente com o Inkscape e as ações de colocar arquivos SVG em *preview* foram realizadas em intervalos de tempo. Assim, depois de colocar um arquivo em *preview*, o Script-Inkscape realiza uma pausa até colocar o próximo arquivo. Esse controle temporal variou de acordo com cada cenário de carga de trabalho

configurado no processo de caracterização (ver Tabela 4.1). Para garantir que no modo variado todas as replicações seriam exatamente iguais, um arquivo contendo números randômicos de 0,497 a 1,019 foi gerado através de um site gerador de números randômicos [RANGON.ORG 2017]. Portanto, a cada pausa o Script-Inkscape lê uma posição desse arquivo randômico. A Figura 4.5 mostra uma visão geral da bancada de testes para o Inkscape.

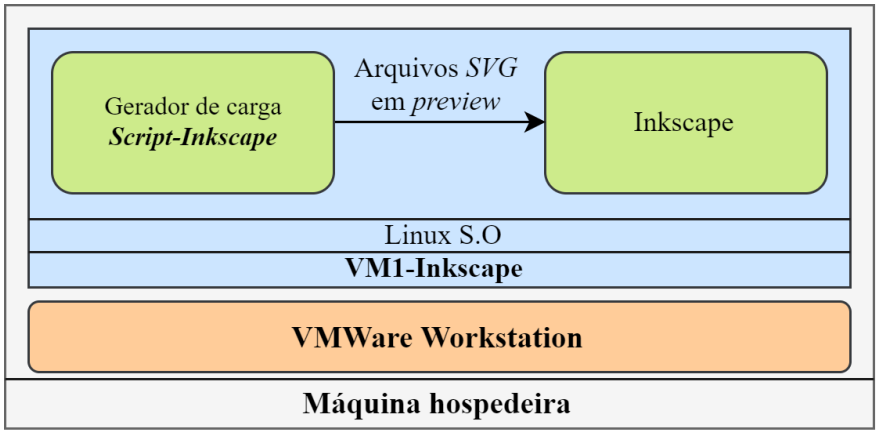


Figura 4.5. Inkscape: visão geral da bancada de testes.

2) Página-Lighttpd: essa página de extensão SHTML, com instruções SSI, do tipo IF/ELIF, responsáveis por explorar o padrão de ativação relativo ao *bug* do Lighttpd (ver Seção 4.3.1.2) foi hospedada na VM2-Lighttpd-1. Assim, a geração de carga de trabalho consistiu simplesmente, em cada replicação experimental dos conjuntos de experimentos, acessar a Página-Lighttpd através de requisições realizadas por um navegador *web* localizado na VM3-Lighttpd-2. Uma vez aberto o navegador *web* e solicitado o acesso à Página-Lighttpd, o processo foi automatizado pelo código *Javascript*, presente nessa página, responsável por atualizar a página em intervalos de tempo. Esse controle temporal variou de acordo com cada cenário de carga conforme visto no processo de caracterização (ver Tabela 4.3). A carga variada também considerou número randômicos de 1,17 a 9,45 [RANGON.ORG 2017] para garantir igualdade entre as replicações. A Figura 4.6 mostra uma visão geral da bancada de testes para o Lighttpd.

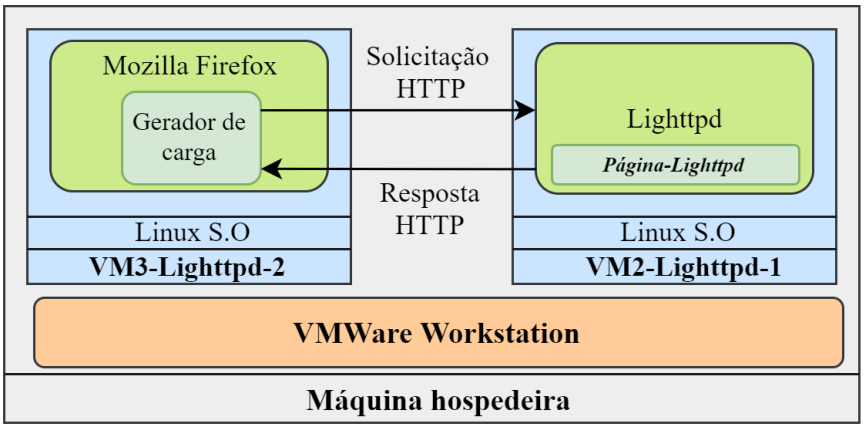


Figura 4.6. Lighttpd: visão geral da bancada de testes.

3) Script-PostgreSQL: esse *Python script* foi responsável por realizar uma conexão a um banco de dados e processar a carga de instruções SQL definidas na

caracterização, fazendo com que os dados retornados de cada *Select* fossem colocados como valor de entrada da função *Perl* – função responsável pelo padrão de ativação do *bug* estudo para o PostgreSQL (ver Seção 4.3.1.3).

Uma vez que a carga de trabalho foi composta de instruções do tipo *Select*; *Update* e *Insert*; então tal *script* criou *Threads*, uma para cada instrução, de forma a processá-las simultaneamente em intervalos de tempo – cada tipo de instrução tem seu respectivo intervalo de tempo. Portanto, depois de realizar uma instrução, a *Thread* responsável faz uma pausa até realizar a próxima. Esse controle temporal variou de acordo com cada cenário de carga de trabalho (ver Tabela 4.5). A carga variada utilizou três arquivos contendo números randômicos (um para cada tipo de instrução) [RANGON.ORG 2017] para garantir que todas as replicações fossem exatamente iguais (ex. o arquivo para *Selects* contemplou números randômicos de 0,83 a 3,32). Portanto, em cada pausa de uma respectiva *Thread* do Script-PostgreSQL uma posição do seu respectivo arquivo randômico é lida. A Figura 4.7 mostra uma visão geral da bancada de testes para o PostgreSQL.

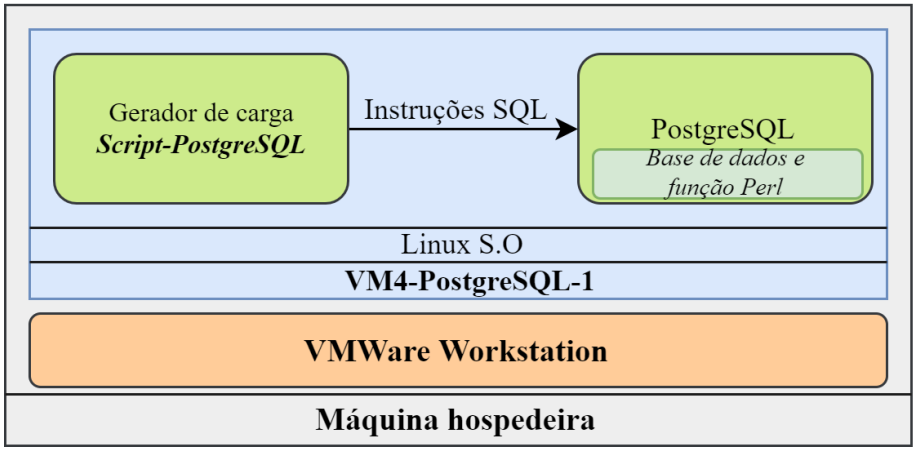


Figura 4.7. PostgreSQL: visão geral da bancada de testes.

4) Página-Squid: essa página de extensão HTML foi criada para ser executada, através de um navegador *web*, pela VM6-Squid-2. Tal página contempla um código *Javascript* responsável por realizar listagens de diretórios FTP, explorando o padrão de ativação relativo ao *bug* do Squid (ver Seção 4.3.1.4), em intervalos de tempo. Assim, com o navegador *web* configurado para ter suas requisições filtradas pelo Squid executando na VM5-Squid-1, o processo de geração de carga de trabalho consistiu em acessar uma lista de 100 URLs FTP, contendo diferentes diretórios de servidores FTP reais na internet, escolhendo, para cada listagem no navegador, uma URL dessa lista.

Portanto, ao escolher uma dessas URLs, a Página-Squid lista seus diretórios abrindo uma nova aba do navegador e realiza uma pausa até a próxima listagem, onde escolhe a próxima URL da lista. Ao fim das 100 posições, a Página-Squid reinicia a leitura da lista pela primeira posição. Esse controle temporal variou de acordo com a definição dos cenários de carga de trabalho para o Squid (ver Tabela 4.7). A carga variada considerou números randômicos de 40,09 a 70,07 para garantir a execução igualitária de todas as replicações [RANGON.ORG 2017]. Portanto, a cada pausa o código *Javascript* presente na Página-Squid lê uma posição desse arquivo randômico. A Figura 4.8 mostra uma visão geral da bancada de testes para o Squid.

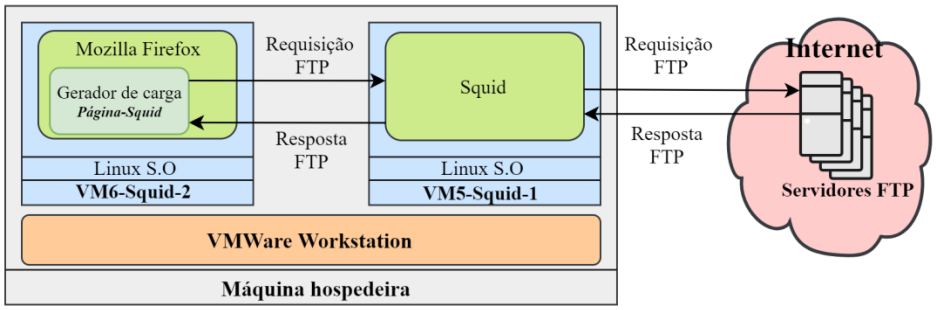


Figura 4.8. Squid: visão geral da bancada de testes.

Com relação à instrumentação para coleta de dados, os indicadores de envelhecimento RSS e HUS foram monitorados a cada replicação de 4h dos conjuntos de replicações experimentais (ver Seção 3.3, etapa 4). Para o monitoramento do RSS, um *shell script* coletando valores de RSS do sistema de arquivos Linux/proc foi utilizado. Para o HUS, todas as operações de alocação e desalocação dinâmica, realizadas em tempo de execução, foram interceptadas. Para isso, um *wrapper* de alocação dinâmica de memória chamado *DebugMalloc*, introduzido em [Costa e Matias 2015], foi utilizado. Uma vez que o RSS foi monitorado a cada 5 segundos, então os dados coletados para HUS foram ajustados para serem apresentados no mesmo intervalo de tempo. Portanto, os resultados apresentados na Seção seguinte são baseados nas séries temporais médias, estimadas pelo DTW, de cada indicador e cenário de carga de trabalho (ver Tabela 4.8). Essas séries temporais médias foram combinadas com as técnicas estatísticas através do cálculo dos valores de divergência.

5. RESULTADOS

5.1 Introdução

De acordo com o que foi explicado no Capítulo 4, a abordagem de análise diferencial de software foi avaliada frente a um conjunto de replicações de experimentos contemplando cenários de cargas de trabalho e aplicações reais. Uma série temporal média, por indicador, foi estimada pelo DTW para cada conjunto de replicações em um cenário de carga de trabalho. Essas séries temporais médias foram usadas no cálculo dos valores de divergência, através da combinação com técnicas estatísticas. Pelos valores de divergência é possível comparar de forma justa as combinações técnica/indicador e estimar a melhor delas para cada cenário e aplicação.

Os resultados obtidos do cálculo dos valores de divergência são exibidos, por aplicação, na Seção 5.2. A Seção 5.3 apresenta uma síntese dos resultados, relacionando os encontrados em cada aplicação e, posteriormente, associando-os com os trabalhos anteriores que foram fundamentais para esta pesquisa. Finalmente, essa associação é utilizada para conclusões em cima das perguntas e hipóteses de pesquisa e são apresentadas no na Seção 4.2.

5.2 Resultados

Os resultados, separados por aplicação, apresentados nesta Seção foram baseados nos seguintes critérios de avaliação:

- **Falso-negativos:** verificar se as técnicas estatísticas estudadas conseguiram detectar o vazamento de memória ou geraram falso-negativos.
- **Falso-positivos:** verificar se as técnicas estatísticas estudadas geraram falso-positivos.
- **Melhor técnica:** verificar, por meio dos tempos de detecção, quais técnicas estatísticas estudadas (que não geraram falso-negativos), foram mais eficientes em detectar o vazamento de memória em cada cenário. Nessa análise também é verificado o melhor tipo de técnica para cada caso (detecção de tendências ou CEP).
- **Melhor indicador:** verificar, por meio dos tempos de detecção, qual indicador foi mais efetivo em detectar o vazamento de memória (RSS ou HUS).
- **Melhor combinação técnica/indicador:** tendo encontrado a melhor técnica e o melhor indicador para cada cenário, verificar qual a melhor combinação geral.

Para ajudar nesta avaliação, foram utilizadas tabelas contendo um *ranking* de desempenho das melhores técnicas por indicador. De acordo com a Seção 3.6, a melhor combinação é aquela com o menor tempo de detecção e que, frequentemente, não possui nenhum falso-positivo (*DivFirstTime* = *DivLastTime* e *DivNumEvents* = 0).

Baseado nestes conceitos, o *rankings* das tabelas foram ordenados pelos tempos de detecção. Assim, mesmo que uma combinação seja ideal (*DivFirstTime* = *DivLastTime*) ela poderá ficar atrás de outra combinação não ideal (DivFirstTime ≠ *DivLastTime* e *DivNumEvents* > 0) se não apresentar um tempo de detecção melhor (embora isso seja uma ocorrência rara). Houve casos em que mais de uma combinação teve o mesmo tempo de detecção. Nessas situações decide-se pelo menor número de falso-positivos. Entretanto, não há como desempatar casos de combinações com o mesmo tempo de detecção e número de falso-positivos, deixando-as na mesma colocação.

Por questão de praticidade, deste em ponto em diante adotar-se-á abreviação do nome das técnicas: Regressão Linear (LR), Média Móvel (MA), Mediana Móvel (MM), Hodrick Prescott (HP), Shewhart (SH), Soma Cumulativa (CS), e Média Móvel Exponencialmente Ponderada (EW).

Para melhor organização deste documento, a maioria dos gráficos de divergência e tabelas contendo *rankings* estão no Apêndice. Os gráficos de divergência contêm a disposição gráfica dos valores de divergência calculados de cada combinação técnica/indicador e cenário de carga de trabalho. A análise de falso-negativos e falso-positivos é, principalmente, fundamentada na observação desses gráficos, onde as linhas pontilhadas em vermelho representam o limite, α = 0,5, estipulado como limiar para detecção de anomalias. Além disso, gráficos contendo as 10 séries temporais por indicador, obtidas das 10 replicações, e o respectivo DTW estimado para elas, determinando a série temporal média usada no cálculo dos valores de divergência, são apresentados, por cenário de carga de trabalho e versão, para cada aplicação. Também por questão organizacional a maioria desses gráficos está no Apêndice A2.

5.2.1 Inkscape

O comportamento das séries temporais obtidas das replicações experimentais mostrou que houve considerável vazamento de memória tanto para RSS quanto para HUS, com incrementos próximos a 3,5 GB. É importante mencionar que na versão base, para ambos os indicadores, houve incremento de memória também, entretanto, esse incremento é muito menor do que o visto na versão alvo e, além disso, os últimos 30 minutos utilizados em cada replicação, onde não há carga de trabalho, mostrou que para versão base a memória decrementou, denotando fortes evidências que o incremento observado para essa versão (base) foi consequência do comportamento natural da aplicação perante a carga de trabalho, diferente do que foi visto na versão alvo, onde não há decremento nos últimos 30 minutos experimentais. A Figura 5.1 mostra o gráfico das séries temporais e o DTW calculado para o cenário de carga constante e alta na versão alvo. Os demais gráficos estão no Apêndice A2 representados pelas Figuras 1.A2, 2.A2, 3.A2, 4.A2, 5.A2, 6.A2 e 7A2.

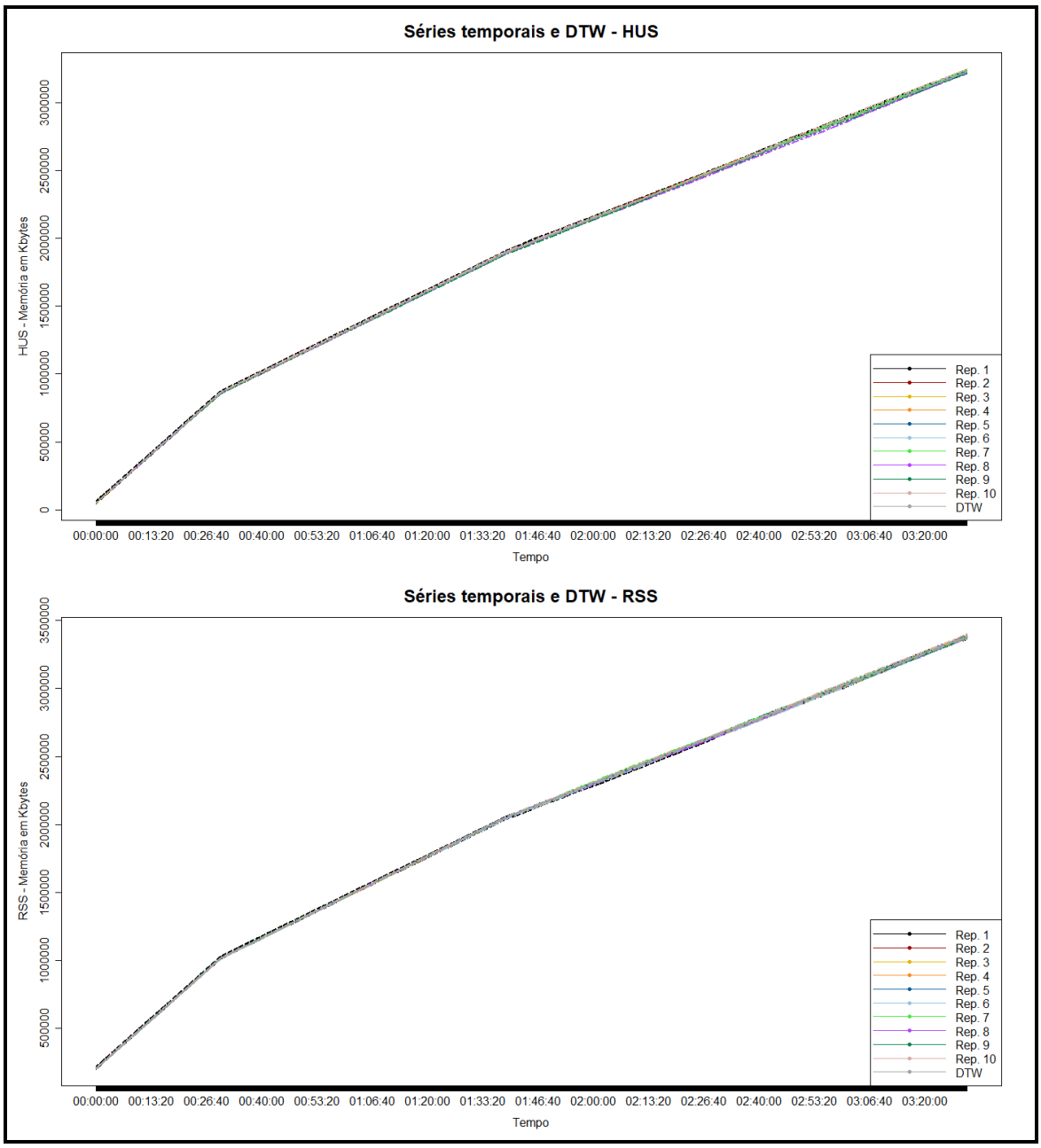


Figura 5.1. Inkscape – cenário de carga constante e alta, versão alvo – séries temporais e DTW.

O cálculo dos valores de divergência para o Inkscape mostrou que, para todos os cenários de carga de trabalho e indicadores, as técnicas estatísticas estudadas foram eficientes em detectar o vazamento de memória, não gerando nenhum falso-negativo. A Figura 5.2 mostra o gráfico de divergência para o cenário de carga de trabalho constante e alta do Inkscape. Os demais gráficos de divergência podem ser encontrados no Apêndice A1 pelas Figuras 1.A1, 2.A1 e 3.A1.

Em todos os gráficos de divergência, as técnicas de detecção de tendência (MM; MA; HP e LR) se mantiveram sempre acima do limite $\alpha = 0,5$, caracterizando apenas um evento de divergência logo no início. Em contrapartida, as técnicas de CEP começaram sempre abaixo do limite, o ultrapassaram em determinado ponto do tempo e depois se mantiveram acima, caracterizando também um único evento de divergência, entretanto, não no início. Dessa forma, como para todas as técnicas apenas um evento de divergência foi identificado, então nenhum falso-positivo foi

gerado (*DivNumEvents* = 0). A Tabela 5.1 apresenta uma visão geral dos tempos de detecção das combinações técnica/indicador para carga constante e alta. As Tabelas 1.A3, 2.A3 e 3.A3, presentes no Apêndice A3, mostram essa visão geral para os demais cenários de carga de trabalho.

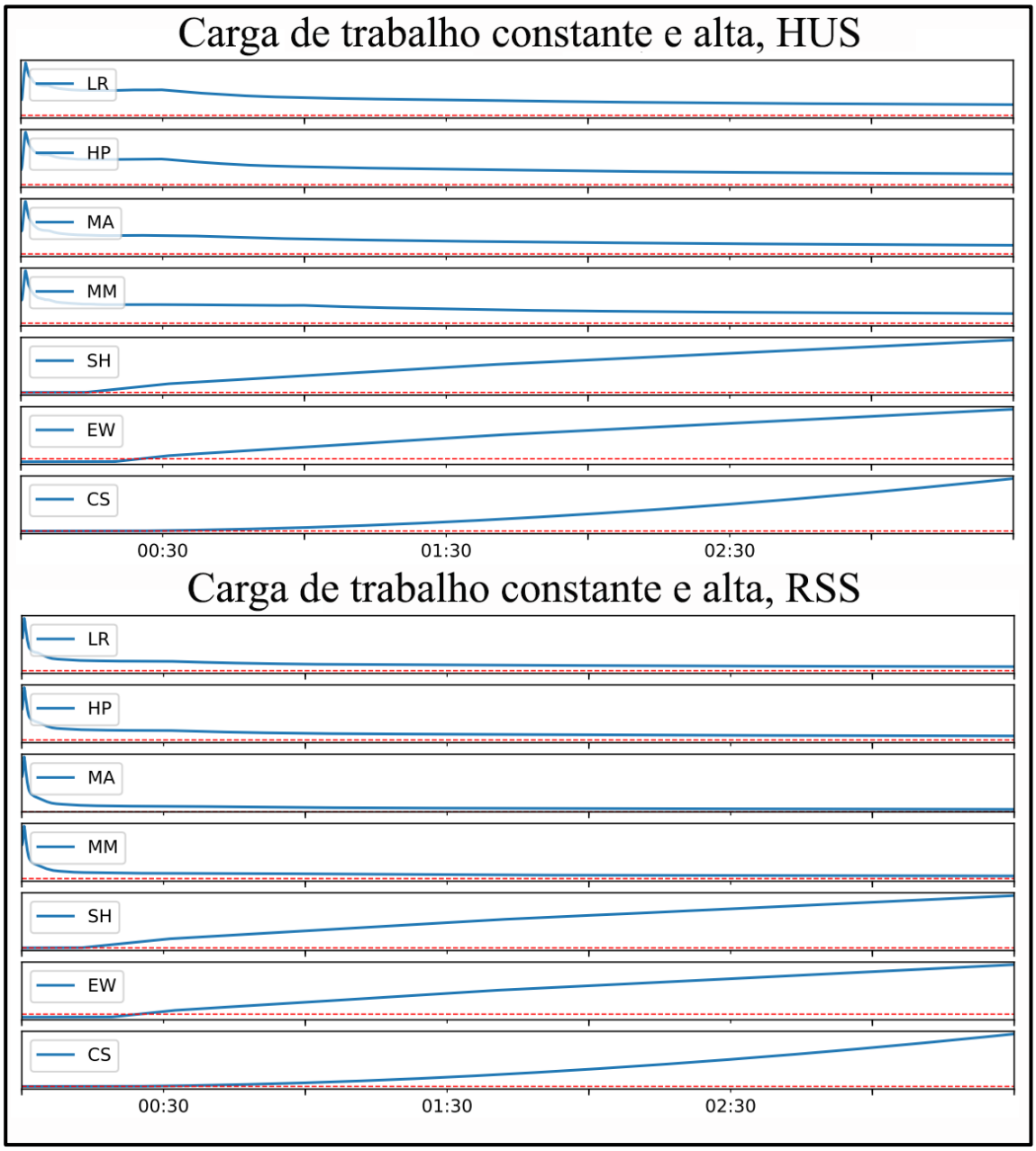


Figura 5.2. Inkscape: gráficos de divergência para o cenário de carga de trabalho constante e alta.

No cenário de carga constante e alta, para o indicador RSS, as técnicas de detecção de tendências mostraram os melhores resultados, detectando o vazamento em menos tempo que as técnicas CEP. Para HUS, o *ranking* segue a mesma ordem que a do RSS, com as técnicas de detecção de tendências em primeiro. A mesma posição de *ranking* vista nesse cenário, tanto para RSS quanto para HUS, se repete para os demais cenários. Assim, as técnicas de detecção de tendência foram sempre melhores, destacando-se entre elas MA, MM e HP. Com relação às técnicas de CEP, SH foi sempre melhor que CS e EW.

A avaliação do melhor indicador mostrou que, independente do cenário de carga de trabalho, RSS e HUS obtiveram o mesmo desempenho, ficando empatados nas primeiras e segundas posições (posições ocupadas pelas técnicas de detecção de tendências). Em contrapartida, o RSS apresentou os menores tempos (diferença mínima para HUS) nas terceiras posições de 3 de 4 cenários – as terceiras posições foram sempre preenchidas pelas técnicas CEP.

Portanto, para o Inkscape, a melhor combinação técnica/indicador foi composta pelas técnicas de detecção de tendências, em especial MA; HP e MM, com ambos os indicadores, RSS e HUS.

Tabela 5.1. Inkscape: tempos de detecção para carga constante e alta.

Indicador: RSS						**Indicador: HUS**				
Ranking		First Time	Last Time	Num Events		Ranking		First Time	Last Time	Num Events
1º	MA; HP; MM	00:00:25	00:00:25	0		1º	MA; HP; MM	00:00:25	00:00:25	0
2º	LR	00:00:30	00:00:30	0		2º	LR	00:00:30	00:00:30	0
3º	SH	00:12:45	00:12:45	0		3º	SH	00:13:55	00:13:55	0
4º	CS	00:20:25	00:20:25	0		4º	CS	00:21:15	00:21:15	0
5º	EW	00:24:50	00:24:50	0		5º	EW	00:24:50	00:25:25	0

5.2.2 Lighttpd

Para o Lighttpd, as séries temporais obtidas das replicações evidenciou vazamento de memória em ambos os indicadores onde, diferente do Inkscape, o vazamento foi bem menor, alcançando valores máximos de 14 MB (para o RSS) e 1 MB (para o HUS). Além disso, as séries temporais da versão base não apresentaram nenhum incremento, mantendo-se constante durante todas as replicações, onde o RSS permaneceu estável em 4,5 MB e o HUS em 300 KB. A Figura 5.3 mostra os gráficos das séries temporais para o cenário de carga constante e alta, além do respectivo DTW calculado, na versão alvo. Os demais gráficos estão no Apêndice A2 pelas Figuras 8.A2, 9.A2, 11.A2, 11.A2, 12.A2, 13.A2 e 14.A2.

O cálculo dos valores de divergência para o Lighttpd mostrou que, para todos os cenários de carga de trabalho e indicadores, as técnicas estatísticas avaliadas foram eficientes em detectar o vazamento de memória, não gerando nenhum falso-negativo. A Figura 5.4 mostra o gráfico de divergência para o cenário de carga de trabalho constante e alta do Lighttpd. Os demais gráficos podem ser encontrados no Apêndice A1 nas Figuras 4.A1, 5.A1 e 6.A1.

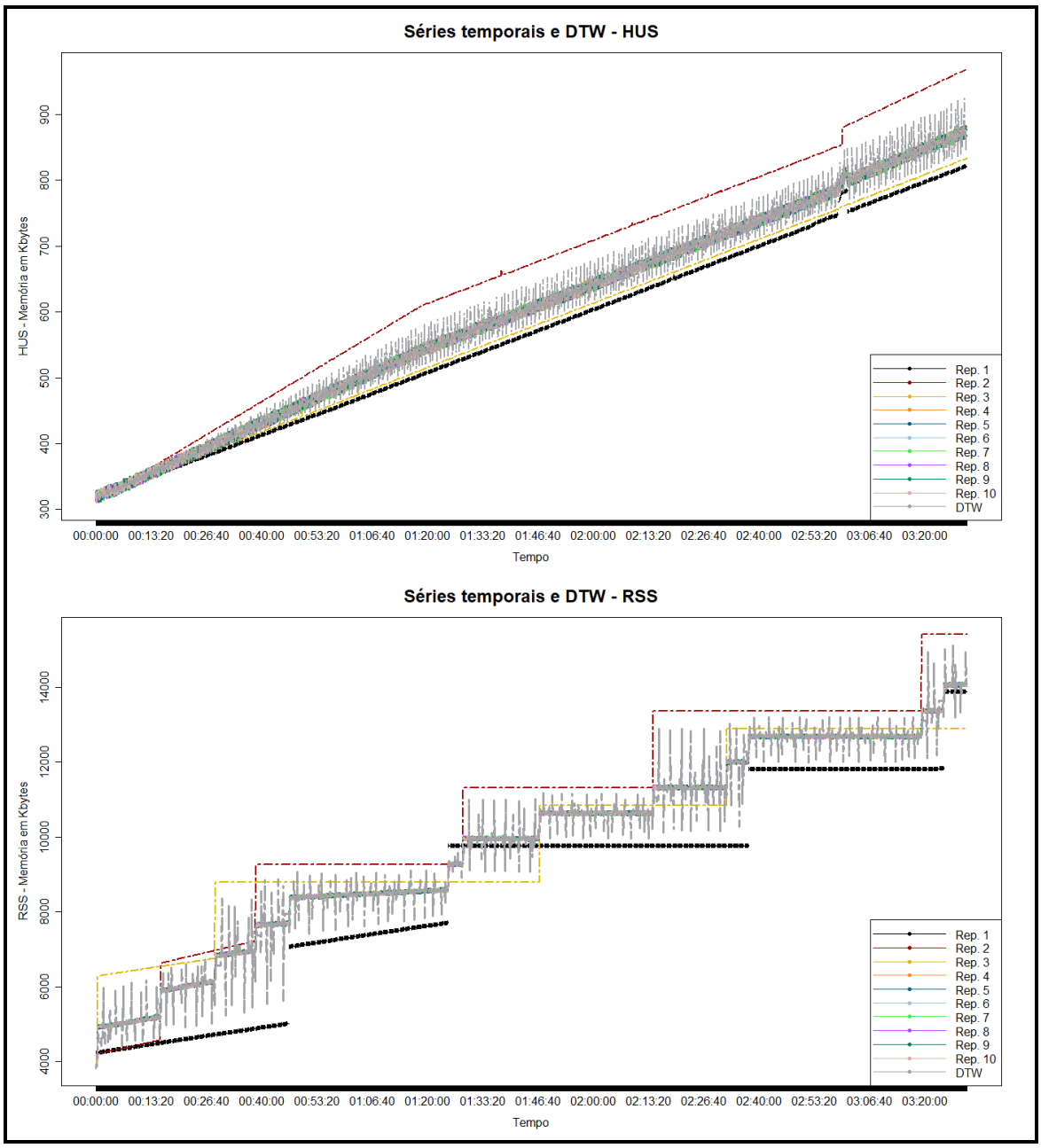


Figura 5.3. Lighttpd: cenário de carga constante e alta, versão alvo – séries temporais e DTW.

Como pode ser constatado, algumas combinações técnica/indicador se mantiveram sempre acima do limite α = 0,5, caracterizando apenas um evento de divergência logo no início. Em contrapartida, outras combinações começaram sempre abaixo do limite, o ultrapassaram logo no início e mantiveram-se sempre acima, caracterizando também um único evento de divergência. Assim, como para todas as técnicas apenas um evento de divergência foi identificado, então nenhum falso-positivo foi gerado (*DivNumEvents* = 0). A Tabela 5.2 apresenta uma visão geral dos tempos de detecção das combinações técnica/indicador para o cenário de carga constante e alta. No Apêndice A3, através das Tabelas 4.A3, 5.A3 e 6.A3, encontra-se essa visão geral para os demais cenários.

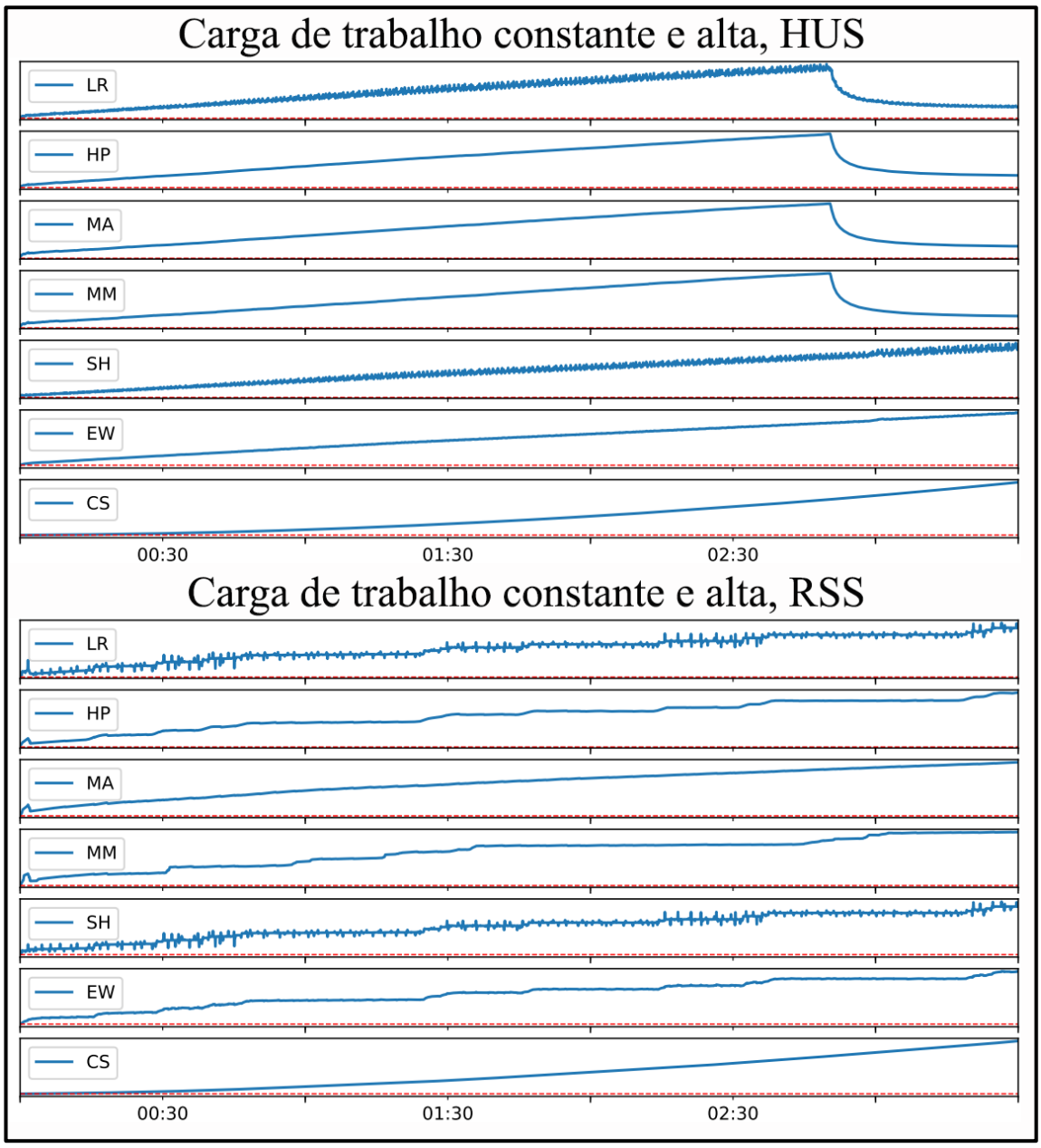


Figura 5.4. Lighttpd: gráficos de divergência para o cenário de carga de trabalho constante e alta.

No cenário de carga constante e alta, para o indicador RSS, as técnicas de CEP mostraram os melhores resultados, detectando o vazamento em pouquíssimo tempo, seguidas de perto pelas técnicas detecção de tendências. Para HUS, o *ranking* segue a mesma ordem e tempos que o do RSS, com as técnicas de CEP liderando. Essa mesma posição de *ranking* para RSS e HUS se repete, com pequenas alternâncias, para todos os outros cenários. Assim, as técnicas de CEP foram as melhores, especialmente SH e CS (EW nunca esteve em primeiro, mas figurou constantemente na segunda colocação).

Além disso, como pode ser visto em todas as tabelas, as diferenças entre a primeira e as demais posições são muito pequenas. Isso que dizer que, para o Lighttpd, todas as técnicas detectaram o vazamento de memória muito rapidamente e que, embora as de CEP tenham sido melhores, as de detecção de tendências também foram muito bem.

A avaliação do melhor indicador mostrou que, independente do cenário de carga de trabalho, RSS e HUS obtiveram o mesmo desempenho, ficando empatados nas primeiras, segundas e terceiras posições. Portanto, podemos concluir que para o Lighttpd não há um indicador que tenha se destacado mais que o outro.

Portanto, a melhor combinação técnica/indicador, para o Lighttpd, é composta pelas técnicas de CEP, especialmente CS e SH, com ambos os indicadores, RSS e HUS.

Tabela 5.2. Lighttpd: tempos de detecção para carga constante e alta.

