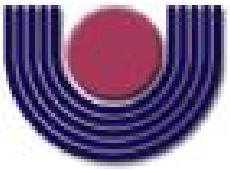

Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Estudo comparativo de técnicas de alocação elástica de memória em nuvens computacionais

Cristiane de Andrade

CASCAVEL
2016

Cristiane de Andrade

Estudo comparativo de técnicas de alocação elástica de memória em nuvens computacionais

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Guilherme Galante

CASCAVEL
2016

Cristiane de Andrade

Estudo comparativo de técnicas de alocação elástica de memória em nuvens computacionais

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Guilherme Galante
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Luiz Antonio Rodrigues
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Marcio Seiji Oyamada
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 15 de fevereiro de 2017.

DEDICATÓRIA

Dedico este trabalho à minha família, ao meu PS e a irmandade a qual eu faço parte e que me ajudaram a chegar até aqui.

"Tome minha vontade e minha vida, oriente-me na minha recuperação, mostre-me como viver."

AGRADECIMENTOS

Agradeço a minha família por me dar apoio, amor e suporte em todos esses anos de graduação, sem eles não seria possível chegar até aqui. Agradeço a irmandade da qual faço parte por tornar possível a conclusão dessa graduação, sem isso eu não estaria aqui. Agradeço ao Poder Superior da minha compreensão por chegar aqui, por me dar forças, me amar incondicionalmente e por ter me dado uma nova oportunidade. Agradeço a minha sobrinha Jade pelo amor, suporte e apoio, por acreditar em mim quando eu não conseguia. Agradeço ao meu padrinho César por todo o apoio, por todo o suporte e por acreditar em mim quando eu não conseguia acreditar. Agradeço as minhas amigas e colegas de graduação, Eloisa, Jessica, Silvia e Suelin, pelas risadas, conversas, jantas e pelo apoio. Agradeço a Alexandra por termos caminhado juntas nessa graduação, dando apoio e suporte uma pra outra, compartilhando todos os momentos, isso fez a diferença nesses anos. Agradeço ao meu orientador Guilherme pela paciência e por todo o conhecimento compartilhado.

Lista de Figuras

2.1 Camadas de serviço em nuvem. Adaptado de [1]. 9

3.1 Alocação estática e alocação elástica. Adaptado de [2]. 19
3.2 Elasticidade Horizontal e Vertical. Adaptado de [1]. 20
3.3 Mecanismo Regra-Condição-Ação. Adaptado de [1]. 22

4.1 Definição de precisão e tempo de provisionamento de recursos. Adaptado de [3] 32

5.1 Trace crescente. 35
5.2 Trace decrescente. 36
5.3 Trace piconst. 37
5.4 Trace const. 37
5.5 Trace picos. 38
5.6 Trace dalDC1. 39
5.7 Trace fipS. 40
5.8 Trace fipapp. 41
5.9 Trace fipnfe. 41
5.10 Trace irWTS. 42
5.11 Trace TVL. 43
5.12 Trace WS. 43
5.13 *Trace const* Comparativo Moltó 10% e 30%. 45
5.14 *Trace piconst* Comparativo Moltó 10% e 30%. 46
5.15 *Trace crescente* Comparativo Moltó 10% e 30%. 47
5.16 *Trace picos* Comparativo Moltó 10% e 30%. 48
5.17 *Trace fipS* Comparativo Moltó 10% e 30%. 49

5.18 *Trace dalDC1* Comparativo Moltó 10% e 30%. 51
5.19 *Trace TVL* Comparativo Moltó 10% e 30%. 52
5.20 *Trace fipapp* Comparativo Moltó 10% e 30%. 53
5.21 *Trace fipnfe* Comparativo Moltó 10% e 30%. 54
5.22 *Trace irWTS* Comparativo Moltó 10% e 30%. 55
5.23 *Trace WS* Comparativo Moltó 10% e 30%. 56
5.24 *Trace crescente* - Oliveira 90% e 70% com λ 1. 58
5.25 *Trace decrescente* - Oliveira 90% e 70% com λ 1. 58
5.26 *Trace picos* - Oliveira 90% e 70% com λ 1. 59
5.27 *Trace piconst* - Oliveira 90% e 70% com λ 1. 59
5.28 *Trace const* - Oliveira 90% e 70% com λ 1. 60
5.29 *Trace fipS* - Oliveira 90% e 70% com λ 1. 61
5.30 *Trace dalDC1* - Oliveira 90% e 70% com λ 1. 61
5.31 *Trace fipapp* - Oliveira 90% e 70% com λ 1. 62
5.32 *Trace fipnfe* - Oliveira 90% e 70% com λ 1. 62
5.33 *Trace irWTS* - Oliveira 90% e 70% com λ 1. 63
5.34 *Trace TVL* - Oliveira 90% e 70% com λ 1. 63
5.35 *Trace WS* - Oliveira 90% e 70% com λ 1. 64
5.36 *Trace crescente* - Oliveira 90% e 70% com λ 0,8. 65
5.37 *Trace decrescente* - Oliveira 90% e 70% com λ 0,8. 65
5.38 *Trace const* - Oliveira 90% e 70% com λ 0,8. 66
5.39 *Trace piconst* - Oliveira 90% e 70% com λ 0,8. 66
5.40 *Trace picos* - Oliveira 90% e 70% com λ 0,8. 67
5.41 *Trace fipS* - Oliveira 90% e 70% com λ 0,8. 68
5.42 *Trace dalDC1* - Oliveira 90% e 70% com λ 0,8. 69
5.43 *Trace fipapp* - Oliveira 90% e 70% com λ 0,8. 70
5.44 *Trace fipnfe* - Oliveira 90% e 70% com λ 0,8. 71
5.45 *Trace irWTS* - Oliveira 90% e 70% com λ 0,8. 72
5.46 *Trace TVL* - Oliveira 90% e 70% com λ 0,8. 73
5.47 *Trace WS* - Oliveira 90% e 70% com λ 0,8. 74

5.48 *Trace crescente* - Oliveira 90% e 70% com λ 1,2. 74
5.49 *Trace decrescente* - Oliveira 90% e 70% com λ 1,2. 75
5.50 *Trace picos* - Oliveira 90% e 70% com λ 1,2. 75
5.51 *Trace piconst* - Oliveira 90% e 70% com λ 1,2. 76
5.52 *Trace const* - Oliveira 90% e 70% com λ 1,2. 76
5.53 *Trace fipS* - Oliveira 90% e 70% com λ 1,2. 77
5.54 *Trace dalDC1* - Oliveira 90% e 70% com λ 1,2. 78
5.55 *Trace fipapp* - Oliveira 90% e 70% com λ 1,2. 78
5.56 *Trace fipnfe* - Oliveira 90% e 70% com λ 1,2. 79
5.57 *Trace irWTS* - Oliveira 90% e 70% com λ 1,2. 79
5.58 *Trace TVL* - Oliveira 90% e 70% com λ 1,2. 80
5.59 *Trace WS* - Oliveira 90% e 70% com λ 1,2. 80
5.60 *Trace Crescente* - Jenitha. 82
5.61 *Trace Decrescente* - Jenitha. 83
5.62 *Trace const* - Jenitha. 84
5.63 *Trace Picos* - Jenitha. 85
5.64 *Trace Piconst* - Jenitha. 86
5.65 *Trace fipS* - Jenitha. 87
5.66 *Trace dalDC1* - Jenitha. 88
5.67 *Trace fipnfe* - Jenitha. 89
5.68 *Trace fipapp* - Jenitha. 90
5.69 *Trace irWTS* - Jenitha. 91
5.70 *Trace TVL* - Jenitha. 92
5.71 *Trace WS* - Jenitha. 93

Lista de Tabelas

3.1 Suporte à elasticidade vertical oferecida pelos principais hipervisores [1]. . . . 21
3.2 Soluções reativas de alocação de memória. 28
3.3 Soluções proativas de alocação de memória. 30

5.1 Comparação dos métodos de alocação de memória. 95

Lista de Abreviaturas e Siglas

API	*Application Program Interface*
CC	*Capacity Base*
CEO	*Chief Executive Office*
CPU	*Central Processing Unit*
EC2	*Elastic Compute Cloud*
FFT	*Fast Fourier Transform*
IaaS	*Infrastructure as a Service*
JME	*Java Micro Edition*
MB	*MegaBytes*
MME	*Média Móvel Exponencial*
MMU	*Memory Management Unit*
MOP	*Memory Overprovisioning Percentage*
MV	*Máquina Virtual*
NIST	*Nacional Institute of Standards and Technology*
PaaS	*Plataforme as a Service*
PC	*Personal Computer*
PDA'S	*Personal Digital Assistants*
PTE	*Produto Espaço Tempo*
QOS	*Quality of Services*
RT	*Response Time*
SaaS	*Software as a Service*
SGBD	*Sistemas de Gerenciamento de Banco de Dados*
SLA	*Service Level Agreement*
SO	*Sistemas Operacionais*
SSH	*Secure Shell*
TI	*Tecnologia da Informação*

Sumário

Lista de Figuras **vii**

Lista de Tabelas **x**

Lista de Abreviaturas e Siglas **xi**

Sumário **xii**

Resumo **xiv**

1 Introdução **1**

2 Computação em Nuvem **5**

2.1 Características 6

2.2 Modelos de Serviço 9

2.2.1 Softwares como Serviço *(Software as a Service - SaaS)* 9

2.2.2 Plataforma como Serviço *(Platform as a Service - PaaS)* 10

2.2.3 Infraestrutura como um Serviço *(Infrastructure as a Service - IaaS)* . . 11

2.3 Modelos de Implantação 13

2.3.1 Nuvem Privada 14

2.3.2 Nuvem Pública 15

2.3.3 Nuvem Comunitária 16

2.3.4 Nuvem Híbrida 16

3 Elasticidade **18**

3.1 Classificação das Soluções de Elasticidade 19

3.2 Elasticidade de Memória 22

3.3 Elasticidade de Memória - Estado da Arte 24

4 Métricas **31**

5 Avaliação de métodos de elasticidade de memória **34**

5.1 Ambiente Computacional 34

5.2 Caracterização dos Estudos de Caso 35

5.3 Avaliação de métodos de elasticidade de memória 44

5.4 Moltó 44

5.5 Oliveira/Dawoud/Heo 56

5.6 Jenitha 81

5.7 Baruchi 93

5.8 Comparação dos metódos 94

6 Conclusão **96**

Referências Bibliográficas **98**

Resumo

A Computação em Nuvem, ou *Cloud computing*, propõe a integração de diversos modelos tecnológicos para o provimento de infraestrutura de hardware, plataformas de desenvolvimento e aplicações na forma de serviços sob demanda com pagamento baseado em uso. Uma das principais características desse modelo de computação é fornecimento de recursos de forma elástica e dinâmica. A elasticidade pode ser definida como a capacidade de um sistema adicionar ou remover dinamicamente recursos computacionais utilizados por uma determinada aplicação ou usuário de acordo com a demanda. De acordo com o relatório da Comissão Europeia sobre o futuro da computação em nuvem, a elasticidade vertical, que é a possibilidade de alterar a capacidade de um sistema em execução, é uma das áreas pouco exploradas, embora a sua importância no fornecimento de um serviço de nuvem eficiente para qualquer aplicação, seja reconhecida. Neste trabalho abordaremos a elasticidade vertical, focando na elasticidade de memória. Alguns trabalhos têm sido feitos com o objetivo de apresentar soluções para alocação de elástica de memória utilizando técnicas proativas e reativas. Nos métodos reativos, os recursos podem ser adicionados e removidos do ambiente se determinada condição ocorrer. Já nos métodos proativos são utilizadas técnicas analítico-matemáticas e heurísticas, para obter uma previsão sobre o comportamento das cargas do sistema e, a partir desses resultados, tomar decisões de como e quando escalar os recursos para alocação de memória e demais recursos. Há também trabalhos que apresentam solução para elasticidade fornecida por API's. Neste contexto, este trabalho apresenta a implementação e a comparação de 4 técnicas reativas de alocação elástica de memória. Das técnicas avaliadas duas obtiveram bons resultados na maioria dos casos testados, as outras duas técnicas não apresentaram resultados satisfatórios, fornecendo alocações incompatíveis com as demandas de memória.

Palavras-chave: Gerenciamento de memória, Elasticidade, Computação em Nuvem

Capítulo 1

Introdução

A Computação em Nuvem, ou *Cloud computing*, propõe a integração de diversos modelos tecnológicos para o provimento de infraestrutura de hardware, plataformas de desenvolvimento e aplicações na forma de serviços sob demanda com pagamento baseado em uso [4]. O *National Institute of Standads and Technology* (NIST) [5] define Nuvem como sendo uma infraestrutura com capacidade de aprovisionamento de recursos (redes, servidores, armazenamento, aplicações e serviços) de forma rápida e elástica, em alguns casos automática, onde é possível aumentar e diminuir o número de recursos, ou seja, os recursos podem ser alocados ou liberados de acordo com a demanda do usuário, sem que ele precise se preocupar com a sua configuração ou localização física [4]. O modelo de cobrança utilizado para a exploração destes repositórios está baseado em pagamento pelo uso (*pay-per-use*) [6].

Os serviços de computação em nuvem apresentam 5 características essenciais, que são: (1) o amplo acesso à rede, no qual os recursos estão disponíveis através da rede e Internet, estando acessíveis por meio de dispositivos computacionais padrões, promovendo sua utilização por plataformas heterogêneas [6] [7]; (2) a capacidade de fornecer serviços mensuráveis, os serviços de computação em nuvem controlam e otimizam o uso de recursos por meio da sua capacidade de medir esses serviços [7] [2]; (3) autosserviço por demanda, permite que o usuário adquira unilateralmente recursos computacionais na medida em que necessite e sem precisar da interação com os provedores de serviço [2]; (4) conjunto (*pool*) de recursos, os recursos computacionais do provedor (armazenamento, tratamento, memória, rede de interconexão, CPU) são organizados em um pool para servir múltiplos usuários usando um modelo *multi-tenant* [2] [8]. Para isso são utilizados recursos físicos e virtuais atribuídos de forma dinâmica de acordo com a demanda do consumidor [7]; e a (5) elasticidade rápida.

Graças a essa última característica citada, a computação em nuvem tem atraído cada vez mais a atenção da indústria, empresas e instituições de pesquisa, e um dos principais motivos para isso é a possibilidade de aquisição de recursos de uma forma dinâmica e elástica [1]. A elasticidade, que pode ser entendida como a capacidade de um ambiente de se adaptar a carga de trabalho [9], é a capacidade de um sistema adicionar ou remover dinamicamente recursos computacionais utilizados por uma determinada aplicação ou usuário de acordo com a demanda [1]. O conceito de elasticidade pode ser estendido para aplicações [1]. Uma aplicação elástica é aquela capaz de se adaptar às alterações na quantidade de recursos ou de solicitar/liberar recursos de acordo com a sua necessidade [10].

Os recursos alocados de forma dinâmica podem ser novos nós de máquinas virtuais, memória, processadores, etc. O que determina o tipo de recurso que será modificado é a modalidade da elasticidade. Ela decide que tratamento será dado aos novos recursos que serão alocados [11]. De modo geral, a elasticidade pode ser classificada como horizontal ou vertical [1].

Uma nuvem com elasticidade horizontal possibilita apenas a adição ou a remoção dinâmica de Maquinas Virtuais (MV's) da plataforma de computação alocada pelo usuário [1]. Um exemplo típico que utiliza a elasticidade horizontal é um aplicativo baseado na web com n MV's, onde as solicitações de entrada são tratadas por um balanceador de carga que os distribui aos n MV's [12].

A elasticidade vertical é caracterizada pela possibilidade de se alterar a capacidade de MV's em execução [13]. Um mecanismo que oferece suporte à elasticidade vertical, normalmente é implementado através da adição de componentes da máquina virtual, tais como CPU, memória e armazenamento [13] [1]. A elasticidade vertical tem a capacidade de modificar os recursos de um único nó, adicionando ou retirando processadores ou memória, aumentando ou diminuindo sua capacidade de processamento [2] [12].

De acordo com o relatório da Comissão Europeia [14] sobre o futuro da computação em nuvem, a elasticidade vertical é uma das áreas pouco exploradas, embora a sua importância no fornecimento de um serviço de nuvem eficiente para qualquer aplicação seja reconhecida. Neste trabalho abordaremos a elasticidade vertical, focando na elasticidade de memória. Tecnologias atuais de virtualização fornecem suporte para elasticidade de memória por meio de técnicas como a de *balloning* [15]. Essa técnica possibilita que a memória física seja alocada de forma

dinâmica de acordo com as necessidades do ambiente.

Alguns trabalhos têm sido feitos com o objetivo de apresentar soluções reativas de alocação elástica de memória [9], [2], [12], [16], [17], [18], [19], nesse método os recursos podem ser adicionados e removidos do ambiente se ultrapassarem limiares pré-estabelecidos. O trabalho apresento por Galante et al. 2014 [1], onde a elasticidade é fornecida por um API. Alguns autores como [14], [20], propõem uma abordagem preditiva para alocação de memória e demais recursos, as abordagens preditivas utilizam técnicas analítico-matemáticas e heurísticas, para obter uma previsão sobre o comportamento das cargas do sistema e, a partir desses resultados, tomar decisões de como e quando escalar os recursos para alocação de memória e demais recursos.

Neste contexto, o objetivo deste trabalho é a implementação e comparação de métodos reativos de alocação elástica de memória, com o intuito de avaliar o desempenho desses métodos utilizando *traces* de consumo de memória sintéticas e reais, de acordo com a métrica apresentada por Herbst et al. 2013 [21]. Foram implementados quatro algoritmos de alocação de memória testados com diferentes parâmetros, em 12 *traces*, destas, 5 são sintéticas com comportamentos bem definidos, 7 são *traces* de consumo real de memória, 6 foram cedidas pela empresa Constel Tecnologia e uma foi coletada de um computador pessoal.

Através dos testes realizados é possível perceber que os algoritmos de alocação elástica de memória que obtiveram melhores resultados foram os apresentados por Oliveira et al. 2015 [2]/Dawoud et al. 2011 [17]/ Heo et al. 2009 [22] e Moltó et al. 2011 [12], tanto nas *traces* sintéticas quanto nas *traces* reais, cumprindo com o SLA ou qualquer exigência do usuário. Já o algoritmo apresentado por Jenitha et al. 2014 [16], o resultado foi insatisfatório na maioria das *traces* de consumo sintéticas e reais, apresentando bom desempenho apenas nas *traces* com comportamento decrescente. O algoritmo apresentado no trabalho de Baruchi et al. 2010 [9], os resultados descritos não foram alcançados apesar de todas as modificações realizadas na tentativa de melhorar o algoritmo. Na realização da avaliação dos métodos através da métrica apresentada por Herbst et al. 2013 [21] é possível verificar que o algoritmo que obtem melhores resultados em todas as *traces* é o método apresentado por Oliveira et al. 2015 [2].