Indicador: RSS						**Indicador: HUS**				
Ranking		First Time	Last Time	Num Events		Ranking		First Time	Last Time	Num Events
1º	CS; SH	00:00:20	00:00:20	0		1º	CS; SH	00:00:20	00:00:20	0
2º	MA; EW; HP; MM	00:00:25	00:00:25	0		2º	MA; EW; HP; MM	00:00:25	00:00:25	0
3º	LR	00:00:30	00:00:30	0		3º	LR	00:00:30	00:00:30	0

5.2.3 PostgreSQL

Nesta aplicação, as séries derivadas das replicações mostrou também vazemento de memória, com incrementos alcançando valores máximos de 2,3 GB (para RSS) e 1,1 GB (para HUS). Na versão base, para o HUS, a memória se manteve estável durante todo experimento. Em contrapartida, para o RSS, houve um pequeno incremento de memória, memória essa que não diminuiu nos 30 minutos finais sem carga de trabalho, ou seja, manteve-se estável depois que a carga parou. Acredita-se que esse comportamento se deve ao fato da presença ruídos no RSS. A Figura 5.5 mostra o gráfico das séries temporais da versão alvo para o cenário de carga constante e alta e o respectivo DTW calculado. Os demais gráficos de séries podem ser encontrados no Apêndice A2 através das Figuras 15.A2, 16.A2 ,17.A2, 18.A2, 19.A2, 20.A2 e 21.A2.

O cálculo dos valores de divergência para o PostgreSQL mostrou que, para todos os cenários de carga de trabalho e indicadores, as técnicas estatísticas avaliadas foram eficientes em detectar o vazamento de memória, não gerando taxas de falso-negativos. A Figura 5.6 mostra o gráfico de divergência para o cenário de carga de trabalho constante e alta do PostgreSQL. Os demais gráficos de divergência podem ser encontrados no Apêndice A1 nas Figuras 7.A1, 8.A1 e 9.A1.

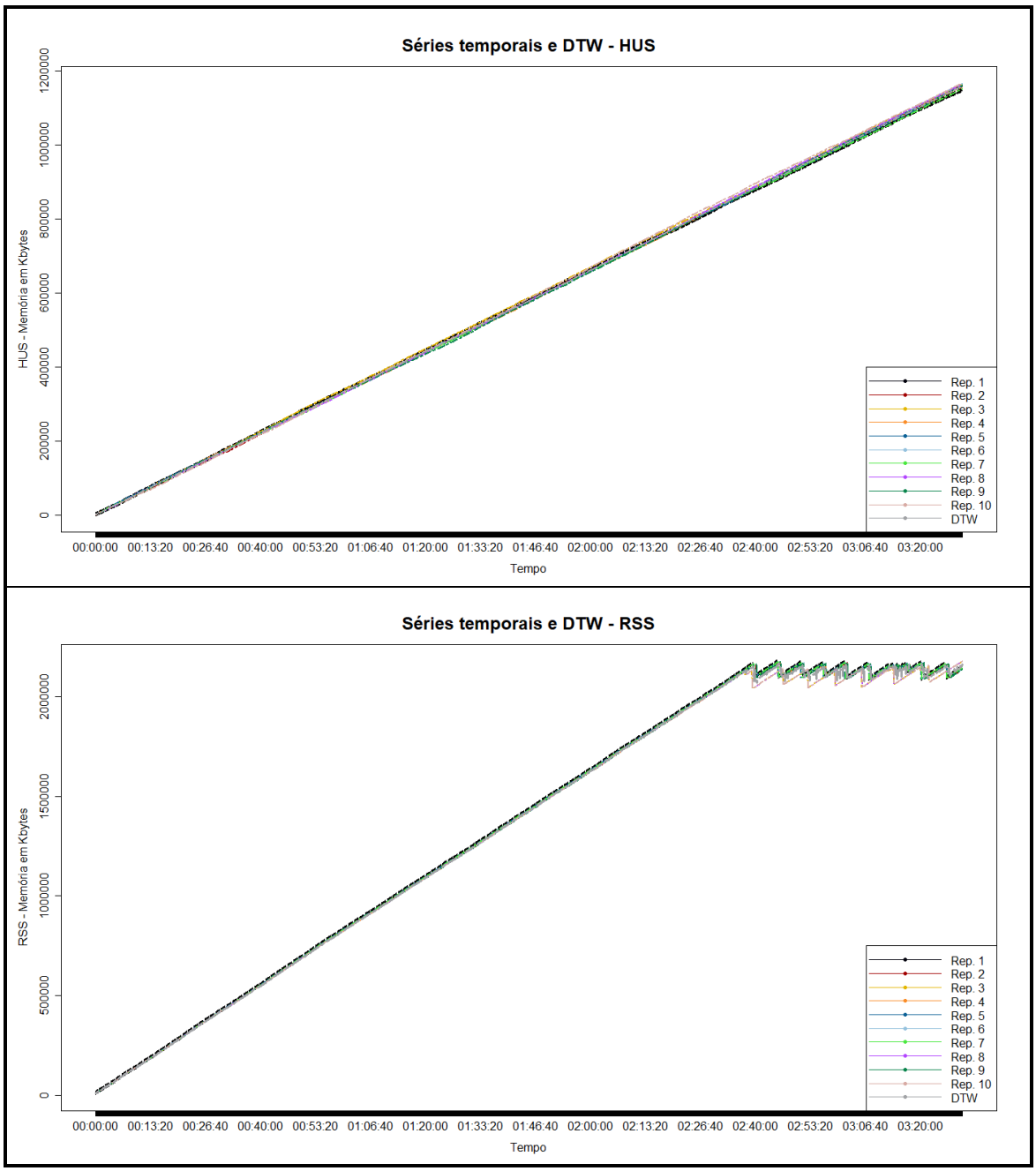


Figura 5.5. PostgreSQL: cenário de carga constante e alta, versão alvo – séries temporais e DTW.

Como pode ser observado, para todos os cenários de cargas de trabalho, as combinações das técnicas de detecção de tendências com o indicador RSS se mantiveram sempre acima do limite $\alpha = 0{,}5$, caracterizando apenas um evento de divergência logo no início. Em contrapartida, outras combinações, como as técnicas de CEP com ambos os indicadores e as técnicas de detecção de tendências com HUS, iniciaram abaixo do limite, ultrapassaram-no em algum momento do tempo e após isso se mantiveram sempre acima, caracterizando apenas um evento de divergência, mas nem sempre no início.

Portanto, todas as combinações, com uma exceção, não geraram taxas de falso-positivos. A exceção aconteceu para o cenário de carga constante e baixa, na combinação MA/HUS. Nesse caso, a série de valores de divergência inicia-se acima do limite, entretanto, logo no início fica por um breve momento abaixo, voltando a

ficar acima logo em seguida e se mantendo até o fim. Isso fez com que mais de um evento de divergência fosse gerado (*DivNumEvents* > 0), caracterizando um falso-positivo. Tal atividade, devido a sua sensibilidade, é dificilmente visualizada a olho nu no gráfico de divergência e, portanto, só foi identificada analisando-se a série dos valores de divergência. A Tabela 5.3 apresenta uma visão geral dos tempos de detecção das combinações técnica/indicador para carga constante e alta – os demais cenários podem ser vistos nas Tabelas 7.A3, 8.A3 e 9.A3 presentes no Apêndice A3.

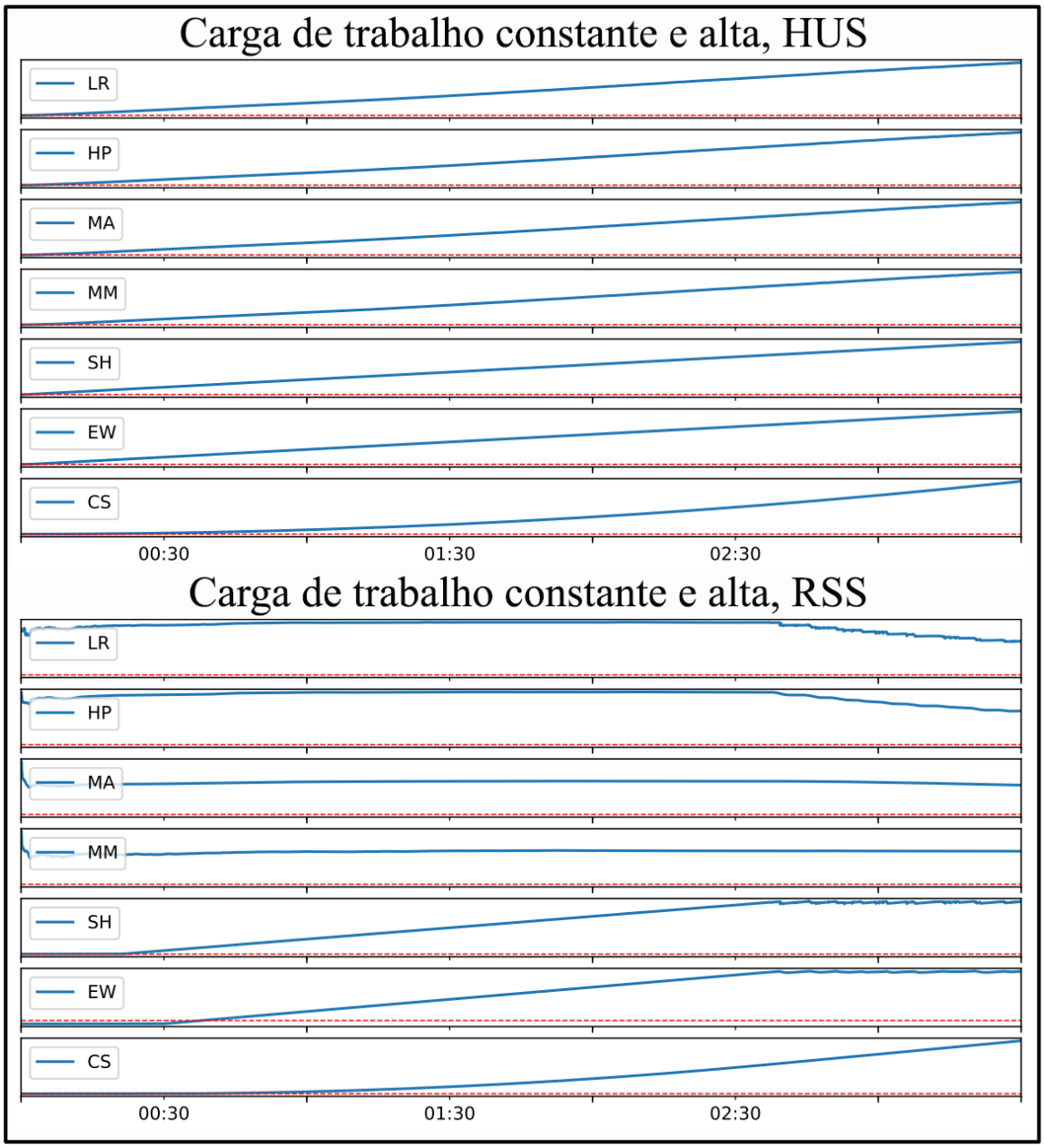


Figura 5.6. PostgreSQL: gráficos de divergência para o cenário de carga de trabalho constante e alta.

No cenário de carga constante e alta, as técnicas de detecção de tendências mostraram os melhores resultados, onde em conjunto com o RSS detectaram o vazamento em um tempo muito menor que as técnicas de CEP. Para HUS, todas as técnicas detectaram o vazamento em curtíssimo tempo, onde a diferença entre as técnicas de detecção de tendências (também melhores) e CEP foi bem menor se

comparado ao RSS. Para os demais cenários, com algumas alternâncias, o panorama se repetiu. Assim, as técnicas de detecção de tendências foram melhores que as de CEP em ambos os indicadores, com destaque para HP, aparecendo na primeira posição em todos os cenários de cargas de trabalho e indicadores (a exceção foi para carga constante e alta, indicador HUS).

A avaliação do melhor indicador mostrou que, para as primeiras e segundas posições, de todos os cenários, RSS sempre foi melhor. Entretanto, é preciso mencionar que para o restante das posições, o indicador HUS se portou muito melhor. Isso aconteceu pelo fato de que as técnicas de CEP combinadas com RSS, posicionadas da terceira posição em diante, tiveram um tempo de detecção muito maior do que a combinação CEP/HUS (também posicionadas de terceiro em diante). Além disso, a diferença entre HUS e RSS para as primeiras e segundas posições não é tão expressiva. Contudo, seguindo o critério de avaliação do melhor indicador, analisando-se pelas primeiras posições, o indicador RSS se apresentou melhor que o HUS.

Portanto, a melhor combinação técnica/indicador, para o PostgreSQL, é composta pelas técnicas de detecção de tendências, em especial HP, com o indicador RSS.

Tabela 5.3. PostgreSQL: tempos de detecção para carga constante e alta.

Indicador: RSS						**Indicador: HUS**				
Ranking		First Time	Last Time	Num Events		Ranking		First Time	Last Time	Num Events
1º	MA; HP; MM	00:00:25	00:00:25	0		1º	LR	00:01:05	00:01:05	0
2º	LR	00:00:30	00:00:30	0		2º	HP	00:01:10	00:01:10	0
3º	SH	00:21:25	00:21:25	0		3º	SH	00:01:15	00:01:15	0
4º	CS	00:31:25	00:31:25	0		4º	CS; EW	00:01:35	00:01:35	0
5º	EW	00:38:30	00:38:30	0		5º	MA; MM	00:01:45	00:01:45	0

5.2.4 Squid

O comportamento das séries temporais obtidas das replicações experimentais mostrou vazamento de memória nos dois indicadores, com incrementos alcançando valores máximos de 1,6 GB (para RSS) e 198 MB (para HUS). Na versão base, para o HUS, a memória se manteve constante. Em contrapartida, para o RSS, assim como no PostgreSQL, a memória apresentou incrementos, não diminuindo no período sem carga. Mais uma vez acredita-se que a presença de ruídos no RSS contribuiu para essa perspectiva. A Figura 5.6 mostra o gráfico das séries e o DTW para carga constante e alta na versão alvo. Os demais podem ser vistos nas Figuras 22.A2, 23.A2 , 24.A2, 25.A2, 26.A2, 27.A2 e 28.A2, presentes no Apêndice A2.

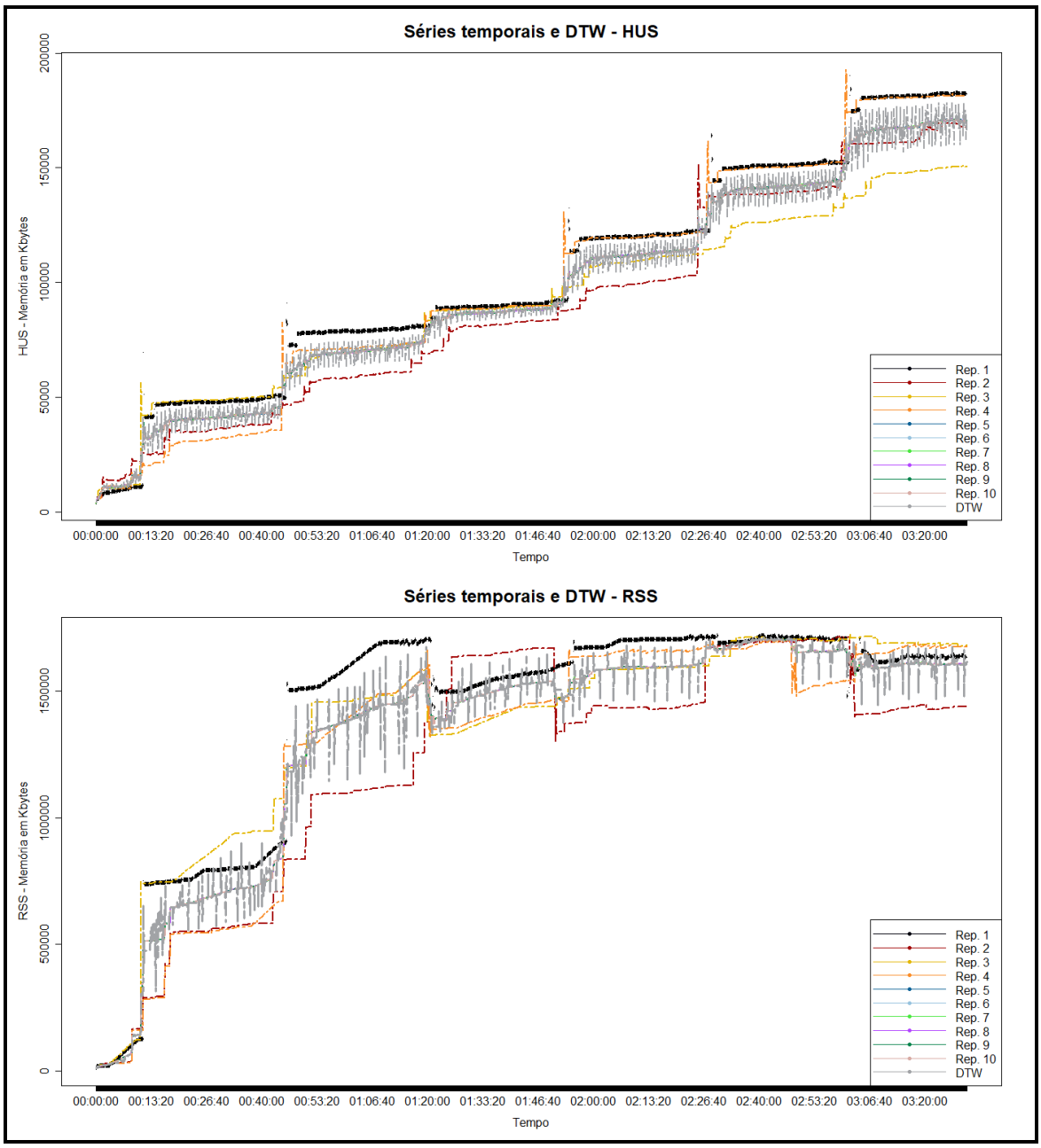


Figura 5.7. Squid: cenário de carga constante e alta, versão alvo – séries temporais e DTW.

O cálculo dos valores de divergência para o Squid mostrou que, para todos os cenários de carga de trabalho e indicadores, as técnicas estatísticas avaliadas foram eficientes em detectar o vazamento de memória, não apresentando falso-negativos. A Figura 5.8 mostra o gráfico de divergência para o cenário de carga de trabalho constante e alta do Squid. Os demais gráficos de divergência podem ser encontrados no Apêndice A1 nas Figuras 10.A1, 11.A1 e 12.A1.

De acordo com os gráficos, para todos os cenários de carga de trabalho, diferentemente das outras aplicações, a maioria das combinações com o indicador RSS apresentaram falso-positivos (*DivNumEvents* > 0), alguns não perceptíveis a olho nu, sendo preciso investigar as séries temporais dos valores de divergência. Com HUS também houveram falso-positivos, mas em um número muito menor. A Tabela 5.4 mostra uma visão geral dos tempos de detecção das combinações

técnica/indicador para carga constante e alta – os outros cenários podem ser vistos no Apêndice A3 através das Tabelas 10.A3, 11.A3 e 12.A3.

Embora tenham surgido várias combinações com mais de um evento de divergência, nenhuma delas empataram em termos de tempo de detecção (*DivLastTime*). As combinações que empataram em termos de *DivLastTime* foram àquelas que não possuem eventos de divergência.

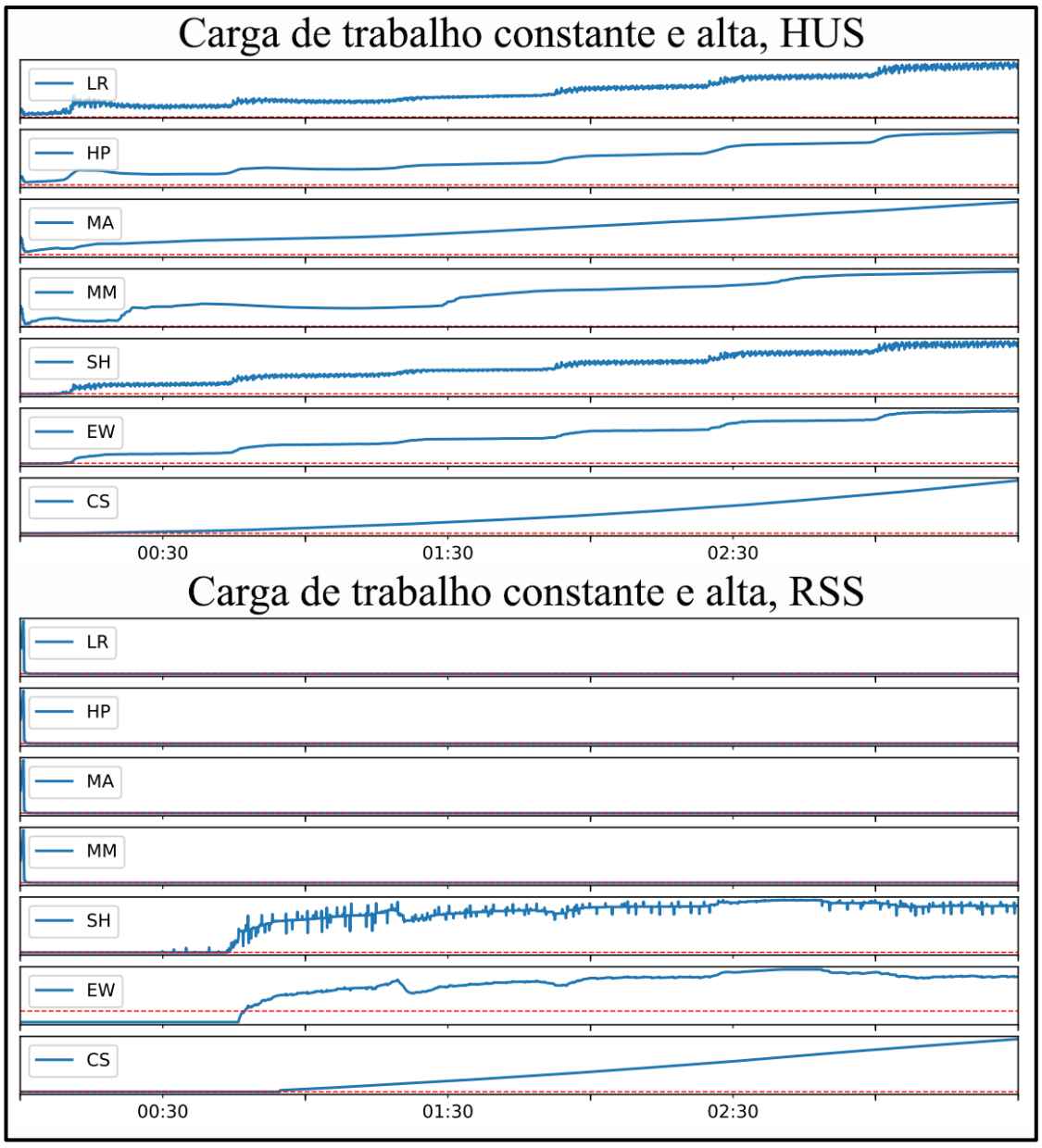


Figura 5.8. Squid: gráficos de divergência para o cenário de carga de trabalho constante e alta.

No cenário de carga constante e alta, para o indicador RSS, as técnicas de CEP apareceram na primeira posição, devido ao desempenho de SH, detectando o vazamento em menor tempo que as técnicas de detecção de tendências. Nesse caso, as técnicas de detecção de tendências apresentaram vários falso-positivos. Para HUS, todas as técnicas detectaram o vazamento em um tempo muito menor se comparado com RSS. Entretanto, as melhores técnicas foram as de detecção de tendências, com tempos de detecção consideravelmente melhores que as de CEP.

Ainda, para HUS, não houve nenhum falso-positivo detectado pelas técnicas estatísticas.

Para carga constante e normal, indicador RSS, SH manteve o bom desempenho, mas não ficou na liderança, pois MA conseguiu apresentar um tempo de detecção melhor. Nesse caso, apenas MA e CS não apresentaram falso-positivos. Para HUS, o desempenho das técnicas foi o mesmo visto no cenário constante e alto, entretanto, com falso-positivos para LR e SH.

Para carga constante e baixa, indicador RSS, as técnicas de detecção de tendências MA e MM obtiveram ótimo desempenho, ficando na liderança e seguidas de longe pelas técnicas de CEP. Para esse indicador e cenário, com exceção de MA, MM e CS, todas as outras técnicas apresentaram falsos-positivos que prejudicaram a apresentação de um melhor tempo de detecção. Para HUS, o mesmo panorama dos outros cenários foi visto. Além disso, as técnicas MA, MM e LR poderiam ter tido um desempenho melhor com HUS não fosse pelos falso-positivos.

Para carga variada, indicador RSS, os resultados do cenário constante a baixo praticamente se repetiram, onde todas as técnicas, com exceção das de CEP e de MA, apresentaram falsos-positivos que prejudicaram a apresentação de um menor tempo de detecção. Justamente por MA não ter apresentado falso-positivos é que essa técnica liderou novamente. Para HUS, outra vez as de detecção de tendência ficaram bem a frente e houveram falso-positivos apenas para SH.

Portanto, no geral as técnicas de detecção de tendências se sobressaíram, principalmente quando combinadas com HUS, com destaque para HP. É importante mencionar que a presença de muitos ruídos nas séries temporais médias do indicador RSS, obtidas da versão base, prejudicou, quase sempre, no desempenho das técnicas de detecção de tendências, especialmente LR e MM.

A avaliação do melhor indicador mostrou que, no geral, os tempos de detecção de HUS foram bem menores que os do RSS. É evidente que, com exceção do cenário de carga de trabalho constante e alta, não há diferença de tempos entre as primeiras posições de ambos os indicadores. Entretanto, para as demais posições, essa diferença se torna muito considerável, denotando um melhor desempenho do HUS.

Assim, a melhor combinação técnica/indicador, para o Squid, é composta pelas técnicas de detecção de tendências, em especial HP, com o indicador HUS.

Tabela 5.4. Squid: tempos de detecção para carga constante e alta.

Indicador: RSS						**Indicador: HUS**				
Ranking		First Time	Last Time	Num Events		Ranking		First Time	Last Time	Num Events
1º	SH	00:43:45	00:43:45	0		1º	MA; HP; MM	00:00:25	00:00:25	0
2º	EW	00:47:35	00:47:35	0		2º	LR	00:00:30	00:00:30	0
3º	CS	00:55:00	00:55:00	0		3º	SH	00:08:20	00:08:20	0
4º	HP	00:00:25	01:25:25	2		4º	CS	00:09:05	00:09:05	0
5º	MM	00:00:25	01:57:35	1		5º	EW	00:09:20	00:09:20	0
6ª	MA	00:00:25	02:35:40	1						
7ª	LR	00:00:30	03:08:50	12						

5.3 Conclusão

5.3.1 Síntese dos Resultados

O objetivo desta Seção é apresentar uma visão geral dos resultados obtidos nas 4 aplicações, buscando responder as perguntas de pesquisa. As Tabelas 5.5 e 5.6 apresentam a estatística descritiva, separamente por cenário de carga de trabalho, indicador e versão (alvo e base), das séries temporais das replicações utilizadas para a aplicação do DTW. O objetivo de apresentar essas informações está em fornecer um resumo simples das séries temporais utilizadas e, além disso, ajudar a entender os gráficos das séries de replicações apresentados na Seção 5.2 e no Apêndice A.2.

Tabela 5.5. Inkscape e Lighttpd: Estatística descritiva das séries temporais das replicações.

	Inkscape				**Lighttpd**			
	RSS	***HUS***	***RSS***	***HUS***	***RSS***	***HUS***	***RSS***	***HUS***
	Cenário: Constante e Alto				Cenário: Constante e Alto			
	Versão Alvo		Versão Base		Versão Alvo		Versão Base	
Média	2.121.605	1.968.203	536.620	408.044	10.008	606	3.878	295
Desvio P.	853.275	857.598	220.247	215.174	2.521	158	0,25	2,49
Mínimo	204.397	50.446	154.211	37.055	3.816	310	3.819	287
Máximo	3.397.699	3.241.127	936.047	799.160	15.408	968	3.934	296
	Cenário: Constante e Normal				*Cenário: Constante e Normal*			
	Versão Alvo		*Versão Base*		*Versão Alvo*		*Versão Base*	
Média	1.336.379	1.183.722	384.257	228.954	9.354	461	3.813	295
Desvio P.	655.258	655.240	131.131	123.591	2.149	84,1	0,24	2,49
Mínimo	199.422	48.606	155.645	12.923	3.790	308	3.763	287
Máximo	2.475.796	2.322.336	620.853	447.510	11.680	616	3.881	296
	Cenário: Constante e Baixo				*Cenário: Constante e Baixo*			
	Versão Alvo		*Versão Base*		*Versão Alvo*		*Versão Base*	
Média	695.716	523.072	287.750	140.686	5.533	403	3.773	295
Desvio P.	272.941	273.127	76.387	69.228	374	51,1	0,25	2,49
Mínimo	221.716	50.503	154.441	16.251	3.764	302	3.730	287
Máximo	1.171.846	998.135	425.398	265.091	5.883	506	3.821	296
	Cenário: Variado				*Cenário: Variado*			
	Versão Alvo		*Versão Base*		*Versão Alvo*		*Versão Base*	
Média	1.588.541	1.439.316	481.633	339.453	5.599	418	3.837	293
Desvio P.	800.8336	800.929	184.892	184.407	1.207	54,7	0,24	2,47
Mínimo	201.682	51.403	155.247	19.430	3.795	309	3.754	286
Máximo	2.984.416	2.831.497	817.666	676.550	9.824	521	4.220	294

Tabela 5.6. PostgreSQL e Squid: Estatística descritiva das séries temporais das replicações.

	PostgreSQL				**Squid**			
	RSS	***HUS***	***RSS***	***HUS***	***RSS***	***HUS***	***RSS***	***HUS***
	Cenário: Constante e Alto				Cenário: Constante e Alto			
	Versão Alvo		Versão Base		Versão Alvo		Versão Base	
Média	1.424.082	576.939	292.904	6.270	1.527.501	88.266	879.051	11.352
Desvio P.	702.232	335.007	162.388	257	450.669	45.841	229.884	1.714
Mínimo	7.320	1.155	8.096	1.261	10.375	3.947	6.748	2.442
Máximo	2.176.913	1.166.658	581.272	7.802	1.723.616	192.808	1.057.292	15.569
	Cenário: Constante e Normal				*Cenário: Constante e Normal*			
	Versão Alvo		*Versão Base*		*Versão Alvo*		*Versão Base*	
Média	753.696	288.958	163.902	3.520	1.484.755	76.007	973.643	9.586
Desvio P.	432.237	167.357	895.482	204	537.205	38.233	248.765	1.982
Mínimo	6.547	1.261	6.698	1.261	10.636	3.959	10.376	2.223
Máximo	1.519.076	583.919	322.424	4.742	1.727.976	163.691	1.058.124	15.301
	Cenário: Constante e Baixo				*Cenário: Constante e Baixo*			
	Versão Alvo		*Versão Base*		*Versão Alvo*		*Versão Base*	
Média	370.968	146.888	105.729	2.456	1.244.655	62.370	961.540	12.050
Desvio P.	210.154	84.302	55.279	190	505.943	27.388	273.831	2.127
Mínimo	6.988	1.202	9.738	1.236	10.486	3.852	8.085	2.412
Máximo	747.784	296.231	204.732	3.156	1.715.232	132.950	1.056.432	16.217
	Cenário: Variado				*Cenário: Variado*			
	Versão Alvo		*Versão Base*		*Versão Alvo*		*Versão Base*	
Média	589.704	242.951	143.549	4.137	1.499.480	70.110	901.629	12.332
Desvio P.	334.538	140.048	77.625	204	516.056	40.436	290.985	2.218
Mínimo	7.556	1.141	9.637	136	10.605	4.234	8.324	2.543
Máximo	1.163.540	493.641	282.780	5.226	1.725.244	161.747	1.058.468	18.994

Como observado anteriormente, em termos de eficiência na detecção, todas as técnicas analisadas não geraram falso-negativos. Em termos de falso-positivos, as combinações técnica/indicador foram precisas na detecção do vazamento de memória em 3 de 4 aplicações. Apenas no Squid houve vários falso-positivos (*DivNumEvents* > 0), fundamentalmente para RSS. Nas demais aplicações, com exceção do cenário constante e baixo do PostgreSQL, combinação MA/HUS, nenhum falso-positivo foi observado. Essa presença de poucas taxas de falso-positivos (em 3 de 4 aplicações) e nenhuma de falso-negativos pode estar diretamente relacionada ao modo como as cargas de trabalho foram planejadas. Isso porque tais cargas consideraram apenas fatores relativos aos padrões de ativação dos *bugs* encontrados, ou seja, sempre que a carga era ativada o *bug* se manifestava, o que contribuiu para atenuar o vazamento de memória.

Já para o Squid, uma justificativa para a ocorrência de tantos falso-positivos pode ser devido a inúmeros ruídos nas séries temporais médias, em especial do indicador RSS, para a versão base. Assim, a presença desses ruídos faz com que essas séries se assemelhem com as séries temporais médias da versão alvo, tornando mais difícil a detecção do vazamento de memória. É por isso que espera-se do HUS sempre um desempenho melhor que o RSS, por não considerar outras partes do processo que o RSS considera, partes essas que podem gerar ruídos.

Ainda é preciso mencionar que a maioria desses falso-positivos do Squid surgiu em combinações com as técnicas de detecção de tendências LR e MM. Se existem muitos ruídos nas séries temporais médias dos indicadores usados, essas

técnicas de tendências podem não desempenhar-se bem. A Tabela 5.7 mostra uma visão geral do número de falso-positivos em cada aplicação.

Tabela 5.7. Quantidade de falso-positivos encontrados.

	Inkscape		**Lighttpd**		**PostgreSQL**		**Squid**	
	Quantidade de falsos alarmes (*DivNumEvents*)							
Cenário	RSS	HUS	RSS	HUS	RSS	HUS	RSS	HUS
Constante e alta	0	0	0	0	0	0	15	0
Constante e normal	0	0	0	0	0	0	21	4
Constante e baixa	0	0	0	0	0	1	11	7
Variada	0	0	0	0	0	0	32	2
Total	**0**	**0**	**0**	**0**	**0**	**1**	**79**	**13**

Com relação ao tipo de técnica estatística mais efetiva na detecção do vazamento de memória, foi possível identificar analisando os tempos de detecção que, com exceção do Lighttpd, em todas as outras aplicações as técnicas de detecção de tendências detectaram o vazamento de memória mais rápido do que as técnicas de CEP.

A Tabela 5.8 ajuda a evidenciar essa conclusão, onde uma visão geral das combinações técnica/indicador, separadas por aplicação, indicador e cenário de carga de trabalho, é apresentada. Em cada cenário, um *ranking* das três melhores técnicas para cada indicador é exibido. As células em vermelho referem-se a posições que tiveram apenas técnicas de detecção de tendências. As células em verde correspondem às técnicas de CEP. As células amareladas identificam os casos em que tiveram ambas as técnicas de detecção de tendência e CEP.

Pela Tabela 5.8 também é possível verificar que, para as situações em que as técnicas de CEP foram melhores, CS e SH foram as que se destacaram, especialmente CS que aparece mais vezes na primeira posição. Para as situações em que as técnicas de detecção de tendências foram melhores (maioria das vezes), HP foi a que se sobressaiu, estando presente mais vezes em primeiro, seguida de perto por MA e MM. A técnica LR poucas vezes liderou, mas marcou bastante presença nas segundas e terceiras colocações, dividindo muitas vezes espaço com as técnicas de CEP. É preciso enfatizar que, embora HP tenha tido os melhores resultados gerais, essa técnica parece não se sair tão bem diante de ruídos nas séries temporais da versão base (caso do Squid); LR e MM também são muito sensíveis a esses ruídos.

Tabela 5.8. *Ranking* das técnicas por indicador, aplicação e cenário de carga de trabalho.

	Inkscape		Lighttpd		PostgreSQL		Squid	
	RSS	*HUS*	*RSS*	*HUS*	*RSS*	*HUS*	*RSS*	*HUS*
Cenário: Constante e Alto								
1ª	MA; HP; MM	MA; HP; MM	CS; SH	CS; SH	MA; HP; MM	LR	SH	MA; HP; MM
	25s	25s	20s	20s	25s	1m05s	43m45s	25s
2ª	LR	LR	MA; EW; HP; MM	MA; EW; HP; MM	LR	HP	EW	LR
	30s	30s	25s	25s	30s	1m10s	47m35s	30s
3ª	SH	SH	LR	LR	SH	SH	CS	SH
	12m45s	13m55s	30s	30s	21m25s	1m15s	55m	8m20s
Cenário: Constante e Normal								
1ª	MA; HP; MM	MA; HP; MM	CS; SH	SH	MA; HP; MM	MA; LR; HP; MM	MA	MA; HP; MM
	25s	25s	20s	20s	25s	30s	25s	25s
2ª	LR	LR	EW	CS; MA; EW; HP; MM	LR	SH	SH	LR
	30s	30s	25s	25s	30s	1m10s	1h01m40s	1m
3ª	SH	SH	MA; LR; HP; MM	LR	SH	CS	CS	CS
	17m35s	17m05s	30s	30s	23m35s	1m20s	1h29m30s	4m10s
Cenário: Constante e Baixo								
1ª	MA; HP; MM	MA; HP; MM	CS; SH	SH	MA; HP; MM	LR; HP; MM	MA; MM	HP
	25s	25s	20s	20s	25s	30s	25s	25s
2ª	LR	LR	MA; EW; HP; MM	CS; MA; HP; MM	LR	SH	SH	MA
	30s	30s	25s	25s	30s	1m10s	1h25m30s	1m45s
3ª	SH	SH	LR	LR; EW	SH	MA	EW	MM
	15m50s	20m40s	30s	30s	28m25s	1m10s	1h36m35s	2m10s
Cenário: Variado								
1ª	MA; HP; MM	MA; HP; MM	SH	CS; SH	LR; HP	HP	MA	MA; HP; MM
	25s	25s	20s	20s	30s	25s	25s	25s
2ª	LR	LR	CS; HP	MA; EW; HP; MM	MA; MM	LR	SH	LR
	30s	30s	25s	25s	40s	30s	58m35s	30s
3ª	SH	SH	LR	LR	SH	MA	EW	EW
	21m40s	22m45s	30s	30s	25m35s	45s	1h08m25s	14m35s

Pela Tabela 5.9 é possível identificar que para as aplicações Inkscape e Lighttpd, RSS e HUS obtiveram o mesmo desempenho. A única exceção foi para as terceiras colocações do Inkscape, onde RSS se mostrou melhor em 3 de 4 cenários. Entretanto, é preciso mencionar que para esse caso (onde não houve empate), as diferenças entre os tempos de detecção de RSS e HUS são mínimas.