Esse trabalho está organizado da seguinte maneira: o Capítulo 2 aborda os principais conceitos sobre computação em nuvem, citando uma série de características essenciais de cada

ambiente de nuvem, junto aos modelos de serviço e implantação introduzidos pela literatura. O Capítulo 3 aborda os principais conceito sobre elasticidade, as modalidades de aplicação, sua politica de alocação. O Capítulo 4 apresenta métricas para avaliação de elasticidade. O Capítulo 5 apresenta os resultados dos métodos implementados e a avaliação de cada método utilizando uma métrica.

Capítulo 2

Computação em Nuvem

A computação em nuvem está associada a um novo paradigma para o fornecimento de infraestrutura de tecnologia da informação (TI) [23] . O conceito utilizado atualmente de Computação em Nuvem (*Cloud Computing*) surgiu em 2006, quando Eric Schmidt, CEO da Google, utilizou pela primeira vez o termo para descrever o modelo de negócios de sua empresa [5].

Atualmente não há uma definição unânime de computação em nuvem, o que demonstra que o conceito ainda está em evolução. Para Buyya et al. 2008 [24], Computação em Nuvem é um sistema distribuído e paralelo, constituído por uma coleção de computadores virtuais e físicos interconectados que proveem um ou mais recursos computacionais, baseados em Acordos de Níveis de Serviços (*Service Level Agreement - SLA's*) preestabelecidos através de uma negociação entre consumidor e provedor.

A computação em nuvem, segundo Armbrust et al. 2009 [25], é um conjunto de serviços de rede ativados que proporcionam escalabilidade, qualidade de serviço e infraestrutura barata de computação sob demanda, que pode ser acessada de uma forma simples e pervasiva. No trabalho de Vaquero et al. 2008 [26] são avaliadas mais de 20 definições de computação em nuvem, com o objetivo de obter uma definição unificada. Para o autor, computação em nuvem pode ser definida como um grande conjunto de recursos virtualizados e compartilhados (hardware, plataformas de desenvolvimento ou software) que são facilmente acessados e utilizados. Esses recursos podem ser alterados e reconfigurados de forma dinâmica, para se adaptar a uma carga de trabalho variável. Essa característica permite que a sua utilização seja otimizada. Este conjunto de recursos é explorado por um modelo de pagamento por utilização (*pay-per-use*), no qual garantias são oferecidas pelo provedor de infraestrutura na forma de acordos de SLA's personalizados.

A partir das definições encontradas [24] [25] [26], pode-se concluir que a computação em nuvem não é apenas um modelo de computação, mas é também um modelo de negócios, pois define os papéis e o fornecimento e aquisição dos recursos computacionais. Neste contexto surgem dois tipos de atores: provedores e usuários. Os provedores são os responsáveis pela adequada implantação e operação da infraestrutura e dos mecanismos que permitem que eles ofereçam serviços. Já os usuários são aqueles que se utilizam dos recursos, devendo ou não pagar pelo uso dos mesmos [1]. Os valores cobrados referentes ao grau de utilização dos recursos são estipulados por meio de contrato entre provedor e usuário.

De acordo com o NIST [5], os serviços de computação em nuvem apresentam 5 características essenciais, 3 modelos de serviço e 4 modelos de implementação, apresentados nas Seções 2.1, 2.3 e 2.3 respectivamente.

2.1 Características

As características essenciais são vantagens que as soluções de computação em nuvem oferecem [27]. Estas características servem para melhor identificar e distinguir a computação em nuvem de outros paradigmas e representam algumas das vantagens deste paradigma [6]. A elasticidade rápida de recursos, amplo acesso e medição de serviço são exemplos de características básicas para compor uma solução de computação em nuvem.

- **Amplo acesso à rede**: Os recursos estão disponíveis através da rede e Internet, estando acessíveis por meio de dispositivos computacionais padrões, promovendo sua utilização por plataformas heterogêneas [6] [7], como celulares, laptops, PDA's. Desta forma, a nuvem seria um ponto de acesso centralizado para as necessidades computacionais dos seus usuários, estando disponível o tempo todo e em qualquer lugar [6]. Essa característica é conhecida como ubiquidade, que segundo Sobragi et al. 2012 [28] é um dos fatores que leva a adoção da computação em nuvem, pois possibilita o acesso aos recursos de qualquer lugar utilizando diferentes dispositivos ou aplicações, e a ubiquidade também leva a um senso de independência em relação à localização, pois os clientes na maioria dos casos não conhecem a localização exata dos recursos que estão utilizando e não tem controle sobre esses recursos. Outra vantagem que o amplo acesso a rede traz é que as interfaces que proporcionam o acesso à nuvem não obrigam os usuários a mudar suas

condições e ambientes de trabalho, como por exemplo, linguagens de programação e sistema operacional. Já os sistemas de software clientes instalados localmente para o acesso à nuvem podem ser leves, como um navegador de Internet [27].

- **A capacidade de fornecer serviço mensuráveis**: Analogamente, os sistemas de gerenciamento da nuvem, controlam e aperfeiçoam o uso dos recursos por meio de medições que consideram cada tipo de serviço provido [29]. Esta monitoração agrega transparência tanto para o provedor quanto para o cliente, sendo que normalmente são utilizados nos SLA's para especificar as características dos serviços, parâmetros de qualidade (*QoS – Quality of services*) e determinar os valores que serão cobrados. Um SLA define os níveis de disponibilidade, funcionalidade, desempenho e outros atributos relativos aos serviços e inclui também penalidades para o caso de violação das regras por qualquer uma das partes [6] [29]. Os serviços de computação em nuvem controlam e otimizam os recursos o uso de recursos por meio da sua capacidade de medir esses serviços [7] [2]. Para que o provedor consiga mensurar os recursos, eles são medidos durante um intervalo de tempo pré-determinado, e podem ser alocados ou liberados de acordo com a demanda do usuário, permitindo que o usuário pague somente por aquilo que for utilizar, tornando possível a próxima característica a ser abordada.

- **Uso de Recursos sob demanda**: Permite que o usuário adquira unilateralmente recursos computacionais na medida em que necessite e sem precisar da interação com os provedores serviço [2]; Essa característica possibilita que os usuários paguem somente pelos recursos que realmente utilizarem e pelo tempo em que esses recursos forem efetivamente utilizados, o que podemos chamar de *pay as you go* [30]. O usuário tem um maior controle de gastos ao usar aplicativos, pois a maioria dos sistemas de computação em nuvem fornece aplicações gratuitamente e, quando não gratuitas, são pagas pelo tempo de utilização dos recursos, números de usuários, tempo de acesso e etc. Usuários que utilizam software como serviço, não precisam pagar por manutenção e licença integral de uso de software [1]. Embora pesquisas em computação tenham investigado vários modelos econômicos de infraestrutura computacional durante a última década, a computação em nuvem tem uma abordagem mais aplicada aos negócios e relacionada ao custo [31]. Segundo Sobragi et al. 2012 [28], o maior benefício da computação em nuvem é financeiro, pois o modelo

pay as you go é muito mais barato para uma empresa ou organização do que o modelo de pagamento antecipado, mas a economia gerada pela computação em nuvem pode ter origens distintas, pois uma grande parte dos custos de infraestrutura de TI vem do custo com a eletricidade que é necessário para rodar hardware, PC‘s e servidores, e também para o resfriamento necessário para diminuir o calor gerado pelo hardware. Segundo Marston et al. 2011 [32] a computação em nuvem diminui dramaticamente os custos de entrada no mercado para as pequenas organizações, beneficiando-as com tecnologias que eram até o momento disponíveis apenas para grandes corporações. Ou seja, essa tecnologia promove acesso imediato aos recursos de hardware sem investimentos de entrada pelos usuários, provendo um precioso tempo de mercado em muitos negócios tendo a TI como despesa operacional.

- **Conjunto (pool) de recursos**: Os recursos computacionais do provedor (armazenamento, tratamento, memória, rede de interconexão, CPU) são organizados em um conjunto para servir múltiplos usuários usando um modelo multiusuário (*multi-tenant*) [2] [8]. Para isso, são utilizados recursos físicos e virtuais atribuídos de forma dinâmica de acordo com a demanda do consumidor [7]. Sem esse modelo de vários clientes utilizando o serviço, o custo de manter esse serviço e, posteriormente, o custo para um cliente se inscrever no serviço, poderia ser tão grande que um prestador de serviços não teria retorno suficiente sobre o investimento. A capacidade de ter vários clientes usando as mesmas licenças e a mesma plataforma traz economia em escala. Quanto maior for o número de clientes utilizando o mesmo conjunto de recursos, maior será a redução dos custos gerais que serão repassados para cada cliente [29]. Como dito anteriormente, os usuários não precisam saber a localização física dos recursos computacionais, podendo somente especificar a localização em um nível mais alto de abstração, tais como o país, estado ou centro de dados [27]. Porém, pode ser possível especificar sua prioridade de localização com relação a país e centro de dados através do SLA [6].

- **Elasticidade rápida** A alocação dos recursos ocorre de forma dinâmica, em tempo de execução de acordo com a necessidade do usuário. Essa característica gera ao usuário a impressão de que os recursos disponíveis são ilimitados [5] [19]. Essa característica será abordada de forma mais completa no Capitulo 3, pois o objetivo deste trabalho é

realizar um estudo comparativo de técnicas de alocação elástica de memória em nuvens computacionais.

2.2 Modelos de Serviço

O modelo conceitual encontrado com maior frequência na literatura é composto por três camadas [9], e de acordo com o nível de abstração e pelos tipos dos recursos fornecidos pelo provedor, podemos dividir os modelos de serviço de computação em nuvem da seguinte forma: (1) Software como Serviço (*Software as a Service - SaaS*); (2) Plataforma como Serviço (*Platform as a Service - PaaS*); e (3) Infraestrutura como Serviço (*Infrastructure as a Service - IaaS*) [1]. A Figura 2.1, retirada de Galante et al. 2014 [1], apresenta a organização das camadas que compõem uma nuvem computacional.

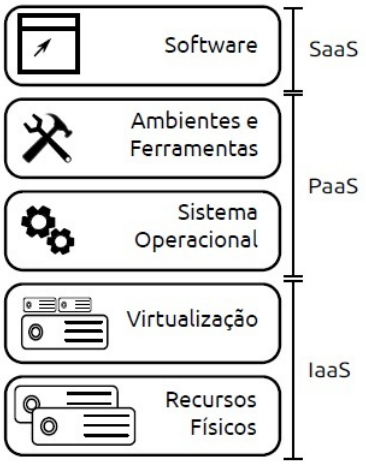

Figura 2.1: Camadas de serviço em nuvem. Adaptado de [1].

2.2.1 Softwares como Serviço *(Software as a Service - SaaS)*

Corresponde a camada mais externa do modelo conceitual, ela é composta por aplicativos que são executados no ambiente da nuvem. Podem ser aplicações completas ou conjuntos de aplicações cujo uso é regulado por modelos de negócios que permitem personalização [6].

Tais aplicações podem ser acessadas por vários dispositivos (PCs, smartphones, tablets) através de uma interface simples de cliente (*thin client*), tal como um navegador web [1] [6].

O modelo de SaaS proporciona sistemas de software com propósitos específicos que estão disponíveis para os usuários através da Internet [27] através de uma interface com um navegador web. Logo devem ser acessíveis de qualquer lugar a partir dos diversos dispositivos dos usuários. Desta forma, novos recursos podem ser adicionados aos sistemas de forma transparente aos usuários, tornando-se assim a manutenção e evolução dos sistemas tarefas bem mais simples [6].

No modelo de SaaS, o cliente não gerencia ou controla a infraestrutura da nuvem, nem mesmo a camada de plataforma, incluindo rede, servidores, sistemas operacionais, armazenamento ou mesmo as características individuais da aplicação, exceto configurações específicas. Com isso, os desenvolvedores se concentram em inovação e não na infraestrutura, levando ao desenvolvimento rápido de sistemas de software [27]. É possível que em alguns casos o usuário configure parâmetros específicos de personalização. No entanto, o provedor mantém controle administrativo sobre a aplicação, cabendo-lhe a instalação, configuração, atualização e gerenciamento de modo a assegurar o funcionamento adequado da mesma [1].

Os exemplos mais comuns de implementações de SaaS são as redes sociais Facebook [1] e Twitter [2], portais de fotos e vídeos como Youtube [3] e Flickr [4] e aplicações desenvolvidas pela Google (gmail, Docs e Maps) [5].

2.2.2 Plataforma como Serviço *(Platform as a Service - PaaS)*

Essa é a camada intermediaria do modelo conceitual e está relacionada aos serviços de plataforma (PaaS) oferecendo tipos específicos de serviços como sistemas operacionais, banco de dados, serviços de mensagens, serviços de armazenamento de dados, etc. Nela são fornecidos ambientes de programação, linguagem de alto nível e um conjunto de *Application Programming Interface* (APIs) bem definidas para facilitar a interação entre os ambientes e as aplicações na nuvem o que facilita para projetar e desenvolver aplicações, testes, implantação, hospedagem, segurança, integração de banco de dados, persistência integração de serviços web, etc. Estes serviços também podem ser configurados como uma solução integrada, oferecida através da

[1] Disponível em: http://www.facebook.com
[2] Disponível em: https://www.twitter.com
[3] Disponível em: http://www.youtube.com
[4] Disponível em: http://www.flickr.com/
[5] Disponível em: http://www.google.com

Internet, pois é possível hospedar e ofertar essas aplicações desenvolvidas pelo usuário para seus clientes e também oferece ferramentas para administrar e monitorar as aplicações por eles instaladas [1] [6] [33].

A vantagem desse serviço é que a implantação de uma nova aplicação de software em uma PaaS é tão simples quanto o *upload* de um arquivo para um servidor Web [33].

Os usuários de PaaS não gerenciam e nem controlam a infraestrutura da nuvem, incluindo rede, servidores, sistemas operacionais ou armazenamento. Geralmente eles possuem acesso apenas às ferramentas de desenvolvimento, a suas aplicações e respectivas configurações [1].

Do ponto de vista do negócio, a PaaS permite aos usuários utilizarem serviços de terceiros, aumentando o uso do modelo de suporte no qual os usuários se inscrevem para solicitações de serviços de TI ou para resoluções de problemas pela Web. Com isso, pode-se melhorar o gerenciamento do trabalho e as responsabilidades das equipes de TI das empresas [27].

Outra vantagem oferecida pelo modelo de PaaS é que, mesmo os dados estando fisicamente espalhados pela rede do provedor ou entre vários provedores, toda gerência de dados, incluindo os de desenvolvimento, é vista pelo usuário de forma centralizada [1].

Em geral, os desenvolvedores dispõem de ambientes escaláveis, mas eles têm que aceitar algumas restrições sobre o tipo de software que se pode desenvolver, desde limitações que o ambiente impõe na concepção das aplicações até a utilização de sistemas de gerenciamento de banco de dados (SGBDs) do tipo chave-valor, ao invés de SGBDs relacionais [27]. Como principal desvantagem pode-se citar o fato do usuário ficar dependente da plataforma, não podendo migrar suas aplicações para outra nuvem (*lock-in*). [1] [6] [27].

Exemplos de fornecedores de PaaS são o Microsoft Azure [6], o Google Apps Engine [7], e o Salesforce [8].

2.2.3 Infraestrutura como um Serviço *(Infrastructure as a Service - IaaS)*

Representa a camada inferior do modelo conceitual. Sua base é composta por plataformas para o desenvolvimento, teste, implantação e execução de aplicações proprietárias.

Os recursos físicos da nuvem, juntamente com as funcionalidades fornecidas pelos geren-

[6]Disponível em: https://azure.microsoft.com/pt-br/
[7]Disponível em: https://appengine.google.com
[8]Disponível em: http://www.salesforce.com

ciadores de virtualização, formam a base para entrega de serviço do tipo IaaS. Os recursos são fornecidos por centros de dados (*datacenters*), que contêm um conjunto de recursos físicos interligados, tais como servidores, armazenamento, dispositivos de rede [1] e outros recursos fundamentais para construir um ambiente sob demanda, que pode incluir sistemas operacionais e aplicativos [27]. Estes servidores são transparentemente gerenciados por meio de virtualização, permitindo que suas capacidades sejam compartilhadas entre as instâncias virtuais [1].

A IaaS possui algumas características, tais como uma interface única para administração da infraestrutura, API para interação com switches, hosts, roteadores e o suporte para a adicionar novos equipamentos de forma simples e transparente. Em geral, o usuário não administra ou controla a infraestrutura da nuvem, mas tem controle total sobre seu ambiente virtual, sendo o responsável por operar, atualizar e configurar os recursos com objetivo de atingir os níveis de desempenho, de segurança e de confiabilidade desejados [1], e, eventualmente, seleciona componentes de rede, tais como *firewalls*.

O usuário recebe os recursos requisitados abstraídos por meio de máquinas virtuais, que são geralmente administradas através de protocolos de acesso remoto (SSH, por exemplo) [1].

A infraestrutura baseia-se na virtualização dos recursos computacionais que podem ser dinamicamente escalados para aumentar ou diminuir os recursos de acordo com as necessidades das aplicações [6].

Máquinas virtuais são frequentemente utilizadas, pois simulam os recursos computacionais de uma máquina real e tornam o ambiente virtualizado criado independente de outras máquinas virtuais e do hardware [24]. Um equipamento computacional virtualizado pode hospedar diversas MV's simultaneamente, compartilhando seus recursos de armazenamento, processamento e comunicação. Quem gerencia o compartilhamento de recursos é um sistema chamado de hipervisor (ou *hypervisor*). Ele gerencia o ciclo de vida das máquinas virtuais e permitir uma alocação de recursos configurável. Um hipervisor é uma camada que acessa o hardware e apresenta ao sistema operacional um conjunto virtual de recursos de hardware [12]. O hipervisor permite que múltiplas MV's sejam executadas e é responsável por escalonar o acesso dessas MV's aos recursos de hardware, provendo um acesso consistente a CPU, memória e dispositivos de Entrada e Saída (E/S) [13]. Alguns desses hipervisores tem como funcionalidades a alocação dinâmica de máquinas virtuais, a consolidação de servidores, o redimensionamento dinâmico

da infraestrutura e o balanceamento de carga automático.

Esse compartilhamento e consolidação de recursos proporciona uma redução de custos com consumo de energia, equipamentos de refrigeração, administração e gerenciamento e, ao mesmo tempo, aumenta a possibilidade de exploração e utilização do substrato computacional [34].

De acordo com Borges et al. 2011 [6] algumas vantagens de trabalhar com IaaS são:

- Liberação de espaço físico na empresa;
- A redução de investimentos em hardware, bem como a preocupação com a depreciação dos mesmos;
- Otimização do desempenho;
- Eliminação de custos com segurança e manutenção;
- Flexibilidade para ampliar e reduzir a capacidade de processamento e/ou armazenamento.

Entre os principais provedores de nuvem públicas de IaaS pode-se citar o Amazon EC2 [9], Eucalyptus [10], Rackspace [11], GoGrid [12] e CloudSigma [13]. Para a implementação de IaaS em nuvens privadas estão disponíveis diversas ferramentas de código aberto, entre elas o Eucalyptus, OpenNebula [14] e OpenStack [15] [1].

2.3 Modelos de Implantação

Devido as diversas abordagens de computação em nuvem existem variados modelos de implantação disponíveis na literatura que consideram diferentes ambientes de acordo com a disponibilidade e acesso [6] [27]. A abertura ou restrição de acesso depende do processo de negócio, do tipo de informação e do nível de visão. Para algumas empresas não é desejável que todos os usuários possam utilizar e acessar determinados recursos no seu ambiente de computação

[9]Disponível em: https://aws.amazon.com/pt/ec2/
[10]Disponível em: http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html
[11]Disponível em: https://www.rackspace.com/pt
[12]Disponível em: https://www.gogrid.com
[13]Disponível em: https://www.cloudsigma.com/
[14]Disponível em: http://opennebula.org/
[15]Disponível em: http://www.openstack.org/

em nuvem. Neste sentido, surge a necessidade de ambientes mais restritos, onde somente alguns usuários devidamente autorizados possam utilizar os serviços providos. Os modelos de implantação da computação em nuvem podem ser divididos em nuvem pública, privada, híbrida e comunitária [6] [27].

2.3.1 Nuvem Privada

No modelo de implantação de nuvem privada, a infraestrutura de nuvem é utilizada exclusivamente para uma organização ou empresa, sendo esta nuvem local ou remota e administrada pela própria empresa, implantadas dentro de *firewalls*, ou por terceiros [27] [35].

No caso de ser administrada pela própria empresa, a nuvem oferece todas as funções básicas da computação como a flexibilidade, escalabilidade, aumento de produtividade, acesso remoto, entre outras, mas o acesso a esses recursos fica restrito a apenas a companhia ou um grupo especifico, sem que haja compartilhamento de recursos com outras empresas ou usuários fora do ambiente corporativo [27] [35]. Nesse formato, é a própria empresa que integra todos os departamentos e áreas com o modelo de Computação em Nuvem, realizando a instalação e manutenção da infraestrutura e da plataforma. Com isso, a nuvem privada se utiliza de uma rede que provê uma experiência de uso confiável, possibilitando o armazenamento e o acesso à informações e dados corporativos com segurança. Empresas que atuam em segmentos altamente regulamentados ou que trabalham com informações confidenciais, têm uma preocupação maior com segurança que precisa ser atendida. Nesse caso, escolher uma nuvem privada pode ser a opção certa, pois seu principal objetivo é justamente o de fornecer mais estabilidade para o armazenamento dos dados corporativos na nuvem, garantindo controle total sobre o ambiente com menor risco de ameaças vindas de terceiros, e fornecendo o acesso em qualquer lugar que o colaborador esteja. Outra vantagem de uma nuvem privada é que ela possui uma alta capacidade de customização, já que é possível aumentar a eficiência de servidores e de Data Centers, diminuindo custos de implantação, aumentando a produtividade da companhia, simplificando as operações e infraestrutura [6] [27] [35].

No caso de a nuvem ser fornecida por um terceiro, a organização pode utilizar redes seguras do tipo *VPN* para acessar a infraestrutura [18]. Neste modelo de implantação são empregados políticas de acesso aos serviços. As técnicas utilizadas para prover tais características podem

ser em nível de gerenciamento de redes, configurações dos provedores de serviços e a utilização de tecnologias de autenticação e autorização [27].

A dificuldade e custo para se estabelecer uma nuvem privada podem às vezes ser proibitivos para empresas de pequeno e médio porte. O custo de operação contínua da nuvem pode exceder o custo de uso de uma nuvem pública. Por isso, torna-se necessária uma boa análise das necessidades da empresa quanto à Computação em Nuvem. [6] [35].

2.3.2 Nuvem Pública

O modelo de Nuvem Pública é um serviço prestado por um fornecedor a usuários comuns ou a empresas, através da Internet. Nesse modelo, a infraestrutura pertence a uma organização que vende serviços para o público e pode ser acessada por qualquer usuário que conheça a localização do serviço, não sendo utilizadas técnicas de restrição de acesso ou autenticação. As nuvens públicas têm por objetivo fornecer aos clientes elementos de TI livres de complexidades, onde o provedor da nuvem assume as responsabilidades de instalação, gerenciamento, disponibilização, proteção e manutenção e cobra desses clientes apenas pelos recursos utilizados, sejam esses recursos de aplicação, infraestrutura física ou softwares [6] [27] [35] [8].

Esse serviço é compartilhado com outras empresas ou usuários. Com isso, sua companhia tem total controle sobre o que faz e registra na Nuvem, mas não sobre as ações dos demais no ambiente[6] [27] [35] [8].

Nesse modelo de implantação, SLA's são publicados pelo provedor e são da sua responsabilidade. Os provedores normalmente possuem mecanismos de replicação de dados para vários locais geograficamente dispersos, às vezes por um custo extra. Este tipo de Nuvem oferece várias vantagens aos clientes, como exemplo, a redução de investimento inicial, otimização de custos operacionais e a transferência de riscos tecnológicos para o fornecedor da infraestrutura [6] [27] [35] [8].

É possível usufruir desse serviço de forma efetiva, porém, é um modelo que tem como um de seus benefícios a redução de custos e, portanto, é uma boa alternativa para empresas de pequeno e médio porte, que tenham um orçamento restrito ou outras prioridades. Para empresas que trabalham com um grande volume de dados confidenciais, essa pode não ser a melhor solução. Outro problema enfrentado ao utilizar esse tipo de nuvem é em relação a limitação da

largura de banda, que varia em torno de 10MB, já que esse serviço está disponível pela Internet [6] [27] [35] [8].

2.3.3 Nuvem Comunitária

No modelo de implantação de nuvem comunitária ocorre o compartilhamento por diversas organizações de uma nuvem, sendo esta suportada por uma comunidade específica que partilhou seus interesses, tais como a missão, os requisitos de segurança, juridição, política e necessidades sobre flexibilidade e compatibilidade. Este tipo de modelo de implantação pode ser administrado internamente ou por terceiros e podem ser hospedados remotamente ou localmente [6] [8] [29].

Instituições acadêmicas e mercados verticais, em particular podem se beneficiar de nuvens comunitárias para abordar preocupações comuns. Por exemplo, empresas de tecnologia que trabalham juntas em uma nova especificação podem utilizar uma nuvem comunitária para compartilhar recursos e provas de conceito [36]. Segundo Felipe et al. 2012 [37], nesta arquitetura a segurança encontra-se em nível intermediário se comparada com as anteriores.

2.3.4 Nuvem Híbrida

No modelo de nuvem híbrida a infraestrutura geral é composta por pelo menos duas nuvens, que podem ser privada, comunitária ou pública, e que permanecem como entidades únicas e preservam as características originais do seu modelo, porém estão interligadas por uma tecnologia que possibilite a portabilidade de informações e de aplicações [6] [8] [27] [35].

Uma nuvem híbrida bem construída pode atender processos seguros que necessitam de um cuidado maior, pois pode ser utilizada uma nuvem privada para garantir a segurança por meio de uma rede exclusiva instalada na companhia. Por exemplo, sistemas críticos ou que manipulam informações confidenciais, como recebimentos de pagamentos de clientes, podem ser hospedados internamente enquanto outros sistemas, que não lidam com dados sigilosos, como seria o caso do processamento de folha de pagamento, podem ser utilizados em uma rede pública [6] [35] [8].

Quando uma empresa demanda de escalabilidade e precisa de capacidade extra de um servidor somente durante um período em particular, a nuvem híbrida consegue atender uma demanda

irregular mais facilmente, devido à sua escalabilidade dinâmica. Esse formato de nuvem permite que uma empresa estabeleça a melhor formação para o seu modelo de negócios, pois aprimora o controle interno de aplicações que a empresa necessita, analisando qual é a melhor opção [8] [35].

A principal limitação deste modelo de computação em nuvem é a dificuldade em se criar e administrar uma solução deste porte. Serviços de fontes diferentes devem ser obtidos e disponibilizados como se fossem originados de um único local, e as interações entre componentes públicos e privados podem tornar a implementação ainda mais complexa [6].

Capítulo 3

Elasticidade

A elasticidade é uma das principais características da computação em nuvem e pode ser entendida como a capacidade de um ambiente de se adaptar a carga de trabalho [9]. Segundo Galante et al. 2014 [10], esta característica consiste na capacidade de adicionar ou remover recursos, sem interrupções e em tempo de execução, de acordo com uma demanda específica. O conceito de elasticidade pode ser estendido para aplicações [1]. Uma aplicação elástica é aquela capaz de se adaptar às alterações na quantidade de recursos ou de solicitar/liberar recursos de acordo com a sua necessidade [10].

A elasticidade é vista de forma diferente pelo consumidor de serviços da nuvem e pelo provedor destes serviços. O usuário ou consumidor de serviços da nuvem não olha o interior das tecnologias da nuvem, mas apenas visualiza a sua interface. Ele interage com uma nuvem normalmente pelo portal de acesso no qual solicita provisionamento e alocação dos recursos computacionais, e os detalhes técnicos ficam escondidos. Já o provedor precisa colocar em operação toda a tecnologia necessária para que a elasticidade seja fornecida [4] [36].

Em nuvens privadas, a empresa precisa adquirir e implementar os recursos computacionais que serão alocados através do modelo da nuvem para os seus usuários. A elasticidade é sentida apenas no nível dos usuários internos da nuvem privada, mas não no Data Center, que precisa ainda investir em capital, como servidores e sistemas operacionais. Uma gestão dinâmica de recursos acelera a velocidade com que estes recursos são provisionados para seus usuários e de acordo com Jesus et al. 2012 [36], diminui em muito a ociosidade média dos servidores em grandes Data Centers.

A alocação elástica de recursos é dependente do comportamento do serviço (ou aplicação) e do nível de serviço (SLA) contratado. Já a alocação estática provisiona recursos sempre para

o pior caso (momentos de pico), correndo o risco de essa previsão ser subestimada. Ainda, há a questão do desperdício de recursos, dado que momentos de pico não caracterizam a carga média no sistema. Ou seja, a alocação elástica traz benefícios tanto para o administrador da nuvem quanto para o usuário, pois o primeiro consegue um melhor uso dos recursos da infraestrutura, uma vez que aqueles liberados por um usuário podem ser repassados para outro sob demanda, enquanto o segundo consegue manter o desempenho de seu serviço em execução em níveis aceitáveis, pagando um preço adequado para isso [38]. Na Figura 3.1 é possível perceber que na alocação estática as MV's 2 e 4 tem mais recursos disponíveis do que necessitam, enquanto as MV's 3 e 5 possuem menos recursos do que o necessário pra funcionar. Já na alocação elástica, os recursos são distribuídos de acordo com a demanda de cada MV sem que aja prejuízo no funcionamento de nenhuma delas.

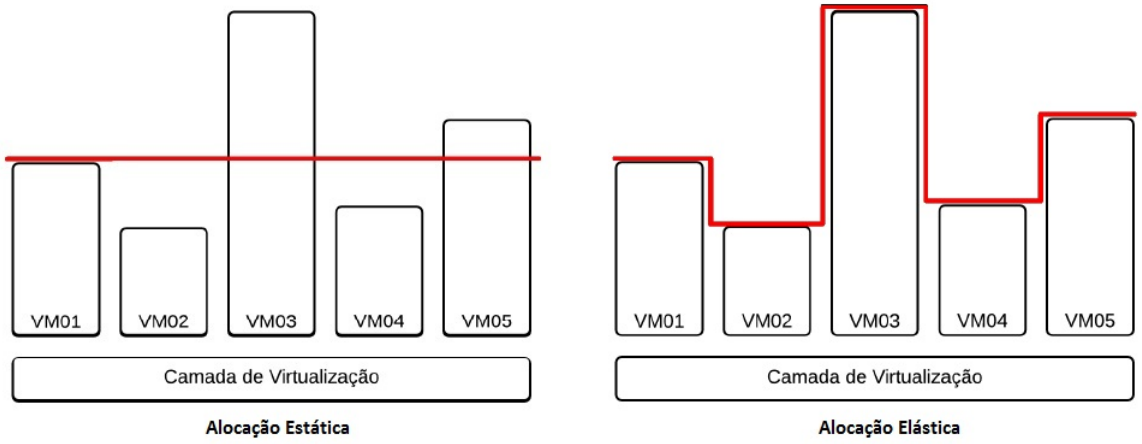

Figura 3.1: Alocação estática e alocação elástica. Adaptado de [2].

Os recursos alocados de forma dinâmica podem ser novos nós de máquinas virtuais, memória, processadores, etc. O que determina o tipo de recurso que será modificado é a modalidade da elasticidade, ela decide que tratamento será dado aos novos recursos que serão alocados [11].

3.1 Classificação das Soluções de Elasticidade

De modo geral, a elasticidade pode ser classificada como horizontal (acréscimo ou diminuição de máquinas virtuais ou nós) ou vertical (redimensionamento de recursos) [1], como mostra a Figura 3.2.

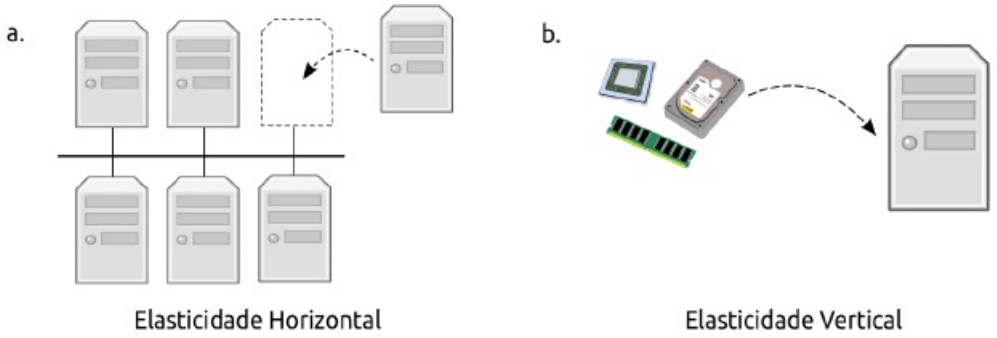

Figura 3.2: Elasticidade Horizontal e Vertical. Adaptado de [1].

A elasticidade do tipo horizontal, é também conhecida como replicação. Ela é a mais comum de se encontrar nos sistemas hoje em dia e consiste em, a partir de um crescimento na demanda, criar novas máquinas virtuais, com a mesma configuração da MV original, para dividir a carga de trabalho entre todas elas. Este é o único método elástico que fornece um modelo de alocação baseado em conjuntos fixos, uma vez que é impossível alterar a configuração de uma MV [38].

Uma nuvem com elasticidade horizontal possibilita apenas a adição ou a remoção dinâmica de MV's da plataforma de computação alocada pelo usuário [1]. Um exemplo típico que utiliza a elasticidade horizontal é um aplicativo baseado na web com n MV's, onde as solicitações de entrada são tratadas por um balanceador de carga que os distribui aos n MV's [12].

A vantagem deste tipo de proposta consiste em uma escalabilidade praticamente infinita, embora existam limitações de software ou outros atributos da infraestrutura. Como vantagem, ela fornece balanceamento de carga e disponibilidade. Em relação as limitações, pode-se citar o exemplo da nuvem pública IaaS da Amazon (EC2) que possui um limite de 20 instâncias (MV's) que podem ser reservadas por um usuário. Uma forma de aumentar o número de instâncias seria pagando mais ao contratar o serviço *on demand*.

A elasticidade vertical é caracterizada pela possibilidade de se alterar a capacidade de MV's em execução [13]. Um mecanismo que oferece suporte à elasticidade vertical, normalmente é implementado através da adição de componentes da máquina virtual, tais como CPU, memória e armazenamento [13] [1]. A elasticidade vertical tem a capacidade de modificar os recursos de um único nó, adicionando ou retirando processadores ou memória, aumentando ou diminuindo sua capacidade de processamento [2] [12].

Em um ambiente virtualizado, os recursos físicos (processador e memória principal) são gerenciados pelo hipervisor. A elasticidade vertical é suportada por hipervisores como VMWare VSphere, KMV, Xen e Microsof HiperV, como mostra a Tabela 3.1.

Tabela 3.1: Suporte à elasticidade vertical oferecida pelos principais hipervisores [1].

Hipervisor	Características
vSphere	CPU (adição e remoção), memória (via *ballooning*), discos (expansão e adição), interfaces de rede (adição), e PCI express.
KMV	CPU (somente adição), memória (*ballooning*), disco (adição), interfaces de rede (adição).
Xen	CPU (adição e remoção), memória (*ballooning*), discos (expansão e adição), interfaces de rede (adição).
HiperV	memória (*ballooning*), discos (expansão e adição).

A elasticidade também é classificada de acordo com a política de alocação de recursos, podendo ser classificada em manual e automática [6] [39].

Um sistema com uma política manual, o usuário é o responsável por monitorar o seu ambiente virtual e suas aplicações e executar todas as ações de elasticidade pertinentes. A interação usuário-nuvem é feita através de uma interface. O provedor de nuvem deve fornecer, pelo menos, uma interface (geralmente uma API) com os quais o utilizador interage com o sistema. Alguns provedores públicos, tais como, GoGrid, Rackspace e Microsoft Azure, são exemplos de sistemas em que os recursos são geridos manualmente [39] [13].

Em sistemas de política automática, o controle e as ações são tomadas por um controlador de elasticidade, de acordo com regras e configurações feitas pelo usuário ou definidas pelo contrato de serviço (SLA). O controlador de elasticidade utiliza informações sobre a carga de trabalho, uso de memória e CPU, tráfego de rede, entre outros, para tomar decisões de quando, como e quanto alocar/liberar os recursos. Estas informações podem ser coletadas pela própria aplicação ou por um sistema de monitoramento [39] [13]. É possível classificar as técnicas utilizadas para acionar as ações de elasticidade automáticas em reativa e preditiva.

Mecanismos reativos com suporte à elasticidade vertical empregam mecanismos do tipo Regra-Condição-Ação, onde cada condição considera um evento ou uma métrica do sistema que é comparado com um limiar determinado e quando uma condição definida em alguma dessas regras é satisfeita, uma ação é disparada. Um limite superior é usado para a alocação

de recursos, enquanto o inferior é útil para a liberação de recursos [2] [1] [39] como mostra a Figura 3.3.

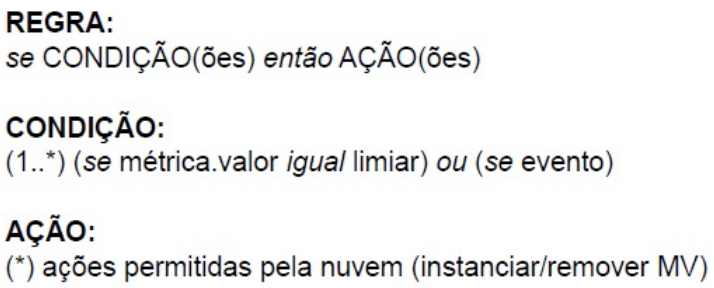

Figura 3.3: Mecanismo Regra-Condição-Ação. Adaptado de [1].

As abordagens preditivas utilizam técnicas analítico-matemáticas e heurísticas, para obter uma previsão sobre o comportamento das cargas do sistema e, a partir desses resultados, tomar decisões de como e quando escalar os recursos. Essas técnicas podem ser análises de séries temporais, teoria de filas, Transformada Rápida de Fourier (FFT), entre outras. A abordagem proativa é mais apropriada para os casos onde a carga de trabalho apresenta padrões bem definidos com periodicidade uniforme, facilitando a previsão de cargas futuras. Um gerenciador da elasticidade proativo pode antecipar a reconfiguração de recursos e, assim, reduzir o tempo de conclusão da aplicação [1] [11] [40].