Tabela 5.9. Ranking das melhores combinações técnica/indicador por aplicação.

	Inkscape	Lighttpd	PostgreSQL	Squid
Cenário: Constante e Alto				
1ª	MA; HP; MM (RSS/HUS)	CS; SH (RSS/HUS)	MA; HP; MM (RSS)	MA; HP; MM (HUS)
	25s	20s	25s	25s
2ª	LR (RSS/HUS)	MA; EW; HP; MM (RSS/HUS)	LR (RSS)	LR (HUS)
	30s	25s	30s	30s
3ª	SH (RSS)	LR (RSS/HUS)	LR (HUS)	SH (HUS)
	12m45s	30s	1m05s	8m20s
Cenário: Constante e Normal				
1ª	MA; HP; MM (RSS/HUS)	SH (RSS/HUS) e CS (RSS)	MA; HP; MM (RSS)	MA (RSS/HUS) e HP; MM (HUS)
	25s	20s	25s	25s
2ª	LR (RSS/HUS)	EW (HUS/RSS) e CS; MA; HP; MM (HUS)	MA; LR; HP; MM (HUS) LR (RSS)	LR (HUS)
	30s	25s	30s	1m
3ª	SH (HUS)	LR (RSS/HUS) e MA; HP; MM (RSS)	SH (HUS)	CS (HUS)
	17m05s	30s	1m10s	4m10s
Cenário: Constante e Baixo				
1ª	MA; HP; MM (RSS/HUS)	CS (RSS) e SH (RSS/HUS)	MA; HP; MM (RSS)	MA; MM (RSS) HP (HUS)
	25s	20s	25s	25s
2ª	LR (RSS/HUS)	HP; MM; MA (RSS/HUS) EW (RSS) CS; (HUS)	LR (RSS/HUS) HP; MM (HUS)	MA (HUS)
	30s	25s	30s	1m45s
3ª	SH (RSS)	LR (RSS/HUS) EW (HUS)	SH (HUS)	MM (HUS)
	15m50s	30s	1m10s	2m10s
Cenário: Variado				
1ª	MA; HP; MM (RSS/HUS)	SH (RSS/HUS) e CS (HUS)	HP (HUS)	MA (RSS/HUS) HP; MM (HUS)
	25s	20s	25s	25s
2ª	LR (RSS/HUS)	HP (RSS/HUS); CS (RSS) e MA; EW; MM (HUS)	LR (RSS/HUS) HP (RSS)	LR (HUS)
	30s	25s	30s	30s
3ª	SH (RSS)	LR (RSS/HUS)	MA; MM (RSS)	EW (HUS)
	21m40s	30s	40s	14m35s

Para o PostgreSQL, o RSS se mostrou melhor que o HUS, ficando nas primeira posição em 3 de 4 cenários de carga de trabalho. Vale citar que isso ocorreu graças ao ótimo desempenho da combinação das técnicas de detecção de tendências com o RSS. Em contrapartida, as técnicas de CEP combinadas com RSS jamais aparecem nas três primeiras posições, pois tal combinação possui tempos de detecção muito maiores do que qualquer outra combinação. Assim, o RSS apresenta os melhores tempos de detecção graças às técnicas de detecção de tendências.

Finalmente, para o Squid, o indicador HUS desempenhou-se melhor que o RSS, estando presente em todas as três primeiras colocações de todos os cenários de carga de trabalho e só não liderou sozinho, pois a técnica Média Móvel (MA) conseguiu igualar o desempenho quando combinada com o RSS. Contudo, para as demais posições, HUS leva ampla vantagem sobre o RSS.

Por fim, para precisar a melhor combinação geral técnica/indicador foi feita uma análise das combinações que mais vezes estiveram nas melhores colocações do *ranking*. Conforme mencionado anteriormente, as técnicas de detecção de tendências foram superiores em 3 de 4 aplicações. Os indicadores RSS e HUS empataram em duas aplicações e cada um foi melhor em outras duas. Entretanto, por mais que RSS e HUS tenham empatado em termos de presença nas primeiras posições, a frequência do HUS no *ranking* para as demais posições é maior. Além disso, os tempos de detecção do HUS, no geral, foram melhores que os tempos do RSS. Assim conclui-se que HUS tem um melhor desempenho que o RSS. A Tabela 5.10 traz um resumo dessa análise. Portanto, como a técnicas de detecção de tendências foram melhores, especialmente HP, e o indicador de vazamento HUS também, então conclui-se que a melhor combinação geral técnica/indicador foi HP/HUS. As células em vermelho na Tabelas 5.9 e 5.10 referem-se a posições que tiveram apenas RSS, as esverdeadas apenas HUS e as amareladas para ambos os indicadores.

Tabela 5.10. Melhor combinação geral técnica/indicador por aplicação.

Inkscape	Lighttpd	PostgreSQL	Squid
MA; HP; MM (RSS/HUS)	SH; CS (RSS/HUS)	HP (RSS)	HP (HUS)

5.3.2 Perguntas e hipóteses de pesquisa

O objetivo desta Seção é realizar uma análise das conclusões gerais à luz das perguntas e hipóteses de pesquisa levantadas no início deste documento. Uma vez que essas perguntas e hipóteses são, principalmente, baseadas nos trabalhos anteriores [Matias *et al*. 2014] e [Matias *et al*. 2016], então a seguir é apresentado um panorama geral dos resultados anteriores e dos resultados vistos nesta pesquisa.

Em relação ao número de falso-negativos, tanto em [Matias *et al*. 2014] quanto em [Matias *et al*. 2016], algumas combinações técnica/indicador não conseguiram detectar o vazamento de memória quanto deveriam, gerando falso-negativos. No primeiro, as técnicas MA e MM foram as que se destacaram negativamente em termos de falso-negativos, tanto combinadas com RSS quanto com HUS. No segundo, essas mesmas técnicas, além de LR, não foram bem quando combinadas com RSS. Entretanto, em um dos 2 cenários avaliados, essas técnicas combinadas com HUS apresentaram bons resultados, fundamentalmente HP que liderou. Nesta pesquisa não houve combinações técnica/indicador que geraram falso-negativos.

Sobre o tipo de técnica mais efetiva, em [Matias *et al*. 2014], as técnicas de CEP aparecem no *ranking* dos três primeiros lugares em aproximadamente 66% do casos, com destaque para CS. O restante fica para as técnicas de detecção de tendências, onde se destaca HP. Já em [Matias *et al*. 2016], foram avaliados somente 2 cenários, então não é interessante aferir a melhor técnica geral, sendo

preferível comentar as melhores para cada cenário: SH para o constante e; LR e HP para o variado.

Os resultados desta pesquisa mostraram que as técnicas de detecção de tendências foram melhores em 3 de 4 aplicações. Essas técnicas aparecem sozinhas, em uma das três primeiras posições do *ranking* de cada aplicação e cenário de carga, em aproximadamente 62% dos casos, com destaque para HP. As técnicas de CEP aparecem em aproximadamente 27% e os dois tipos se mostram juntos (empatados) em aproximadamente 11%. É preciso mencionar que os únicos casos em que as técnicas de CEP aparecem sozinhas em primeiro lugar no *ranking* foi para o Lighttpd.

Com relação aos tempos de detecção, em [Matias *et al*. 2014] os melhores tempos variam entre aproximadamente 2 e 60 minutos. Essa variação acontece de acordo com o cenário de carga de trabalho. Já para o trabalho de [Matias *et al*. 2016] os melhores tempos foram: 1) cenário constante: 7m45s e; 2) cenário variado: 3m. Nesta pesquisa, os tempos de detecção foram bem menores, variando entre 20s e 25s.

Quanto ao indicador de vazamento de memória, em [Matias *et al*. 2014], o indicador HUS aparece no *ranking* do três primeiros lugares em aproximadamente 60% dos casos, contra 40% do RSS. Em [Matias *et al*. 2016], nos dois cenários avaliados, o HUS preenche todas as três primeiras posições. Nesta pesquisa, os resultados mostraram que os indicadores RSS e HUS empataram em duas aplicações e cada um liderou as restantes. Contudo, o indicador HUS domina o *ranking* dos três primeiros lugares aparecendo em aproximadamente 29% dos casos, contra aproximadamente 17% do RSS – os dois aparecem juntos empatados em 54% das situações.

Finalmente, com relação a melhor combinação técnica/indicador, em [Matias *et al*. 2014] a melhor, no geral, foi CS/HUS. Já o trabalho de [Matias *et al*. 2016] a melhor combinação para cada cenário foi: 1) constante: SH/HUS e; 2) variado: LR/HUS e HP/HUS. Nesta pesquisa, a melhor combinação geral foi: HP/HUS.

Portanto, apresentado os resultados gerais desta pesquisa e dos trabalhos anteriores, é possível relacioná-los com as perguntas e hipóteses de pesquisa:

1ª Pergunta: O desempenho das técnicas estatísticas e dos indicadores de envelhecimento, usados para detectar vazamento de memória com cargas sintéticas definidas *adhoc* [Matias *et al*. 2014], será o mesmo frente a aplicações reais e cargas planejadas o mais próximo possível de situações reais de uso dessas aplicações (*não adhoc*)?

Hipótese: O desempenho dos indicadores de envelhecimento será o mesmo visto com carga de trabalho sintética *adhoc* [Matias *et al*. 2014]. Isso porque se espera que os efeitos do vazamento reflitam antes no HUS que no RSS. A mesma expectativa também é prevista para as técnicas estatísticas, pois acredita-se que as técnicas de CEP vão manter os melhores desempenhos. Com relação ao número de falso-negativos e falso-positivos, crê-se que a utilização de cargas planejadas para apenas ativar os *bugs* de vazamento encontrados irá influenciar nessa perspectiva, apresentando um número menor de taxas do que com carga sintética *adhoc*, como a de [Matias *et al*. 2014], projetada para considerar também fatores que não estão relacionados à ativação de *bugs* de envelhecimento.

É importante mencionar que essa comparação com os trabalhos anteriores deve ser vista com muito cuidado. Primeiro que em [Matias *et al.* 2014] a carga sintética utilizada, definida *adhoc*, não considerou apenas a ativação das faltas de envelhecimento injetadas no programa sintético que foi objetivo da análise. Em contrapartida, nesta pesquisa, a carga projetada em 3 de 4 aplicações focou apenas em explorar os padrões de ativação dos *bugs* (ou faltas) de vazamento de memória encontrados. Assim, a comparação desses resultados não é muito equilibrada, pois se referem a escopos de cargas de trabalho diferentes. Entretanto, como uma das perguntas de pesquisa era verificar se, mesmo sob perspectivas diferentes, os resultados se repetiriam, então uma análise breve é feita a seguir.

De acordo com os resultados vistos, o desempenho das técnicas estatísticas nesta pesquisa não foi o mesmo encontrado com carga sintética *adhoc* [Matias *et al*. 2014]. Isso porque as técnicas de detecção de tendências foram melhores que as de CEP – em [Matias *et al*. 2014] as de CEP foram as melhores. Contudo, é preciso citar que em um dos 2 cenários avaliados em [Matias *et al*. 2016], onde a carga também era sintética *adhoc*, mas o sistema alvo do estudo era uma aplicação real onde a carga focava apenas em ativar o *bug* de vazamento estudado nessa aplicação real, as técnicas de detecção de tendências também se saíram melhores. Com relação ao desempenho dos indicadores, assim como nos trabalhos anteriores, HUS se apresentou melhor que o RSS. Além disso, conforme esperado, a utilização de cargas planejadas com foco em apenas ativar os *bugs* de vazamento considerados, apresentou poucas taxas de falso-negativos (nenhuma) e falso-positivos.

2ª Pergunta: Qual o efeito do tipo de aplicação (ex. servidor *web* ou banco de dados) sobre os resultados das combinações técnica/indicador?

Hipótese: O efeito do tipo de aplicação não irá influenciar em termos de falso-negativos e falso-positivos nas combinações técnica/indicador e sim a utilização de cargas projetadas apenas para ativar os *bugs* de vazamento considerados.

A utilização de aplicações reais com cargas sintéticas baseadas no comportamento real de uso dessas aplicações estudadas (não *adhoc*) apresentou como resultado nenhuma taxa de falso-negativos e falso-positivos consideráveis em apenas 1 das 4 aplicações estudadas. Assim, conforme esperado, o modo como as cargas usadas foram planejadas, focando apenas em ativar o *bug* de vazamento estudado em cada aplicação, influenciou nessa panorama. Portanto, acredita-se que se as cargas utilizadas incluírem também fatores não relativos aos padrões de ativação dos *bugs* estudados, então taxas de falso-positivos serão maiores e falso-negativos podem surgir.

Finalmente, a conclusão mais relevante a se realizar é que a abordagem de análise diferencial também se mostrou efetiva perante testes de sistema considerando aplicações reais com cargas de trabalho baseadas em um comportamento real dessas aplicações. Portanto, uma vez que todas as combinações técnica/indicador produziram bons resultados, ameças de validade externa da abordagem foram diminuídas através dessa pesquisa.

6. CONCLUSÕES DA PESQUISA

6.1 Síntese dos Resultados

Este trabalho objetivou avaliar a validade externa da abordagem de análise diferencial de software introduzida em [Matias *et al*. 2014]. Para isso, executou um conjunto de replicações experimentais frente a aplicações reais. As aplicações utilizadas foram aquelas que satisfizeram as condições necessárias para os *bugs* encontrados em fóruns oficiais de relatórios de *bugs*. Tais condições estão relacionadas aos testes prévios de confirmação do vazamento de memória e de localização das versões base e alvo. As cargas de trabalho foram definidas através de um estudo de caracterização dos padrões de ativação de cada *bug* estudado, apoiando um planejamento experimental com cenários de carga que representassem o comportamento real de cada aplicação frente ao padrão de ativação do seu respectivo *bug*. Assim, os valores de divergência foram usados para determinar as melhores combinações técnica/indicador em cada caso.

A análise dos valores de divergência e seus respectivos gráficos mostram que, nas aplicações e cenários de carga considerados, todas as combinações técnica/indicador conseguiram detectar o vazamento de memória de forma precisa, não gerando falso-negativos. Isso mostra como a técnica de análise diferencial também é efetiva em testes de sistema considerando aplicações reais e cargas definidas com um comportamento o mais próximo possível do real.

Os valores de divergência também mostraram que todas as combinações técnica/indicador, com exceção no Squid, não geraram quase nenhuma taxa de falso-positivo. No Squid, um número considerável de falso-positivos surgiu na combinação das técnicas estatísticas, principalmente LR e MM com o indicador RSS. Acredita-se que o comportamento similar das séries temporais médias do indicador RSS entre as versões alvo e base contribuíram para esse resultado. Portanto, conclui-se que essas técnicas de detecção de tendências não são efetivas na presença de maior instabilidade na série. Além disso, essa instabilidade é esperada para o indicador RSS, por considerar várias partes do processo carregadas na memória principal, sendo mais susceptível a ruídos do que o HUS. Para as demais aplicações, onde taxas de falso-positivos não são vistas, crê-se que o modo como as cargas foram planejadas, considerando apenas a ativação dos *bugs* de vazamentos analisados, contribuiu para esse panorama.

As técnicas de detecção de tendências foram mais efetivas que as de CEP em 3 das 4 aplicações estudadas e, embora algumas delas (LR e MM) não tenham ido bem quando combinadas com o RSS no Squid, as demais lideraram o *ranking* tanto para RSS quanto para HUS. Para a única aplicação em que as técnicas de CEP foram melhores, Lighttpd, a diferença entre os tempos de detecção de ambos os tipos de técnica foi muito pequena. Entre as técnicas de detecção de tendências, destacaram-se HP e MA. Para as de CEP, CS e SH foram as melhores. Portanto, é

imperativo concluir que a utilização de aplicações reais com cargas o mais próximo do real, projetadas para apenas considerar a ativação dos *bugs*, tendem a apresentar as técnicas de detecção de tendências como mais efetivas que as de CEP, além de mostrar novamente o indicador HUS como mais preciso que o RSS.

Na literatura em SAR, existem muitos estudos que utilizaram técnicas de detecção de tendências em combinação com indicadores para compreender o fenômeno do envelhecimento e estimar o tempo até a falha. Conforme mostrou [Machida *et al*. 2013], nem sempre essas simples combinações conseguem detectar precisamente o envelhecimento, apresentando muitos falso-positivos e falso-negativos, principalmente por não combinar as melhores técnicas e/ou indicadores para detectar determinado efeito de envelhecimento. É por isso que [Machida *et al*. 2013] sugere um protocolo global que precisa ser seguido considerando vários itens, entre eles, a determinação das melhores combinações para uma dada classe de efeito.

Como pode ser visto na Tabela 5.10, que mostra a melhor combinação técnica/indicador geral para cada aplicação, a técnica HP foi mais frequente na liderança e demais posições do *ranking*. Assim, a melhor combinação geral do estudo realizado nesta pesquisa foi HP/HUS.

Portanto, uma das contribuições desta pesquisa de mestrado para literatura em SAR foi produzir evidências, por meio de experimentos controlados com aplicações reais, da eficiência da análise diferencial em detectar vazamentos de memória. Essas evidências podem ser usadas por outros trabalhos para subsidiar suas escolhas em termos de técnicas e indicadores. O fato dos resultados terem sido obtidos com aplicações reais ajuda a buscar, no futuro, generalizações neste campo de pesquisa.

A utilização do método DTW permitiu capturar semelhanças flexíveis entre as séries das replicações gerando uma série temporal média para cada cenário e indicador. Por esse método não ter sido usado nos trabalhos anteriores e sua aplicação ter apresentado benefícios no cálculo dos valores de divergência, então se torna outra contribuição desta pesquisa.

Finalmente, o processo de caracterização das cargas de trabalho baseando-se o mais próximo possível de comportamentos reais de uso frente aos padrões de ativação de cada *bug*, também é uma contribuição desta pesquisa. Primeiro, por que a maioria dos trabalhos em SAR realizam testes de sistemas com cargas de trabalho definidas *adhoc*. Segundo, pois apresenta de forma sistemática dados específicos sobre problemas de vazamento de memória em aplicações reais que permitem outros trabalhos utilizá-los para reproduzir esses problemas para novas análises.

6.1 Ameaças à Validade

Como esta pesquisa trabalha com dados obtidos de forma empírica, consequentemente existem limitações que devem ser consideradas ao interpretar seus resultados. Nas seções seguintes são destacadas algumas ameaças de validade interna, externa[Siegmund *et al*. 2015].

Uma das principais preocupações relativas a esta pesquisa foi a de diminuir erros experimentais. Para mitigar essa ameaça, múltiplas replicações experimentais foram realizadas para cada aplicação avaliada e as séries temporais

obtidas dessas replicações foram estimadas pelo método DTW. Esse processo garantiu que o cálculo dos valores de divergência não fosse realizado com dados extraídos apenas de uma execução experimental e que as séries temporais médias calculadas pelo DTW considerassem as semelhanças flexíveis entre as séries das replicações.

Além disso, outra preocupação foi relativa ao processo de geração da carga de trabalho. Para garantir que todas as replicações fossem realizadas igualmente, programas de geração de carga de trabalho foram implementados baseados no padrão de ativação de cada *bug* considerado nesta pesquisa. Isso permitiu que as cargas de trabalho fossem acionadas de forma automatizada, evitando a influência de erros humanos nos experimentos.

Também é preciso citar que a instrumentação utilizada para coletar dados do indicador HUS, através de um *wrapper* de alocação dinâmica de memória chamado *DebugMalloc*, considerou a precisão de nano segundos para registrar data/hora de cada pedido de alocação e desalocação. O *DebugMalloc* gera dois arquivos para cada replicação experimental: o primeiro relacionado às alocações e o segundo as desalocações. Assim, para produzir uma série temporal, com valores a cada 5 segundos, foi preciso criar um programa para mesclar esses arquivos e calcular o HUS nessa regularidade de tempo. Esse cálculo leva em conta a precisão de nano segundos registrada nos arquivos, que é limitada à qualidade do temporizador de *hardware* [Matias *et al*. 2016].

Sobre os aspectos relacionados com a generalização dos resultados encontrados, uma das principais motivações desta pesquisa foi de contribuir para a validade externa da abordagem de análise diferencial de software. Assim, 4 aplicações reais foram analisadas. Todas elas representativas em suas áreas, sendo 3 do tipo servidor: Lighttpd (*web*); PostgreSQL (banco de dados) e; Squid (*proxy cache*); e uma do tipo *desktop*: Inkscape (imagem). Além disso, para todas as aplicações avaliadas, cenários de carga de trabalho foram planejados através de um estudo de caracterização da carga de trabalho, baseado nos padrões de ativação de cada *bug*. Assim, espera-se que os cenários planejados refletiram um comportamento aproximado ao típico para cada aplicação analisada.

Entretanto, é preciso mencionar que com exceção do PostgreSQL, os cenários de cargas de trabalho planejados para as demais aplicações foram exclusivamente baseados na ativação de seus respectivos *bugs* de vazamento, não incluindo na carga de trabalho a exploração de outras funcionalidades onde os *bugs* não fossem ativados. Portanto, a ocorrência dessa situação pode explicar alguns comportamentos observados, como a ausência de falso-positivos para o Inkscape e Lighttpd. Já para o PostgreSQL, onde ocorreu um caso de falso-positivo, o planejamento dos cenários de carga considerou a execução de outras ações não relativas à ativação de seu *bug* de vazamento. Ainda, é preciso mencionar que as várias ocorrências de falso-positivos para o Squid foram ocasionadas, principalmente, pela presença de ruídos nas séries temporais do RSS coletadas da versão base.

A técnica de análise diferencial é projetada, principalmente, para uso no ciclo de vida de uma aplicação, onde na fase de testes uma nova versão é colocada sob investigação de envelhecimento (alvo) e comparada com a versão atualmente no ambiente de produção (base). Nesta pesquisa, diferente da situação de uso em um ambiente de desenvolvimento, todas as versões alvo foram previamente

confirmadas com o vazamento, pois o objetivo principal aqui era verificar a efetividade da análise diferencial em detectar vazamentos. Entretanto, qual seria o comportamento da abordagem, em termos de falso-positivos, quando a versão alvo não contivesse vazamento de memória? Para responder isso é que em [Matias *et al.et al.* 2016] foram realizados experimentos onde a versão alvo não tinha vazamento. Os resultados mostraram a efetividade da análise diferencial em não gerar falso-positivos. Portanto, existem fortes evidências que a análise diferencial vai se portar bem em uma situação real de uso, onde não há certeza que a versão alvo contém envelhecimento.

Além disso, a utilização de aplicações reais e cargas definidas o mais próximo da realidade aumentou a validade externa da abordagem de análise diferencial de software para detecção do envelhecimento. Isso porque todas as combinações técnica/indicador foram efetivas na detecção vazamento de memória, fazendo com que os resultados vistos aqui possam ser considerados em ambientes e aplicações reais similares de forma mais seguira que os resultados dos trabalhos anteriores.

6.2 Dificuldades encontradas

A principal dificuldade encontrada foi relativa ao processo de busca dos *bugs* de vazamentos e, consequentemente, das aplicações utilizadas. Esse processo está descrito nas etapas 1 e 2 da Seção 3.3. Vários fóruns oficiais de *bugs* não padronizam o formulário de relatórios, então muitos usuários reportam *bugs* com poucas informações para reproduzi-los. Além disso, é preciso, primeiramente, conhecer bem sobre determinadas aplicações antes de reproduzir seus *bugs*, de modo que a falta de *expertise* não prejudique o processo. Assim, muitos *bugs* de vazamento foram encontrados e posteriormente descartados, pois nessas ocasiões ou o *bug* de vazamento não se manifestou ou se manifestou e não se tratava de vazamento, desperdiçando um tempo considerável até chegar a essa conclusão.

Posteriormente, depois de confirmar o vazamento, foi necessário encontrar as versões alvo e base. Contudo, todos os *bugs* estudados nesta pesquisa não começaram a se manifestar em suas respectivas aplicações nas versões em que foram reportados no fórum, ou seja, a versão alvo não era a mesma versão informada no relatório do *bug*. Portanto, para cada *bug* foi preciso encontrar o repositório de versões de sua respectiva aplicação para instalar e configurar versões anteriores da mesma, de modo a testar uma a uma para constatar em qual versão o *bug* começou a se manifestar. Esse processo se torna muito demorado, principalmente pela incompatibilidade de bibliotecas do SO com as versões mais antigas de aplicações. Assim, frequentemente, para instalar uma versão mais antiga, foi necessário alterar várias versões de bibliotecas do SO. Dessa forma, em muitas ocasiões, *bugs* de vazamento que satisfizeram a etapa de verificação foram descartados por não possuírem uma versão base, desperdiçando também um enorme tempo.

Outra dificuldade a ser citada foi quanto à caracterização da carga de trabalho. Essa caracterização foi feita com base em trabalhos acadêmicos e/ou repositórios de informações relacionados aos padrões de ativação dos *bugs* estudados. Assim, especificamente para os *bugs* das aplicações PostgreSQL e Inkscape, foram encontrados pouquíssimas referências úteis para o planejamento da carga de trabalho, sendo inclusive necessário realizar experimentos com

pessoas de diversos níveis de conhecimento em informática para caracterizar os cenários de carga do Inkscape.

6.3 Trabalhos futuros

Como trabalho futuro pretende-se criar uma nova abordagem para detectar, mais eficientemente do que outras existentes, não só um tipo de efeito de envelhecimento, mas sim múltiplos efeitos. Uma vez que a análise diferencial de software demonstrou ser efetiva na detecção do vazamento de memória, pois diferencia o comportamento natural do software do comportamento do envelhecimento, então ela será mantida como ponto central dessa nova abordagem.

Para isso, planeja-se considerar o trabalho de [Valentim *et al*. 2016] e outros para verificar os indicadores e técnicas frequentemente utilizadas para cada efeito. Portanto, esses indicadores e técnicas serão combinados através de experimentos de análise diferencial frente a estudos de casos abrangendo múltiplos efeitos, objetivando determinar as melhores combinações para cada um. É preciso lembrar que não existem estudos que determinam melhores combinações para vários efeitos na literatura em SAR. Os estudos de caso serão realizados frente a aplicações e cargas definidas também através de um processo de caracterização para diminuir limitações de validade externa e permitir a generalização dos resultados encontrados para outros tipos de aplicações. Para isso, assim como foi feito nesta pesquisa, será preciso realizar uma busca minuciosa de *bugs* de software e elaborar caracterizações de carga de trabalho baseadas nos padrões de ativação dos *bugs* encontrados.

Contudo, essa nova abordagem que pretende-se propor não restringirá em apenas aumentar o número de indicadores, técnicas, aplicações reais e efeitos estudados. Tem-se como objetivo também realizar um mapeamento das características e particularidades de técnicas anteriormente propostas. Tal mapeamento pode evidenciar limitações dessas técnicas que, quando encontradas, se relevantes, serão consideradas em termos de planejamento de melhorias. Ainda, o mapeamento poderá trazer elementos importantes que poderão ser combinados na proposta de detecção dos múltiplos efeitos.

REFERÊNCIAS BIBLIOGRÁFICAS

[Aach e Church 2001] AACH, J.; CHURCH, G. M.. Aligning gene expression time series with time warping algorithms. *Bioinformatics*, v. 17, n. 6, p.495-508, 1 jun. 2001. Oxford University Press (OUP).

[Ahmad et al. 2009] AHMAD, M.; ABOULNAGA, A.; BABU, S.. Query interactions in database workloads. *Proceedings Of The Second International Workshop On Testing Database Systems - Dbtest '09*, p.1-6, 2009. ACM Press.

[Alonso et. al 2010] ALONSO, J. et al. A Comparative Evaluation of Software Rejuvenation Strategies. *2011 Ieee Third International Workshop On Software Aging And Rejuvenation*, p.26-31, nov. 2011. IEEE.

[Al-Naymat et al. 2009] AL-NAYMAT, G.; CHAWLA, S.; TAHERI, J.. SparseDTW: A Novel Approach to Speed up Dynamic Time Warping. *The 2009 Australasian Data Mining*, v. 101, p. 117-127, dez. 2009. ACM Digital Library.

[Araujo et. al 2011] ARAUJO, J. et al. Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. *Proceedings Of The Middleware 2011 Industry Track Workshop On - Middleware '11*, Nova Iorque, p.1-7, 2011. ACM Press.

[Arguel e Jamet 2009] ARGUEL, A.; JAMET, E. Using video and static pictures to improve learning of procedural contents. *Computers In Human Behavior*, [s.l.], v. 25, n. 2, p.354-359, mar. 2009. Elsevier BV.

[Arlitt e Williamson 1996] RLITT, M. F.; WILLIAMSON, C. L.. Web server workload characterization. *Acm Sigmetrics Performance Evaluation Review*, [s.l.], v. 24, n. 1, p.126-137, 15 maio 1996. Association for Computing Machinery (ACM).

[Arlitt e Williamson 1997] ARLITT, M. F.; WILLIAMSON, C. L.. Internet Web servers: workload characterization and performance implications. *Ieee/acm Transactions On Networking*, [s.l.], v. 5, n. 5, p.631-645, 1997. Institute of Electrical and Electronics Engineers (IEEE).

[Arlitt et al. 1999] ARLITT, M.; FRIEDRICH, R.; JIN, T. Workload characterization of a Web proxy in a cable modem environment. *Acm Sigmetrics Performance Evaluation Review*, Nova Iorque, v. 27, n. 2, p.25-36, 1 set. 1999. Association for Computing Machinery (ACM).

[Armstrong et al. 2013] ARMSTRONG, T. G. et al. LinkBench. *Proceedings Of The 2013 International Conference On Management Of Data - Sigmod '13*, Nova Iorque, p.1185-1196, 2013. ACM Press.

[Avritzer e Weyuker 1997] AVRITZER, A.; WEYUKER, E. J.. Monitoring Smoothly Degrading Systems for Increased Dependability. *Empirical Software Engineering*, [s.l.], v. 2, n. 1, p.59-77, 1997. Springer Nature.

[Avižienis et al. 2004] AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *Ieee Transactions On Dependable And Secure Computing*, Los Angeles, v. 1, n. 1, p.11-33, jan. 2004. Institute of Electrical and Electronics Engineers (IEEE).

[Bao et al. 2005] BAO, Y.; SUN, X.; TRIVEDI, K. S.. A Workload-Based Analysis of Software Aging, and Rejuvenation. *Ieee Transactions On Reliability*, [s.l.], v. 54, n. 3, p.541-548, set. 2005. Institute of Electrical and Electronics Engineers (IEEE).

[Barbetta et al. 2010] BARBETTA, P. A.; REIS, M. M.; BORNIA, A. C.. *Estatística para Cursos de Engenharia e Informática*. 3. ed. São Paulo: Atlas, 2010. 410 p.

[Benattar e Trébuchet 2011] BENATTAR, G.; TRÉBUCHET, P. Trend Analysis in Polls, Topics, Opinions and Answers. *Laboratório de Informática de Paris*. Paris, jun. 2011.

[Bent et al. 2006] BENT, L. et al. Characterization of a large web site population with implications for content delivery. *Proceedings Of The 13th Conference On World Wide Web - Www '04*, Nova Iorque, p.522-533, 2004. ACM Press.

[Berger et al. 2000] BERGER, E. D. et al. Hoard. *Acm Sigops Operating Systems Review*, Nova Iorque, v. 34, n. 5, p.117-128, 1 dez. 2000. Association for Computing Machinery (ACM).

[Black 2004] BLACK, P. E.. Euclidean distance: definition. 2004. Disponível em: <https://xlinux.nist.gov/dads/HTML/euclidndstnc.html>. Acesso em: 18 jul. 2017.

[Bovenzi et al. 2012] BOVENZI, A. et al. On the Aging Effects Due to Concurrency Bugs: A Case Study on MySQL. *2012 IEEE 23rd International Symposium On Software Reliability Engineering*, Dallas, p.211-220, nov. 2012. IEEE.

[Bronson et al. 2013] BRONSON, N. et al. TAO: Facebook's distributed data store for the social graph. *USENIX Annual Technical Conference*, San Jose, jun. 2013. USENIX.

[Bugs Launchpad 2017] *Serious memory leakage while browsing svg files.* 2008. Bugs Launchpad Web Site. Disponível em: <https://bugs.launchpad.net/inkscape/+bug/294969>. Acesso em: 18 jul. 2017.

[*Bugs* Squid 2015] *Bug 4193 - Memory leak on FTP listings.* 2015. *Bugs* Squid *Web* Site. Disponível em: <http://*bugs*.squid-*cache*.org/show_*bug*.cgi?id=4193>. Acesso em: 18 jul. 2017.

[Cardamom 2017] CARDAMOM: An Enterprise Middleware for Building Mission and Safety Critical Applications. 2006. Cardamom Home Page. Disponível em: <http://cardamom.ow2.org/>. Acesso em: 18 jul. 2017.

[Carrozza et al. 2010] CARROZZA, G. et al. Memory leak analysis of mission-critical middleware. *Journal Of Systems And Software*, Nova Iorque, v. 83, n. 9, p.1556-1567, set. 2010. Elsevier BV.

[Carvalho 2012] CARVALHO, A. M. M.. *Controle Estatístico de Processos de Predição de Tráfego de Redes de Computadores*. 2012. 124 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Uberlândia, Uberlândia, 2012. Cap. 3.

[Castelli et al. 2001] CASTELLI, V. et al. Proactive management of software aging. *IBM Journal Of Research And Development*, Riverton, v. 45, n. 2, p.311-332, mar. 2001. IBM.

[Claffy et al. 1998] CLAFFY, K. et al. The Nature of the Beast: Recent Traffic Measurements From an Internet Backbone. *International Networking Conference (INET)*, Geneva, jul. 2001

[Costa et al. 2005] COSTA, A. F. B.; EPPRECHT, E. K.; CARPINETTI, L. C. R.. *Controle Estatístico de Qualidade*. 2. ed. [s.l.]: Atlas, 2008. 334 p.

[Costa e Matias 2015] COSTA, D.; MATIAS, R.. Characterization of Dynamic Memory Allocations in Real-World Applications: An Experimental Study. *2015 IEEE 23rd International Symposium On Modeling, Analysis, And Simulation Of Computer And Telecommunication Systems*, Atlanta, p.93-101, out. 2015. IEEE.

[Cotroneo et al. 2010] COTRONEO, D. et al. Software Aging Analysis of the Linux Operating System. *2010 IEEE 21st International Symposium On Software Reliability Engineering*, San Jose, p.71-80, nov. 2010. IEEE.

[Cotroneo et al. 2011] COTRONEO, D. et al. Software Aging and Rejuvenation: Where We Are and Where We Are Going. *2011 IEEE Third International Workshop On Software Aging And Rejuvenation*, Hiroshima, p.1-6, nov. 2011. IEEE.
[Cross-Validation 1997] *Cross Validation*. 1997. Disponível em: <http://www.cs.cmu.edu/~schneide/tut5/node42.html>. Acesso em: 18 jul. 2017.
[DARPA 2017] *Defense Advanced Research Projects Agency (DARPA)*. 2017. DARPA Web Site. Disponível em: <http://www.darpa.mil/>. Acesso em: 18 jul. 2017.
[DreamHost 2012] *Web server performance comparison - DreamHost*. 2012. DreamHost *Web* Site. Disponível em: <https://help.dreamhost.com/hc/en-us/articles/215945987-*Web*-server-performance-comparison>. Acesso em: 18 jul. 2017.
[Eldin et al. 2014] ELDIN, A. A. et al. How will Your Workload Look Like in 6 Years? Analyzing Wikimedia's Workload. *2014 IEEE International Conference On Cloud Engineering*, Boston, p.348-354, mar. 2014. IEEE.
[Elnaffar et al. 2002] ELNAFFAR, S.; MARTIN, P.; HORMAN, R.. Automatically classifying database workloads. P*roceedings Of The Eleventh International Conference On Information And Knowledge Management - CIKM'02*, New York, p.622-624, 2002. ACM.
[Eucalyptus 2017] *Eucalyptus Software Support Services.* 2017. Eucalyptus Web Site. Disponível em: <http://www.dxc.techno*log*y/cloud/offerings/140041/140149-eucalyptus_software_support_services>. Acesso em: 18 jul. 2017.
[Euclidean distance 2017] *Euclidean distance*. 2017. Disponível em: <http://www.nist.gov/dads/HTML/euclidndstnc.html>. Acesso em: 18 jul. 2017.
[Erman et al. 2006] ERMAN, J.; ARLITT, M.; MAHANTI, A.. Traffic classification using clustering algorithms. *Proceedings Of The 2006 SIGCOMM Workshop On Mining Network Data - MineNet '06*, New York, p.281-286, 2006. ACM Press.
[Farias e Laurencel 2006] FARIAS, A. M. L.; LAURENCEL, L. C.. *Estatística Descritiva*. Rio de Janeiro: Universidade Federal Fluminense, 2006. 128 p.
[Fedora Project 2015] *Index of /lookaside/extras/lighttpd*. (2015) Fedora Project Repository. Disponível em: <http://pkgs.fedoraproject.org/lookaside/extras/lighttpd/>. Acesso em: 18 jul. 2017.
[Ferreira et al. 2011] FERREIRA, T. B. et al. An Experimental Study on Memory Allocators in Multicore and Multithreaded Applications. *2011 12th International Conference On Parallel And Distributed Computing, Applications And Technologies*, Gwangju, p.92-98, out. 2011. IEEE.
[Freedman 2009] FREEDMAN, D. *Statistical Models*: Theory and Practice. [s.I.]: Cambridge University Press, 2005. 414 p.
[Frey e Dueck 2007] FREY, B. J.; DUECK, D.. Clustering by Passing Messages Between Data Points. *Science*, [s.l.], v. 315, n. 5814, p.972-976, 16 fev. 2007. American Association for the Advancement of Science (AAAS).
[GNU C Library 2017] *The Gnu C Library*: Freeing after Malloc. 2017. Disponível em <http://www.gnu.org/software/libc/manual/html_node/Freeing-after-Malloc.html>. Acesso em: 18 jul. 2017.
[Grottke et al. 2006] GROTTKE, M. et al. Analysis of Software Aging in a Web Server. *IEEE Transactions On Reliability*, [s.l.], v. 55, n. 3, p.411-420, set. 2006. Institute of Electrical and Electronics Engineers (IEEE).

[Grottke et al. 2008] GROTTKE, M.; MATIAS, R.; TRIVEDI, K. S.. The fundamentals of software aging. *2008 IEEE International Conference On Software Reliability Engineering Workshops (ISSRE WKSP)*, Seattle, p.1-6, nov. 2008. IEEE.
[Grottke e Trivedi 2007] GROTTKE, M.; TRIVEDI, K.S.. Fighting bugs: remove, retry, replicate, and rejuvenate. *Computer*, [s.l.], v. 40, n. 2, p.107-109, fev. 2007. Institute of Electrical and Electronics Engineers (IEEE).
[Goel e Jha 2013] GOEL, N.; JHA, C. K.. Analyzing Users Behavior from Web Access Logs using Automated Log Analyzer Tool. *International Journal Of Computer Applications*, [s.l.], v. 62, n. 2, p.29-33, 18 jan. 2013. Foundation of Computer Science.
[Gujarati 2000] GUJARATI, D. *Econometria Básica*. 5. ed. São Paulo: Makron Books, 2011.
[Gundavaram 1996] GUNDAVARAM, S. *CGI Programming*: on the World Wide Web. Sebastopol: O'reilly Media, 1996. 450 p.
[Hankins et al. 2003] HANKINS, R. et al. Scaling and characterizing database workloads: bridging the gap between research and practice. *22nd Digital Avionics Systems Conference. Proceedings (cat. No.03ch37449)*, San Diego, p.151-157, dez. 2003. IEEE Comput. Soc.
[Hodrick e Prescott 1980] HODRICK, R.; PRESCOTT, E. C. Post-war U.S. business cycles: An Empirical investigation. *Journal of Money*, v. 29, n. 1. jan. 1980.
[Huang et al. 1995] HUANG, Y. et al. Software rejuvenation: analysis, module and applications. *Twenty-fifth International Symposium On Fault-tolerant Computing*. Digest Of Papers, Pasadena, p.381-390, 1995. IEEE Comput. Soc. Press.
[Hyndman 2014] DAGUM, E. B. Moving Averages. *Encyclopedia Of Statistical Sciences*, [s.l.], p.866-869, 15 ago. 2006. John Wiley & Sons, Inc..
[Inkscape 2017a] *Visão geral do Inkscape*. 2017. Inkscape Web Site. Disponível em: <https://inkscape.org/pt-br/sobre/>. Acesso em: 18 jul. 2017.
[Inkscape 2017b] *Frequently Asked Questions - for Inkscape Users*. 201. Inkscape *Web* Site. Disponível em: <https://inkscape.org/en/learn/faq/#how-did-inkscape-start>. Acesso em: 18 jul. 2017.
[Itakura 1975] ITAKURA, F.. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions On Acoustics, Speech, And Signal Processing*, [s.l.], v. 23, n. 1, p.67-72, fev. 1975. Institute of Electrical and Electronics Engineers (IEEE).
[I-Shou University FTP Server Usage 2005] *Usage Statistics for http://ftp.isu.edu.tw ftp.isu.edu.tw.* 2017. I-Shou University FTP Server Usage. Disponível em: <http://ftp.isu.edu.tw/usage/>. Acesso em: 18 jul. 2017.
[JAIST 2017a] *JAIST Home Page.* 2017. JAIST Web Site. Disponível em: <https://www.jaist.ac.jp/english/>. Acesso em: 18 jul. 2017.
[JAIST 2017b] *JAIST Public Mirror Service*. 2017. JAIST *Web* Site. Disponível em: <https://ftp.jaist.ac.jp/>. Acesso em: 18 jul. 2017.
[JAIST FTP Server Usage 2017] *FTP Usage Statistics for ftp.jaist.ac.jp.* 2015. JAIST FTP Server Usage. Disponível em: <http://ftp.jaist.ac.jp/ftp-admin/awffull/ftp/>. Acesso em: 18 jul. 2017.
[Jong e Sakarya 2013] JONG, R. M; SAKARYA, N. The Econometrics of the Hodrick-Prescott Filter. *Review Of Economics And Statistics*, [s.l.], v. 98, n. 2, p.310-317, maio 2016. MIT Press - Journals.
[Kamra et al. 2008] KAMRA, A.; TERZI, E.; BERTINO, E.. Detecting anomalous access patterns in relational databases. *The VLDB Journal*, New York, v. 17, n. 5, p.1063-1077, 17 maio 2007. Springer Nature.

[Keogh e Ratanamahatana 2002] KEOGH, E.; RATANAMAHATANA, C. A.. Exact indexing of dynamic time warping. *Knowledge And Information Systems*, New York, v. 7, n. 3, p.358-386, mar. 2005. Springer Nature.
[Kiskis e Shin 1996] KISKIS, D.L.; SHIN, K.G. SWSL: A synthetic workload specification language for real-time systems. *IEEE Transactions On Software Engineering*, [s.l.], v. 20, n. 10, p.798-811, 1994. Institute of Electrical and Electronics Engineers (IEEE).
[Kornfield et al. 2004] KORNFIELD, E.M; MANMATHA, R. e ALAN, J. Text Alignment with Handwritten Documents, *Proceedings. 1st International workshop on Document Image Analysis for Libraris (DIAL)*, [s.I.], pp. 195-209. 2004.
[Laaksonen et al. 1998] LAAKSONEN, Jorma et al. Comparison of Adaptive Strategies for On-Line Character Recognition. *ICANN 98*, [s.l.], p.245-250, 1998. Springer London.
[Langner e Andrzejak 2013a] LANGNER, F.; ANDRZEJAK, A.. Detecting Software Aging in a Cloud Computing Framework by Comparing Development Versions, *Proceedings 13th IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2013)*, [s.I], maio 2013.
[Langner e Andrzejak 2013b] LANGNER, F; ANDRZEJAK, A.. Detection and Root Cause Analysis of Memory-Related Software Aging Defects by Automated Tests. *2013 IEEE 21st International Symposium On Modelling, Analysis And Simulation Of Computer And Telecommunication Systems*, San Francisco, p.365-369, ago. 2013. IEEE.
[Li et al. 2002] LI, L.; VAIDYANATHAN, K.; TRIVEDI, K.S.. An approach for estimation of software aging in a Web server. *Proceedings International Symposium On Empirical Software Engineering*, Nara, p.1-10, 2002. IEEE Comput. Soc.
[Lighttpd 2017a] *Lighttpd - Fly light.* 2017. Lighttpd Web Site. Disponível em: <https://www.lighttpd.net/>. Acesso em: 18 jul. 2017.
[Lighttpd 2017b] *Lighttpd - Fly light Powered by Lighttpd.* 2017. Lighttpd Web Site. Disponível em: <https://www.lighttpd.net/2007/4/4/powered-by-lighttpd/>. Acesso em: 18 jul. 2017.
[Liu et al. 2009] LIU, H. et al. Timing, Timing, Timing: Fast Decoding of Object Information from Intracranial Field Potentials in Human Visual Cortex. *Neuron*, [s.l.], v. 62, n. 2, p.281-290, abr. 2009. Elsevier BV.
[Liu e Williamson 2016] LIU, Y.; WILLIAMSON, C.. Workload Study of a Media-Rich Educational Web Site. *Proceedings Of The 25th International Conference Companion On World Wide Web - WWW '16 Companion*, Québec, p.495-500, 2016. ACM Press.
[Lo et al. 1992] LO, J. L. et al. An analysis of database workload performance on simultaneous multithreaded processors. *Proceedings. 25th Annual International Symposium On Computer Architecture*, Barcelona, p.39-50, 1992. IEEE Comput. Soc.
[Linux Filesystem Hierarchy 2017] *Linux Filesystem Hierarchy. Chapter 1*. Linux Filesystem Hierarchy. Disponível em: <http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html>. Acesso em: 18 jul. 2017.
[Lucas 1985] LUCAS, G. S.. One-Sided Cumulative Sum (CUSUM) Control Charts for the Erlang-Truncated Exponential Distribution. *Computational Methods In Science And Technology*, Dodoma, v. 19, n. 4, p.229-234, 30 dez. 2013. ICHB PAS Poznan Supercomputing and Networking Center.
[Lucas e Saccucci 1990] LUCAS, J. M.; SACCUCCI, M. S.. Exponentially Weighted Moving Average Control Schemes: Properties and Enhancements. *Technometrics*, Alexandria, v. 32, n. 1, p.1-29, fev. 1990.

[Macedo et al. 2010] MACEDO, A.; FERREIRA, T. B.; MATIAS, R.. The mechanics of memory-related software aging. *2010 IEEE Second International Workshop On Software Aging And Rejuvenation*, San Jose, p.1-5, nov. 2010. IEEE.
[Machida et al. 2013] MACHIDA, F. et al. On the effectiveness of Mann-Kendall test for detection of software aging. *2013 IEEE International Symposium On Software Reliability Engineering Workshops (ISSREW)*, Pasadena, p.269-274, nov. 2013. IEEE.
[Magalhães e Silva 2010] MAGALHAES, J. P.; SILVA, L. M.. Prediction of performance anomalies in web-applications based-on software aging scenarios. *2010 IEEE Second International Workshop On Software Aging And Rejuvenation*, San Jose, p.1-7, nov. 2010. IEEE.
[Mahanti et al. 2009] MAHANTI, A.; WILLIAMSON, C.; WU, L. Workload Characterization of a Large Systems Conference Web Server. *2009 Seventh Annual Communication Networks And Services Research Conference*, Moncton, p.55-64, maio 2009. IEEE.
[Maravelakis e Castagliola 2009] MARAVELAKIS, P. E.; CASTAGLIOLA, P.. An EWMA chart for monitoring the process standard deviation when parameters are estimated. *Computational Statistics & Data Analysis*, Amsterdam, v. 53, n. 7, p.2653-2664, maio 2009. Elsevier BV.
[Matias 2015] Matias, R. *Envelhecimento e Rejuvenescimento de Software*, Notas de Aula, 3 mar. 2015, 31 jul. 2015.
[Matias e Filho 2006] MATIAS, R.; FILHO, F. J. P.. An Experimental Study on Software Aging and Rejuvenation in Web Servers. *30th Annual International Computer Software And Applications Conference (COMPSAC'06)*, Chicago, p.189-196, 2006. IEEE.
[Matias et al. 2010a] MATIAS, R. et al. Accelerated Degradation Tests Applied to Software Aging Experiments. *IEEE Transactions On Reliability*, [s.l.], v. 59, n. 1, p.102-114, mar. 2010. Institute of Electrical and Electronics Engineers (IEEE).
[Matias et al. 2010b] MATIAS, R. et al. Measuring software aging effects through OS kernel instrumentation. *2010 IEEE Second International Workshop On Software Aging And Rejuvenation*, San Jose, p.1-6, nov. 2010. IEEE.
[Matias et al. 2010c] MATIAS, R.; TRIVEDI, K. S.; M., Paulo R. M.. Using Accelerated Life Tests to Estimate Time to Software Aging Failure. *2010 IEEE 21st International Symposium On Software Reliability Engineering*, San Jose, p.211-219, nov. 2010. IEEE.
[Matias et al. 2011] MATIAS, R.; FERREIRA, T. B.; MACEDO, A.. An experimental study on user-level memory allocators in middleware applications. *2011 IEEE International Conference On Systems, Man, And Cybernetics*, Anchorage, p.2431-2436, out. 2011. IEEE.
[Matias et al. 2012] MATIAS, R.; COSTA, B. E.; MACEDO, A.. Monitoring Memory-Related Software Aging: An Exploratory Study. *2012 IEEE 23rd International Symposium On Software Reliability Engineering Workshops*, Dallas, p.247-252, nov. 2012. IEEE.
[Matias et al. 2014] MATIAS, R. et al. A Systematic Differential Analysis for Fast and Robust Detection of Software Aging. *2014 IEEE 33rd International Symposium On Reliable Distributed Systems*, Nara, p.311-320, out. 2014. IEEE.
[Matias et al. 2016] MATIAS, Rivalino et al. Software Aging Detection Based on Differential Analysis: An Experimental Study. *2016 IEEE International Symposium On Software Reliability Engineering Workshops (ISSREW)*, Ottawa, p.71-77, out. 2016. IEEE.

[Mohan e Reddy 2015] MOHAN, B. R; REDDY, G. R. M.. The effect of software aging on power usage. 2015 *IEEE 9th International Conference On Intelligent Systems And Control (ISCO)*, Coimbatore, p.1-3, jan. 2015. IEEE.
[Montgomery et al. 2008] MONTGOMERY, D. C.; JENNINGS, C. L.; KULAHCI, M.. *Introduction to Time Series Analysis and Forecasting.* Hoboken: John Wiley & Sons, Inc, 2008. 472 p. (Wiley Series in Probability and Statistics).
[Montgomery 2009] MONTGOMERY, Douglas C.. *Introdução ao Controle Estatístico da Qualidade*. 4. ed. São Paulo: Grupo Editora Nacional, Gen, 2009. 513 p.
[Montgomery et al. 2012] MONTGOMERY, D. C.; PECK, E. A.; VINING, G. G.. *Introduction to Linear Regression Analysis.* 5. ed. Hoboken: John Wiley & Sons, Inc, 2012. 672 p. (Wiley Series in Probability and Statistics).
[Montgomery 2013] MONTGOMERY, D. C.; *Design And Analysis of Experiments.* 8. ed. Hoboken: John Wiley & Sons, Inc, 2012. 752 p.
[Morettin e Toloi 2006] MORETTIN, P. A.; TOLOI, C. M. C.. *Análise de Séries Temporais.* 2. ed. São Paulo: Editora Edgard Blucher, 2006. 564 p.
[Mueen e Keogh 2016] MUEEN, A.; KEOGH, E.. Extracting Optimal Performance from Dynamic Time Warping. *Proceedings Of The 22nd ACM SIGKDD International Conference On Knowledge Discovery And Data Mining - KDD '16*, San Francisco, p.2129-2130, 2016. ACM Press.
[Ni e Sun. 2008] NI, Q.; SUN, W.; MA, S.. Memory Leak Detection in Sun Solaris OS. *2008 International Symposium On Computer Science And Computational Technology*, Shanghai, p.703-707, 2008. IEEE.
[Niennattrakul e Ratanamahatana 2007] NIENNATTRAKUL, V.; RATANAMAHATANA, C. A.. Inaccuracies of Shape Averaging Method Using Dynamic Time Warping for Time Series Data. *International Conference On Computational Science: ICCS 2007: Computational Science – ICCS 2007*, [s.i], p.513-520, 2007.
[NLUUG/SURFnet FTP Server Usage 2017] NLUUG/SURFnet FTP Server Usage. NLUUG/SURFnet FTP Server Usage. Disponível em: <http://ftp.nluug.nl/.statistics/>. Acesso em: 18 jul. 2017.
[Openhub 2017a] Openhub Inkscape. Openhub Web Site. Disponível em: <https://www.openhub.net/p/inkscape>. Acesso em: 18 jul. 2017.
[Openhub 2017b] Openhub Lighttpd. Openhub Web Site. Disponível em: <https://www.openhub.net/p/lighttpd>. Acesso em: 18 jul. 2017.
[Openhub 2017c] Openhub PostgreSQL Database Server. Openhub Web Site. Disponível em: <https://www.openhub.net/p/postgres>. Acesso em: 18 jul. 2017.
[Openhub 2017d] Openhub Squid Cache. Openhub Web Site. Disponível em: <https://www.openhub.net/p/squid>. Acesso em: 18 jul. 2017.
[Page 1954] PAGE, E. S.. Continuous Inspection Schemes. *Biometrika*, Oxford, v. 41, n. 1/2, p.100-115, jun. 1954. JSTOR.
[Petitjean et al 2011] PETITJEAN, F.; KETTERLIN, A.; GANÇARSKI, P.. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, New York, v. 44, n. 3, p.678-693, mar. 2011. Elsevier BV.
[PostgreSQL 2017a] PostgreSQL About. PostgreSQL Web Site. Disponível em: <https://www.postgreSQL.org/about/>. Acessado em Abril de 2017.
[PostgreSQL 2017b] PostgreSQL History. PostgreSQL Web Site. Disponível em: <https://www.postgreSQL.org/about/history/>. Acessado em Abril de 2017.
[PostgreSQL 2017c] BUG #9223: plPerlu result memory leak. PostgreSQL Web Site. Disponível em: <https://www.postgreSQL.org/message-

id/20140214163125.24620.66179@wrigleys.postgreSQL.org>. Acessado em Abril de 2017.
[Pressman e Maxim 2014] PRESSMAN, R. S.; MAXIM, B.. S*oftware Engineering: A Practitioner's Approach*. 8. ed. New York: Mcgraw-hill Science/engineering/math, 2014. 976 p.
[PyAutoGUI 2014] PyAutoGUI Introduction. PyAutoGUI Web Site. Disponível em: <https://pyautogui.readthedocs.io/en/latest/introduction.html>. Acessado em Abril de 2017.
[Rabiner et al. 1978] RABINER, L.; ROSENBERG, A.; LEVINSON, S.. Considerations in dynamic time warping algorithms for discrete word recognition. *IEEE Transactions On Acoustics, Speech, And Signal Processing*, [s.l.], v. 26, n. 6, p.575-582, dez. 1978. Institute of Electrical and Electronics Engineers (IEEE).
[RANGON.ORG 2017] True Random Number Service. RANDON.ORG Web Site. Disponível em: <https://www.random.org/>. Acessado em Abril de 2017.
[Ratanamahatana e Keogh 2004] RATANAMAHATANA, C. A.; KEOGH, E.. Everything you know about dynamic time warping is wrong. *Third Workshop On Mining Temporal And Sequential Data*, [s.i], p.1-11, 2004.
[Redmine Lighttpd 2017] mod_ssi Memory Leak. Lighttpd - light labs. Disponível em: <https://redmine.lighttpd.net/issues/1753>. Acessado em Abril de 2017.
[Sakoe e Chiba 1971] SAKOE, H.; CHIBA, S.. A Dynamic Programming Approach to Continuous Speech Recognition. *Proceedings Of The Seventh International Congress On Acoustic*s, Budapest, v. 3, p.65-69, 1971.
[Sakoe e Chiba 1978] SAKOE, H.; CHIBA, S.. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions On Acoustics, Speech, And Signal Processing*, [s.l.], v. 26, n. 1, p.43-49, fev. 1978. Institute of Electrical and Electronics Engineers (IEEE).
[Sisodia e Verma 2012] SISODIA, D. S.; VERMA, S.. Web usage pattern analysis through web logs: A review. *2012 Ninth International Conference On Computer Science And Software Engineering (JCSSE)*, [s.l.], p.49-53, maio 2012. IEEE.
[Shabtai et al. 2012] SHABTA, A.; ELOVICI, Y.; ROKACH, L.. *A Survey of Data Leakage Detection and Prevention Solutions*. [s.I.]: Springerbriefs In Computer Science, 2012. 92 p.
[Shao et al. 2005] SHAO, M.; AILAMAKI, A.; FALSAFI, B.. DBmbench: fast and accurate database workload representation on modern microarchitecture. *CASCON '05 Proceedings Of The 2005 Conference Of The Centre For Advanced Studies On Collaborative Research*, Ontario, p.254-267, 17 out. 2005.
[Sharma e Gupta 2013] SHARMA, A. K.; GUPTA, P.C. Analysis of Web Server Log Files to Increase The Effectiviness of the Website Using Web Mining Tool. *International Journal Of Advanced Computer And Mathematical Sciences*, [s.I], v. 4, n. 1, p.1-8, 2013.
[Shereshevsky et al. 2003] SHERESHEVSKY, M. et al. Software aging and multifractality of memory resources. *2003 International Conference On Dependable Systems And Networks*, 2003. Proceedings., San Francisco, p.721-730, 2003. IEEE.
[Siegmund et al. 2015] SIEGMUND, J.; SIEGMUND, N.; APEL, S.. Views on Internal and External Validity in Empirical Software Engineering. *2015 IEEE/ACM 37th IEEE International Conference On Software Engineering*, Florence, p.9-19, maio 2015. IEEE.
[Silva e Batista 2016] SILVA, D. F.; BATISTA, G. E. A. P. A.. Speeding Up All-Pairwise Dynamic Time Warping Matrix Calculation. *Proceedings Of The 2016 Siam*

International Conference On Data Mining, [s.l.], p.837-845, 30 jun. 2016. Society for Industrial and Applied Mathematics.
[Souza et al. 2008] SOUZA, L. M. de et al. Eficiência dos gráficos de controle xbarra, ewma e cusum. *Revista Produção e Engenharia*, [s.l.], v. 1, n. 1, p.81-94, 24 jan. 2016. Fundação Centro Tecnológico de Juiz de Fora (FCT).
[Squid 2017a] The Squid Software Foundation. Squid Web Site. Disponível em: <http://www.squid-*cache*.org/>. Acessado em Abril de 2017.
[Squid 2017b] Squid-based Products. Squid Web Site. Disponível em: <http://www.squid-*cache*.org/Support/products.html>. Acessado em Abril de 2017.
[Statistics How To 2013] Statistics for the rest of us. Statistics How To. Disponível em: <http://www.statisticshowto.com/mean-squared-error/>. Acessado em Abril de 2017.
[Thompson et al. 1997] THOMPSON, K.; MILLER, G. J.; WILDER, R.. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, [s.l.], v. 11, n. 6, p.10-23, 1997. Institute of Electrical and Electronics Engineers (IEEE).
[TPC 2017a] *TPC Homepage V5*. 2017. TPC *Web* Site. Disponível em: <http://www.tpc.org/>. Acesso em: 18 jul. 2017.
[TPC 2017b] TPC-H. TPC Web Site. Disponível em: <http://www.tpc.org/tpch/>. Acessado em Abril de 2017.
[TPC 2017c] TPC-C. TPC Web Site. Disponível em: <http://www.tpc.org/tpcc/>. Acessado em Abril de 2017.
[Trivedi et al. 2011] TRIVEDI, K. S. et al. Recovery from Failures Due to Mandelbugs in IT Systems. *2011 IEEE 17th Pacific Rim International Symposium On Dependable Computing*, Pasadena, p.224-233, dez. 2011. IEEE.
[Vahalia 1995] VAHALIA, U.. *UNIX Internals: The New Frontiers*. New Jersey: Pearson Prentice Hall, 1995. 601 p.
[Valentim et al. 2016] VALENTIM, N. A.; MACEDO, A.; MATIAS, R.. A Systematic Mapping Review of the First 20 Years of Software Aging and Rejuvenation Research. *2016 IEEE International Symposium On Software Reliability Engineering Workshops (ISSREW)*, Ottawa, p.57-63, out. 2016. IEEE.
[Wasserman 1995] WASSERMAN, G. S.. An adaptation of the EWMA chart for short run SPC. *International Journal Of Production Research*, [s.l.], v. 33, n. 10, p.2821-2833, out. 1995. Informa UK Limited.
[Webartz 2017] Pacote com mais de 1000 Ícones PNG para Download. Webartz *Web* Site. Disponível em: <http://www.*web*artz.com.br/design/design-e-inspiracoes/pacote-com-mais-de-1000-icones-png-para-*download*/>. Acessado em Abril de 2017.
[WebTechster 2017] Lighttpd Web Server Usage. (2017). WebTechster *Web* Site. Disponível em: <http://www.WebTechster.com/trends/*web*-server/lighttpd.html>. Acessado em Abril de 2017.
[Weisstein 2017] MathWorld - Wolfram Web Resource - Created, developed, and nurtured by Eric Weisstein at Wolfram Research. Disponível em: <http://mathworld.wolfram.com/StatisticalMedian.html> Acessado em 20/02/2017.
[Wetherill e Brown 1991] WETHERILL, G. B.; BROWN, D. W.. *Statistical Process Control*: Theory and Practice. 3. ed. London: Chapman And Hall/CRC, 1991. 416 p. (Chapman & Hall/CRC Texts in Statistical Science).

[Wikimedia Blog 2014] How we made editing Wikipedia twice as fast. Wikimedia Blog Web Site. Disponível em: <https://blog.wikimedia.org/2014/12/29/how-we-made-editing-wikipedia-twice-as-fast/>. Acessado em Abril de 2017.
[Wikitech 2017] Analytics/AQS/Pageviews. Wikitech Web Site. Disponível em: <https://wikitech.wikimedia.org/wiki/Analytics/PageviewAPI>. Acessado em Abril de 2017.
[Williams et al. 2005] WILLIAMS, A. et al. Web Workload Characterization: Ten Years Later. *Web Content Delivery: Web Information Systems Engineering and Internet Technologies Book Series*, [s.i], v. 2, n. 1, p.3-21, 2005.
[Woodall 1985] WOODALL, W. H.. The Statistical Design of Quality Control Charts. *The Statistician*, [s.l.], v. 34, n. 2, p.155-160, 1985.
[Yao et al. 2005] YAO, Q.; AN, A.; HUANG, X.. Finding and Analyzing Database User Sessions. *Database Systems For Advanced Applications*, Beijing, p.851-862, 2005.
[Yu et al. 1992] YU, P. S. et al. On workload characterization of relational database environments. *IEEE Transactions On Software Engineering*, New Jersey, v. 18, n. 4, p.347-355, abr. 1992. Institute of Electrical and Electronics Engineers (IEEE).
[Yuksel et al. 2000] YUKSEL, M. et al. Workload Generation for ns Simulations of Wide Area Networks and the Internet. *Communication Networks And Distributed Systems Modeling And Simulation Conference*, [s.i], p.1-16, 2000.
[Zivot e Wang 2006] ZIVOT, E.; WANG, J.. *Modeling Financial Time Series with S-PLUS®*. 2. ed. New York: Springer, 2006. 998 p.
[Zhu e Shasha 2003] ZHU, Y.; SHASHA, D.. Warping indexes with envelope transforms for query by humming. *Proceedings Of The 2003 ACM Sigmod International Conference On On Management Of Data - SIGMOD '03*, San Diego, p.181-192, 2003. ACM Press.

APÊNDICE

A.1 Gráficos de divergência

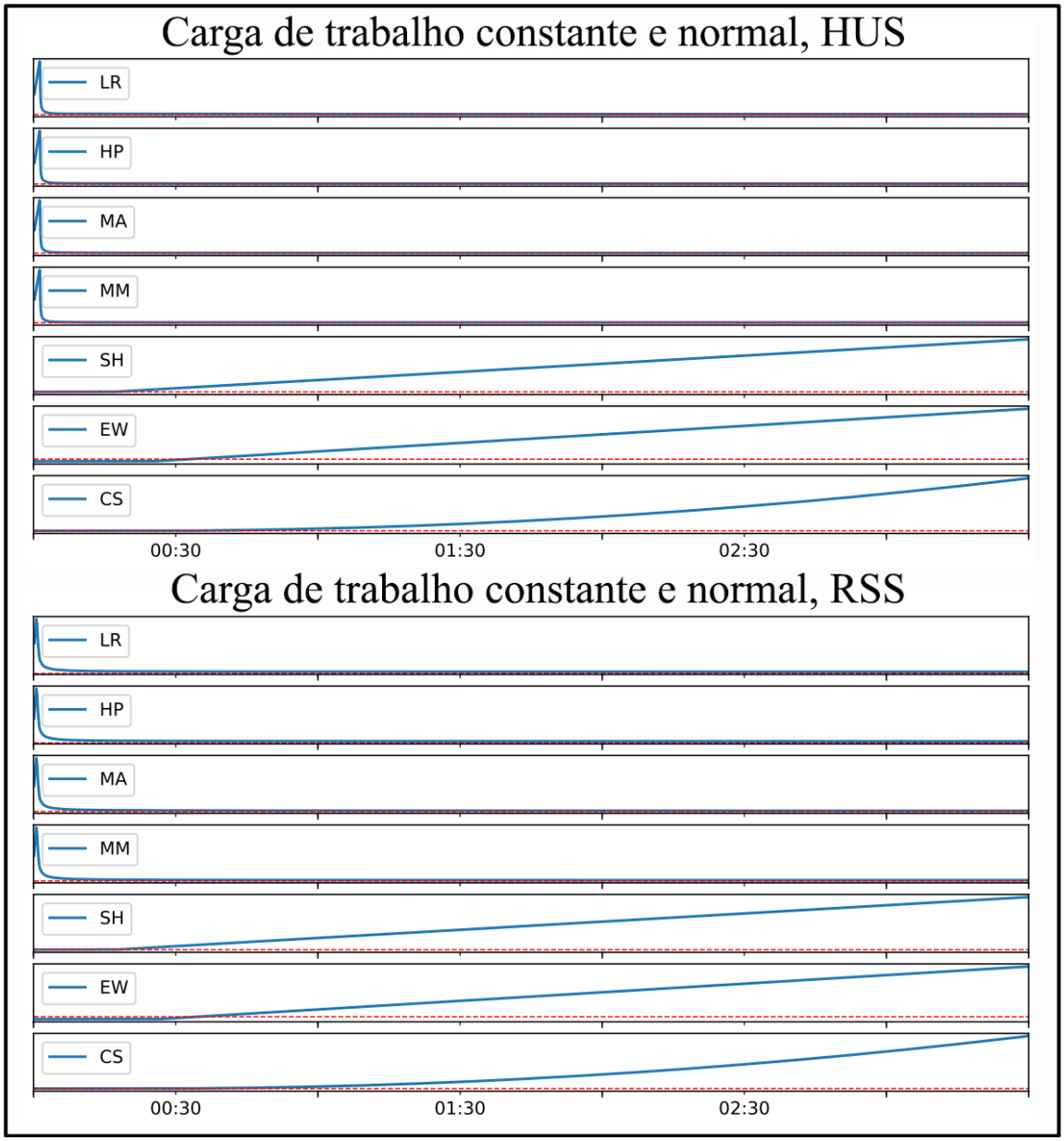


Figura 1.A1. Inkscape: gráficos de divergência para o cenário de carga de trabalho constante e normal.

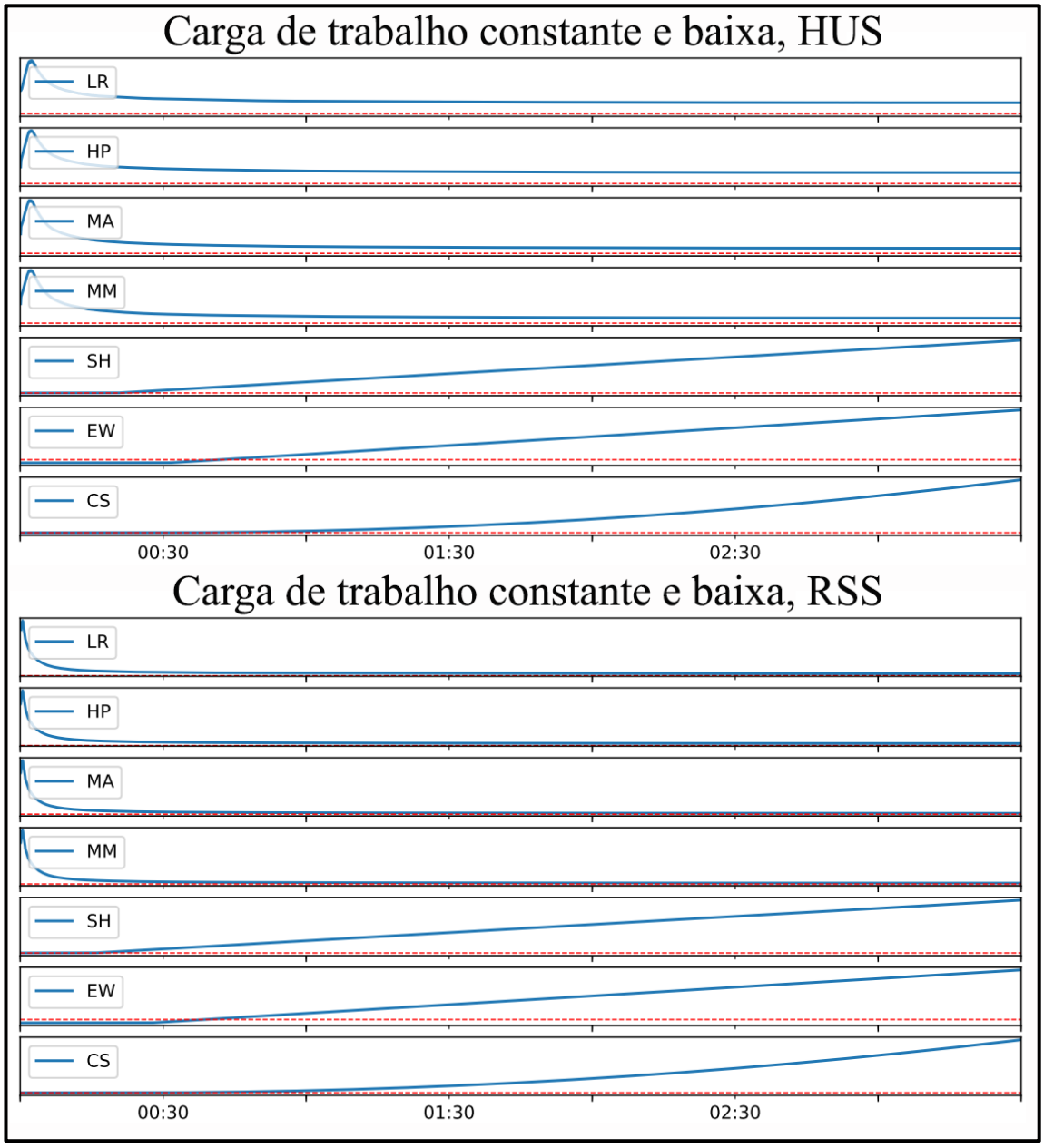


Figura 2.A1. Inkscape: gráficos de divergência para o cenário de carga de trabalho constante e baixa.

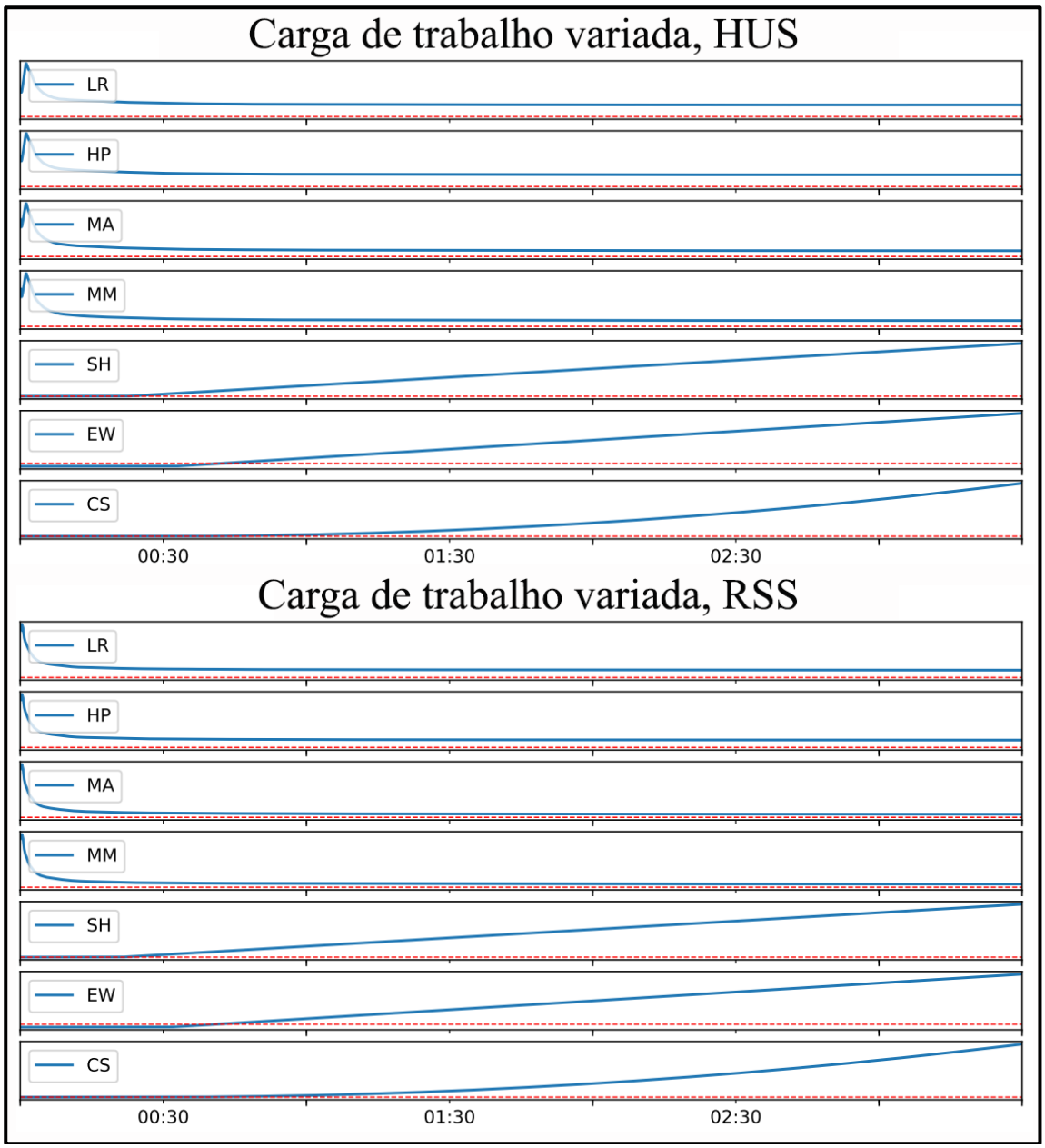


Figura 3.A1. Inkscape: gráficos de divergência para o cenário de carga de trabalho variada.