3.2 Elasticidade de Memória

A elasticidade de memória é vantajosa para a computação em nuvem, pois cada aplicação e cada MV possui requisitos de memória diferentes. Para os provedores de computação em nuvem a possibilidade de implementar a elasticidade de memória representa uma oportunidade para melhorar a consolidação de servidores e, assim, reduzir ainda mais o consumo de energia e os custos de administração [41].

Segundo Moltó et al. 2011 [12], a capacidade de modificar a memória de uma MV em tempo de execução, sem qualquer interrupção de serviço, representa um recurso importante para aplicação que possuem requisitos dinâmicos de memória. Adaptar a memória de uma aplicação de forma dinâmica requer uma observação precisa e frequente da memória livre. Para isso, um projeto adequado de normas de elasticidade é necessário para diminuir e aumentar o

tamanho da memória da MV sem ocasionar um provisionamento excessivo de memória nem *thrashing* [1].

Em ambientes virtualizados o processo implementado pelos hipervisores para a alocação de memória funciona da mesma forma do que a alocação feita por sistemas operacionais, ou seja, a memória é disponibilizada para os sistemas operacionais convidados como se eles fossem aplicações normais e posteriormente esses endereços virtuais são mapeados para endereços físicos. O hipervisor fornece para cada Sistema Operacional em execução, um conjunto de endereços virtuais lineares e contíguos. Esses endereços apresentam o mesmo limite e base para todos os SO's considerados e com auxilio do MMU (*Memory Management Unit*), que é disponibilizado pelo hardware. Esses endereços são divididos em páginas e convertidos em endereços físicos, que são divididos em *frames*[2] de paginas do mesmo tamanho da página virtual, onde os dados estão realmente localizados [42].

Neste contexto podemos destacar quatro técnicas de alocação elásticas de memória de acordo com [33]:

1. Compartilhamento de páginas, que é uma técnica destinada a reduzir o número de páginas utilizadas em um SO hospedeiro, isso é alcançado através da eliminação de páginas duplicadas (compartilhadas entre as MV's), da memória física do hospedeiro;

2. Compressão de memória, essa técnica recupera de forma rápida a memória evitando as operações de *Swapping* [3] no âmbito do hipervisor uma vez que o acesso ao disco é extremamente mais lento que o tempo consumido ao comprimir as páginas;

3. *Swapping* do hipervisor, que é semelhante ao processo de *swapping*, a diferença está no fato de o sistema de virtualização selecionar páginas de uma máquina virtual especifica

[1] *Thrashing* pode ser definido como sendo a excessiva transferência de páginas/segmentos entre a memória principal e a memória secundária. Esse problema está presente em sistemas que implementam tanto paginação como segmentação. É a situação onde uma grande quantidade de recurso computacional é utilizada para fazer uma quantidade mínima de trabalho, com o sistema em um estado contínuo de contenção de recursos. O *trashing* no sistema ocorre quando existem mais processos competindo por memória principal que espaço disponível.

[2] *Frames*: no contexto dos sistemas operacionais, a paginação da memória do computador é um processo de virtualização da memória que consiste na subdivisão da memória física em pequenas partições (*frames*), para permitir uma utilização mais eficiente da mesma. As (*frames*) da memória física correspondem a páginas de memória virtual.

[3] *Swapping* é uma técnica aplicada à gerência de memória, explorando o conceito de Memória Virtual, onde o sistema escolhe um programa residente que é levado da memória para o disco (*swap out*), retornando posteriormente para a memória principal como se nada tivesse ocorrido (*swap in*).

para ser enviada ao disco, sendo tais páginas, inclusive, armazenadas em arquivos separados por máquinas virtuais;

4. *Ballooning*, nessa técnica a memória física é alocada de forma dinâmica de acordo com as necessidades do ambiente. Ela é realizada através da instalação de um pseudo-*driver* no núcleo do Sistema operacional convidado, criando um canal de comunicação entre o sistema operacional que roda na máquina virtual e o hipervisor, notificando a sua demanda real de memória. Este *drive* estimula os algoritmos de paginação do SO instalado na MV a procurar páginas de memória descartáveis. Ao necessitar de memória no SO hospedeiro o hipervisor "infla o balão" no interior do SO convidado, fazendo com que o mesmo aloque páginas, que inicialmente estavam em cache, para a aplicação *ballooning*. A etapa de "desinflar o balão" acontece à medida que a demanda por memória diminui. O hipervisor se comunica através do *driver*, com a aplicação balão, solicitando que a mesma libere a memória anteriormente reservada [42] [43]. Se o balão se expande, a memória física disponível na MV é reduzida, o que obriga o SO convidado a reduzir o consumo de memória de outros processos, por exemplo, pela passagem de algumas de suas páginas de memória para o espaço de *swap*. Então, a memória alocada pelo processo de balão no SO convidado pode ser recuperada pelo SO hospedeiro, que pode ser eventualmente usado por outras MV's.

Alguns trabalhos têm sido feitos com o objetivo de apresentar soluções para alocação de elástica de memória. Na Seção 3.3 apresenta-se e descrevem-se as principais propostas descritas na literatura técnica com o objetivo de estabelecer o estado-da-arte da elasticidade de memória em nuvens computacionais.

3.3 Elasticidade de Memória - Estado da Arte

Esta seção tem o objetivo de apresentar o estado-da-arte das soluções de elasticidade em memória provenientes de pesquisas acadêmicas.

Oliveira et al. 2015 [2] e Dawoud et al. 2011 [17] propõem em seus trabalhos gerenciadores para alocação elástica de memória em ambientes virtuais reativos, baseados no trabalho de Heo et al. 2009 [22] que apresenta uma arquitetura para provisionamento elástico de memória e

CPU. Nesta plataforma existem dois controladores, um para CPU e outro para memória, que analisam a taxa de uso de seu respectivo recurso para realizar alocações ou desalocações. A alocação dinâmica de memória utiliza *ballooning* e a alocação de CPU é implementada usando o *Xen Credit Scheduler*. Esses gerenciadores de memória (*Memory Manager*) implementados com suporte do *Xen Balloon Driver* visam alterar dinamicamente a alocação de memória para as máquinas virtuais. Inicialmente cada MV é configurada com um valor máximo de memória (maxmem), que representa a soma do valor do balão e do valor disponível para alocação. O atuador de memória realiza o teste de limiar e aloca mais memória de forma agressiva quando a memória utilizada está próxima da saturação (por exemplo, 90%), e diminui de forma lenta a alocação de memória quando a alocação está em uma região longe de gargalos. Esse teste de limiar é realizando utilizando as fórmulas 3.1 e 3.2.

$$u_{mem}(k+1) = u_{mem}(k) + \lambda \times v_{mem}(k) \times (u_{ref}^{mem} - r_{mem}(k))/u_{ref}^{mem} \tag{3.1}$$

$$r_{mem} = \frac{v_{mem}(k)}{u_{mem}(k)} \tag{3.2}$$

Assim, o atuador calcula a próxima alocação de memória $u_{mem}(k+1)$, baseando-se na última alocação de memória $u_{mem}(k)$ e na memória consumida $v_{mem}(k)$. O parâmetro λ é a constante de incremento, que determina a agressividade do atuador. O fator $r_{mem}(k)$ é uma média entre a memória usada e a memória alocada no tempo $k-1$. A desvantagem da abordagem é a necessidade da definição manual do valor de referência sobre a taxa de utilização de memória u_{ref}^{mem}, necessária para manter um determinado SLA. Os resultados obtidos mostram que o atuador de memória trouxe um ganho de desempenho considerável em praticamente todas as aplicações testadas [2].

No trabalho de Moltó et al. 2013 [12] apresenta-se um mecanismo reativo no qual realiza-se o monitoramento do uso de memória da MV e, caso o percentual de memória livre esteja fora de determinada faixa, pode-se alocar ou liberar memória usando o *balloon driver* do hipervisor KMV. Nesse mecanismo o gerenciador de memória monitora as informações sobre a utilização de memória a cada 5 segundos. A memória utilizada por uma MV é estimada subtraindo-se a memória livre, a memória em cache e em buffers da memória total. As regras de elasticidade são aplicadas se a porcentagem de memória livre da MV é menor que 80% ou maior que 120% do *Memory Overprovisioning Percentage*, o MOP - que é a porcentagem de memória livre da

MV que o usuário define na inicialização da MV, por exemplo, se o usuário escolher um MOP = 10%, quer dizer que o mecanismo de alocação irá ser acionado quando a memória livre da MV for menor ou maior do que 10% do total de memória disponível para aquela MV, se a quantidade de memória livre for menor do 8% o mecanismo aloca mais memória, se for maior do que 12% o mecanismo libera memória. Assim a fórmula que define o tamanho da memória a ser alocada é:

$$VMmemorysize = usedmemory \times (1 + MOP) \tag{3.3}$$

Os testes foram realizados com valores de MOP de 10% e 30%, e com duas amostras de consumo de memória, uma que cresce o consumo rapidamente e decresce de forma lenta (GFSL), e outra amostra que cresce o consumo de forma lenta e decresce de forma rápida (GSSF). A abordagem de 10% visa reduzir a memória não utilizada da VM, mas tem uma maior chance de ocorrer *thrashing*. Já o valor de MOP de 30% visa reduzir a chance de ocorrer *thrashing* caso o consumo de memória cresça rapidamente.

A amostra de MOP de 30% obteve melhores resultados na alocação de memória, sem ocasionar *thrashing* e sem perda de desempenho para aplicação. Porém a desvantagem de um MOP tão alto é que a quantidade de memória provisionada é maior do que o requisitado pela aplicação.

No trabalho apresentado Baruchi et al. 2010 [9], é apresentado um mecanismo de alocação de memória dinâmica em MV's utilizando o método média móvel exponencial (MME) que define quando a memória deve ser alocada e quando deve ser removida. Nessa técnica, as amostras de utilização de memória mais recentes têm maior peso que as amostras mais antigas. São utilizados dois períodos de amostragem que tem por objetivo melhorar a identificação de tendências de alta e baixa demanda de memória e suavizar momentos de picos de utilização. O cálculo da MME pode ser observado na seguinte equação:

$$MME_{atual} = MME_{atual-1} + \alpha \times (MEM_{atual} - MME_{atual-1}) \tag{3.4}$$

onde:

$$\alpha = 2 \div (QtdeAmostras + 1); \tag{3.5}$$

e

$$MEM_{atual} = QuantidadeDeMemoriaUsadaNaUltimaMedida \tag{3.6}$$

São utilizados dois períodos de amostragem: 5 segundos e 25 segundos. A memória é alocada quando a amostra de 5 segundos cruza o gráfico com a amostra de 25 segundos no sentido crescente, e é liberada quando a amostra de 5 segundos cruza com a de 25 segundos no sentido decrescente.

A avaliação do mecanismo foi realizada com dois *benchmarks*. Para a demonstração da qualidade de alocação de memória foi utilizado o Produto Espaço Tempo (PTE), que mensura a quantidade de memória demandada e a alocada no ambiente [44].

Outro trabalho que utiliza Média Móvel Exponencial(MME) é descrito por Jenitha et al. 2014 [16] que propõem um método de alocação elástica de memória também utilizando MME, é um sistema que libera de forma dinâmica as páginas de memória que não estão sendo utilizadas, o que ajuda a melhorar o desempenho global do sistema. O sistema é composto por quatro módulos: coletor estatístico de memória, gerenciador de recursos, computação verde e lista de detalhes do *balloon*. O gerenciador de recursos é responsável por prever a demanda futura de memória utilizando uma equação baseada em médias móveis exponenciais ponderadas.

$$E(t) = \alpha \times E(t-1) + (1-\alpha) \times O(t-1) \tag{3.7}$$

E(t) é a memória estimada no tempo t

E(t - 1) é a memória estimada no tempo t-1

O(t-1) é a memória usada no tempo t-1

E alfa é obtido através da média da memória que foi utilizada no tempo t-1 pela memória estimada no tempo t-1

$$\alpha = \frac{O(t-1)}{E(t-1)} \tag{3.8}$$

Os métodos reativos são mostrados na Tabela 3.2.

Tabela 3.2: Soluções reativas de alocação de memória.

Autor	Proposta
Heo/Oliveira/Dawoud	Gerenciador de memória reativo, esse gerenciador aloca mais memória de forma agressiva quando a memória utilizada está próxima da saturação (por exemplo, 90%), e diminui de forma lenta a alocação de memória quando a alocação está em uma região longe de gargalos.
Moltó	É proposto um mecanismo que monitora o uso de memória da MV e, caso o percentual de memória utilizada esteja fora de determinada faixa, pode-se alocar ou liberar memória usando o *balloon driver* do hipervisor.
Baruchi	Propõem um mecanismo que utiliza Média Móvel Exponencial para definir quando a memória deve ser alocada/liberada. Média ponderada, pois as amostras mais recentes têm maior peso que amostras antigas.
Jenitha	Propõem um método de alocação elástica de memória também utilizando MME, é um sistema que libera de forma dinâmica as páginas de memória que não estão sendo utilizadas, o que ajuda a melhorar o desempenho global do sistema.

Além da abordagem reativa, abordagens proativas têm sido utilizadas para disponibilizar soluções de elasticidade automática. A principal característica dessa classe é o uso de técnicas que preveem o comportamento das cargas do sistema para guiar a tomada de decisões de quando e como alocar os recursos.

No trabalho de Gong et al. 2010 [20], é apresentado um sistema de elasticidade preditivo chamado PRESS, que é baseado em assinaturas geradas com a Transformada Rápida de Fourier (FFT). Para cada padrão de carga de CPU e memória, é gerada uma assinatura que será utilizada na predição do comportamento de aplicações que contém estruturas repetitivas. Para aplicações que não contem estruturas repetitivas e que não possuem um padrão, emprega-se cadeias de Markov com um número finito de estados para construir uma previsão de curto prazo para os futuros valores de carga. Os experimentos mostram uma boa precisão na previsão de recursos futuramente necessários, com menos de 5% de erro de sobre estimação e quase zero de erro de subestimação. O sistema faz predições a cada minuto e usa as informações geradas para alocar/liberar recursos para a aplicação no próximo minuto.

Farokhi et al. 2016 [45] propõem um modelo híbrido de alocação elástica de memória, que possui um controlador de desempenho (*performance-based* - PC), e um controlador de capaci-

dade (*capacity-based* - CC). O controlador de desempenho toma a decisão de alocar/desalocar memória de acordo com a performance da aplicação. Já o controlador de capacidade toma a decisão baseado na utilização de recursos. O sistema é composto por três módulos: o sensor, que coleta informações em tempo real dos aplicativos e das MV's, informações como o nível de desempenho, tempo médio de resposta e memória média utilizada pela aplicação; o controlador, que é responsável por ajustar a quantidade de memória necessária para cada aplicativo de acordo com os valores de tempo de resposta (*Response Time* - RT) e com a utilização de memória (U_{mem}) dada pelo modulo sensor; o atuador que é o modulo responsável por alocar/desalocar a memória.

No trabalho de Spinner et al. 2015 [46], é implementado um sistema proativo de alocação elástica de memória. Esse sistema prevê a utilização de memória de uma MV baseado em um treinamento que dura trinta dias, onde a carga de utilização de memória é monitorada e armazenada, e a partir de métodos de análise de series é possível prever a carga de trabalho da MV durante o dia, identificando pontos de gargalo e ponto de baixa utilização e também a quantidade de memória mínima e máxima que essa MV utiliza. Mas antes mesmos de realizar essa etapa de treinamento o sistema consegue fazer previsões analisando a carga de memória utilizada pelas MV's durante três dias durante a semana e por três dias durante os finais de semana. A atualização da quantidade de memória é feita durante um horário de baixo uso da MV. Nesse sistema é também monitorado, durante 3 dias, a carga de trabalho para identificar quando a utilização de memória é menor, quando se identifica esse período, o sistema utiliza o *ballooning* para liberar a memória excedente.

Em Hines et al. 2011 [47] é proposto um sistema de gerenciamento de memória em nuvem computacional chamado Ginkgo. Ele possui um modulo que coleta informações sobre a utilização da memória de cada MV; possui um correlator que analisa os dados recolhidos, utilizando média móveis ponderadas e a função do 95º percentil, depois os dados passam por uma regressão linear, e a partir disso constrói um conjunto de modelo que correlaciona o desempenho do aplicativo monitorado para um determinado tamanho de memória; e possui um estimador que utiliza os modelos gerados no correlator para calcular uma atribuição de memória viável por MV, para otimizar uma função objetivo que pode ser por exemplo, o desempenho máximo ou uso mínimo de memória.

Salomie et al. 2013 [48] propõem o ABL, um API que permite a alocação/liberação de memória entre diferentes aplicativos (como por exemplo, bancos de dados e JVM's) e o drive *ballooning* do SO convidado.

Tan et al. 2012 [49] apresenta o PREPARE (*Predictive Performance Anomaly Prevention*), um sistema de previsão e prevenção de anomalias, possui módulos responsáveis por: detectar a anomalia na utilização de recursos antes de ela ocorrer, esse modulo utiliza cadeias de MARKOV, e redes Bayesiana; módulo responsável por identificar a melhor ação a ser tomada para a resolução da anomalia, seja alocação de CPU, memória ou Entrada/Saída, ou a migração de maquinas virtuais; e um modulo responsável por alocar/desalocar os recursos, ou realizar a migração de MV, no caso da memória a alocação/liberação é realizada utilizando a técnica de *ballooning*.

A Tabela 3.3 mostra os métodos preditivos de alocação elástica de memória.

Tabela 3.3: Soluções proativas de alocação de memória.

Autor	Proposta
Gong et al. 2010	*PRESS* - sistema de elasticidade, que é baseado em assinaturas geradas com a Transformada Rápida de Fourier (FFT).
Farokhi et al. 2016	Modelo híbrido de alocação elástica de memória, que possui um controlador de desempenho, e um controlador de capacidade.
Spinner et al. 2015	Baseado no monitoramento da utilização de memória durante 30 dias, e a necessidade de alocação/liberação é realizada utilizando analise de séries.
Hines et al. 2011	*Ginkgo* – sistema que possui quatro módulos pra a predição de utilização de memória. Os dados recolhidos, são analisados utilizando média móveis ponderadas e a função do 95º percentil, depois os dados passam por uma regressão linear, e a partir disso constrói um conjunto de modelo que correlaciona o desempenho do aplicativo monitorado para um determinado tamanho de memória.
Salomie et al. 2013	*ABL* – API que aloca/libera memória.
Tan et al. 2012	*Prepare* – sistema que utiliza cadeias de Markov e redes Bayesianas para identificar a melhor ação para a alocação de recursos.

Capítulo 4

Métricas

Existem diversos categorias de métricas que são utilizadas com o objetivo de avaliar o desempenho da elasticidade. Porém existem vários aspectos associados a elasticidade que são implementados de diversas maneiras, como desalocação, alocação e utilização de recursos, o que torna avaliar o desemprenho da elasticidade algo complicado [50].