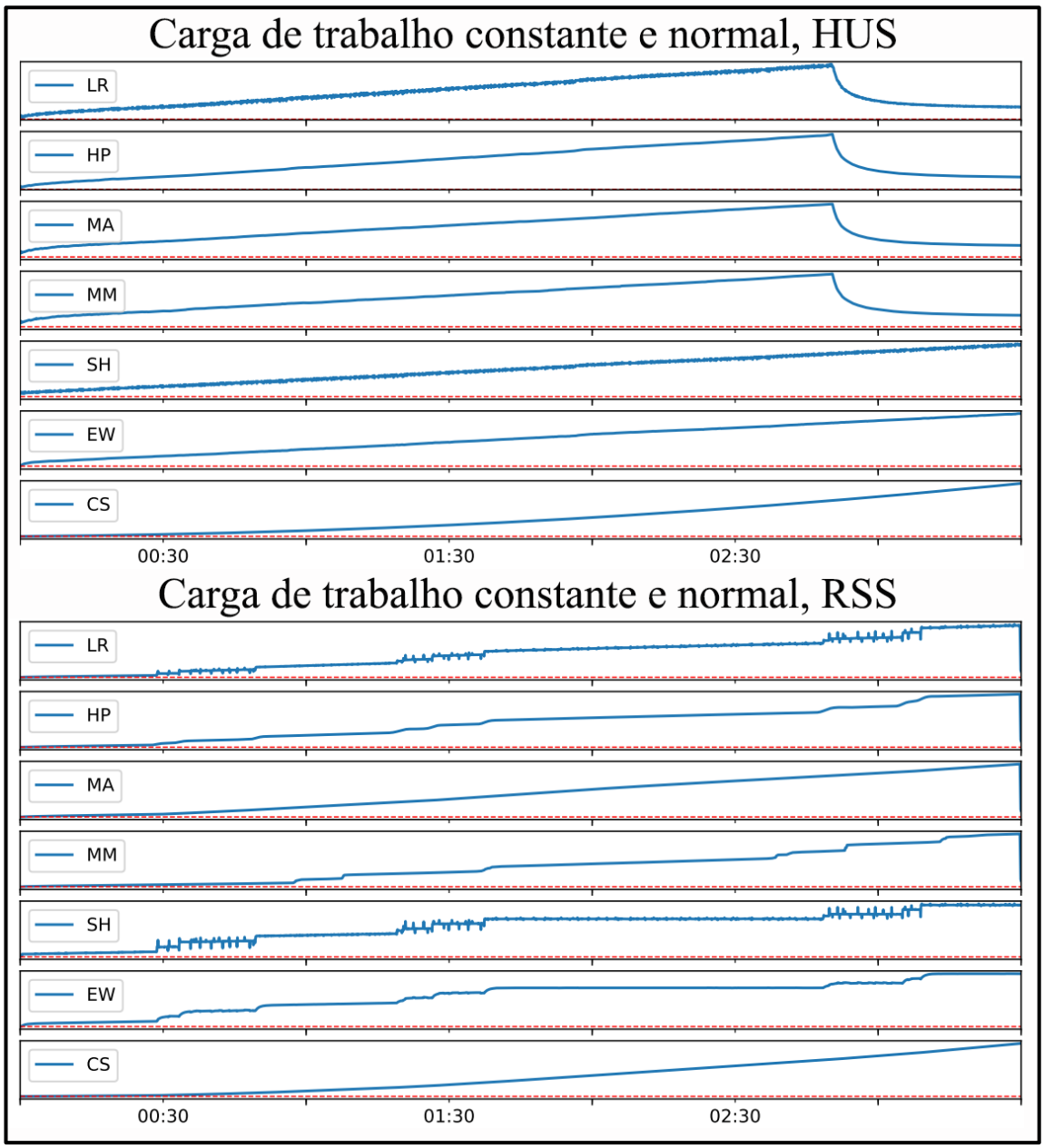


Figura 4.A1. Lighttpd: gráficos de divergência para o cenário de carga de trabalho constante e normal.

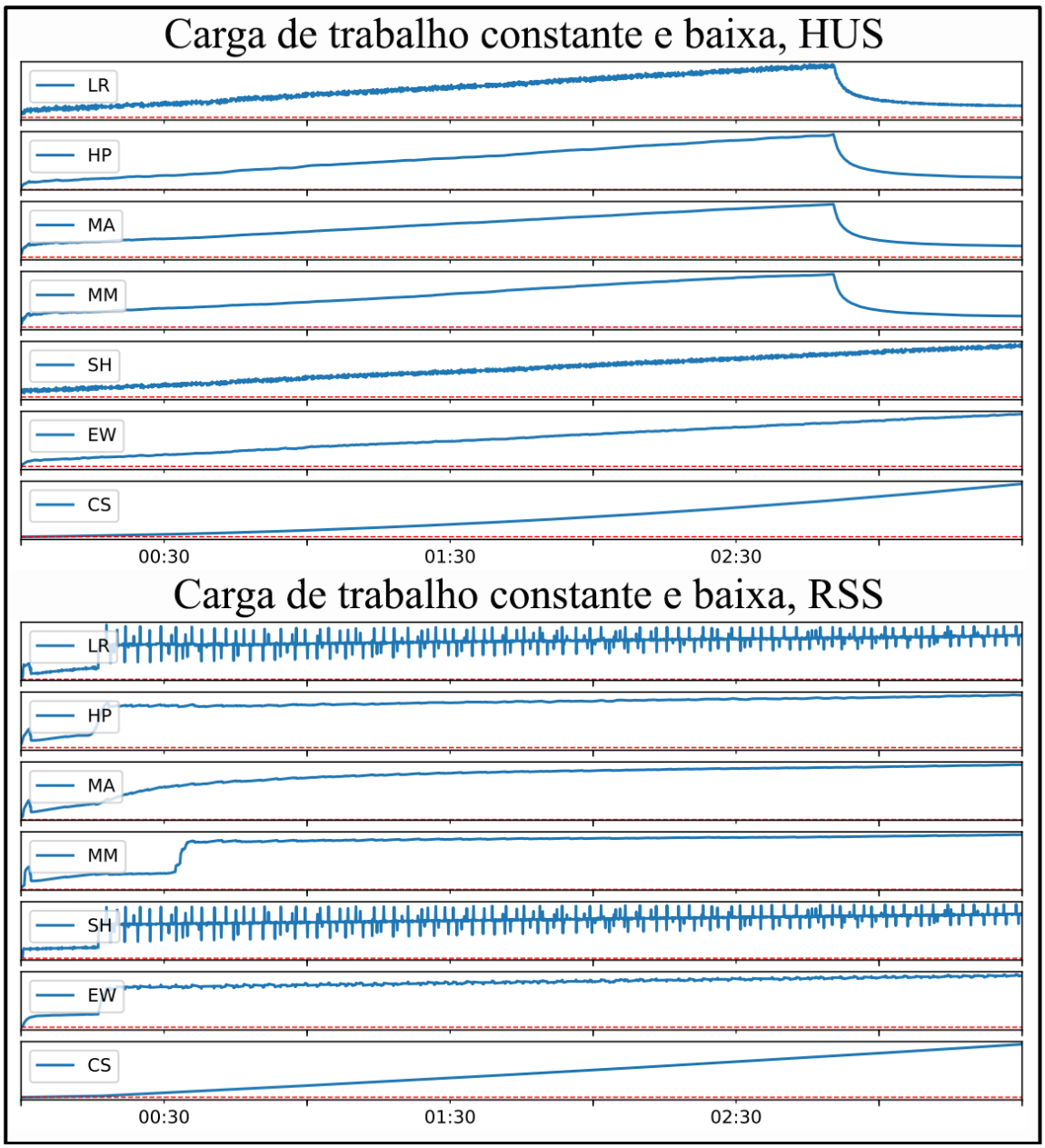


Figura 5.A1. Lighttpd: gráficos de divergência para o cenário de carga de trabalho constante e baixa.

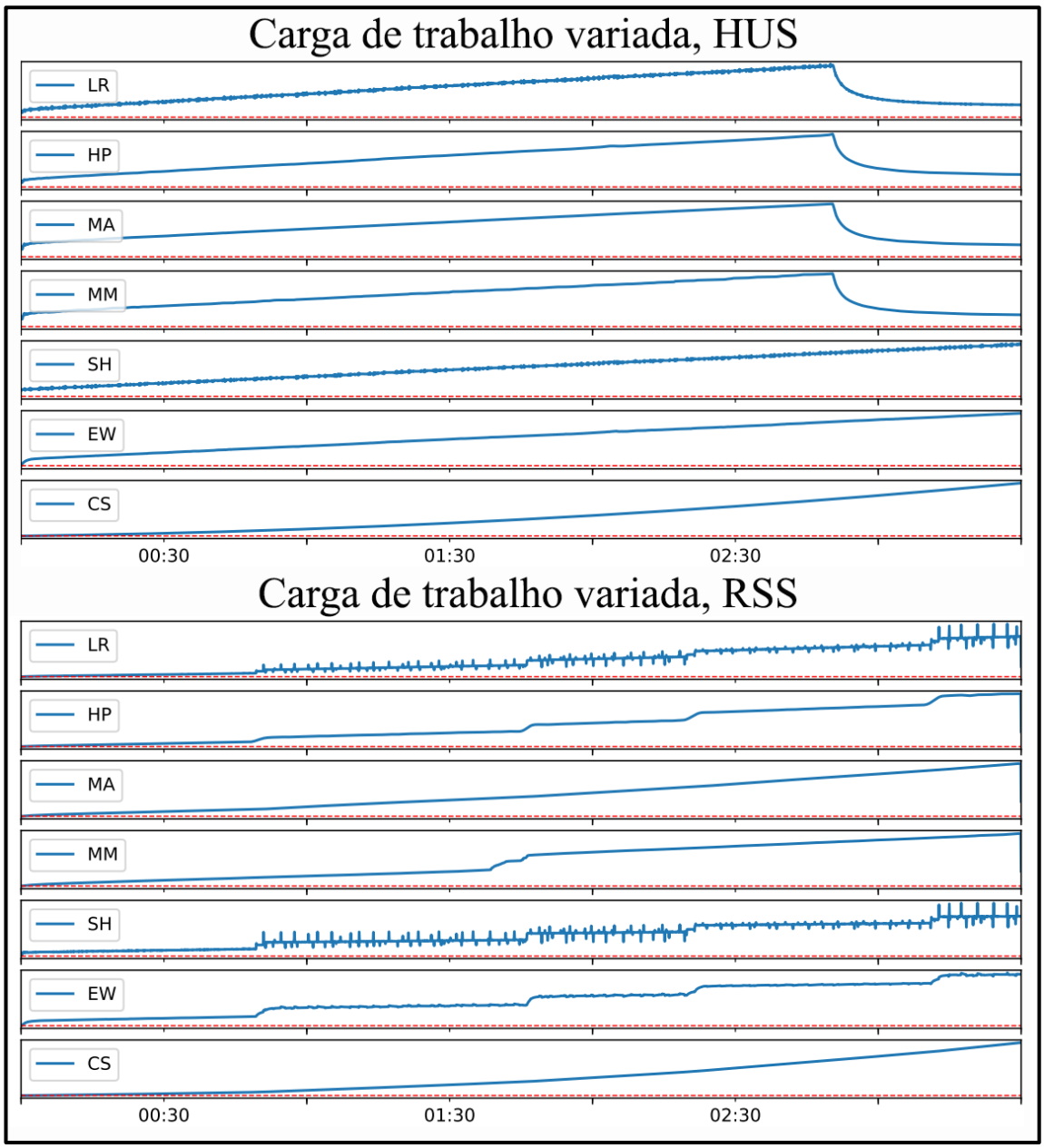


Figura 6.A1. Lighttpd: gráficos de divergência para o cenário de carga de trabalho variada.

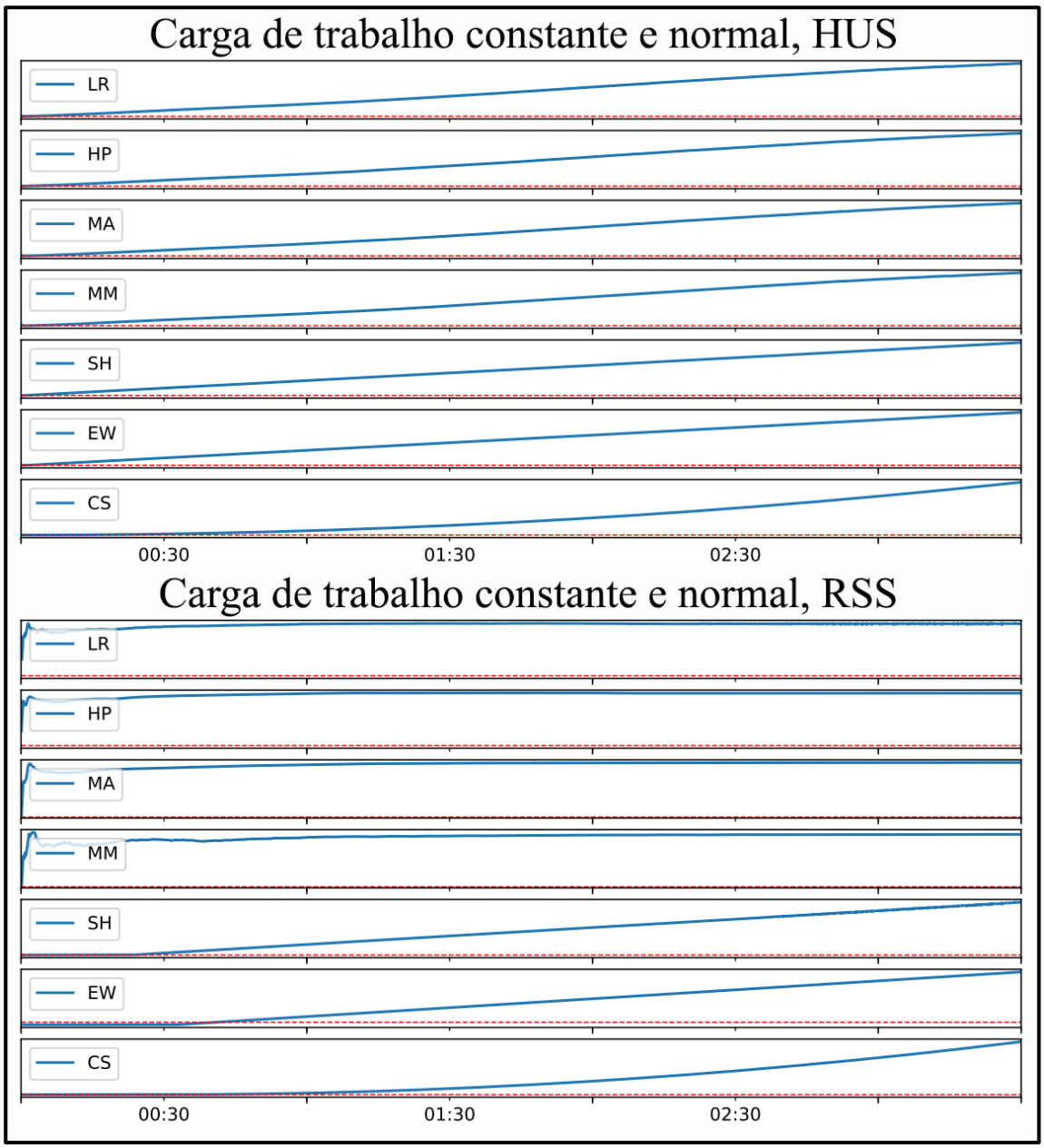


Figura 7.A1. PostgreSQL: gráficos de divergência para o cenário de carga de trabalho constante e normal.

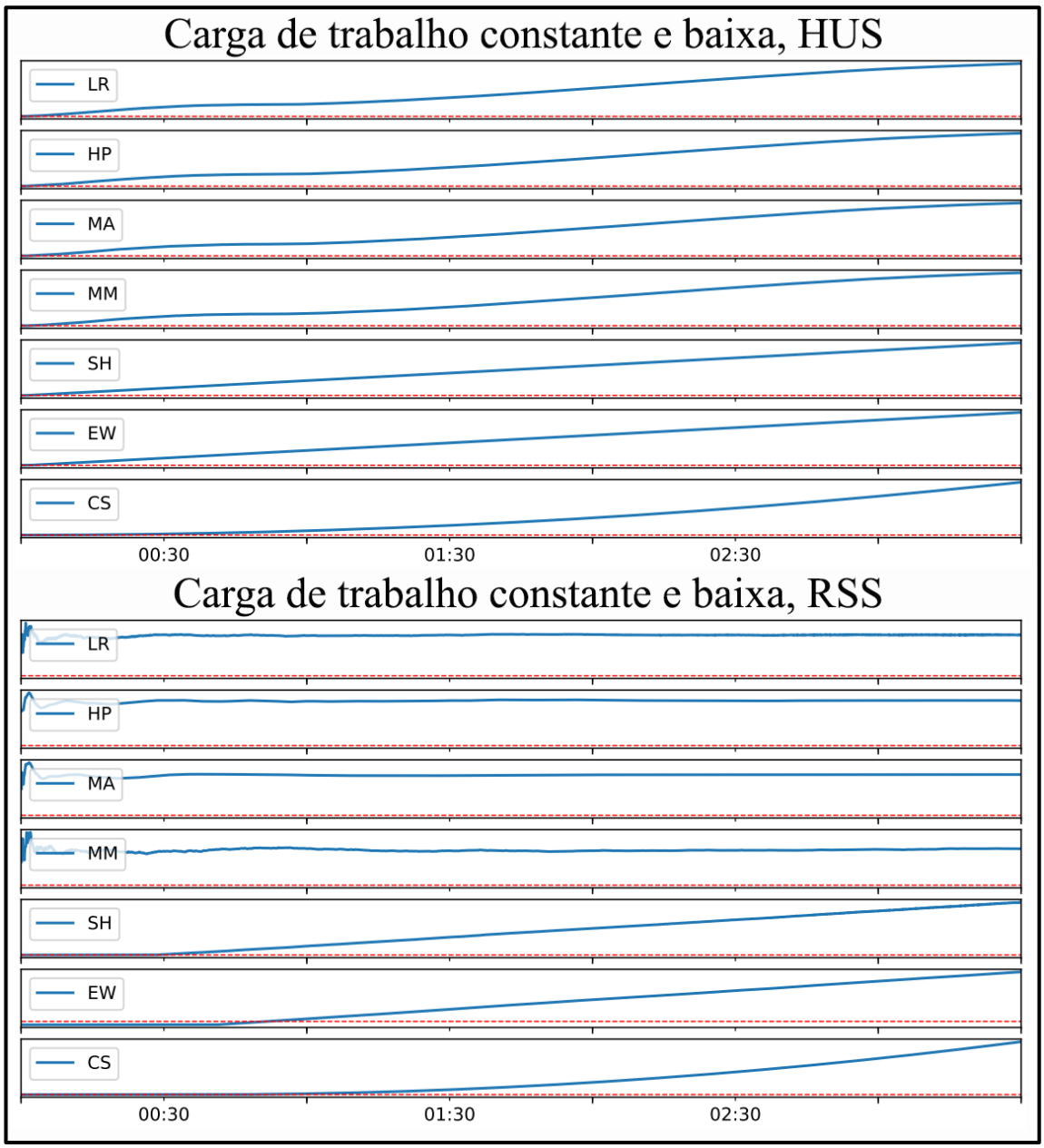


Figura 8.A1. PostgreSQL: gráficos de divergência para o cenário de carga de trabalho constante e baixa.

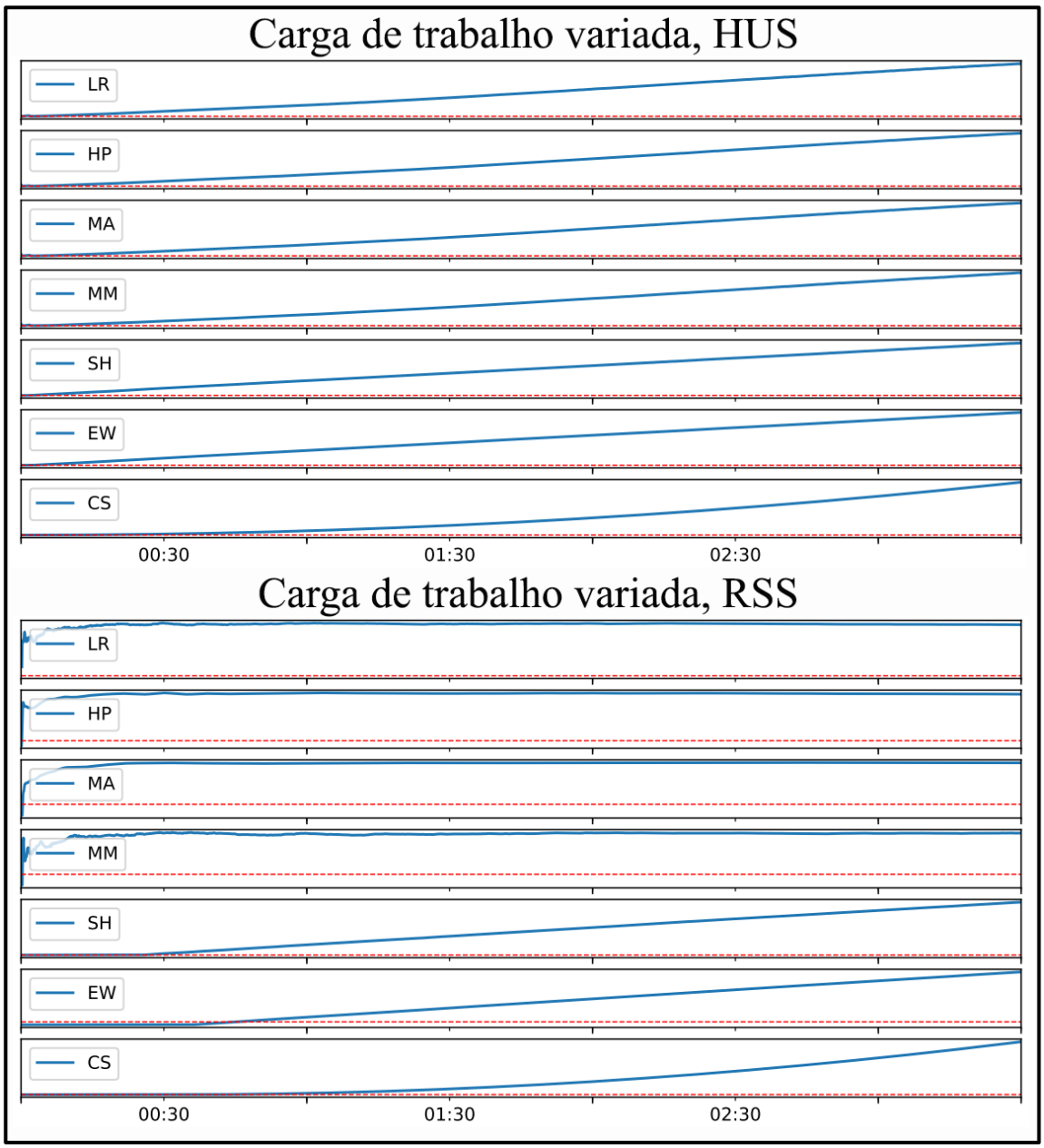


Figura 9.A1. PostgreSQL: gráficos de divergência para o cenário de carga de trabalho variada.

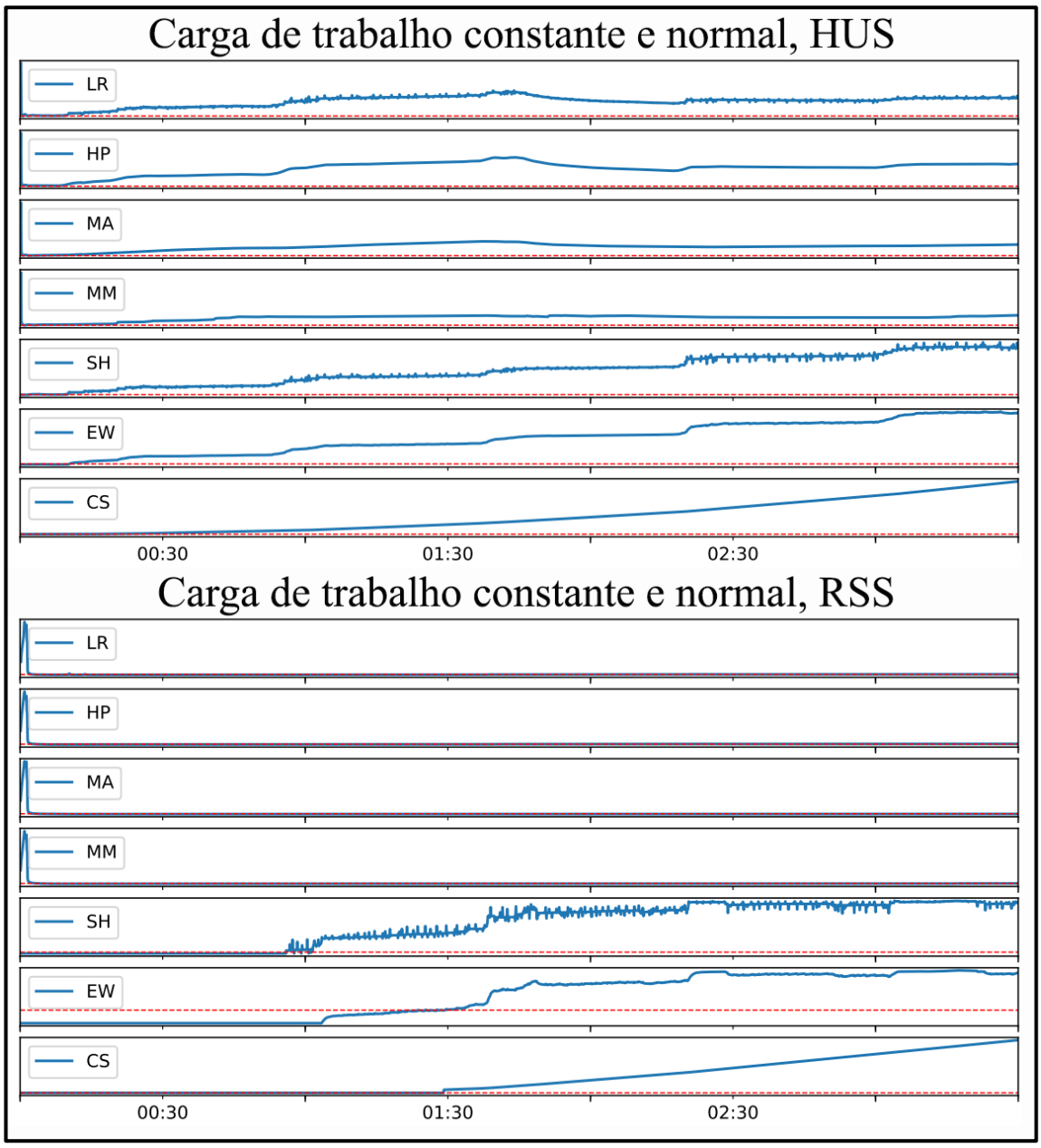


Figura 10.A1. Squid: gráficos de divergência para o cenário de carga de trabalho constante e normal.

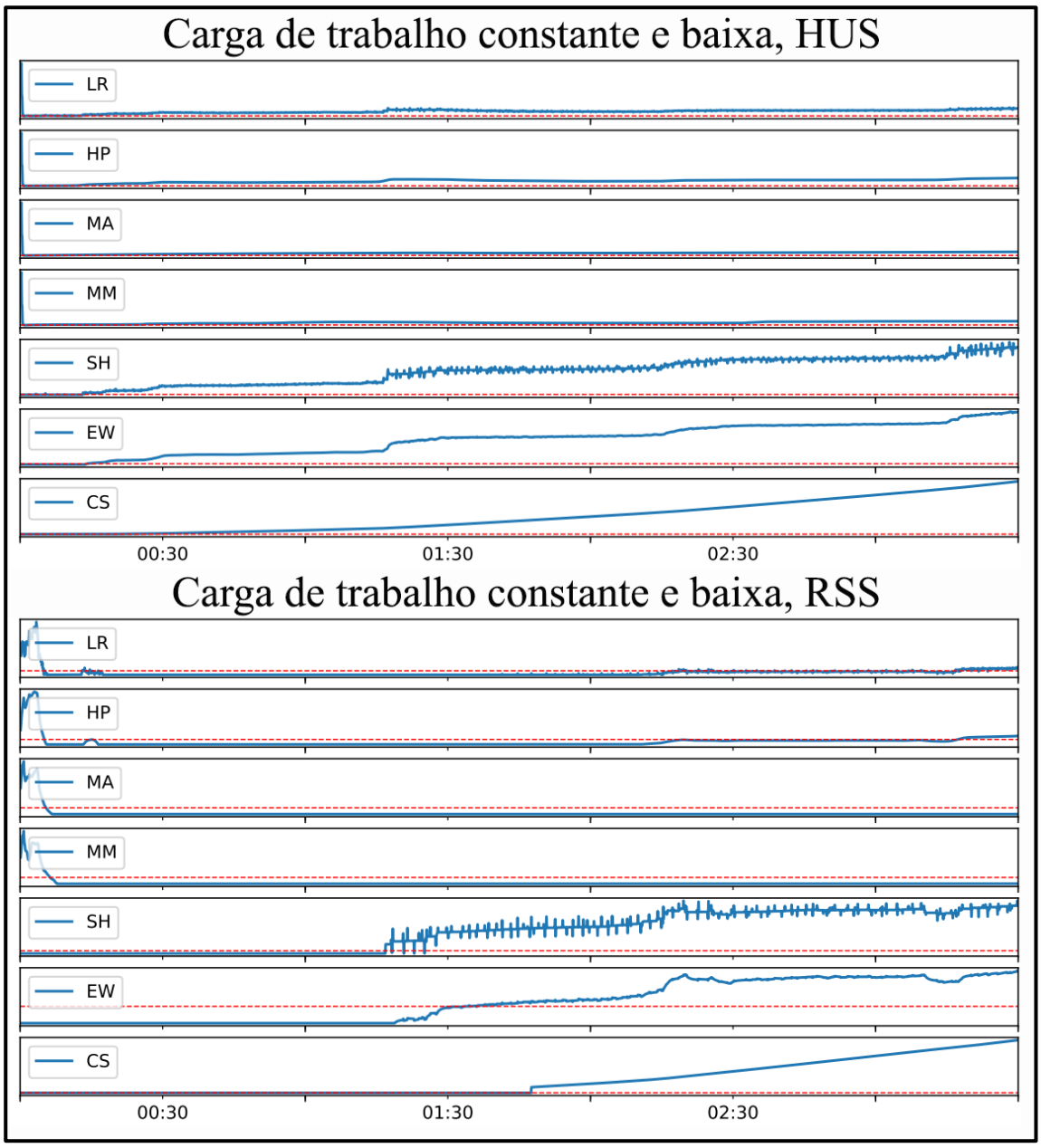


Figura 11.A1. Squid: gráficos de divergência para o cenário de carga de trabalho constante e baixa.

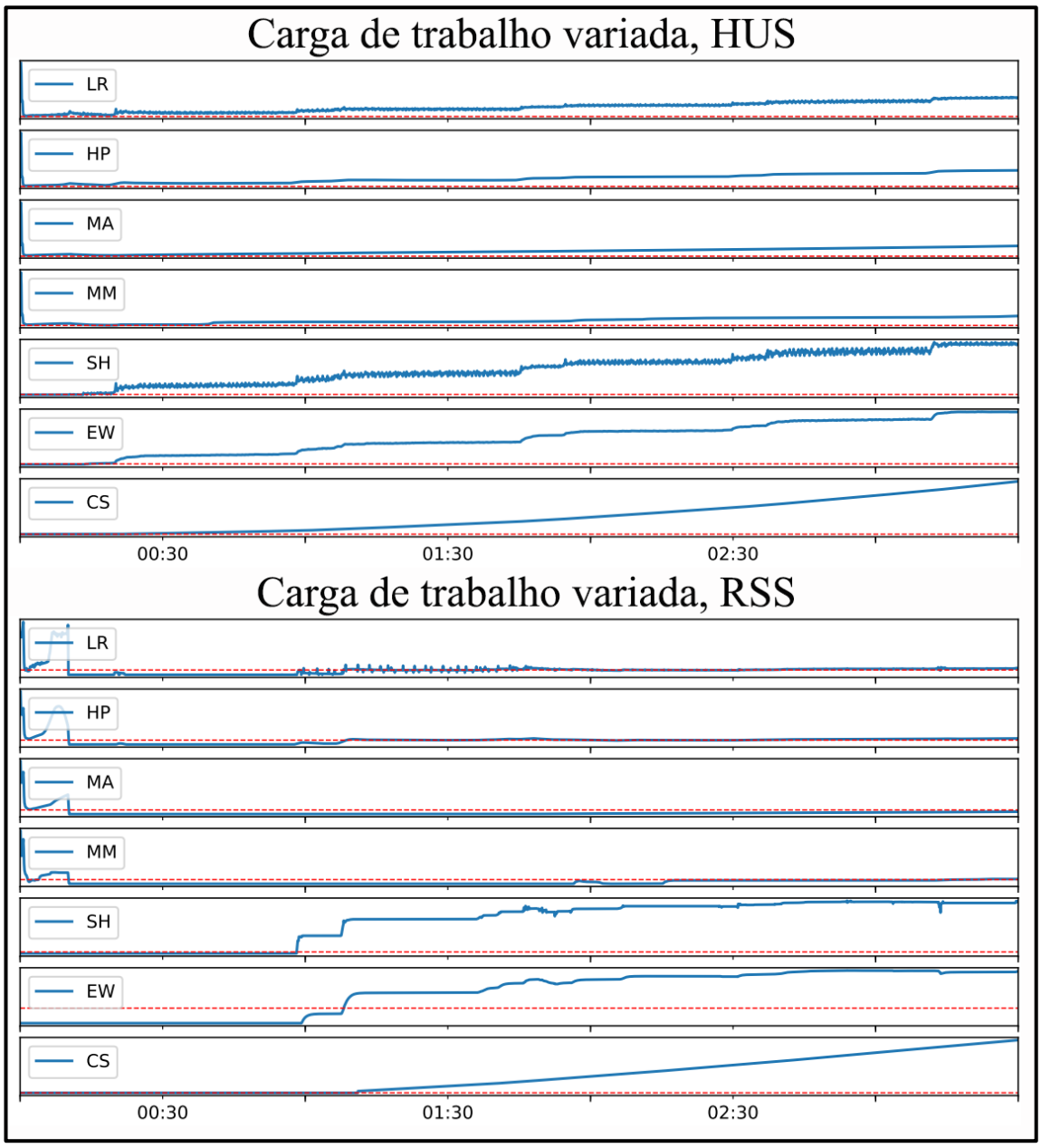


Figura 12.A1. Squid: gráficos de divergência para o cenário de carga de trabalho variada.

A.2 Gráficos das séries temporais obtidas das replicações

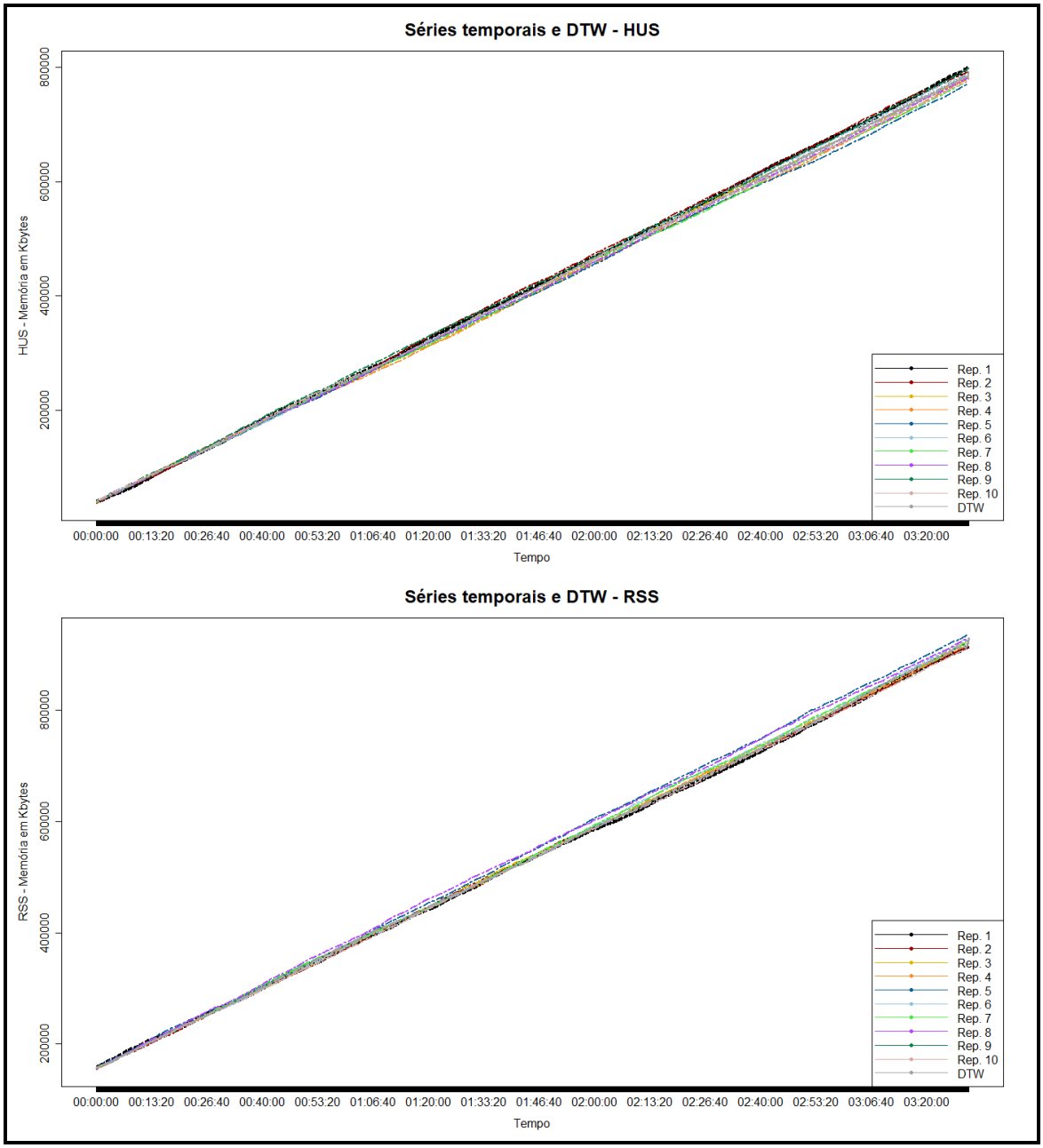


Figura 1.A2. Inkscape: cenário de carga constante e alta, versão base – séries temporais e DTW.