Não há um padrão na maneira de se medir a elasticidade. Além disso, existe uma grande quantidade de tecnologias e estratégias para o provimento da elasticidade [51]. Por fim, não é comum a utilização de métricas específicas para a medição da elasticidade de uma nuvem computacional, e sim métricas indiretamente associadas [50]. Tais como a apresentada por Islam et al. 2012 [51] que define formas de quantificar a elasticidade, que é medida e avaliada pelo ponto de vista do usuário. Foram criadas cargas de trabalho que crescem rapidamente e diminuem de forma lenta e outras que crescem de forma lenta e diminuem de forma rápida, e pra cada uma dessas cargas é examinada a forma como a plataforma responde. Nesse trabalho são considerados os aspectos de qualidade e penalidades em caso de não cumprimento do acordo de serviço, é feito um calculo monetário de quanto dinheiro é desperdiçado pelo usuário caso a quantidade de recursos alocados forem maiores do que o que é requerido (*overprovisioning*), e é também utilizado um componente para medir as perdas que o usuário tem quando os recursos alocados são insuficientes (*underprovisioning).* Um modelo matemático baseado em penalidades, recursos e demanda foi descrito. Experimentos foram realizados na Amazon EC2 com o benchmark TPC-W e o aplicativo JMeter para geração de cargas de trabalho.

Herbst, Kounev e Reussner et al. 2013 [21] propõem uma métrica que considera dois aspectos para a adaptação de recursos computacionais, a velocidade e a elasticidade:

- Velocidade - nesse trabalho a velocidade é definida como o tempo que leva sair de um estado de *underprovisioned* para um estado ótimo de provisionamento de recursos. E o tempo que leva para sair de um estado *overprovisioned* para um estado ótimo de alocação de recursos.

- Precisão - a precisão do dimensionamento de recursos é calculada em relação a quantidade de recursos provisionados e a quantidade de recursos realmente utilizados.

E para capturar os aspectos de precisão e elasticidade foram definidas as métricas a seguir.

- $\overline{A}$ mede a quantidade de tempo que o sistema sobre subprovisionamento de memória.
- $\sum \overline{A}$ tempo acumulado de subprovisionamento.
- $\overline{U}$ quantidade de recursos subprovisionados.
- $\sum \overline{U}$ quantidade acumulada de recursos subprovisionados.
- $\overline{B}$ mede a quantidade de tempo que o sistema sobre superprovisionamento de memória.
- $\sum \overline{B}$ tempo acumulado de superprovisionamento.
- $\overline{O}$ quantidade de recursos superprovisionados.
- $\sum \overline{O}$ quantidade acumulada de recursos superprovisionados.

A Figura 4.1 mostra a forma como essas métricas são medidas durante o período de coleta de uma *trace*.

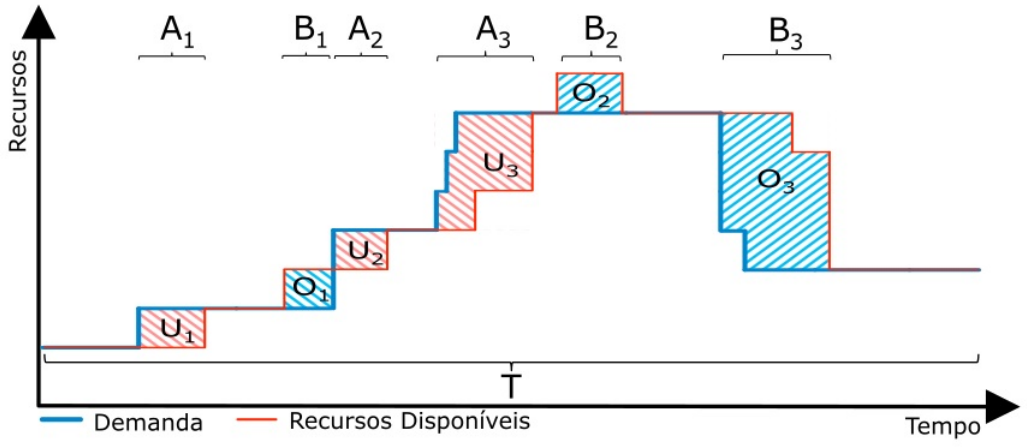

Figura 4.1: Definição de precisão e tempo de provisionamento de recursos. Adaptado de [3]

$$P_u = \sum \overline{U}/T \tag{4.1}$$

Onde T é a duração total da execução da *trace* a qual está sendo feita a analise. P_u é definida como a precisão do método para desalocação de memória caso ocorra subprovisionamento, ou seja, quando não há subprovisionamento, P_u é igual a 0.

$$P_d = \sum \overline{O}/T \tag{4.2}$$

P_d é definida como a precisão do método para alocação de memória caso ocorra superprovisionamento, ou seja, quando não há superprovisionamento, P_d é igual a 0.

Provisioning Timeshare: As duas métricas de precisão apresentadas anteriormente, não permitem concluir se a quantidade média de recursos subprovisionados ou superprovisionados é resultado de grandes desvios entre demanda e oferta ou se é causada por um pequeno desvio constante. Para solucionar isso, as duas métricas a seguir são projetadas para fornecer mais informações sobre a proporção de tempo em que ocorre o subprovisionamento ou superprovisionamento.

$$Subprovisionamento : timeshareU[\%] = \sum \overline{A}/T \tag{4.3}$$

Onde T é a duração total da execução da *trace* a qual está sendo feita a analise. $\overline{A}$ mede a quantidade de tempo que o sistema sofre subprovisionamento de memória.

$$Superprovisionamento : timeshareO[\%] = \sum \overline{B}/T \tag{4.4}$$

Onde T é a duração total da execução da *trace* a qual está sendo feita a analise. $\sum \overline{B}$ tempo acumulado de superprovisionamento.

Para a avaliação dos métodos de alocação de memória utilizaremos a métrica apresentada por Herbst dado que é adequada para avaliar a elasticidade de memória.

Capítulo 5

Avaliação de métodos de elasticidade de memória

Concluída a revisão do estado da arte sobre a alocação elástica de memória, o objetivo deste Capítulo é apresentar os resultados das técnicas reativas de alocação elástica de memória .

Os testes foram realizados em 12 *traces* de utilização de memória, dessas, 5 são sintéticas e foram criadas para testar o desempenho de cada algoritmo em situações bem específicas. As outras 7 *traces* são de utilização real, uma dessas *traces* foi coletada de um computador de uso pessoal, e as outras 6 *traces* foram coletadas de máquinas virtuais do DataCenter da empresa Constel Tecnologia.

Na Seção 5.4 apresentamos os resultados da técnica implementada por Moltó et al. 2013 [12]. Na Seção 5.5 os resultados de Oliveira et al. 2015 [2] e Dawoud et al. 2011 [17], na Seção 5.6 os resultados da técnica apresentada por Jenitha et al. 2014 [16], e na Seção 5.7 apresentamos a tecnica proposta por Baruchi et al. 2010 [9] avaliando a aplicação desses métodos em diferentes cenários, com diferentes cargas de consumo de memória.

5.1 Ambiente Computacional

Para realização dos testes foi desenvolvido um simulador em linguagem de programação C++, nesse simulador foram implementados os algoritmos reativos citados na bibliografia apresentados no Capítulo 3. O computador utilizado para rodar os testes possui 6 GB de memória RAM, Processador Intel Core i5-2450M 2.50 GHz, Sistema Operacional Windows 10, utilizando a máquina virtual VMware Workstation 12 Player, com o Sistema Operacional Ubuntu 14.04 LTS, 32 bits.

5.2 Caracterização dos Estudos de Caso

Para a realização dos testes foram utilizadas 12 *traces* de utilização de memória, dessas, 5 são *traces* sintéticas com características bem específicas para testar o comportamento de cada algoritmo nessas situações. Abaixo são descritos detalhes de cada *trace* utilizadas nos testes.

- *crescente* - essa *trace* possui o comportamento crescente, como mostra a Figura 5.1 com a utilização de memória sendo incrementada gradativamente de 6 em 6 MB (Mega Bytes), a cada minuto, iniciando com o consumo de 4000 MB e atingindo o máximo de 4498 MB.

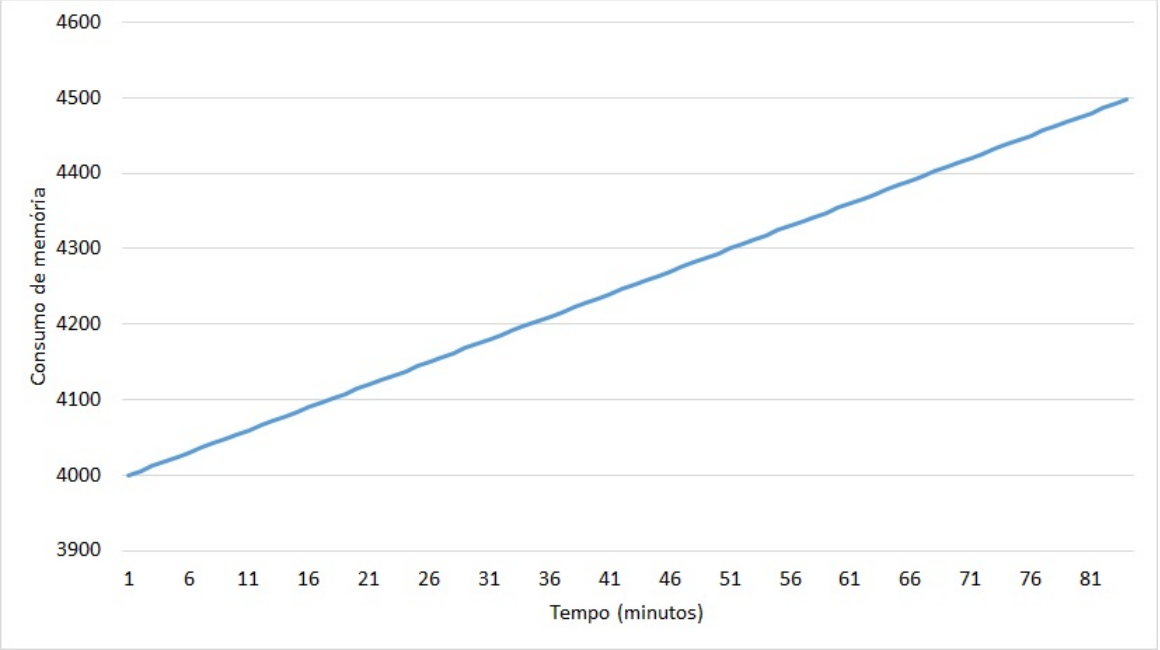

Figura 5.1: Trace crescente.

- *decrescente* - essa *trace* possui a característica de decrescer o consumo de memória. O uso da memória é decrementado de 6 em 6 MB a cada minuto, iniciando com consumo de memória de 4500 MB, e que termina com o consumo de 4002 MB, como mostra a Figura 5.2.

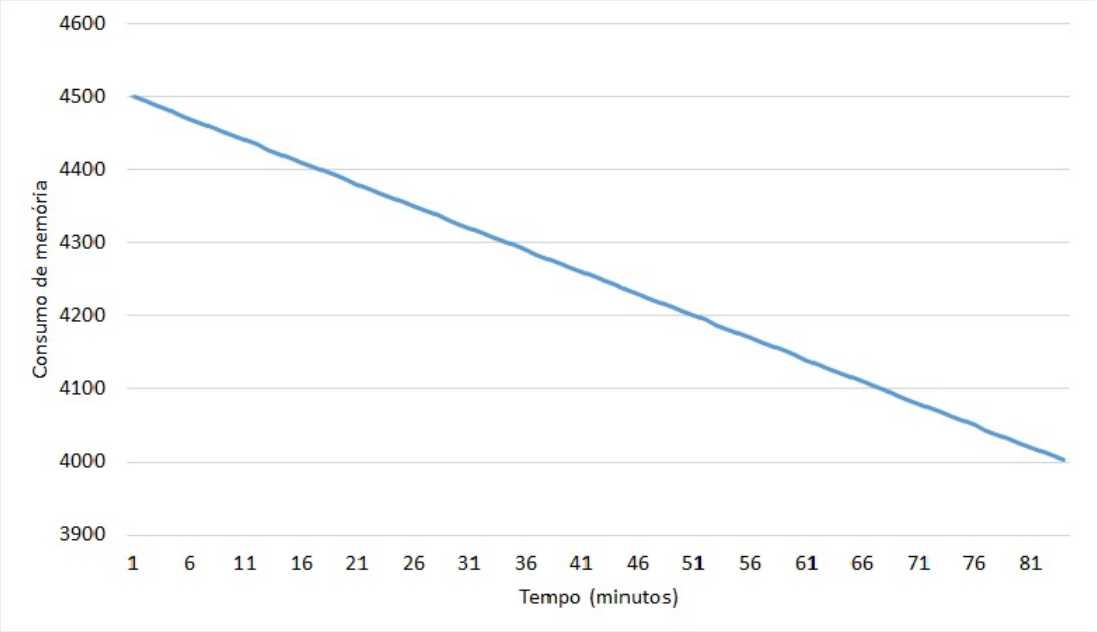

Figura 5.2: Trace decrescente.

- *piconst* - essa *trace* tem um comportamento que cresce rapidamente. Mantém uma utilização contínua da memória no topo e cai o consumo de forma rápida. O consumo de memória se inicia em 4000 MB, e esses mesmos 4000 MB são mantidos nas partes de menores consumos da trace. A parte crescente da trace é incrementada de 20 em 20 MB por minuto, chegando ao topo de consumo de 4100 MB, e o decréscimo na utilização de memória também cai 20 MB a cada minuto é apresentada na Figura 5.3.

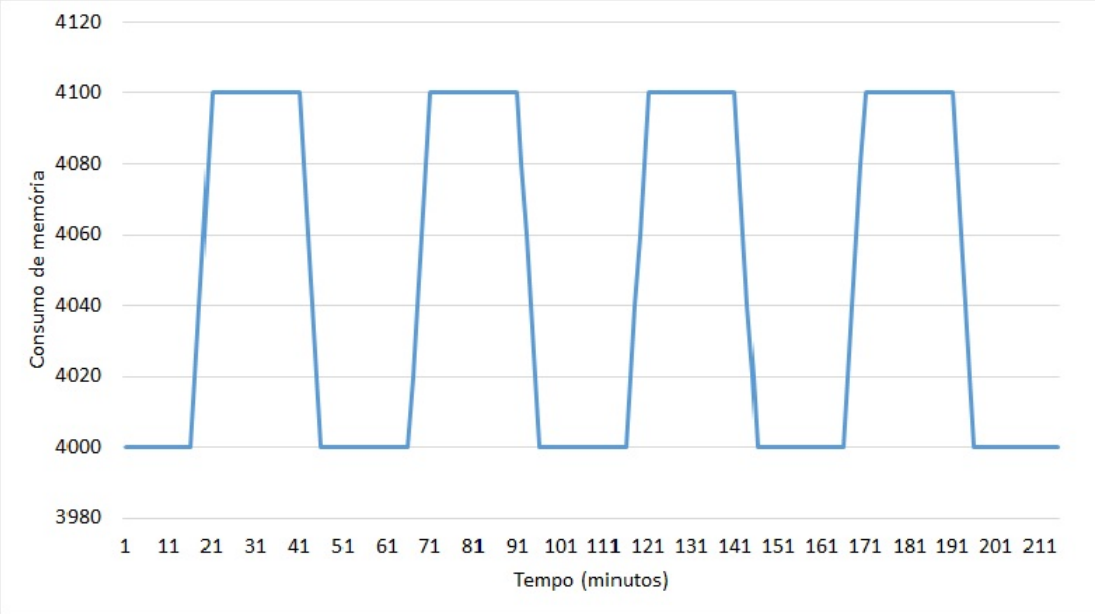

Figura 5.3: Trace piconst.

- *const* - essa *trace*, que pode ser vista na Figura 5.4, cresce 8 MB por minuto, iniciando em 4000 MB, atingindo o consumo máximo de 4104 MB e permanecendo assim por 45 minutos, e depois o consumo começa a cair 8 MB por minuto.

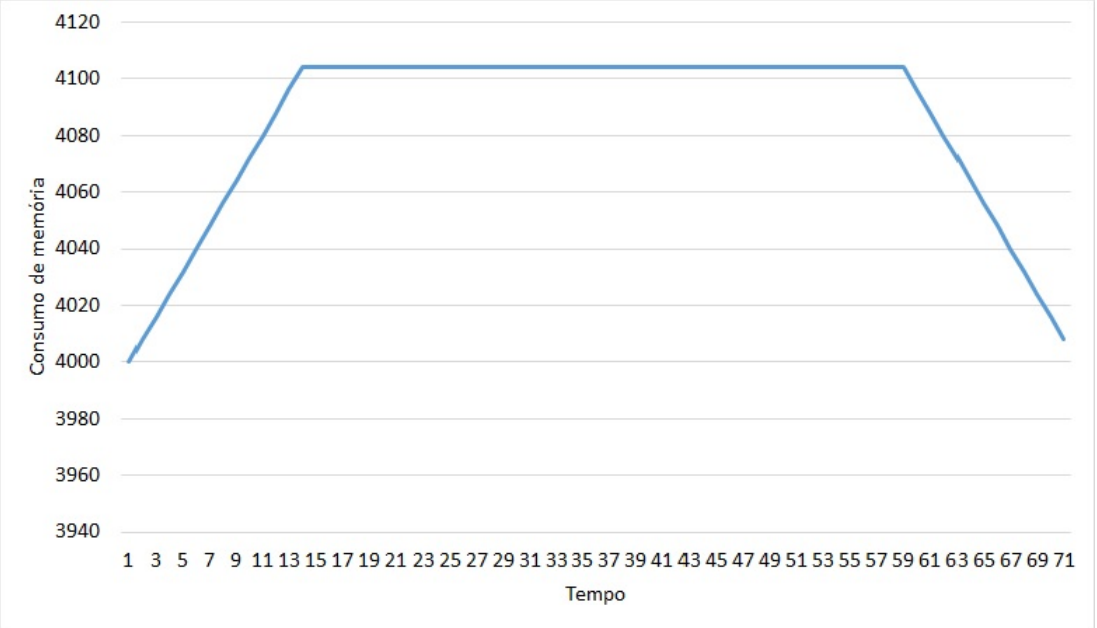

Figura 5.4: Trace const.