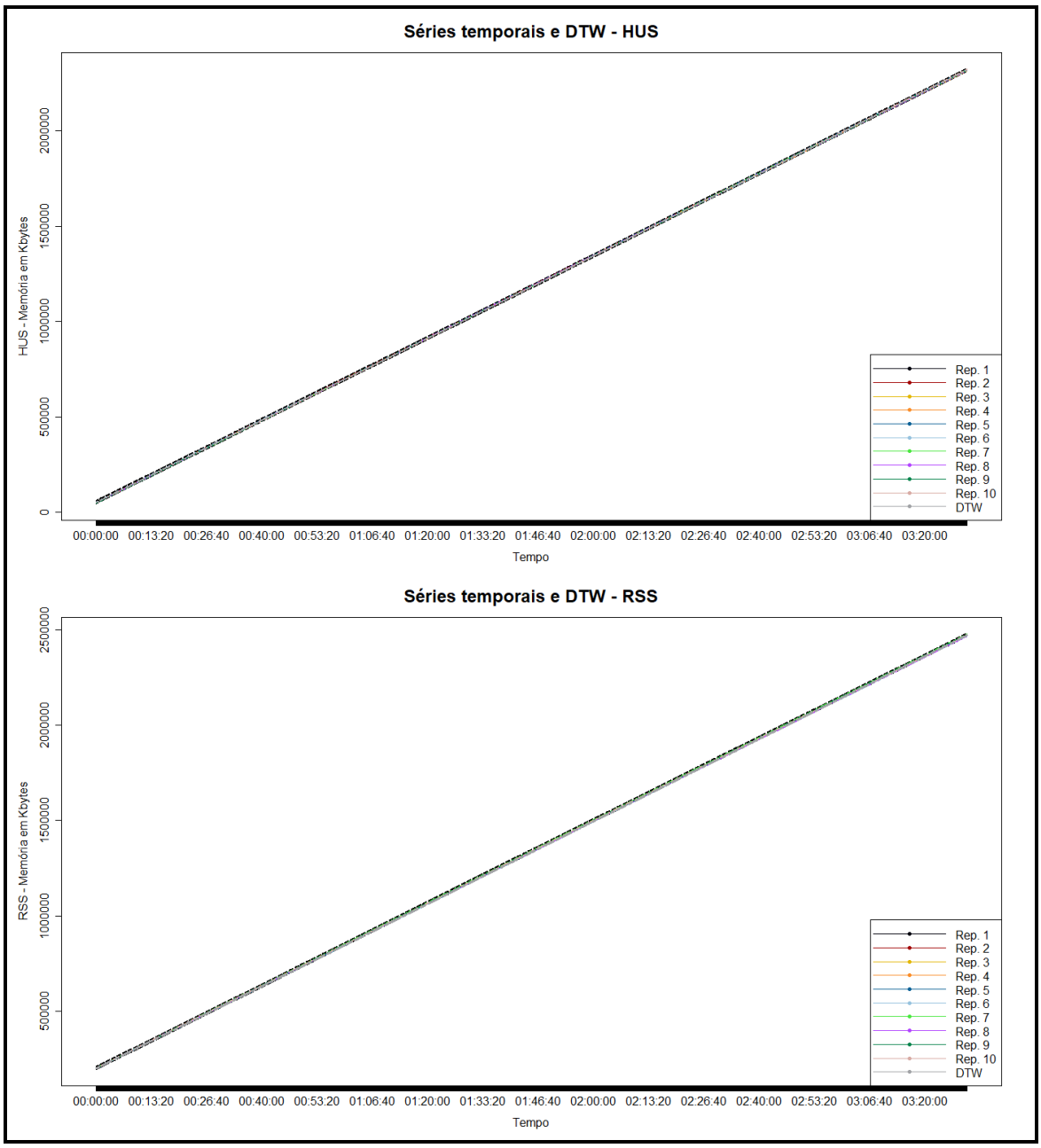


Figura 2.A2. Inkscape: cenário de carga constante e normal, versão alvo – séries temporais e DTW.

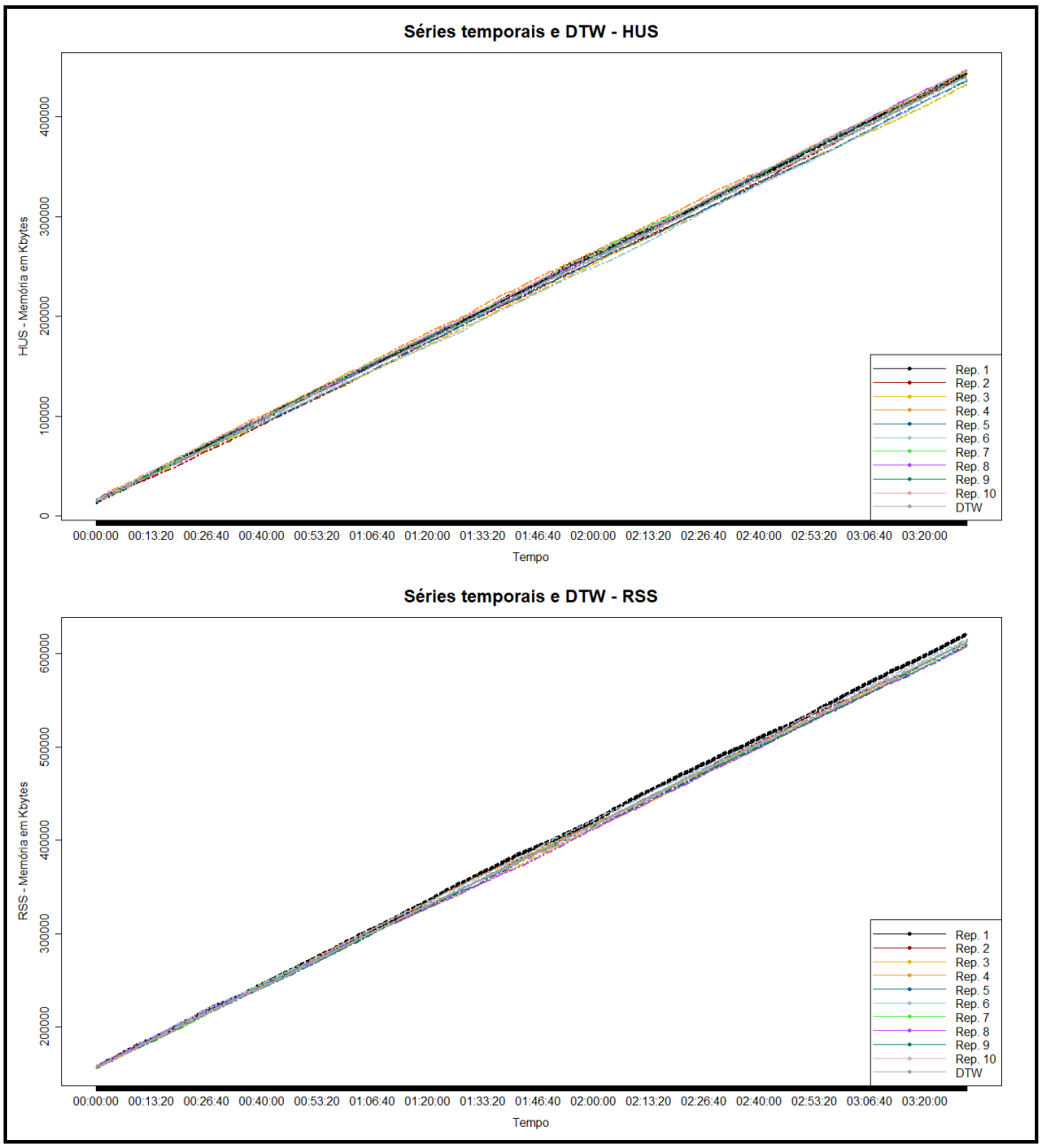


Figura 3.A2. Inkscape: cenário de carga constante e normal, versão base – séries temporais e DTW.

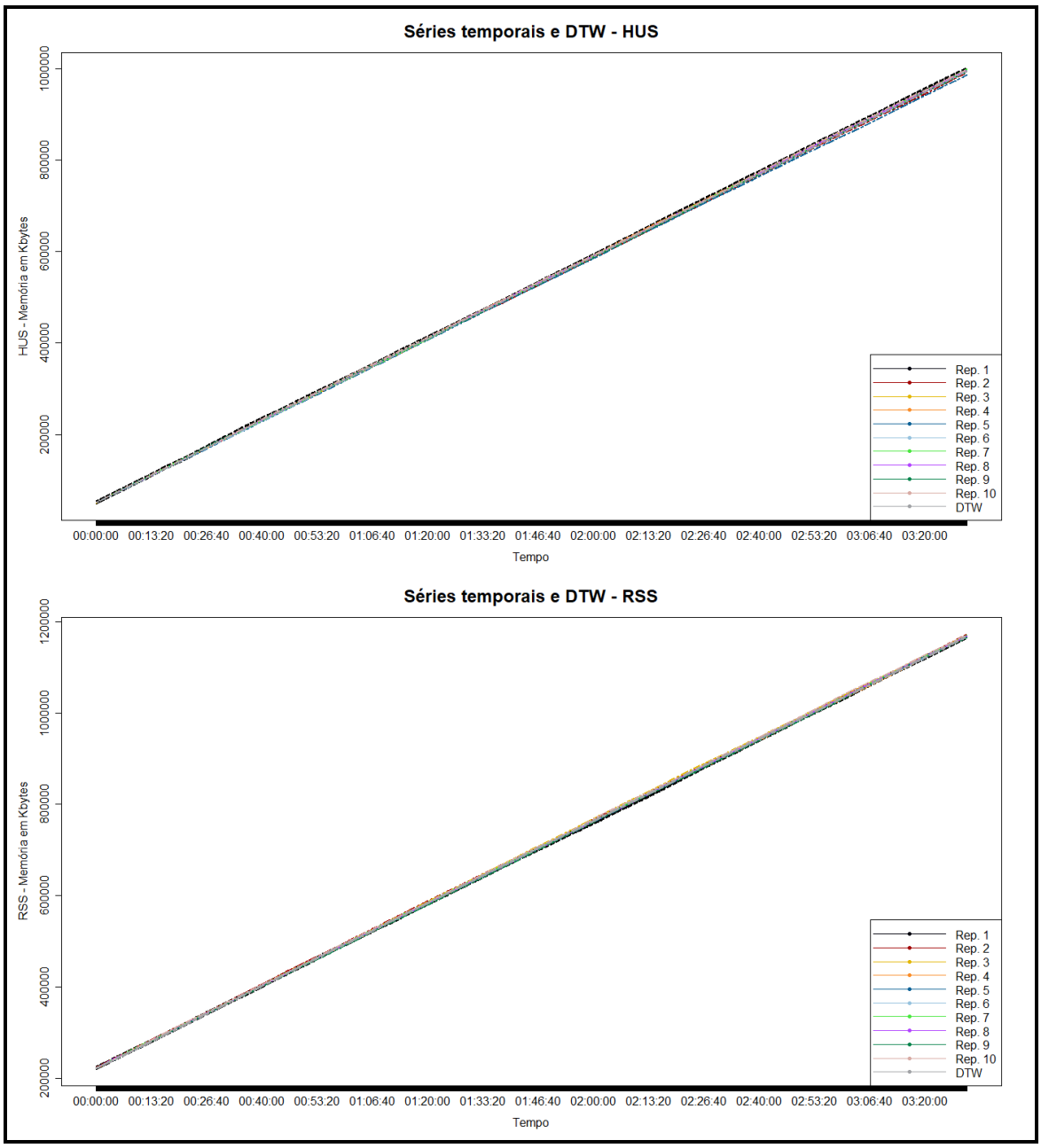


Figura 4.A2. Inkscape: cenário de carga constante e baixa, versão alvo – séries temporais e DTW.

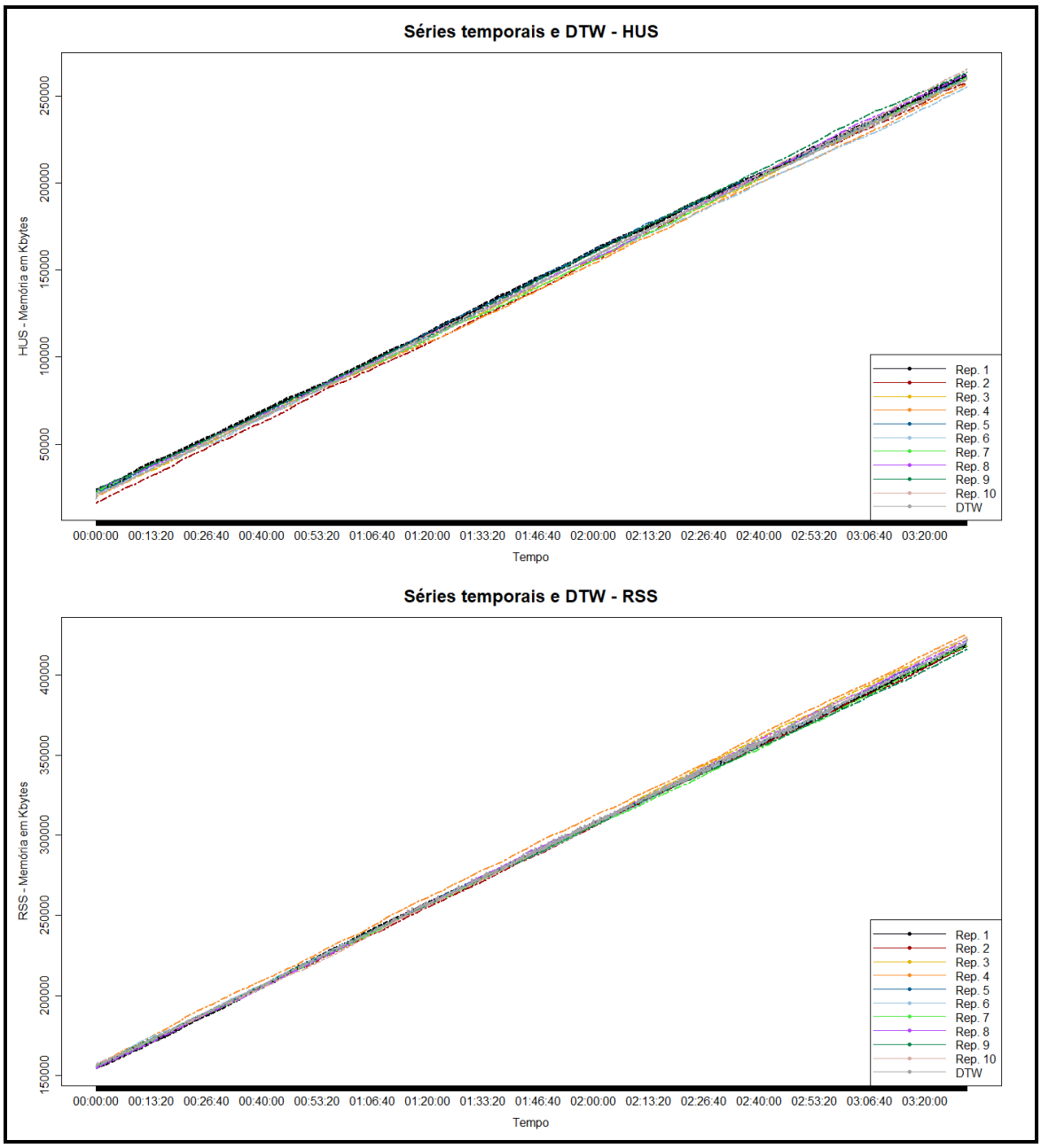


Figura 5.A2. Inkscape: cenário de carga constante e baixa, versão base – séries temporais e DTW.

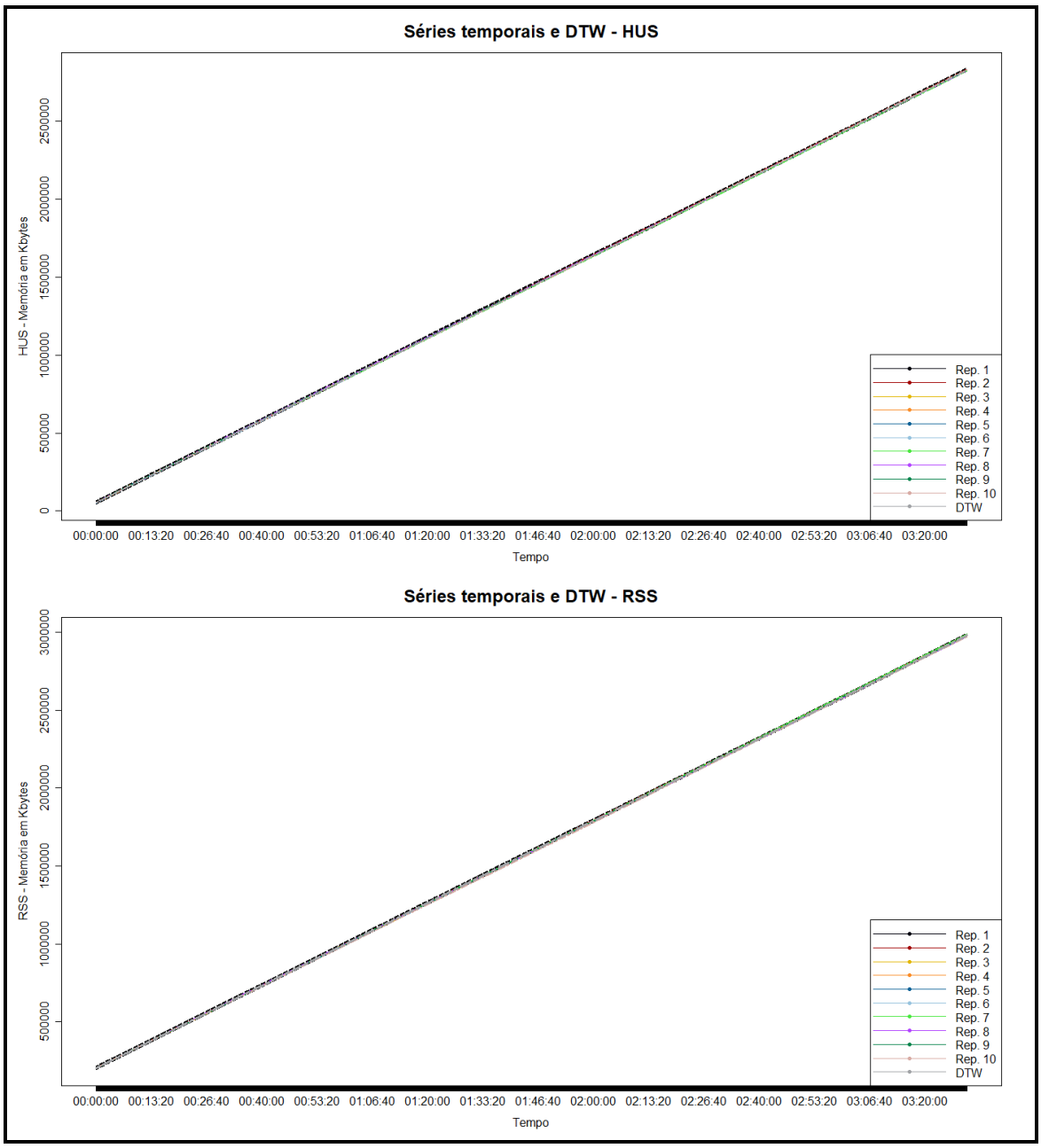


Figura 6.A2. Inkscape: cenário de carga variada, versão alvo – séries temporais e DTW.

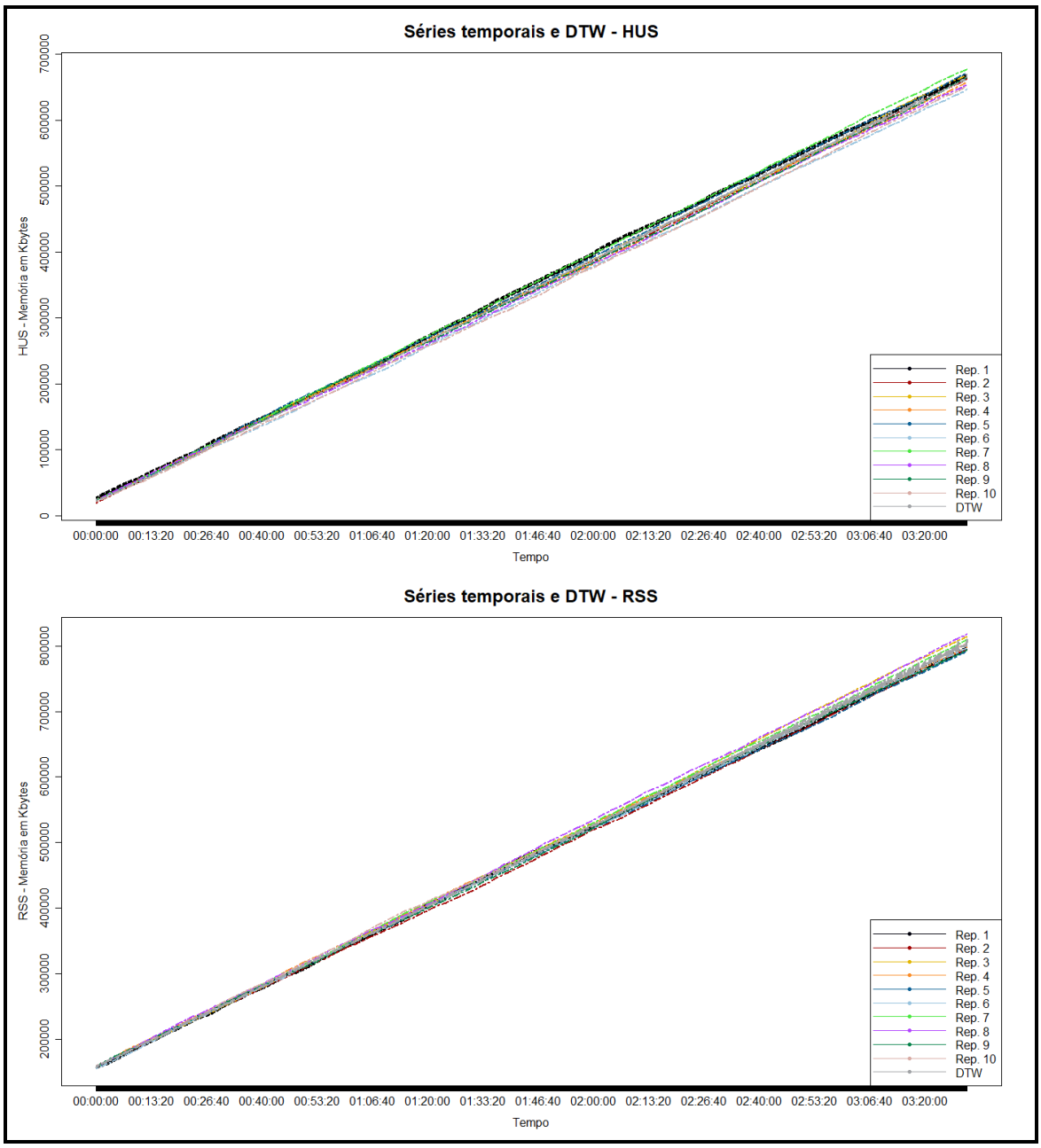


Figura 7.A2. Inkscape: cenário de carga variada , versão base – séries temporais e DTW.

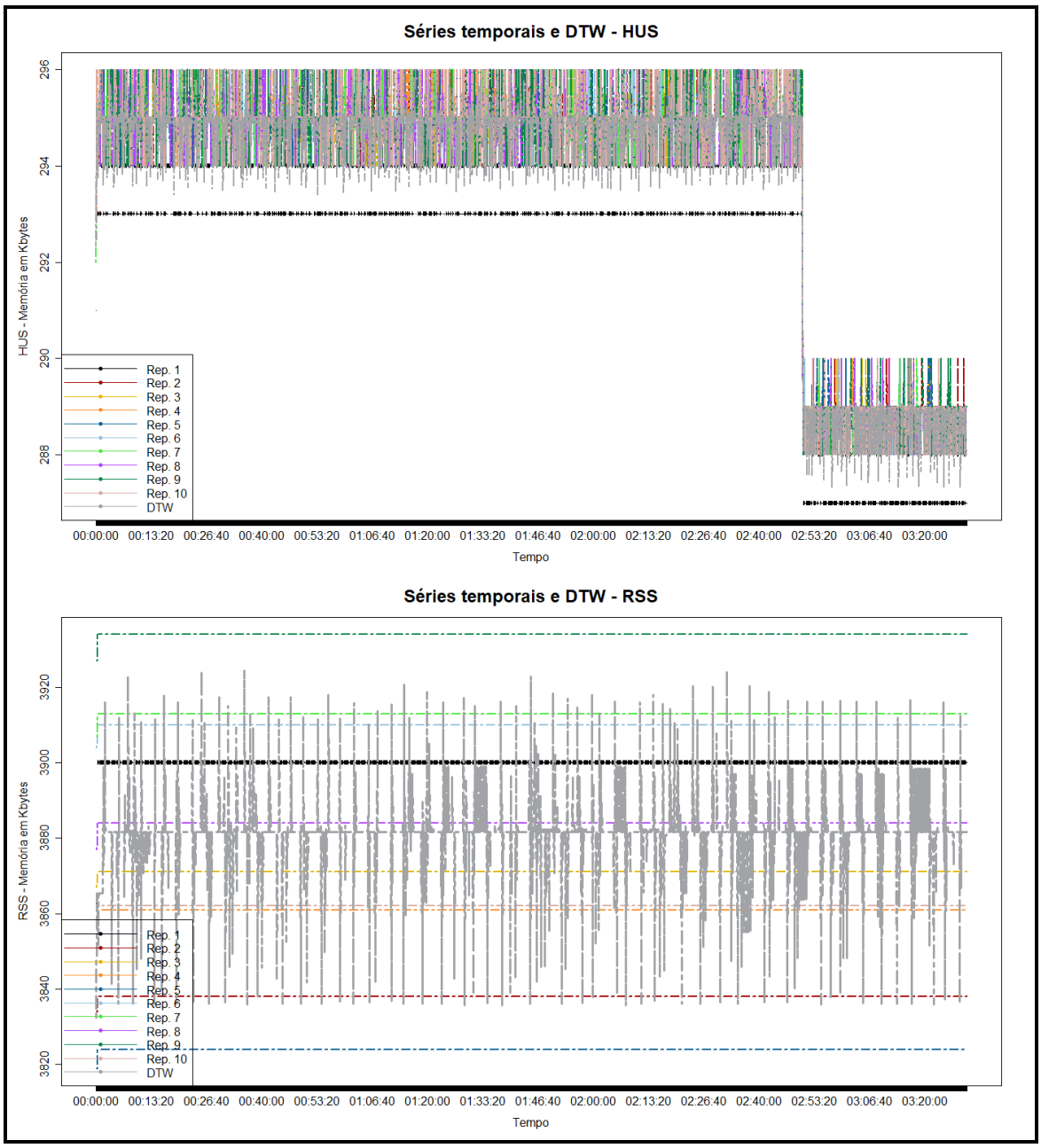


Figura 8.A2. Lighttpd: cenário de carga constante e alta, versão base – séries temporais e DTW.

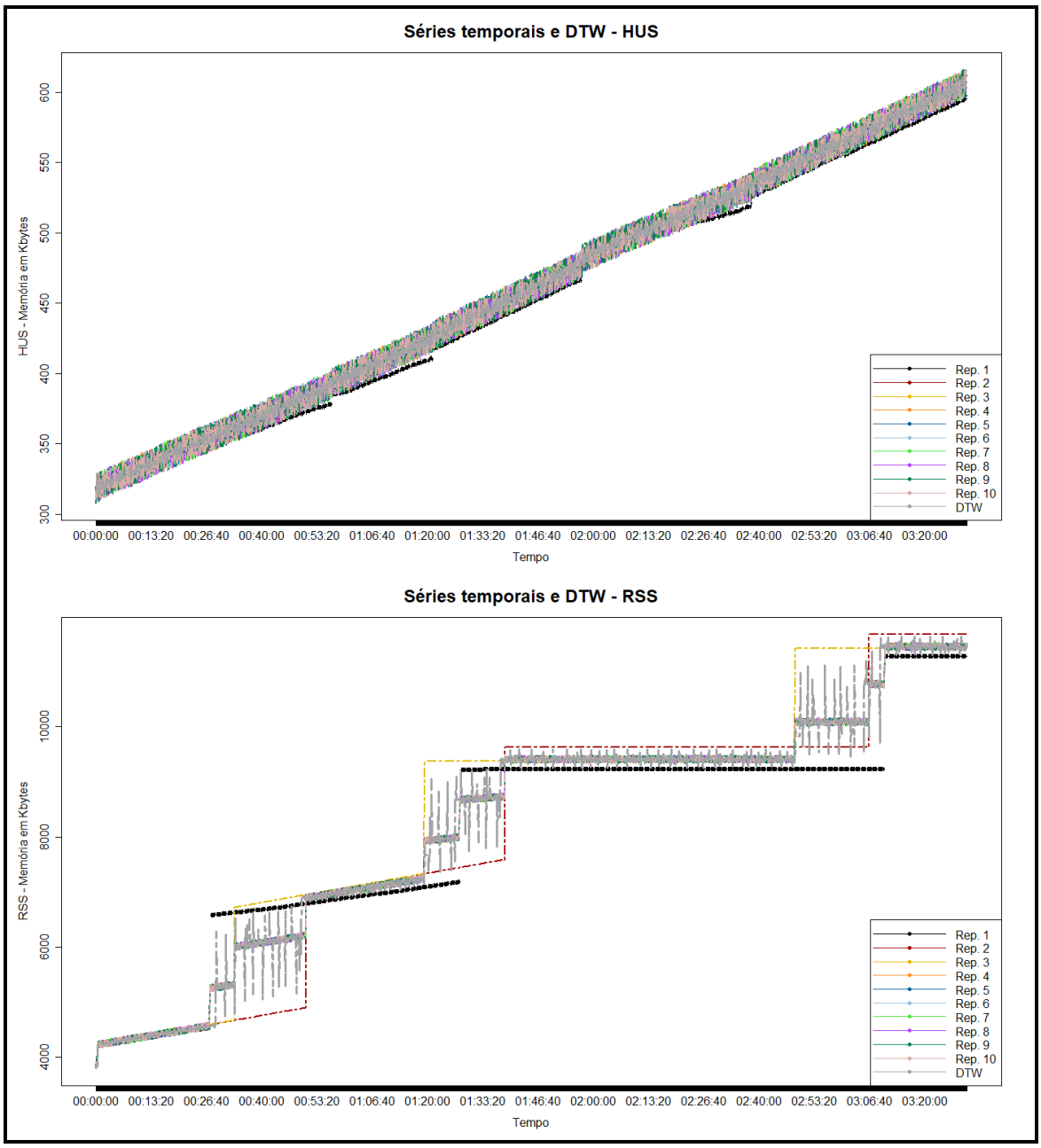


Figura 9.A2. Lighttpd: cenário de carga constante e normal, versão alvo – séries temporais e DTW.

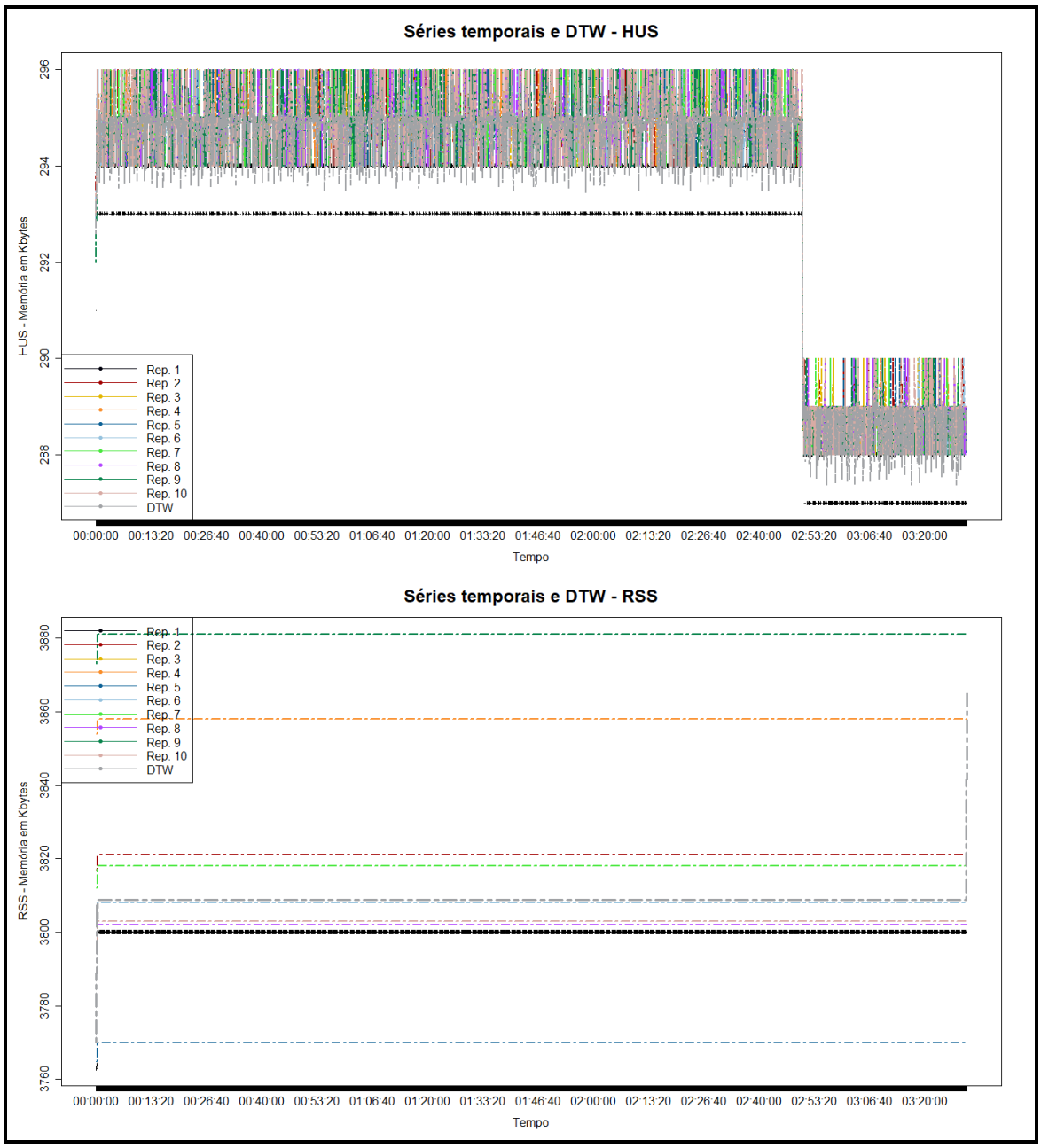


Figura 10.A2. Lighttpd: cenário de carga constante e normal, versão base – séries temporais e DTW.