- *picos* - *trace* que possui picos de consumo de memória, que iniciam com o consumo de

4000 MB, cresce 4 MB a cada minuto, atingindo picos máximos de 4100 MB de consumo de memória, a queda no consumo de memória decresce 4 MB por minuto, como mostra a Figura 5.5

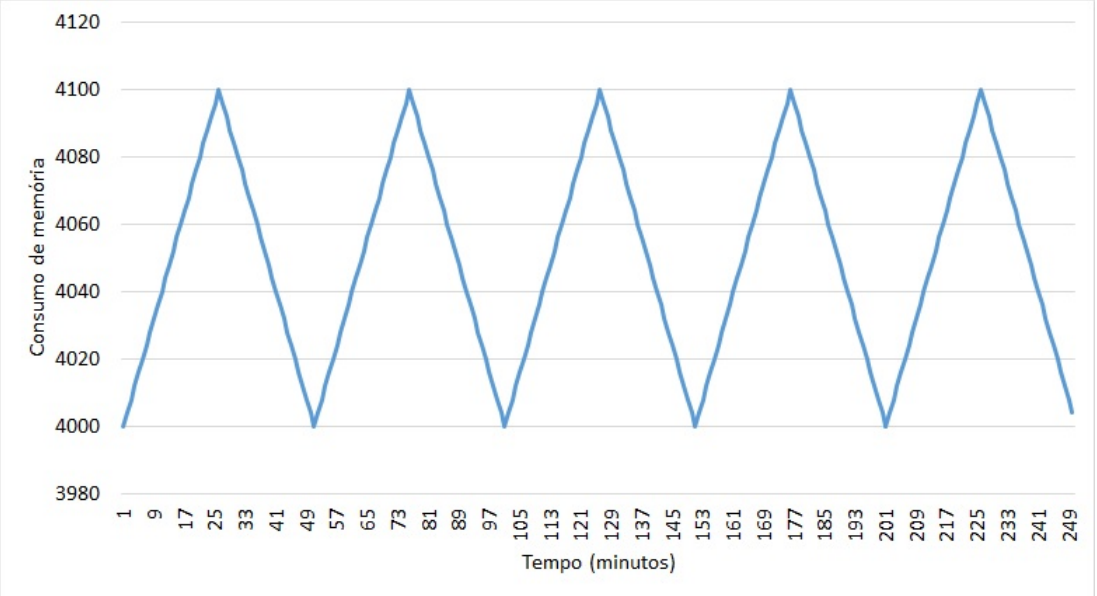

Figura 5.5: Trace picos.

Das 7 *traces* de consumo real, 6 delas foram cedidas pela empresa Constel Tecnologia. O consumo de memória das MV's foi coletado durante uma semana. A medição é feita a cada um minuto e as MV's utilizam o hipervisor Hyper V da Microsoft. E 1 das *traces* foi coletada de um computador pessoal, durante 1 dia.

- *dalDC1* - domain controller: O controlador de domínio é o responsável por manter os dados referentes aos usuários e grupos aos quais pertence, bem como suas permissões e privilégios, através do gerenciamento de políticas de grupo. As principais funções são autenticar e autorizar usuários a se conectarem na rede gerenciada pelo controlador, permitindo ou negando a execução de ações pelo usuário, em conformidade com as políticas de grupo definidas. Essa MV possui um mínimo de 1 GB de memória, e tem um limite máximo de 4 GB de memória, dado o servidor físico em que a MV está hospedada, ou seja, o consumo de memória dessa MV pode variar de acordo com a demanda da empresa contratante, dentro deste intervalo que foi o contratado, como mostra a Figura 5.6.

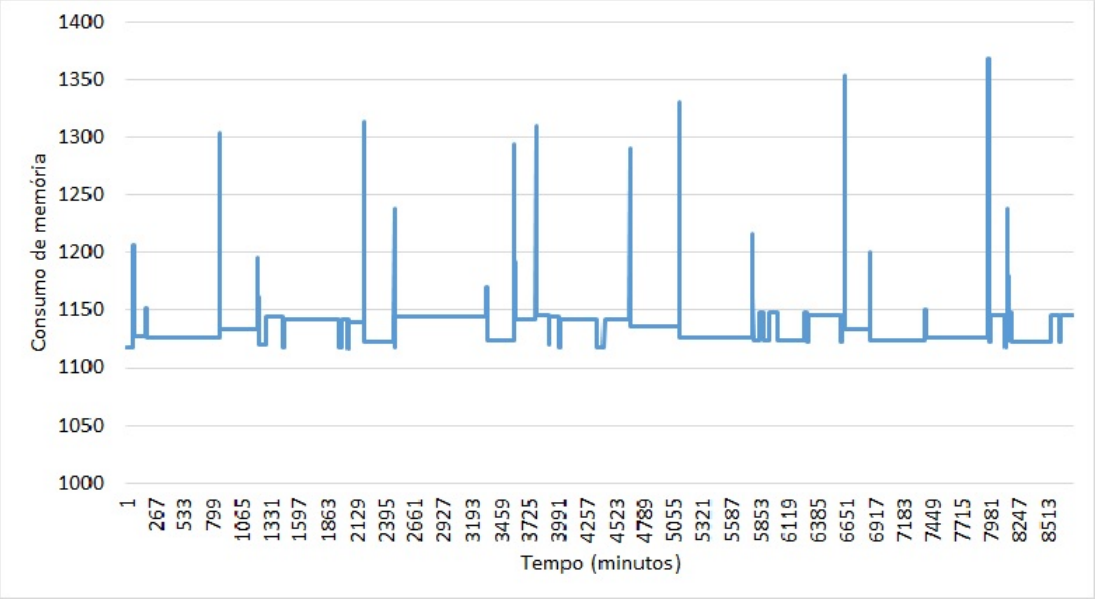

Figura 5.6: Trace dalDC1.

- *fipS* - servidor de impressões: O servidor de impressão é o responsável por centralizar e controlar a fila de tarefas de impressão de uma determinada rede, permitindo controlar a prioridade da fila, alterar propriedades de forma centralizada, como quantidade de cópias ou qualidade de impressão e também auditar o uso do recurso de impressão pelos usuários, como quantidade de páginas impressas, custo de impressão e reposição de suprimentos. Em conjunto com o controlador de domínio pode limitar o uso de determinadas impressoras para cada usuário, baseada na autenticação exigida do usuário no momento de sua conexão com a rede. Essa MV possui um mínimo de 1 GB de memória, e tem um limite máximo de 16 GB de memória, como mostra a Figura 5.7.

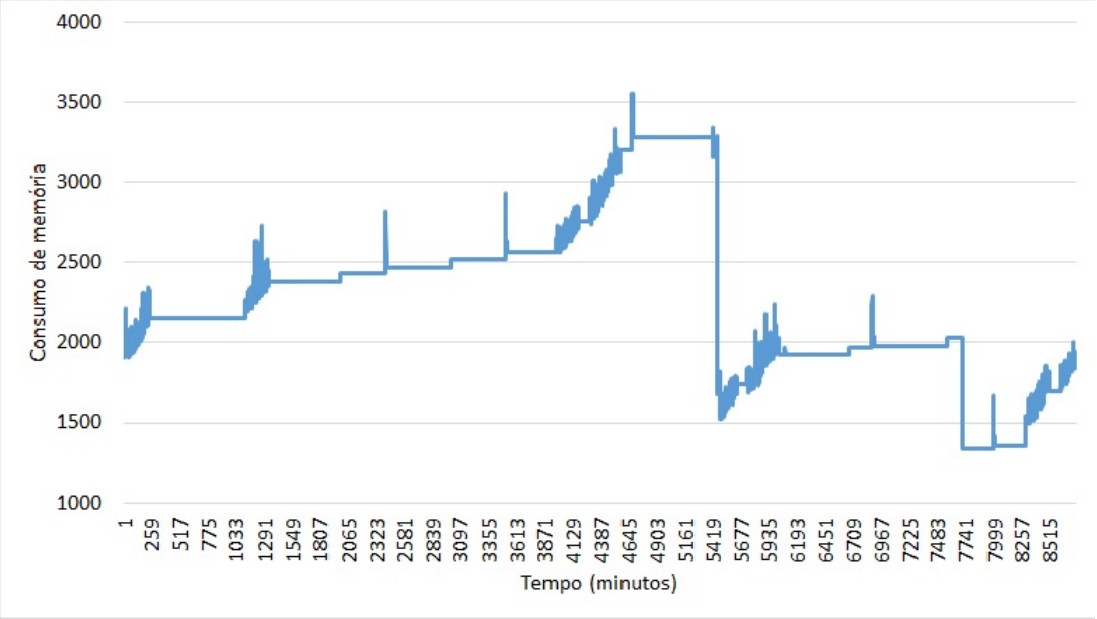

Figura 5.7: Trace fipS.

- *fipapp* - servidor de aplicações: Em um ambiente multicamadas, temos ambientes diferentes executando cada parte da aplicação utilizada por determinado usuário. O servidor de aplicações é a camada onde o software aplicativo está instalado, consumindo recursos do servidor nas atividades de processamento das requisições realizadas pelo usuário. Essas requisições normalmente são geradas na camada de apresentação (interface do usuário) e são processadas pelo servidor de aplicações, que pode também gerar requisições para o servidor de banco de dados ou outros servidores de aplicações. Essa MV possui um mínimo de 4 GB de memória, e tem um limite máximo de 16 GB de memória, como mostra a Figura 5.8.

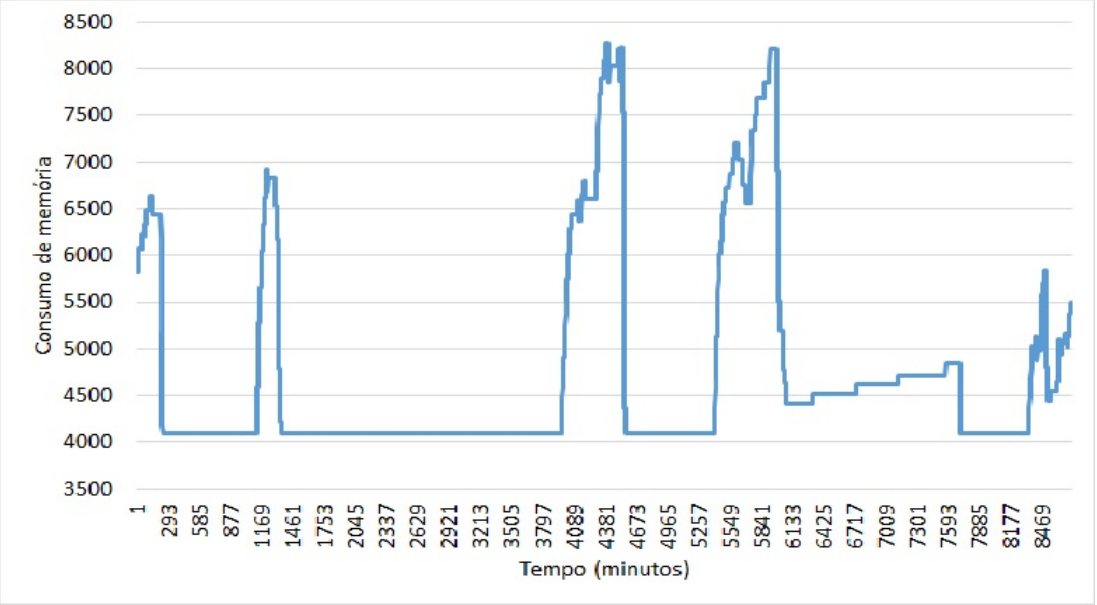

Figura 5.8: Trace fipapp.

- *fipnfe* - emissão de nota fiscal eletrônica: É um servidor de aplicação responsável pela parte do software aplicativo que se comunica, via internet, com a Receita Estadual para autorizar a emissão das notas fiscais eletrônicas. Essa MV possui um mínimo de 0 GB de memória, e tem um limite máximo de 4 GB de memória, como mostra a Figura 5.9.

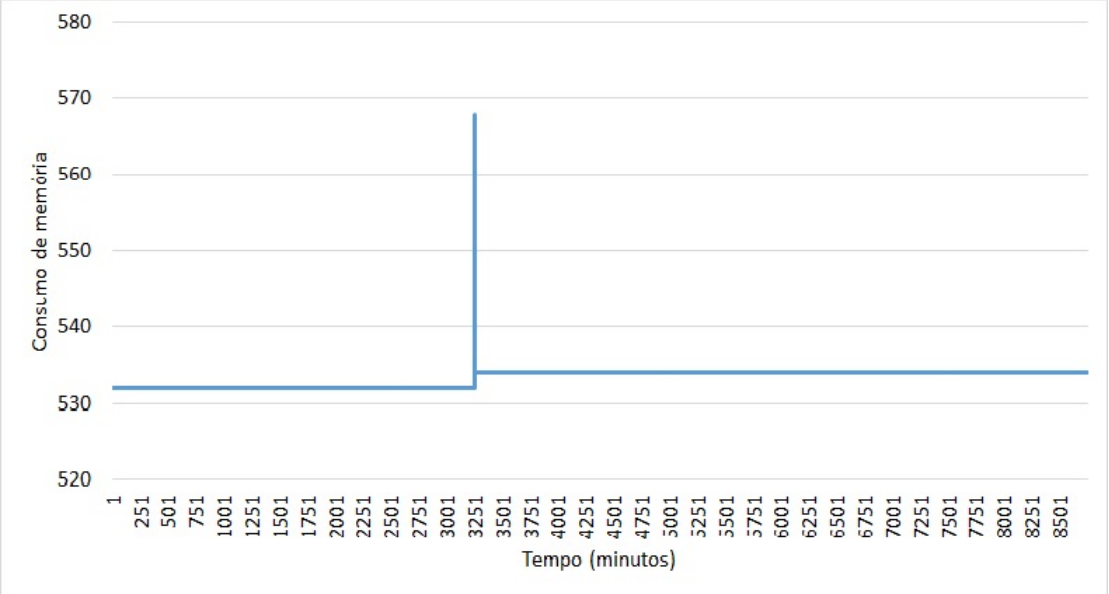

Figura 5.9: Trace fipnfe.

- *irWTS* - servidor de aplicações: Consume recursos do servidor nas atividades de pro-

cessamento das requisições realizadas pelo usuário e que pode também gerar requisições para o servidor de banco de dados ou outros servidores de aplicações. Essa MV possui um mínimo de 0 GB de memória, e tem um limite máximo de 2 GB de memória, como mostra a Figura 5.10.

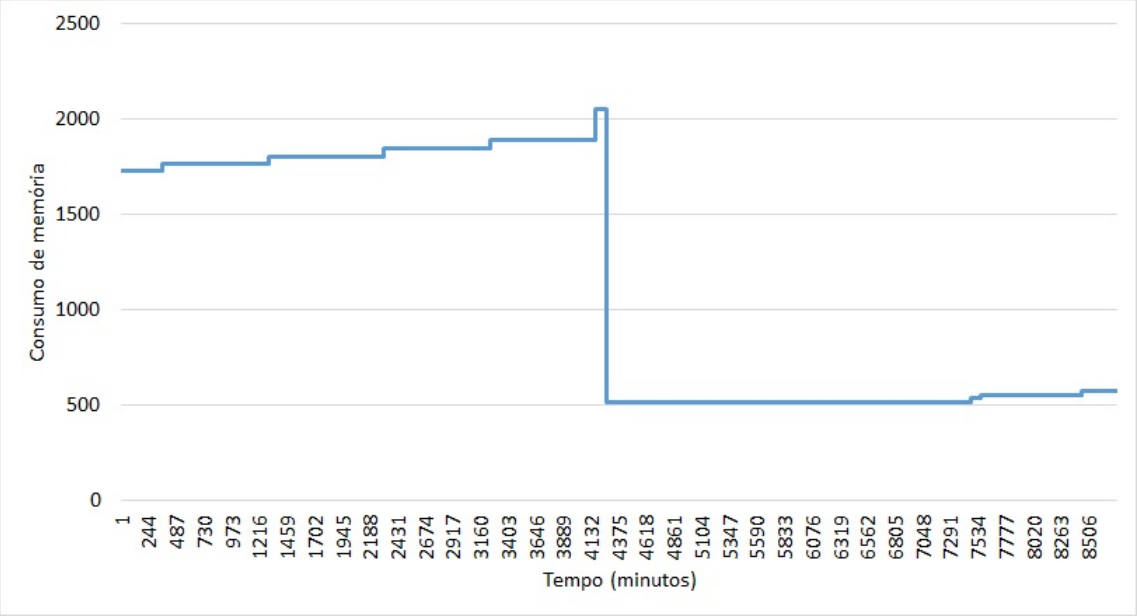

Figura 5.10: Trace irWTS.

- *TVL* - servidor de aplicações: Consume recursos do servidor nas atividades de processamento das requisições realizadas pelo usuário e que pode também gerar requisições para o servidor de banco de dados ou outros servidores de aplicações. Essa MV possui um mínimo de 1 GB de memória, e como limite máximo 10 GB de memória, como mostra a Figura 5.11.

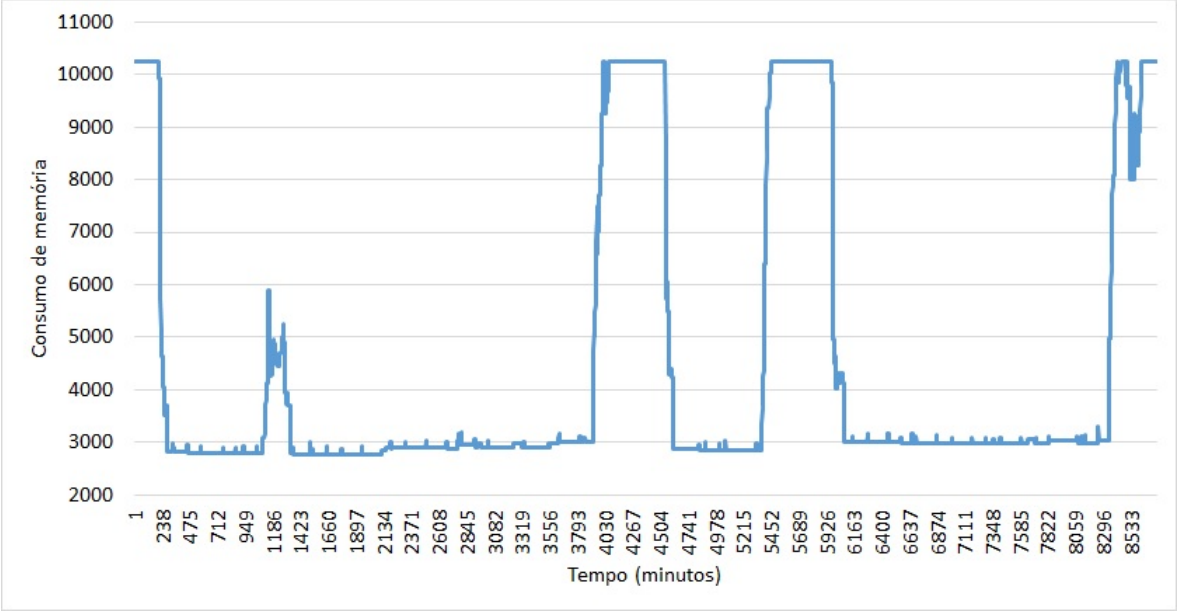

Figura 5.11: Trace TVL.

- *WS*: *trace* coletada de uma estação de trabalho pessoal, durante um dia, a medição foi feita a cada segundo, essa estação conta com um máximo de memória de 8 GB, como mostra a Figura 5.12.