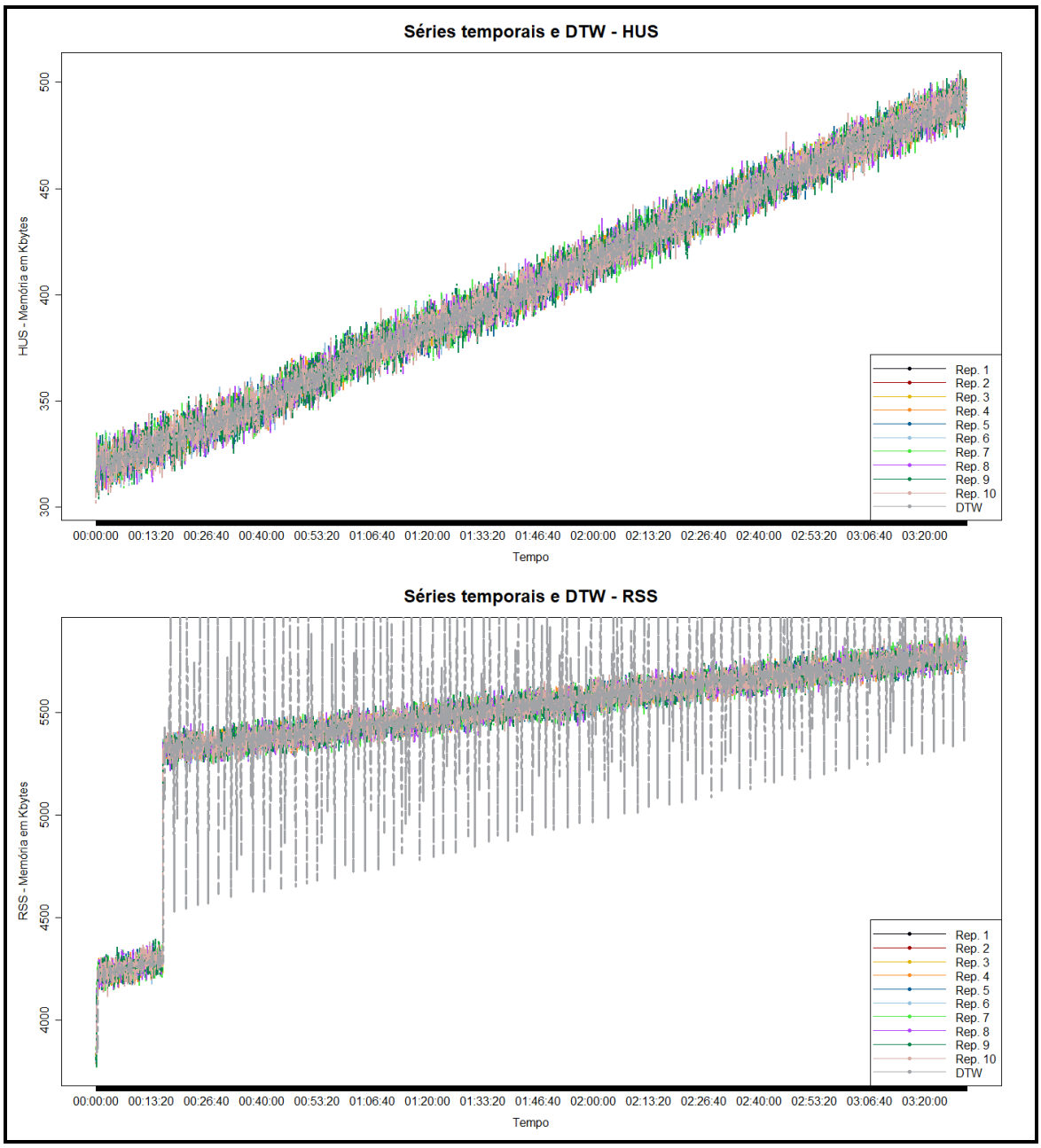


Figura 11.A2. Lighttpd: cenário de carga constante e baixa, versão alvo – séries temporais e DTW.

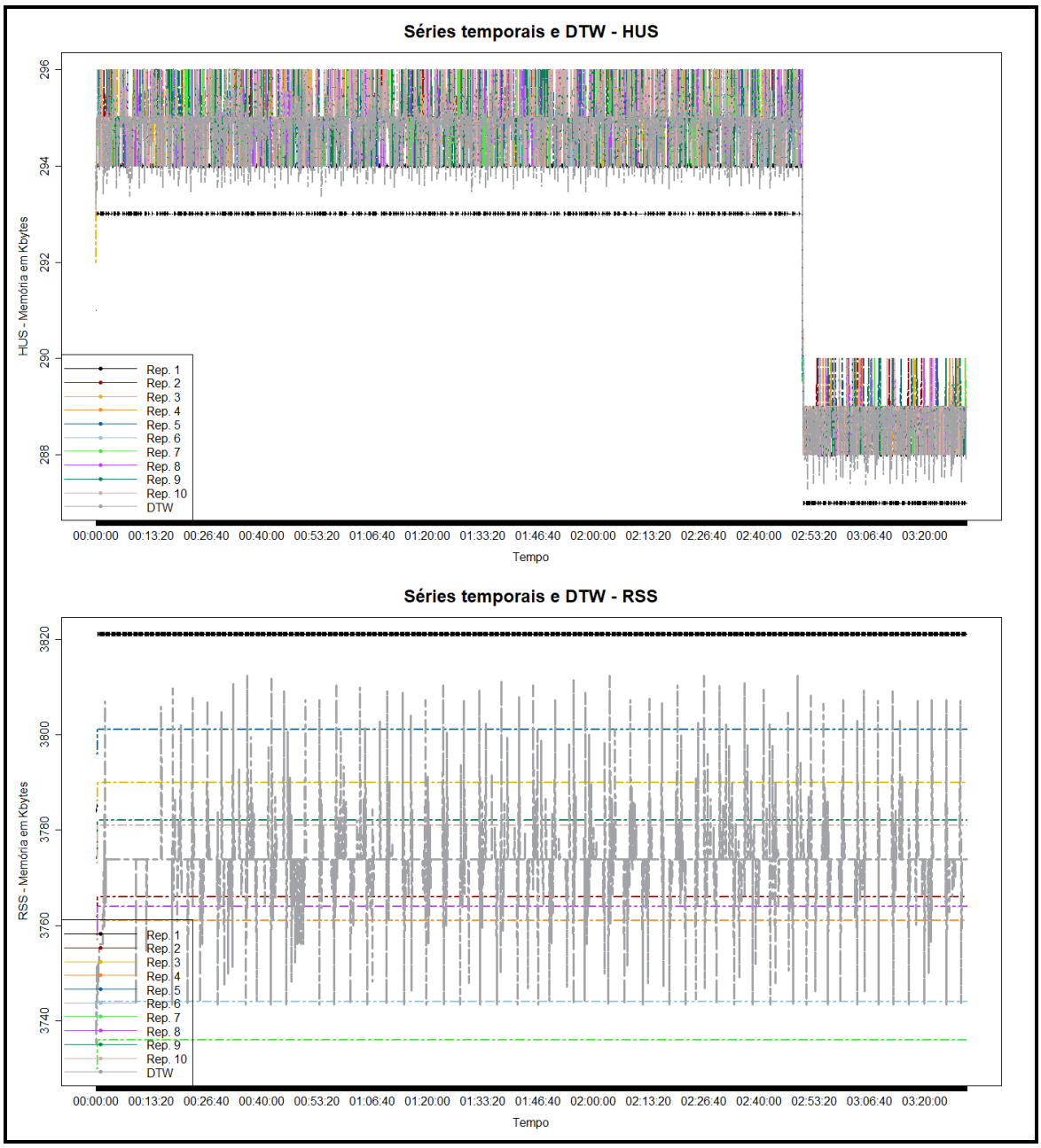


Figura 12.A2. Lighttpd: cenário de carga constante e baixa, versão base – séries temporais e DTW.

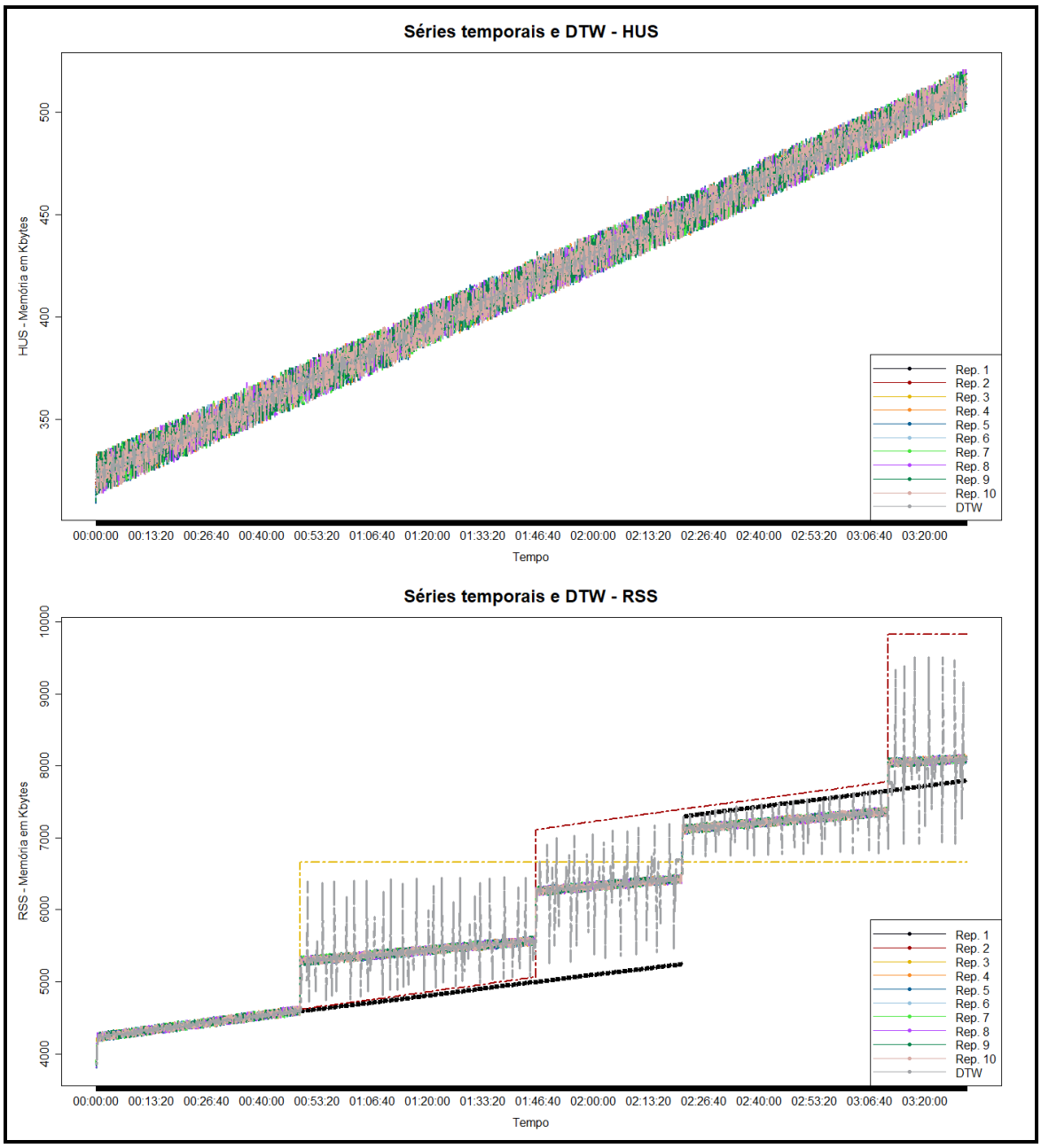


Figura 13.A2. Lighttpd: cenário de carga variada, versão alvo – séries temporais e DTW.

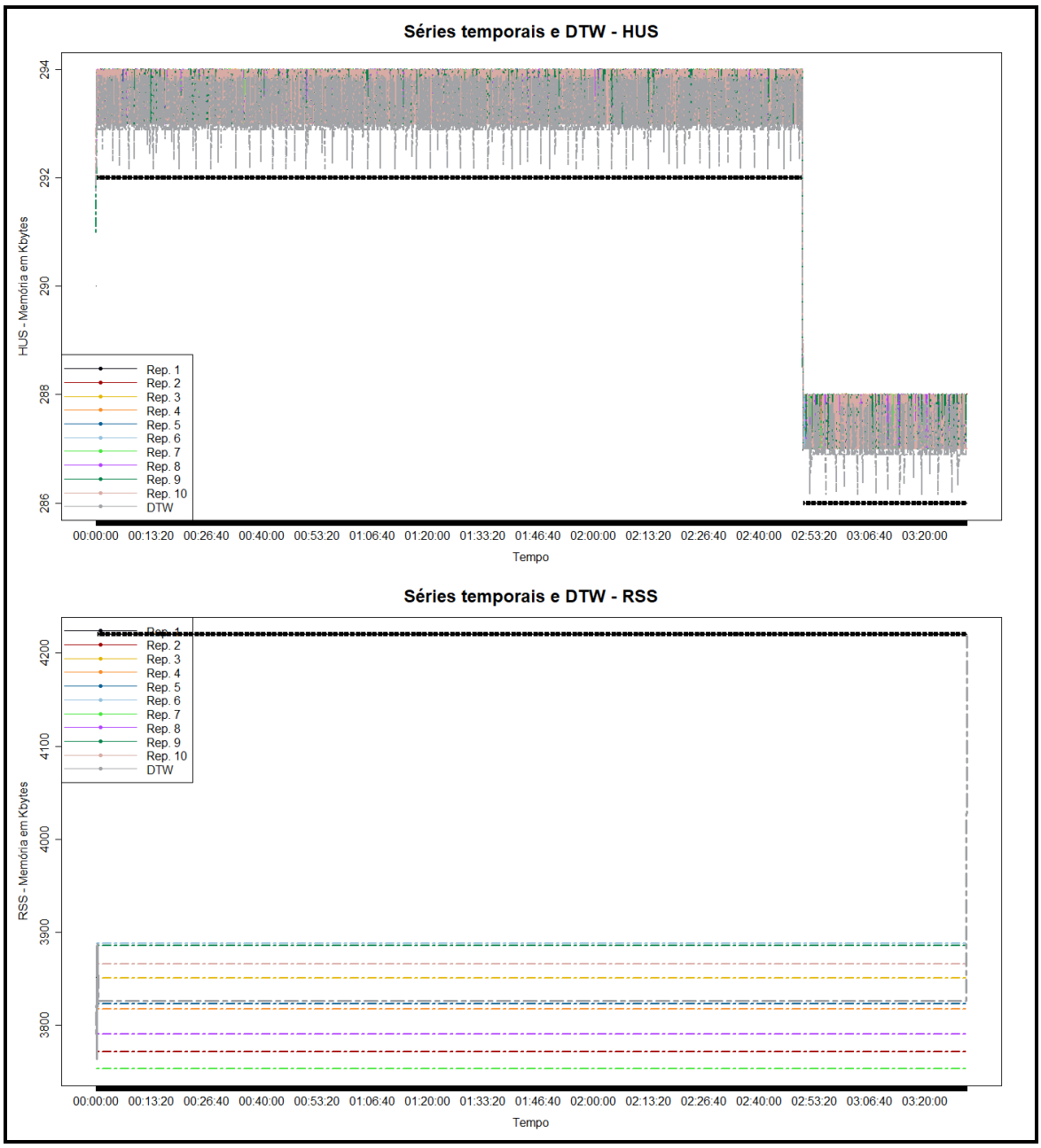


Figura 14.A2. Lighttpd: cenário de carga variada , versão base – séries temporais e DTW.

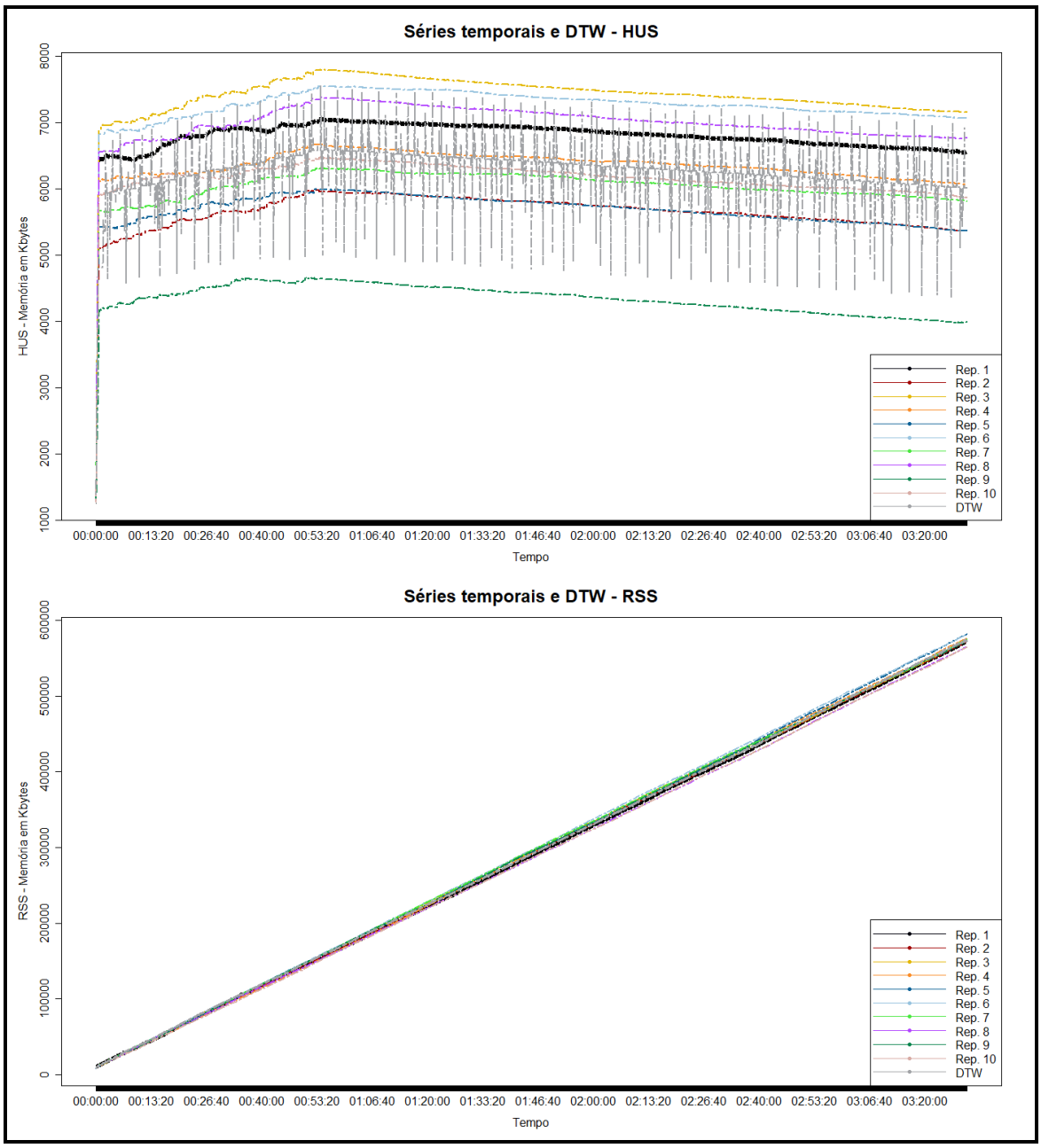


Figura 15.A2. PostgreSQL: cenário de carga constante e alta, versão base – séries temporais e DTW.

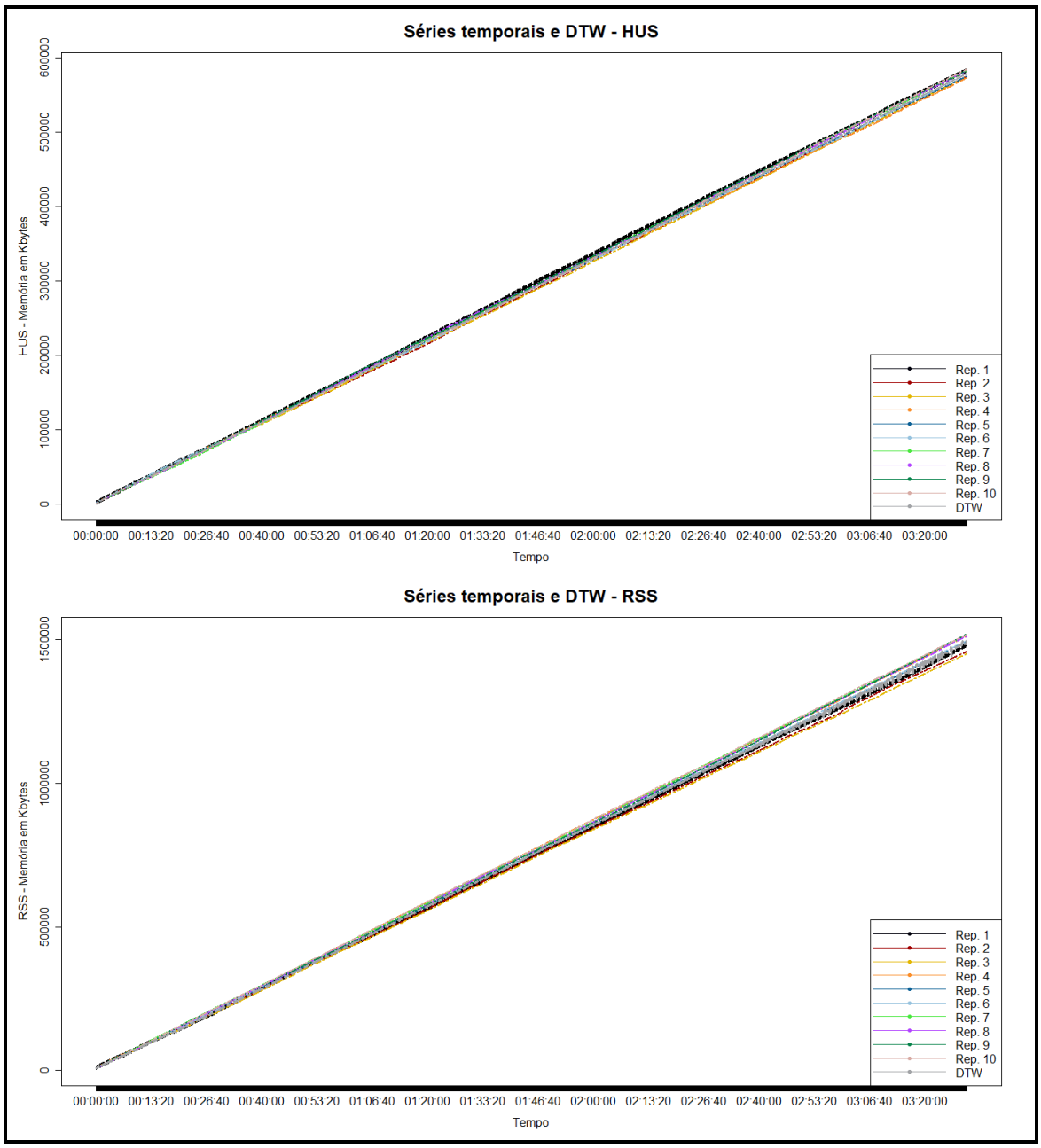


Figura 16.A2. PostgreSQL: cenário de carga constante e normal, versão alvo – séries temporais e DTW.

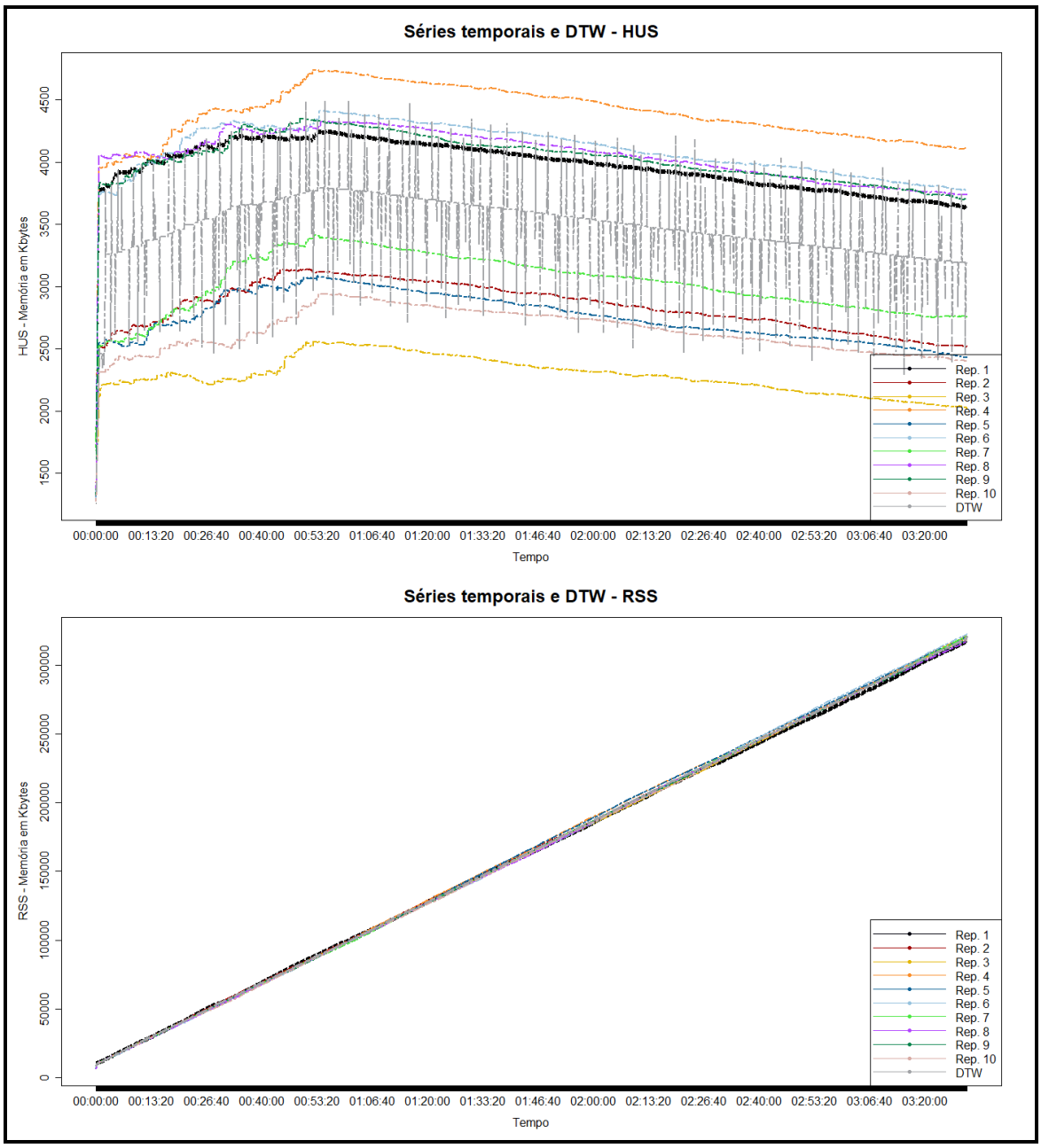


Figura 17.A2. PostgreSQL: cenário de carga constante e normal, versão base – séries temporais e DTW.

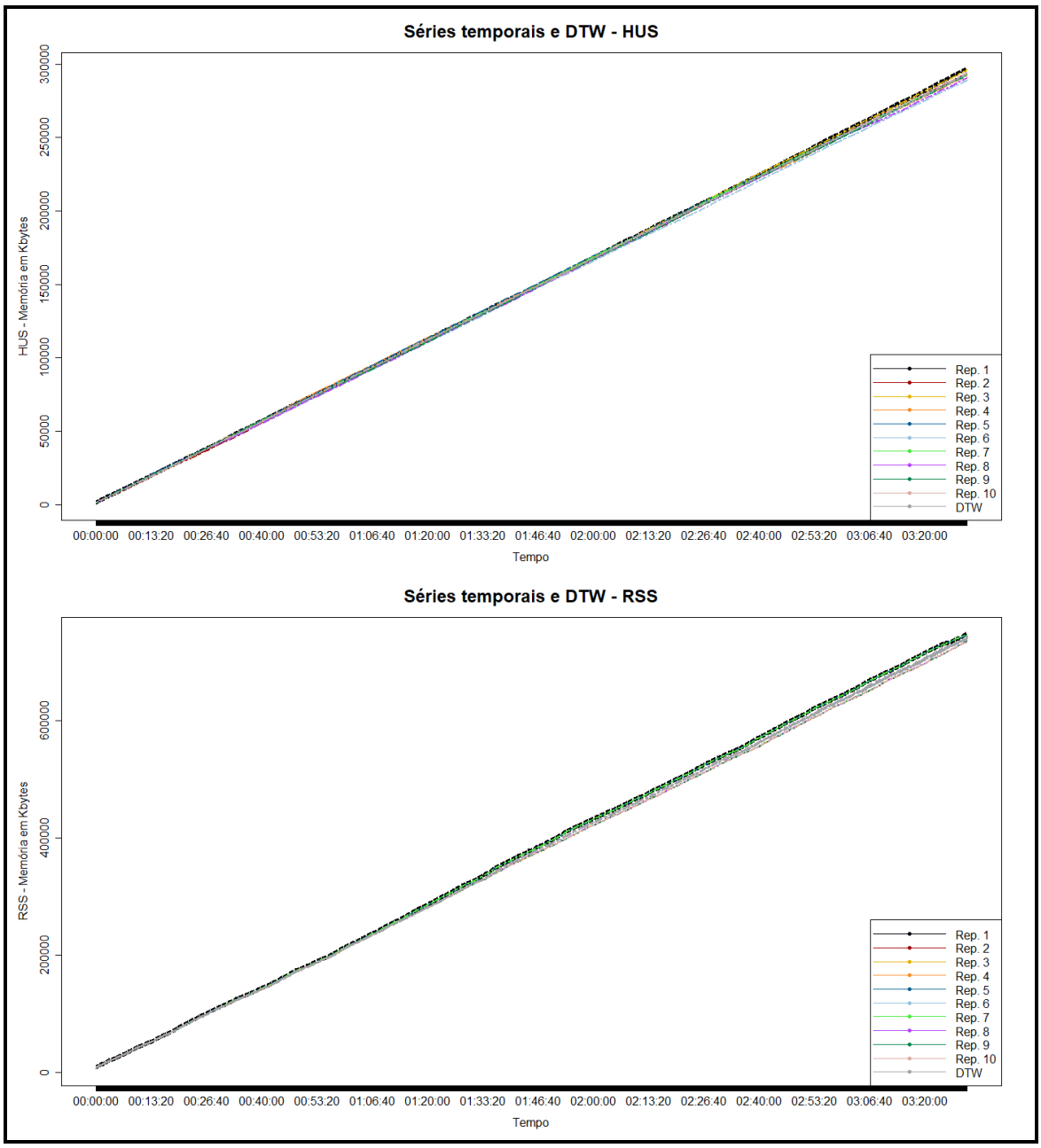


Figura 18.A2. PostgreSQL: cenário de carga constante e baixa, versão alvo – séries temporais e DTW.

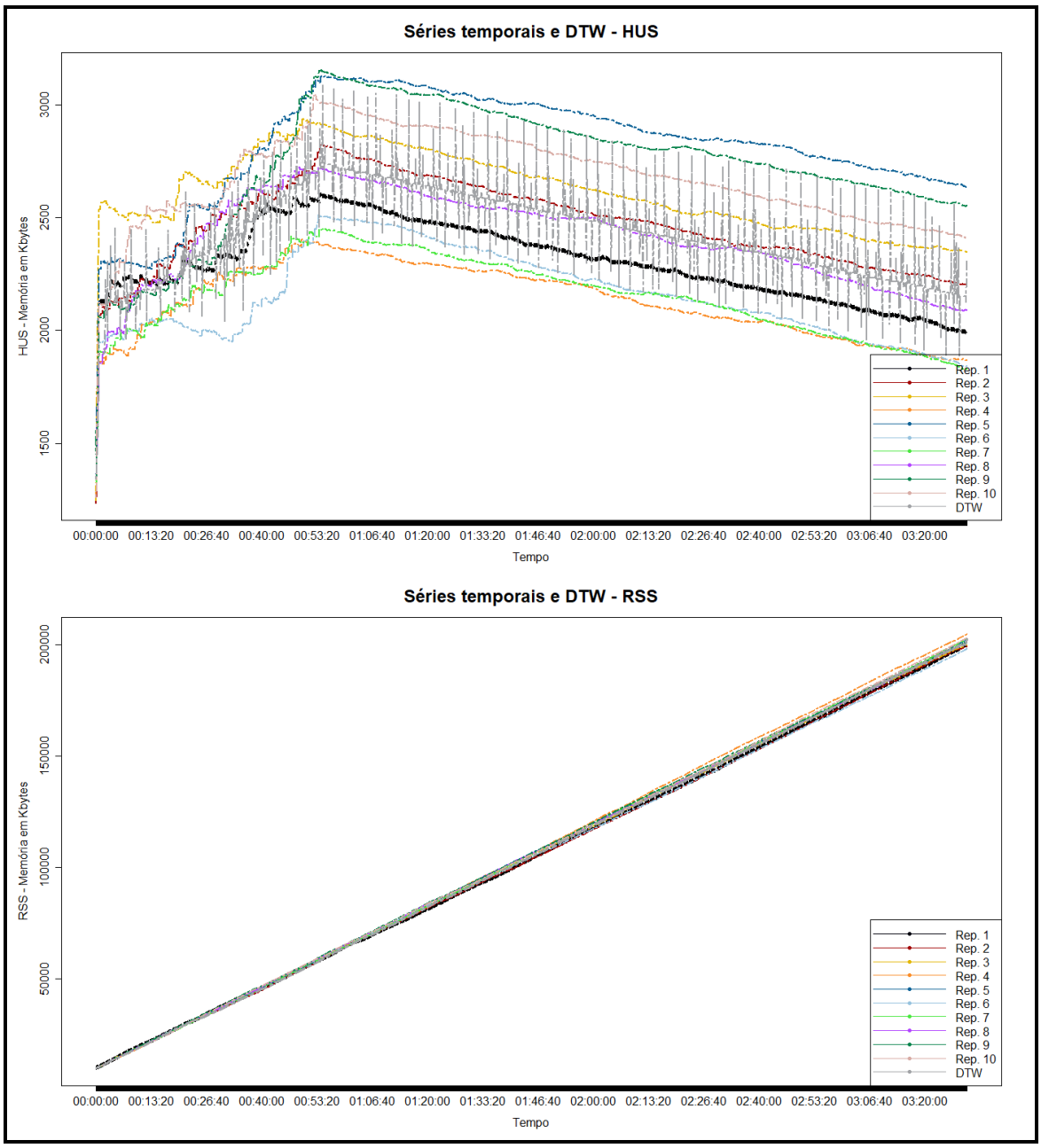


Figura 19.A2. PostgreSQL: cenário de carga constante e baixa, versão base – séries temporais e DTW.

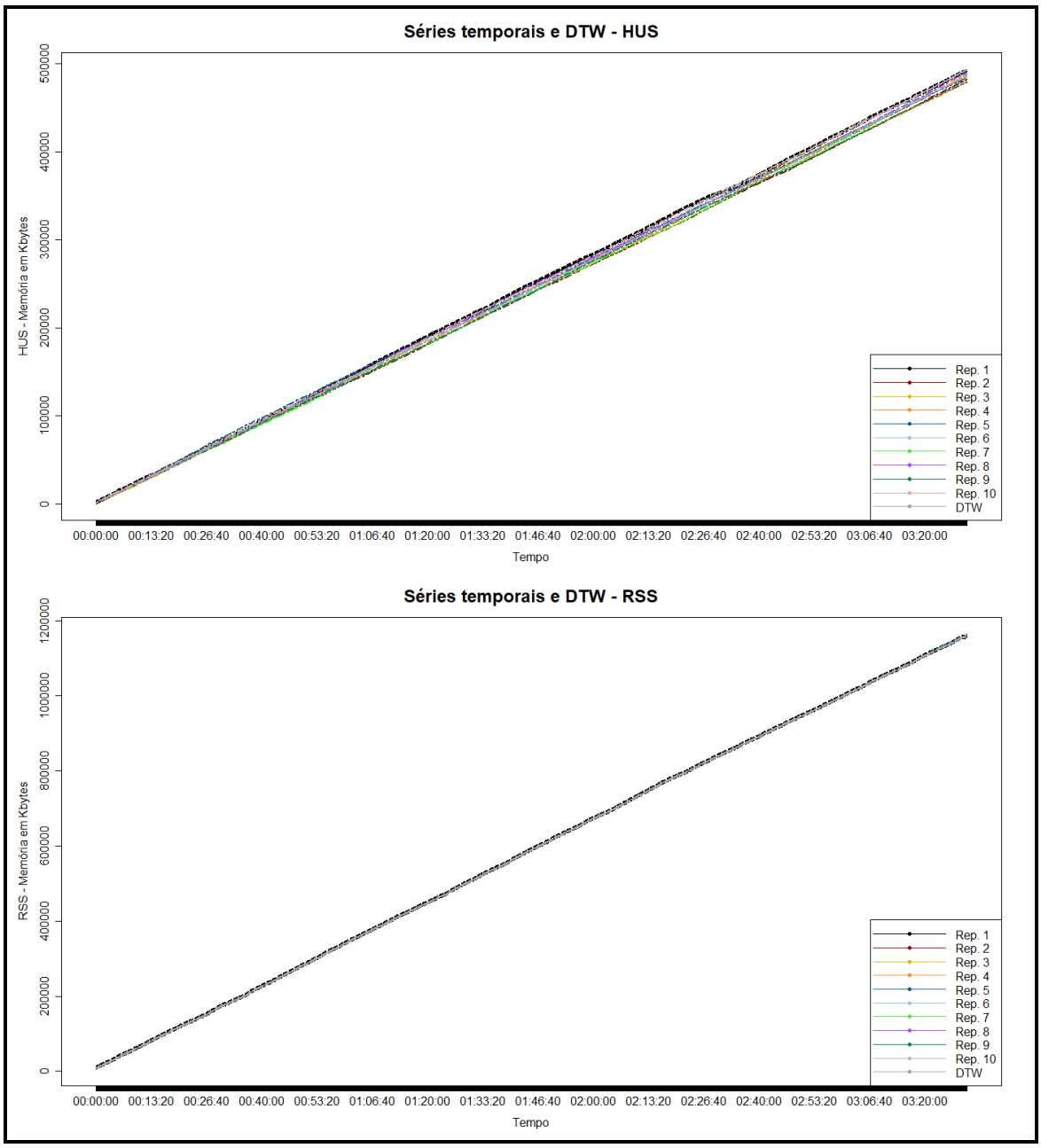


Figura 20.A2. PostgreSQL: cenário de carga variada, versão alvo – séries temporais e DTW.