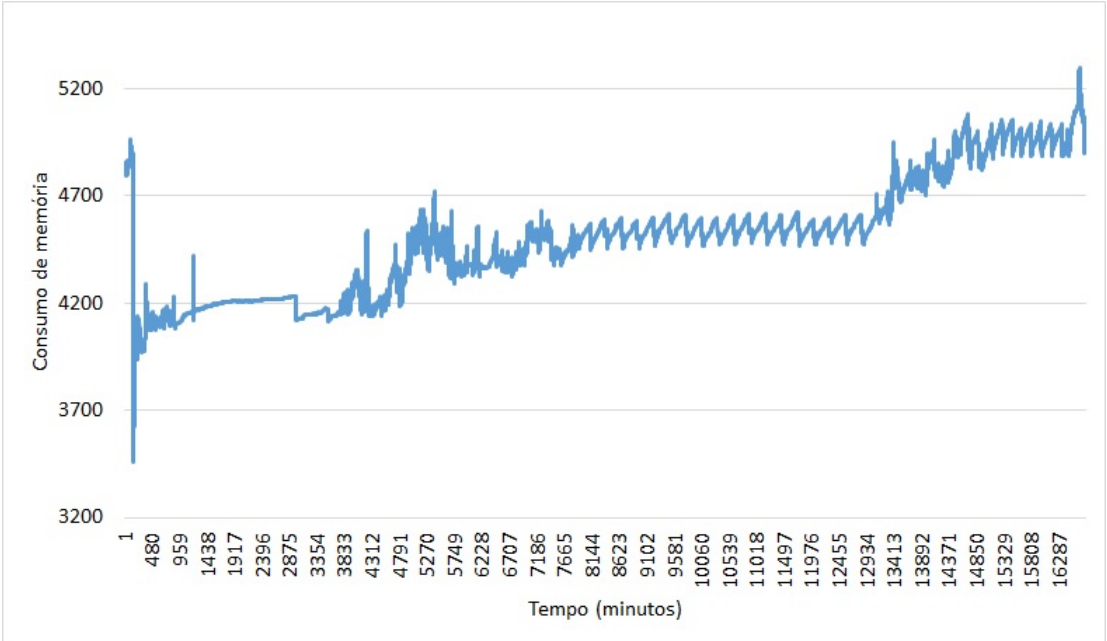

Figura 5.12: Trace WS.

5.3 Avaliação de métodos de elasticidade de memória

O objetivo dessa Seção é apresentar os resultados dos testes realizados em cada *trace* utilizando os métodos implementados, a avaliação de cada método utilizado foi realizada com a métrica descrita no trabalho de Herbst et al. 2013 [21] apresentada no Capítulo 4.

5.4 Moltó

Como descrito no Capítulo 3, a fórmula que define o tamanho da memória a ser alocada utilizando o método implementado no trabalho de Moltó é:

$$VMmemorysize = usedmemory \times (1 + MOP) \tag{5.1}$$

Nessa fórmula a quantidade de memória que será alocada é dada pelo termo *VM memory size*. O termo *used memory* diz respeito a quantidade de memória efetivamente utilizada no último intervalo de tempo, e o MOP que é a porcentagem de memória livre que o usuário necessitar ter na MV e que o mesmo define esse valor na inicialização da MV.

Para não haver alocação e liberação de memória sempre que a quantidade de memória livre for menor ou maior do que o valor do MOP, é definido um intervalo de tolerância para acionar o método, esse intervalo corresponde a 80% abaixo da faixa de MOP, ou 120% acima. Utilizando o valor de MOP igual a 10% como exemplo, se a quantidade de memória livre for menor do 8% o mecanismo aloca mais memória, se for maior do que 12% o mecanismo libera memória.

A utilização de um MOP = 30%, quer dizer que o mecanismo de alocação será acionado quando utilizando como referência de que a MV tenha pelos menos 30% de memória livre. E de acordo com o intervalo de tolerância delimitado, se a quantidade de memória livre for menor do que o correspondente a 80% desses 30%, o mecanismo aloca mais memória, se for maior do que o correspondente a 120% desses 30% de memória livre, o mecanismo libera memória.

Os testes realizados com o método proposto por Moltó [12], foram feitos com um MOP no valor de 10% e de 30%. Esse método foi implementado e testado utilizando as 12 *traces*.

O teste realizado com as *traces* sintéticas mostram um bom desempenho do algoritmo, pois realiza a alocação de memória em grande parte das *traces* nos momentos certos. Com MOP no valor de 10% apenas 3 *traces* sintéticas apresentaram resultados negativos. Já com MOP de 30% nenhuma *trace* sintética apresentou problema.

Abaixo na Figura 5.13 apresenta o comparativo entre a alocação realizada com MOP igual a 10% e 30%.

Para essa *trace* o valor de 10% apresentou superprovisionamento de memória como mostram as métricas: $\sum \overline{B}$ 1 minuto, tempo que a memória alocada ficou acima do necessário; $\sum \overline{O}$ de 400 MB que é a soma da memória alocada acima do necessário. P_d = 5 que é a precisão da alocação de memória, nesse caso o resultado foi ruim, pois o valor é acima de 0.

A métrica $timeshareO$ obteve 0,014, ou seja durante 1,4% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

O superprovisionamento ocorre quando cai o consumo de memória e permanece assim até o final da *trace*.

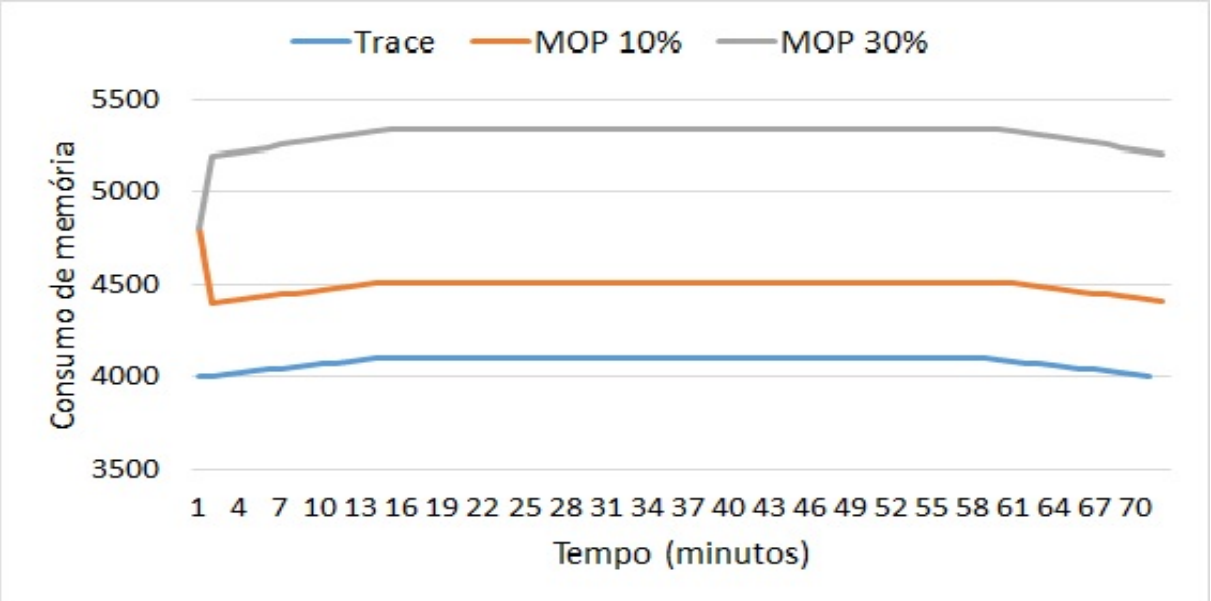

Figura 5.13: *Trace const* Comparativo Moltó 10% e 30%.

Para esta *trace* o valor de MOP de 30% não apresentou nenhum tipo de problema.

Abaixo na Figura 5.14 apresenta o comparativo entre a alocação realizada com MOP igual a 10% e 30%.

Para essa *trace* o valor de 10% apresentou superprovisionamento de memória como mostram as métricas:$\sum \overline{B}$ 21 minutos, tempo que a memória alocada ficou acima do necessário. $\sum \overline{O}$ de 804 MB que é a soma da memória alocada acima do necessário. P_d = 3 que é a precisão da alocação de memória, nesse caso o resultado foi ruim, pois o valor é acima de 0.

A métrica $timeshareO$ obteve 0,097, ou seja durante 9,7% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Os minutos de superprovisionamento dessa *trace* com MOP igual a 10%, ocorre quando acontece uma queda no consumo de memória e só é normalizado quando o consumo de memória volta a ser constante.

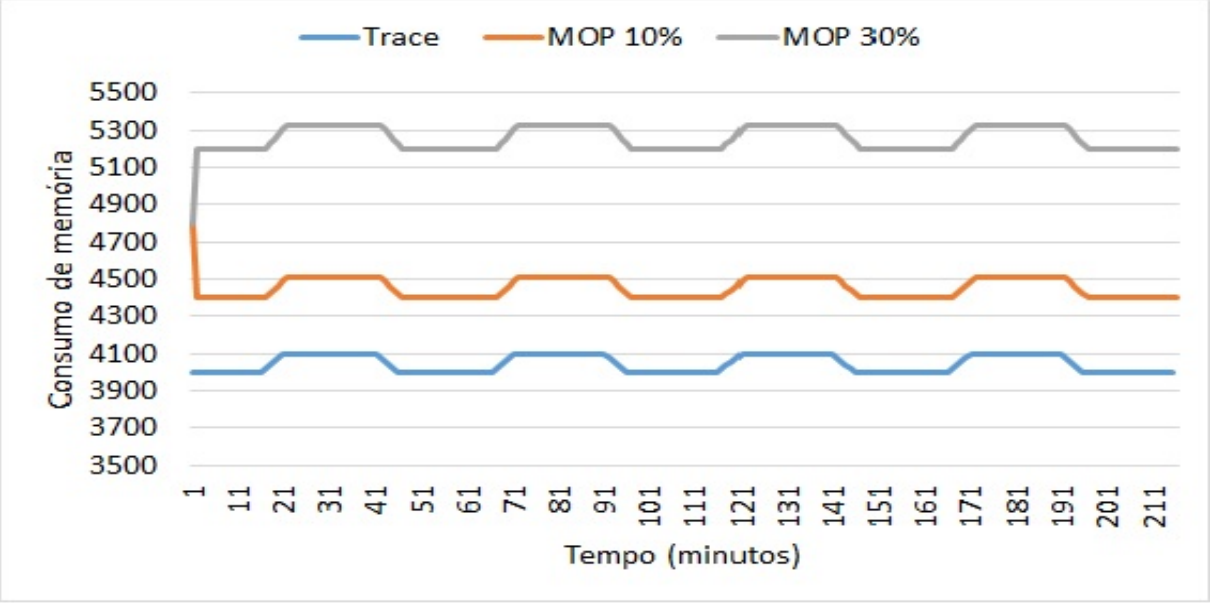

Figura 5.14: *Trace piconst* Comparativo Moltó 10% e 30%.

Na *trace crescente* houve superprovisionemento de memória com MOP de 10%, a Figura 5.15 mostra o comparativo da alocação dos dois valores de MOP.

A avaliação feita pela métrica apresentou o seguinte resultado: $\sum \overline{B}$ 1 minuto tempo que a memória alocada ficou acima do necessário. $\sum \overline{O}$ de 400 MB que é a soma da memória alocada acima do necessário. P_d = 4 que é a precisão da alocação de memória, nesse caso o resultado foi ruim, pois o valor é acima de 0.

A métrica $timeshareO$ obteve 0,011, ou seja durante 1,1% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Esse superprovisionamento ocorreu no primeiro minuto da *trace* que é quando o algoritmo inicia.

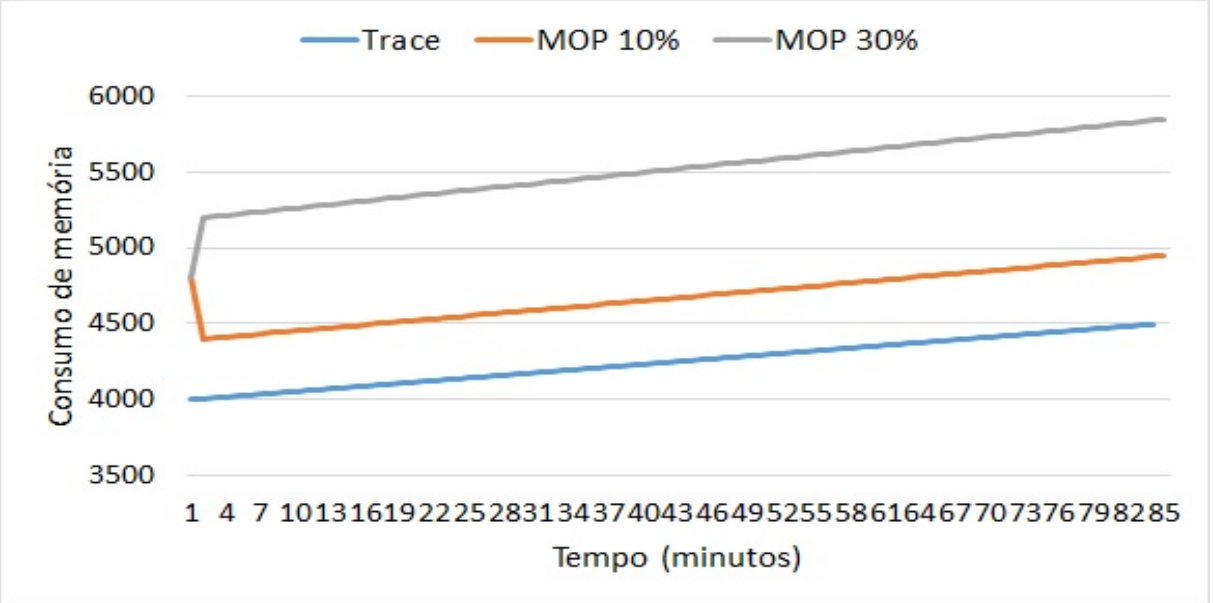

Figura 5.15: *Trace crescente* Comparativo Moltó 10% e 30%.

A Figura 5.16 mostra o comparativos da alocação com os dois valores de MOP para a *trace picos*.

A avaliação feita pela métrica apresentou o seguinte resultado: $\sum \overline{B}$ 101 minutos tempo que a memória alocada ficou acima do necessário. $\sum \overline{O}$ de 840 MB que é a soma da memória alocada acima do necessário. P_d = 3 que é a precisão da alocação de memória, nesse caso o resultado foi ruim, pois o valor é acima de 0.

A métrica $timeshareO$ obteve 0,402, ou seja durante 40,2% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

O superprovisionamento ocorre no momento que existe um pico de queda no consumo de memória, mas se normaliza assim que o consumo começa a subir novamente.

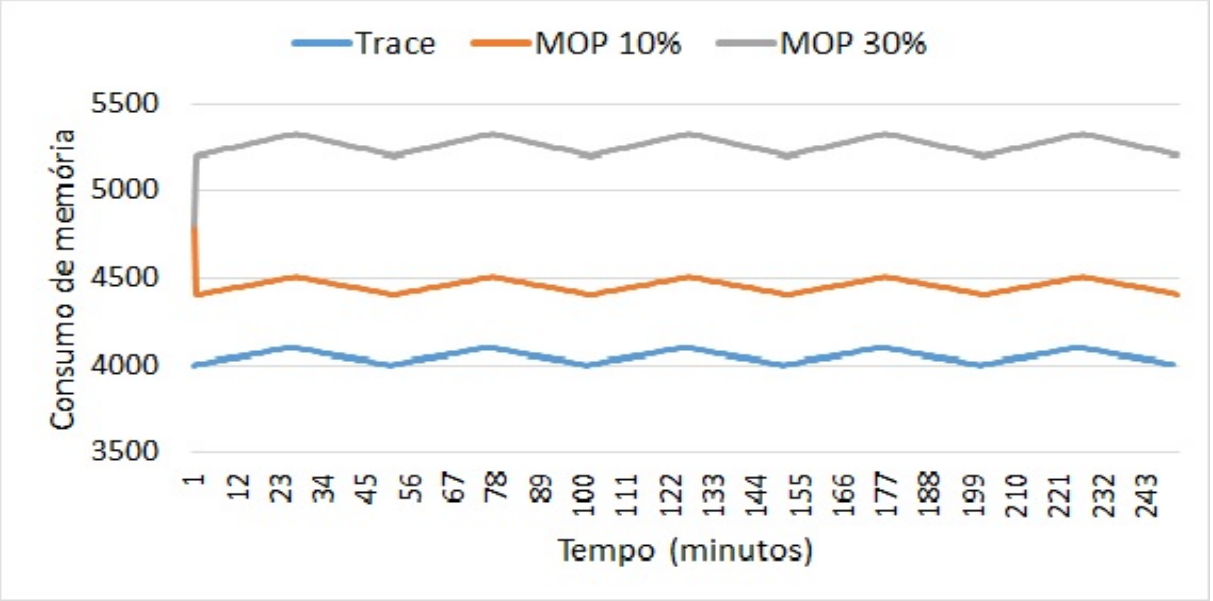

Figura 5.16: *Trace picos* Comparativo Moltó 10% e 30%.

As demais *traces* sintéticas apresentaram bons resultados para os dois valores de MOP, ou seja, as variáveis: $\sum \overline{A}$, $\sum \overline{B}$,$\sum \overline{O}$, $\sum \overline{U}$, P_u, P_d, $timeshareU$ e $timeshareO$ ficaram iguais a 0.

Nas *traces* de consumo real o MOP com valor de 10% ocorreu superprovisionamento em 4 *traces* e subprovisionamento em 3, já o MOP de valor 30% houve superprovisionamento em 2 *traces* e não houve subprovisionamento em nenhuma *trace* de consumo real.

A Figura 5.17 mostra o comparativo da alocação com os dois valores de MOP para a *trace fipS*.

Para esta *trace*, com valor de MOP de 10%, houve subprovisionamento e superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: $\sum \overline{A}$ 10 minutos, que é o tempo total de subprovisionamento, porém não consecutivos. $\sum \overline{U}$ é de 404 MB que é a quantidade total de memória subprovisionada. P_u próximo a 0 que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor próximo a 0 significa que, apesar de ter ocorrido subprovisionamento, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,00011, ou seja durante 0,11% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

Os momentos de subprovisionamento não são consecutivos e ocorrem tanto em momentos de pico de consumo de memória como em momentos de queda no consumo.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 304 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 34124 MB.

Apresentando um valor de P_d igual a 4, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,034, ou seja durante 3,47% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Com MOP no valor de 30% não houve subprovisionamento nem superprovisionamento de memória.