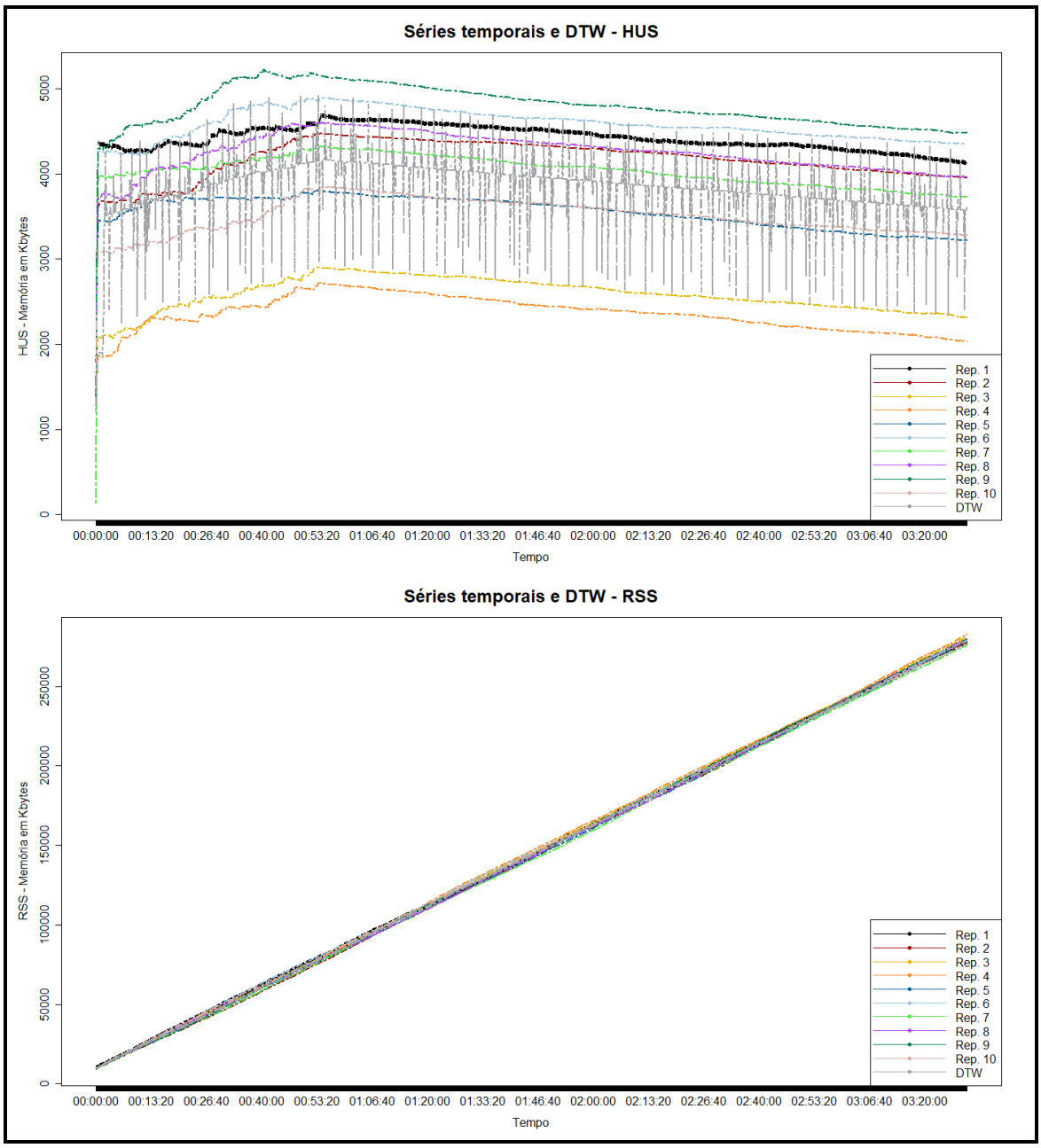


Figura 21.A2. PostgreSQL: cenário de carga variada , versão base – séries temporais e DTW.

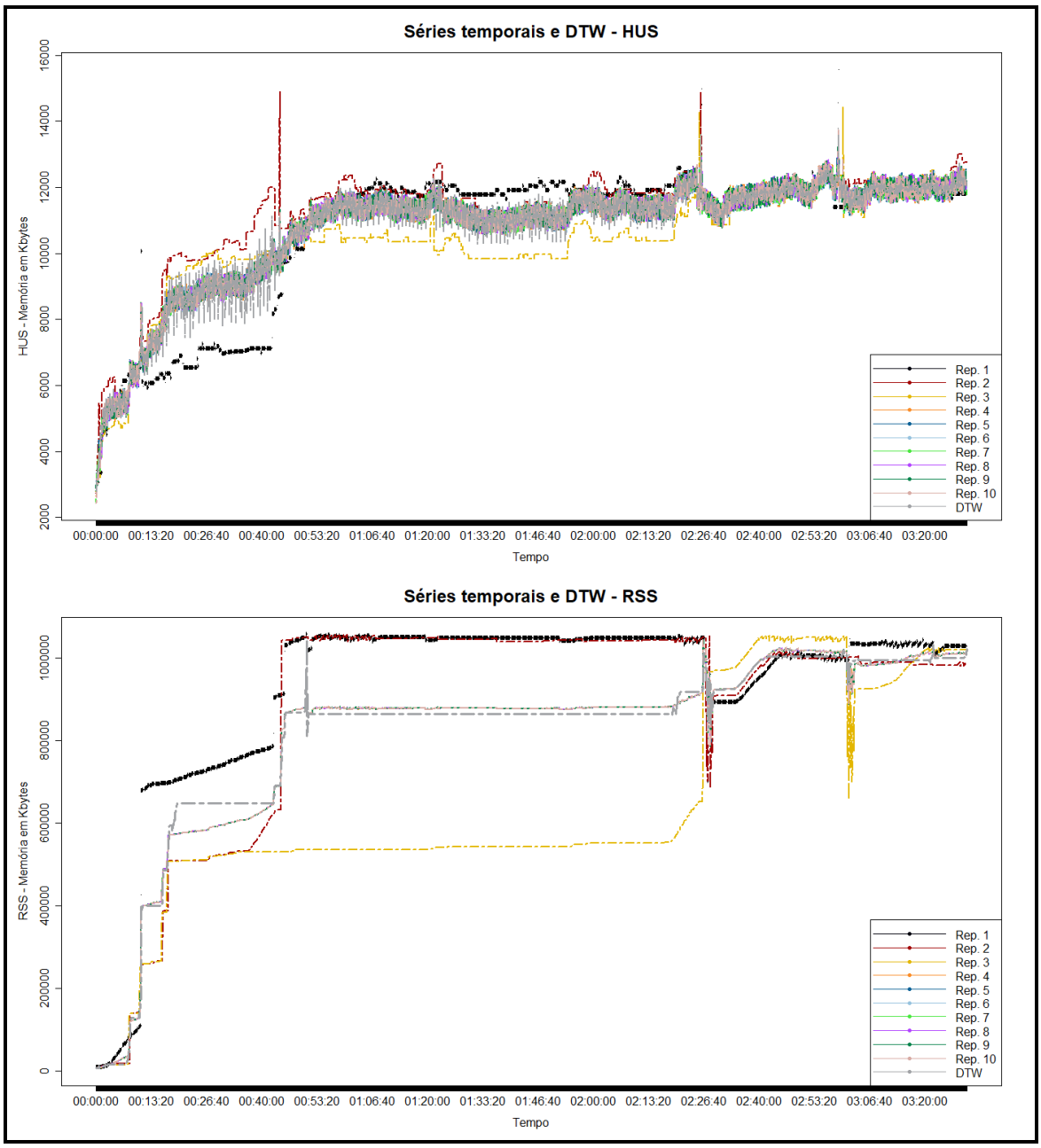


Figura 22.A2. Squid: cenário de carga constante e alta, versão base – séries temporais e DTW.

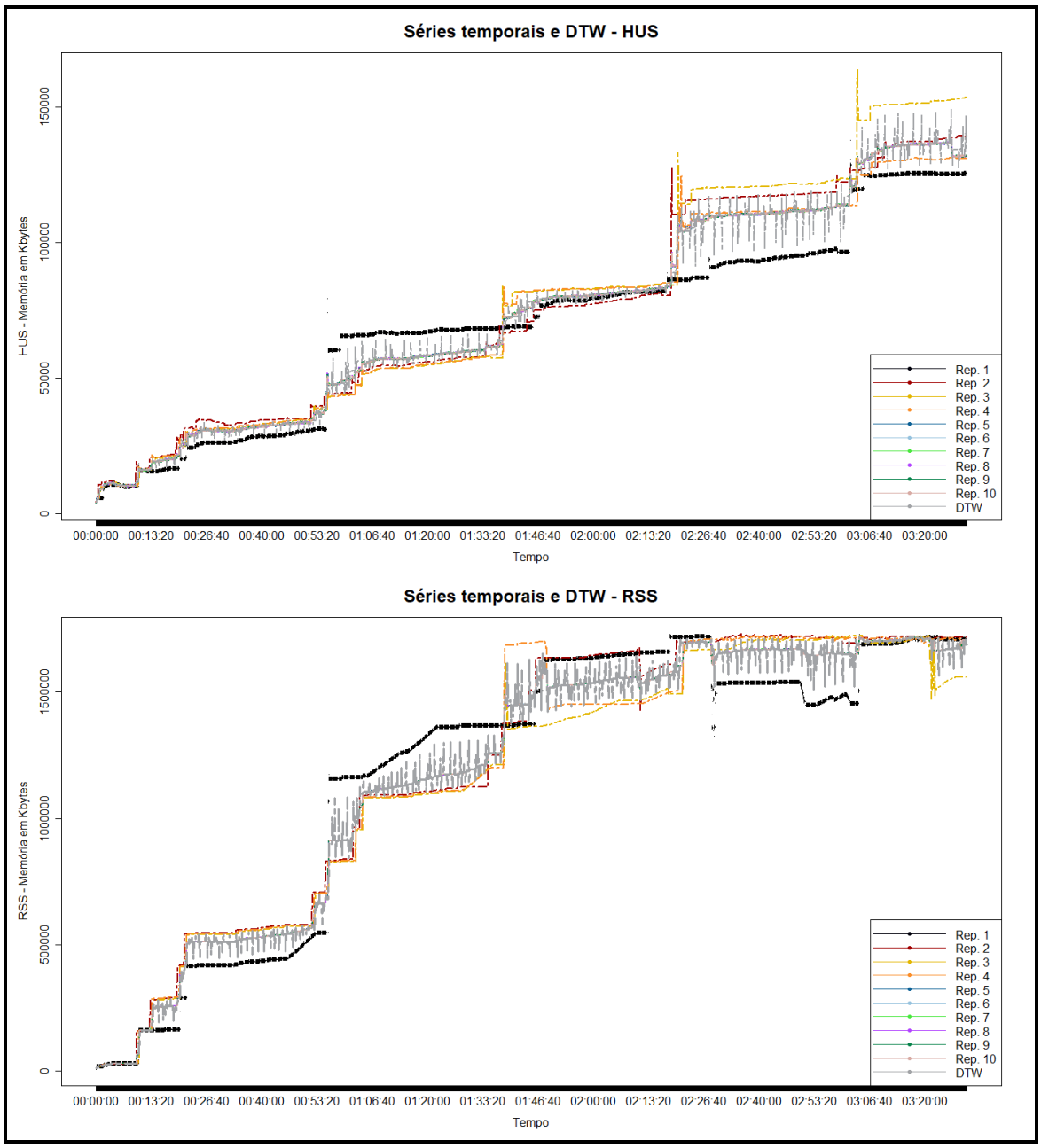


Figura 23.A2. Squid: cenário de carga constante e normal, versão alvo – séries temporais e DTW.

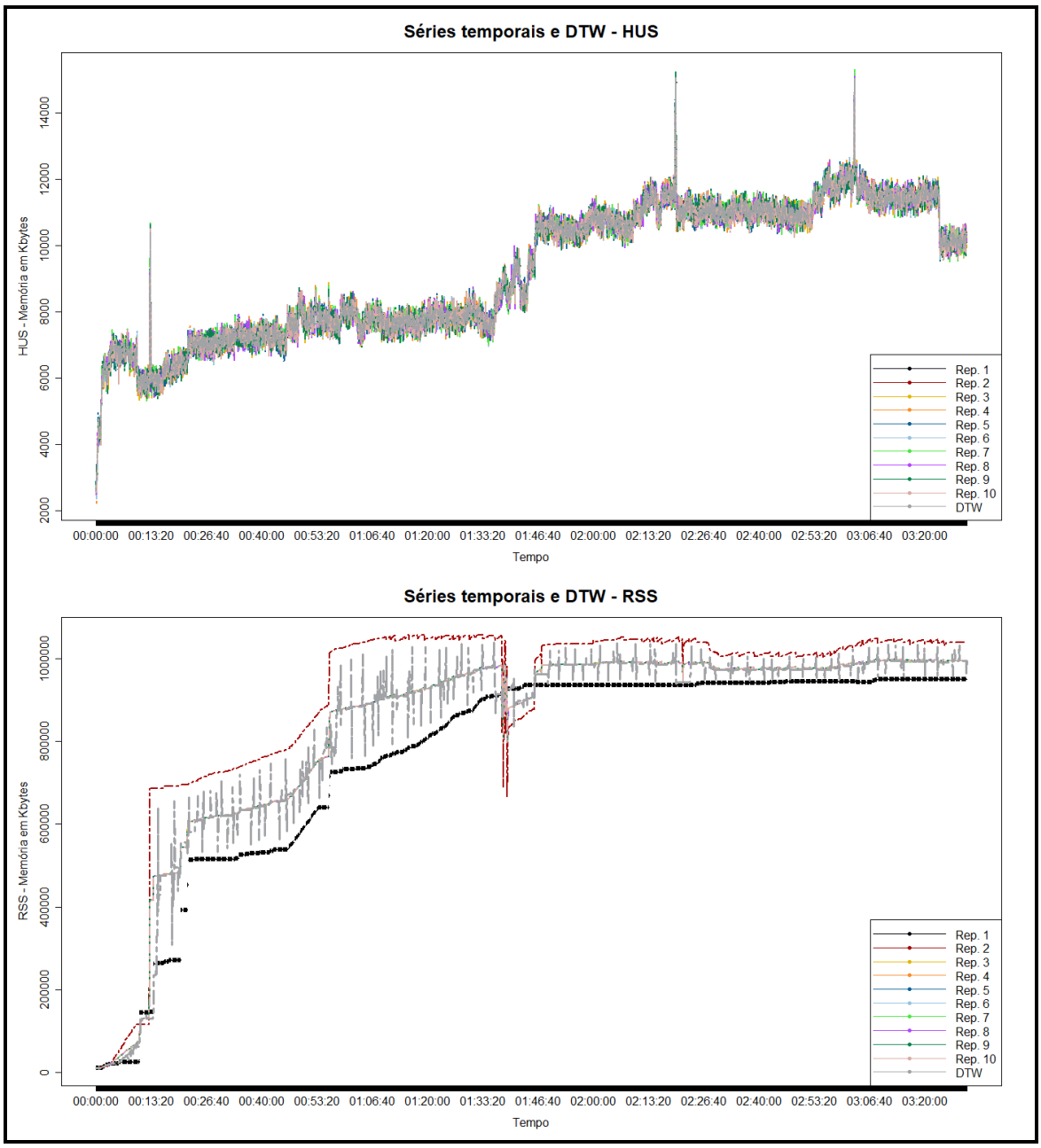


Figura 24.A2. Squid: cenário de carga constante e normal, versão base – séries temporais e DTW.

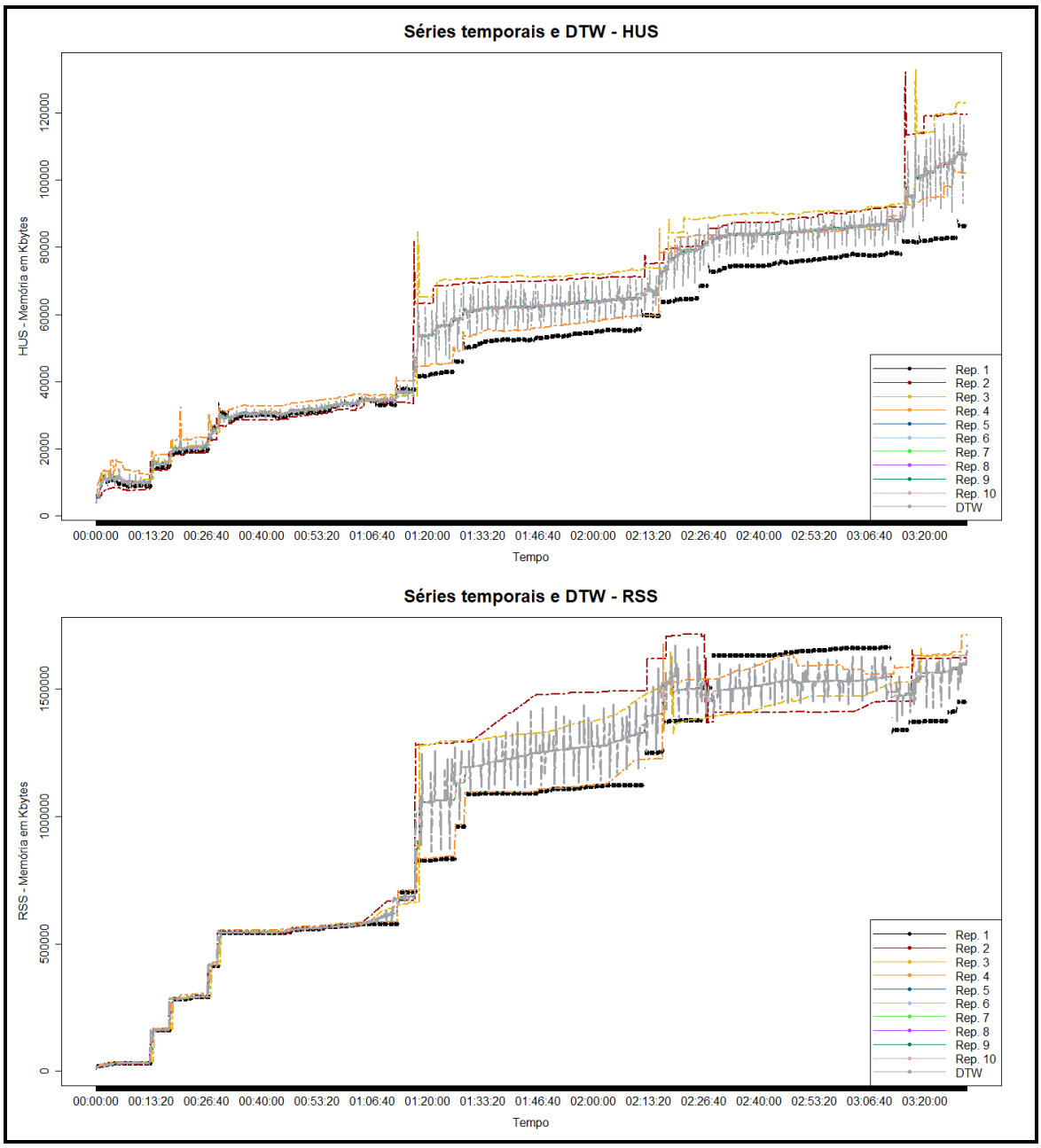


Figura 25.A2. Squid: cenário de carga constante e baixa, versão alvo – séries temporais e DTW.

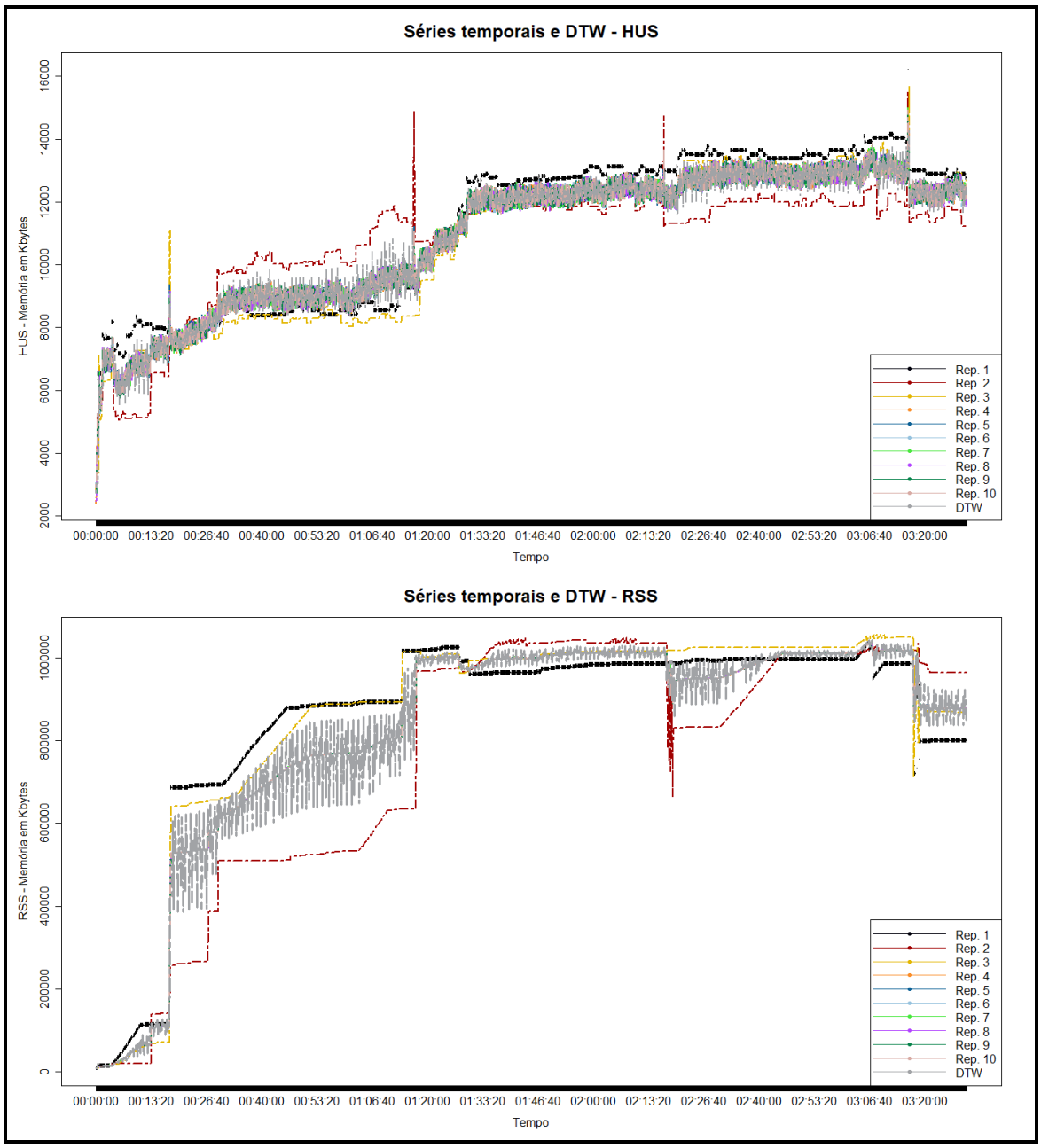


Figura 26.A2. Squid: cenário de carga constante e baixa, versão base – séries temporais e DTW.

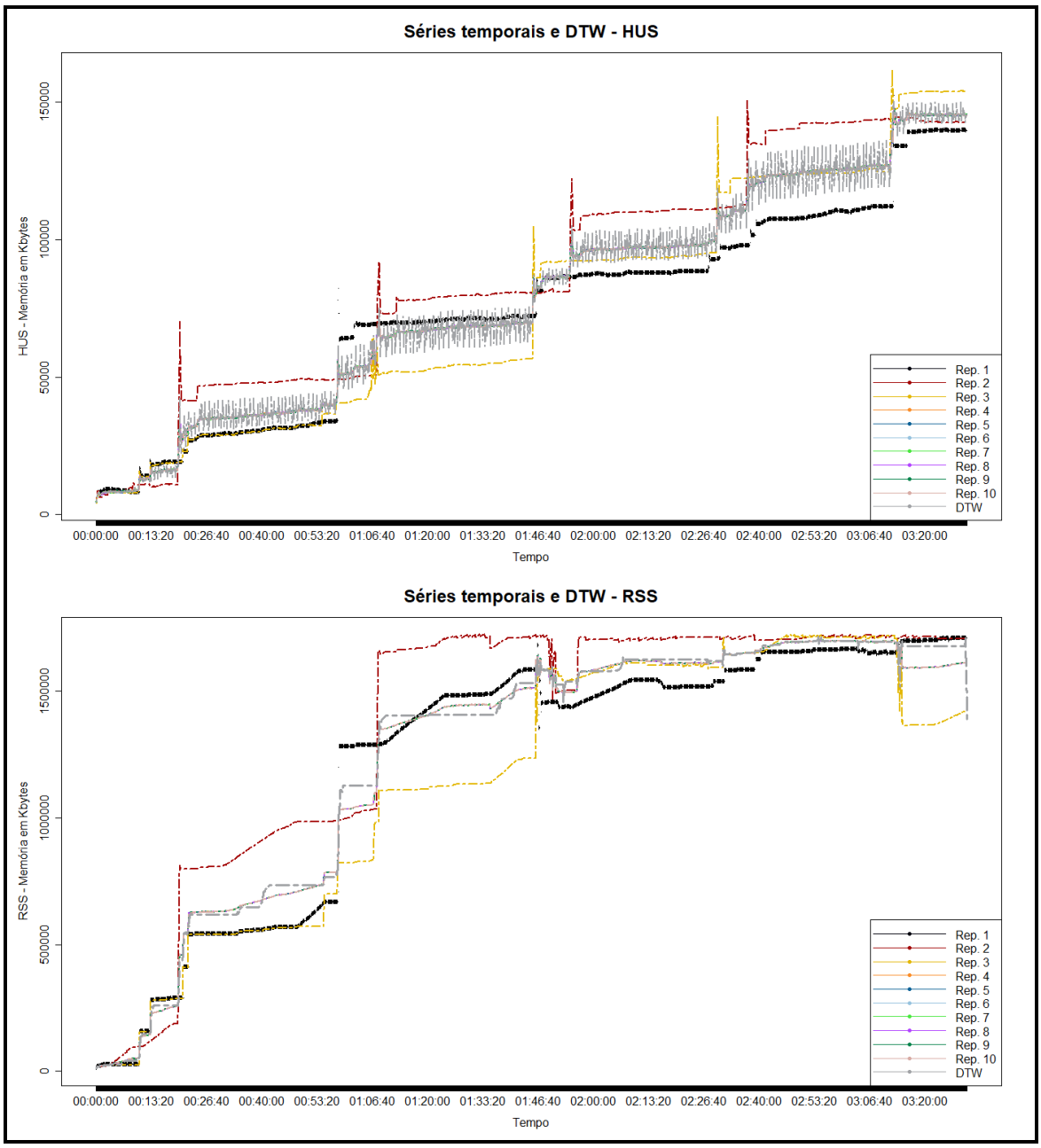


Figura 27.A2. Squid: cenário de carga variada, versão alvo – séries temporais e DTW.

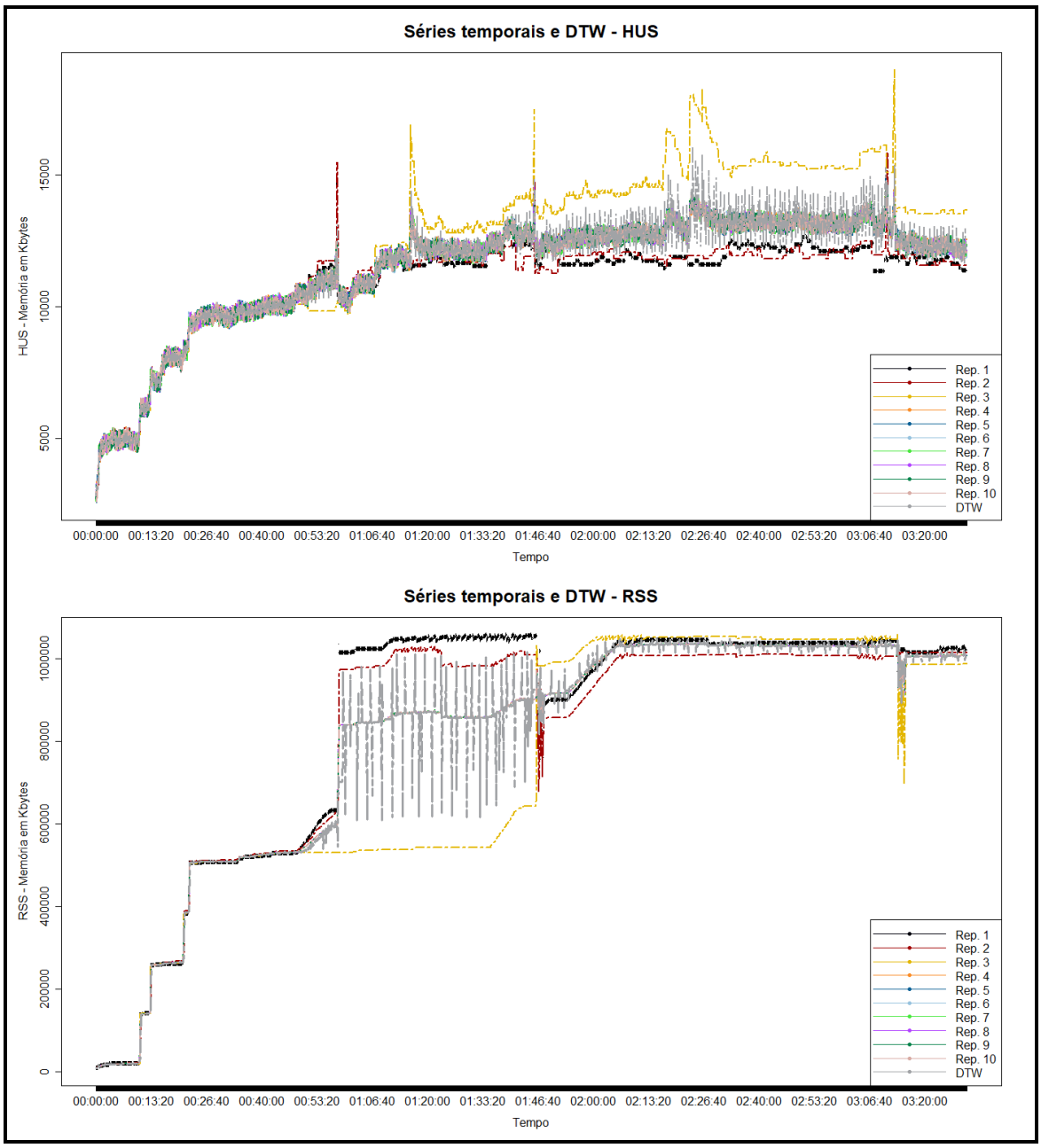


Figura 28.A2. Squid: cenário de carga variada , versão base – séries temporais e DTW.

A.3 Tabelas com tempos de detecção

Tabela 1.A3. Inkscape: Tempos de detecção para carga constante e normal.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	MA; HP; MM	00:00:25	00:00:25	0	1º	MA; HP; MM	00:00:25	00:00:25	0
2º	LR	00:00:30	00:00:30	0	2º	LR	00:00:30	00:00:30	0
3º	SH	00:17:35	00:17:35	0	3º	SH	00:17:05	00:17:05	0
4º	CS	00:28:00	00:28:00	0	4º	CS	00:26:40	00:26:40	0
5º	EW	00:35:10	00:35:10	0	5º	EW	00:33:10	00:33:10	0

Tabela 2.A3. Inkscape: Tempos de detecção para carga constante e baixa.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	MA; HP; MM	00:00:25	00:00:25	0	1º	MA; HP; MM	00:00:25	00:00:25	0
2º	LR	00:00:30	00:00:30	0	2º	LR	00:00:30	00:00:30	0
3º	SH	00:15:50	00:15:50	0	3º	SH	00:20:40	00:20:40	0
4º	CS	00:28:50	00:28:50	0	4º	CS	00:32:45	00:32:45	0
5º	EW	00:39:15	00:39:15	0	5º	EW	00:42:05	00:42:05	0

Tabela 3.A3. Inkscape: Tempos de detecção para carga variada.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	MA; HP; MM	00:00:25	00:00:25	0	1º	MA; HP; MM	00:00:25	00:00:25	0
2º	LR	00:00:30	00:00:30	0	2º	LR	00:00:30	00:00:30	0
3º	SH	00:21:40	00:21:40	0	3º	SH	00:22:45	00:22:45	0
4º	CS	00:33:00	00:33:00	0	4º	CS	00:34:05	00:34:05	0
5º	EW	00:41:30	00:41:30	0	5º	EW	00:42:25	00:42:25	0

Tabela 4.A3. Lighttpd: Tempos de detecção para carga constante e normal.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	CS; SH	00:00:20	00:00:20	0	1º	SH	00:00:20	00:00:20	0
2º	EW	00:00:25	00:00:25	0	2º	CS; MA; EW; HP; MM	00:00:25	00:00:25	0
3º	MA; LR; HP; MM	00:00:30	00:00:30	0	3º	LR	00:00:30	00:00:30	0

Tabela 5.A3. Lighttpd: Tempos de detecção para carga constante e baixa.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	CS; SH	00:00:20	00:00:20	0	1º	SH	00:00:20	00:00:20	0
2º	MA; EW; HP; MM	00:00:25	00:00:25	0	2º	CS; MA; HP; MM	00:00:25	00:00:25	0
3º	LR	00:00:30	00:00:30	0	3º	LR; EW	00:00:30	00:00:30	0

Tabela 6.A3. Lighttpd: Tempos de detecção para carga variada.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	SH	00:00:20	00:00:20	0	1º	CS; SH	00:00:20	00:00:20	0
2º	CS; HP	00:00:25	00:00:25	0	2º	MA; EW; HP; MM	00:00:25	00:00:25	0
3º	LR	00:00:30	00:00:30	0	3º	LR	00:00:30	00:00:30	0
4º	MA; EW	00:00:35	00:00:35	0					
5º	MM	00:00:40	00:00:40	0					

Tabela 7.A3. PostgreSQL: Tempos de detecção para carga constante e normal.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	MA; HP; MM	00:00:25	00:00:25	0	1º	MA; LR; HP; MM	00:00:30	00:00:30	0
2º	LR	00:00:30	00:00:30	0	2º	SH	00:01:10	00:01:10	0
3º	SH	00:23:35	00:23:35	0	3º	CS	00:01:20	00:01:20	0
4º	CS	00:34:00	00:34:00	0	4º	EW	00:01:30	00:01:30	0
5º	EW	00:41:25	00:41:25	0					

Tabela 8.A3. PostgreSQL: Tempos de detecção para carga constante e baixa.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	MA; HP; MM	00:00:25	00:00:25	0	1º	LR; HP; MM	00:00:30	00:00:30	0
2º	LR	00:00:30	00:00:30	0	2º	SH	00:01:10	00:01:10	0
3º	SH	00:28:25	00:28:25	0	3º	MA	00:00:30	00:01:10	1
4º	CS	00:42:15	00:42:15	0	4º	CS	00:01:30	00:01:30	0
5º	EW	00:52:10	00:52:10	0	5º	EW	00:01:45	00:01:45	0

Tabela 9.A3. PostgreSQL: Tempos de detecção para carga variada.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	LR; HP	00:00:30	00:00:30	0	1º	HP	00:00:25	00:00:25	0
2º	MA; MM	00:00:40	00:00:40	0	2º	LR	00:00:30	00:00:30	0
3º	SH	00:25:35	00:25:35	0	3º	MA	00:00:45	00:00:45	0
4º	CS	00:37:25	00:37:25	0	4º	MM	00:00:50	00:00:50	0
5º	EW	00:45:30	00:45:30	0	5º	SH	00:01:55	00:01:55	0
					6º	CS	00:02:00	00:02:00	0
					7º	EW	00:02:15	00:02:15	0

Tabela 10.A3. Squid: Tempos de detecção para carga constante e normal.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	MA	00:00:25	00:00:25	0	1º	MA; HP; MM	00:00:25	00:00:25	0
2º	SH	00:56:20	01:01:40	4	2º	LR	00:00:30	00:01:00	1
3º	CS	01:29:30	01:29:30	0	3º	CS	00:04:10	00:04:10	0
4º	EW	01:25:35	01:29:45	2	4º	SH	00:02:10	00:09:55	3
5º	HP	00:00:25	02:15:10	2	5º	EW	00:10:40	00:10:40	0
6ª	LR	00:00:30	02:53:25	11					
7ª	MM	00:00:25	03:28:50	2					

Tabela 11.A3. Squid: Tempos de detecção para carga constante e baixa.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	MA; MM	00:00:25	00:00:25	0	1º	HP	00:00:25	00:00:25	0
2º	SH	01:17:15	01:25:30	4	2º	MA	00:00:25	00:01:45	1
3º	EW	01:35:00	01:36:35	1	3º	MM	00:00:25	00:02:10	1
4º	CS	01:47:55	01:47:55	0	4º	LR	00:00:30	00:12:40	5
5º	HP	00:00:25	03:17:35	1	5º	SH	00:13:35	00:13:35	0
6ª	LR	00:00:30	03:24:00	5	6ª	CS	00:14:05	00:14:05	0
					7ª	EW	00:14:30	00:14:30	0

Tabela 12.A3. Squid: Tempos de detecção para carga variada.

Indicador: RSS					Indicador: HUS				
Ranking		First Time	Last Time	Num Events	Ranking		First Time	Last Time	Num Events
1º	MA	00:00:25	00:00:25	0	1º	MA; HP; MM	00:00:25	00:00:25	0
2º	SH	00:58:35	00:58:35	0	2º	LR	00:00:30	00:00:30	0
3º	EW	01:08:25	01:08:25	0	3º	EW	00:14:35	00:14:35	0
4º	CS	01:11:25	01:11:25	0	4º	SH	00:13:40	00:15:20	2
5º	HP	00:00:25	02:31:40	5	5º	CS	00:15:30	00:15:30	0
6ª	LR	00:00:30	03:14:05	24					
7ª	MM	00:00:25	03:21:35	2					