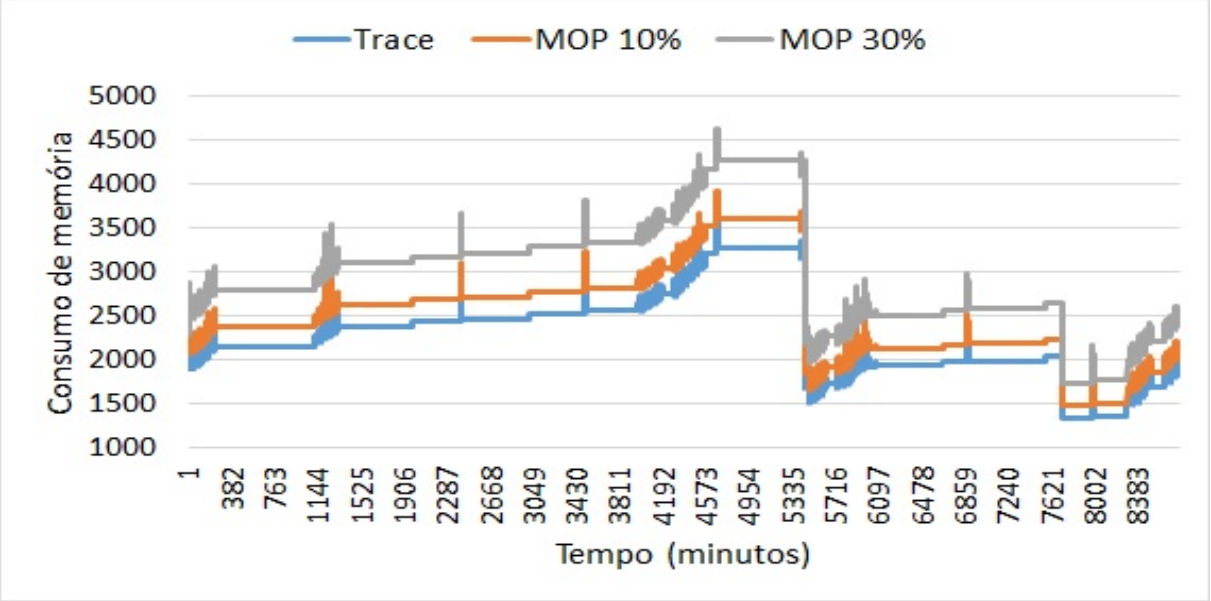

Figura 5.17: *Trace fipS* Comparativo Moltó 10% e 30%.

A Figura 5.18 mostra o comparativo da alocação com os dois valores de MOP para a *trace dalDC1*.

Para esta *trace*, com valor de MOP de 10%, houve subprovisionamento e superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 7 minutos, que é o tempo total de subprovisionamento, porém não consecutivos; $\sum \overline{U}$ é de 467 MB que é a quantidade total de memória subprovisionada; P_u próximo a 0 que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor

próximo a 0 significa que, apesar de ter ocorrido subprovisionamento, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,00008, ou seja durante 0,08% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 38 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 2843 MB.

Apresentando um valor de P_d próximo a 0, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um bom resultado, pois quanto mais próximo a 0 for o valor de P_d melhor é o desempenho do método.

A métrica $timeshareO$ obteve 0,0004, ou seja durante 0,4% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Com MOP no valor de 30% houve superprovisionamento com $\sum \overline{B}$ 125 minutos e $\sum \overline{O}$ de 3326 MB, com um P_d muito próximo de 0, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um bom resultado, pois quanto mais próximo a 0 for o valor de P_d melhor é o desempenho do método.

A métrica $timeshareO$ obteve 0,014, ou seja durante 1,4% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Ou seja, para essa *trace*, com MOP de 30% obteve um pior resultado em relação a superprovisionamento, porém esse com esse valor não houve subprovisionamento.

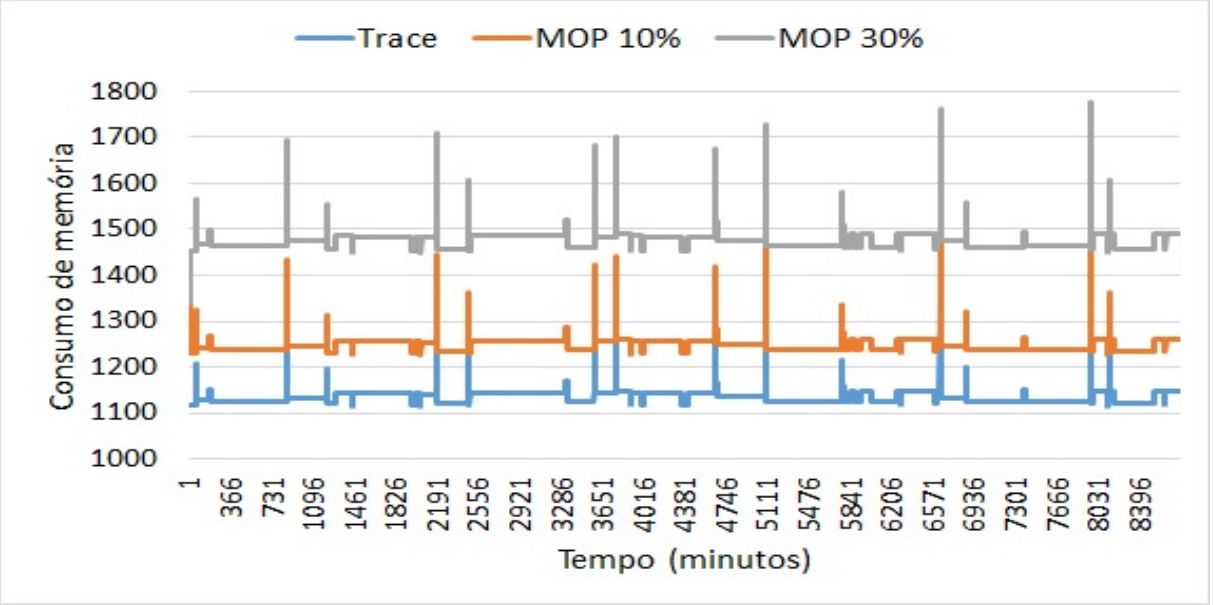

Figura 5.18: *Trace dalDC1* Comparativo Moltó 10% e 30%.

A Figura 5.19 mostra o comparativo da alocação com os dois valores de MOP para a *trace TVL*.

Para esta *trace*, com valor de MOP de 10%, houve subprovisionamento e superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 7 minutos, que é o tempo total de subprovisionamento, porém não consecutivos; $\sum \overline{U}$ é de 741 MB que é a quantidade total de memória subprovisionada; P_u próximo a 0 que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor próximo a 0 significa que, apesar de ter ocorrido subprovisionamento, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,00008, ou seja durante 0,08% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 137 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 53743 MB.

Apresentando um valor de P_d igual a 6, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior for o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,015, ou seja durante 1,5% do tempo de execução da *trace*

ocorreu superprovisionamento de memória.

Com MOP no valor de 30% houve superprovisionamento com $\sum \overline{B}$ 1602 minutos e $\sum \overline{O}$ de 3326 MB, com um P_d igual a 6, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior for o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,18, ou seja durante 18% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Ou seja, para essa *trace*, com MOP de 30% obteve um pior resultado em relação a superprovisionamento, apesar de a métrica de precisão ter dado o mesmo resultado a métrica de tempo para o método com MOP de 30% deu um resultado muito maior, porém com esse valor não houve subprovisionamento.

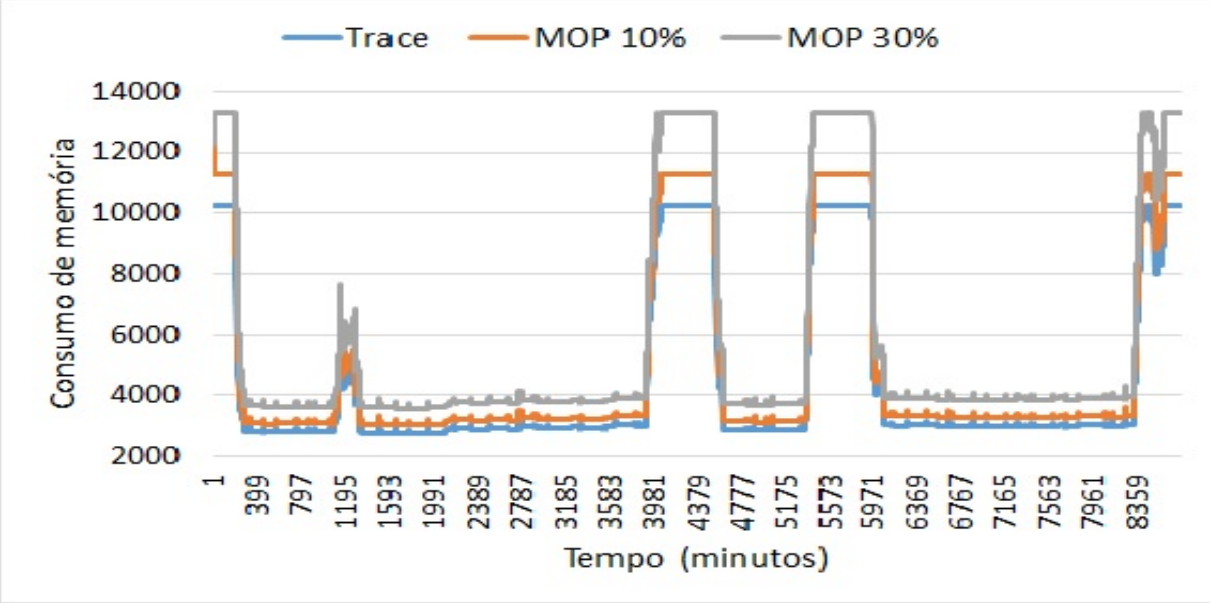

Figura 5.19: *Trace TVL* Comparativo Moltó 10% e 30%.

Nas demais *traces* reais o valor de MOP de 30% não apresentou nenhum tipo de problema de alocação de memória, já para o valor de MOP de 10% houve superprovisionamento de memória.

A Figura 5.20 mostra o comparativo da alocação com os dois valores de MOP para a *trace fipapp*.

Para esta *trace*, com valor de MOP de 10%, houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 47 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 20560 MB.

Apresentando um valor de P_d igual a 2, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,00053, ou seja durante 0,53% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Com MOP no valor de 30% não houve subprovisionamento nem superprovisionamento de memória.

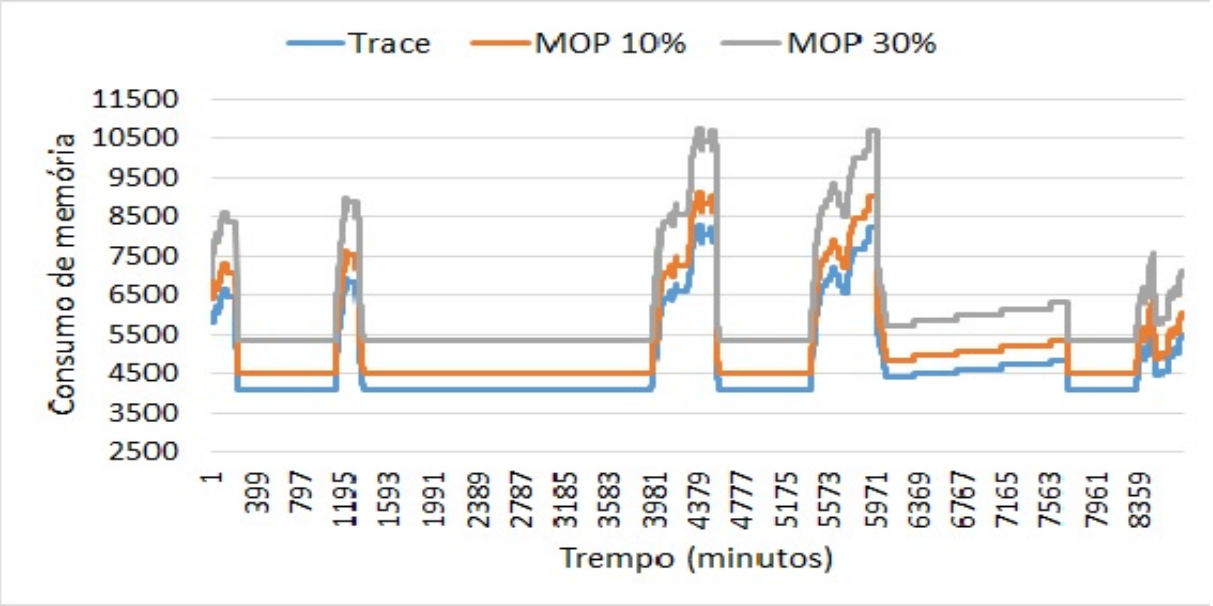

Figura 5.20: *Trace fipapp* Comparativo Moltó 10% e 30%.

A Figura 5.21 mostra o comparativo da alocação com os dois valores de MOP para a *trace fipnfe*.

Para esta *trace*, com valor de MOP de 10%, houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 2 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 90 MB.

Apresentando um valor de P_d próximo a 0, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um bom resultado, pois quanto mais próximo a 0 for o valor de P_d melhor é o desempenho do método.

A métrica $timeshareO$ obteve 0,00002, ou seja durante 0,022% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Com MOP no valor de 30% não houve subprovisionamento nem superprovisionamento de memória.

Nesse caso, a utilização do valor de MOP igual a 10% seria mais vantajoso, pois não apresentou subprovisionamento, e a quantidade de tempo e de MB de superprovisionamento foram baixos. Se utilizássemos o valor de MOP igual a 30% o desperdício na quantidade de memória livre alocada seria maior, já que esse valor de MOP deixa sempre 30% a mais de memória livre para o sistema.

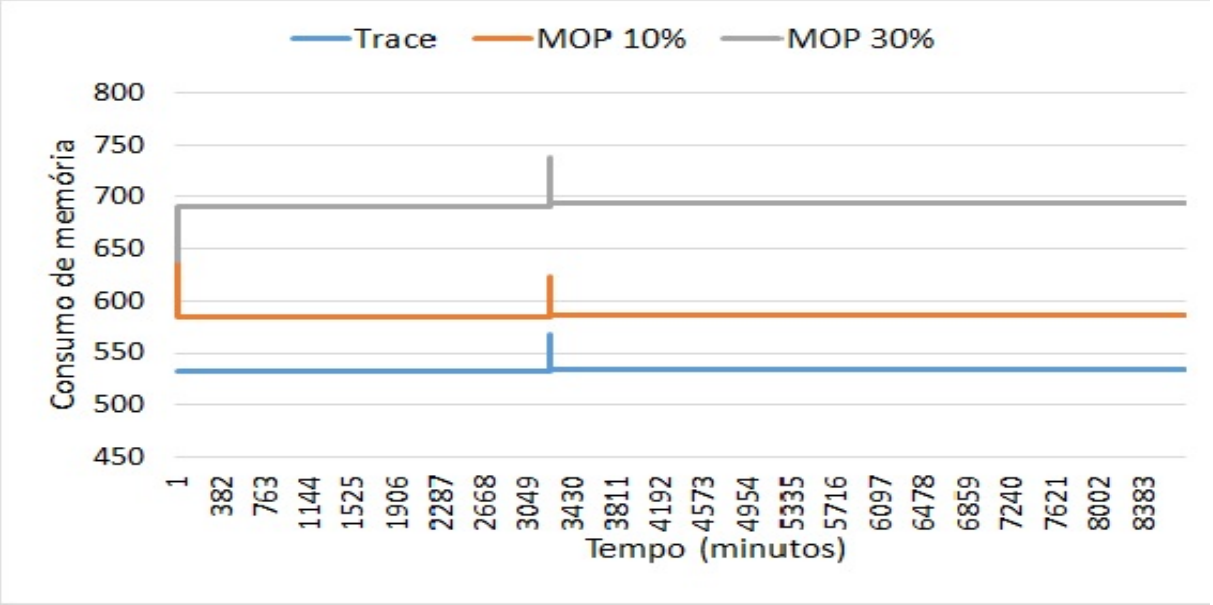

Figura 5.21: *Trace fipnfe* Comparativo Moltó 10% e 30%.

A Figura 5.22 mostra o comparativo da alocação com os dois valores de MOP para a *trace irWTS*. Para esta *trace*, com valor de MOP de 10%, houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 2 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 1862 MB.

Apresentando um valor de P_d próximo a 0, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um bom resultado, pois quanto mais próximo a 0 for o valor de P_d melhor é o desempenho do método.

A métrica $timeshareO$ obteve 0,000022, ou seja durante 0,022% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Com MOP no valor de 30% não houve subprovisionamento nem superprovisionamento de memória.

Nesse caso, a utilização do valor de MOP igual a 10% seria mais vantajoso, pois não apresentou subprovisionamento, e a quantidade de tempo e de MB de superprovisionamento foram baixos. Se utilizássemos o valor de MOP igual a 30% o desperdício na quantidade de memória livre alocada seria maior, já que esse valor de MOP deixa sempre 30% a mais de memória livre para o sistema.

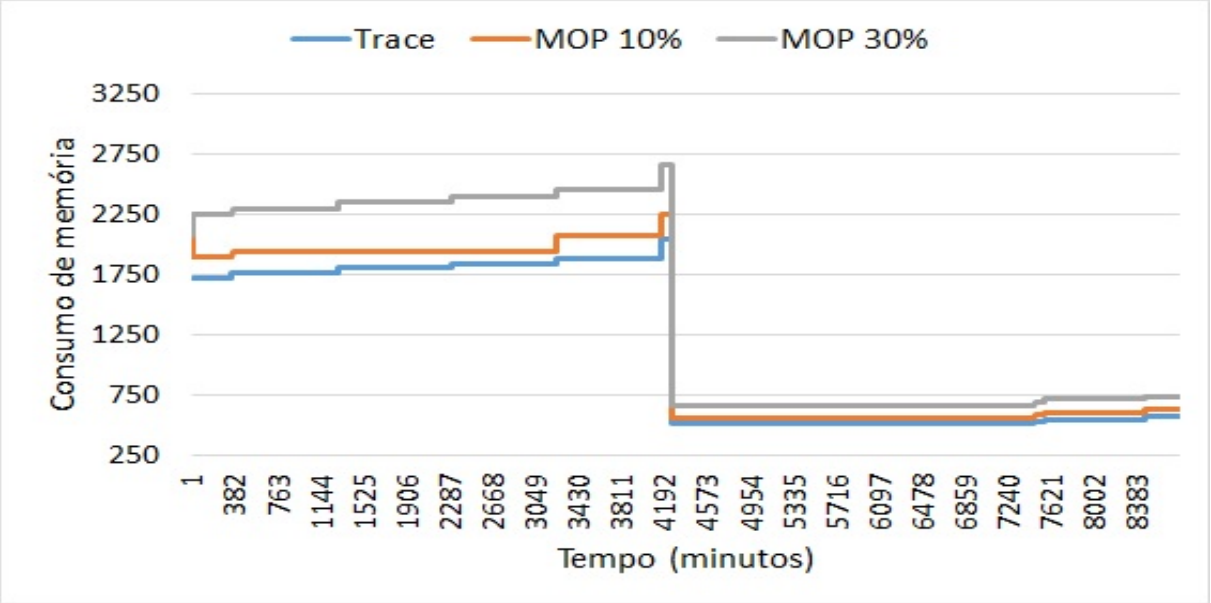

Figura 5.22: *Trace irWTS* Comparativo Moltó 10% e 30%.

A Figura 5.23 mostra o comparativo da alocação com os dois valores de MOP para a *trace WS*.

Para esta *trace*, com valor de MOP de 10%, houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 2828 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 33459 MB.

Apresentando um valor de P_d próximo a 0, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um bom resultado, pois quanto mais próximo a 0 for o valor de P_d melhor é o desempenho do método.

A métrica $timeshareO$ obteve 0,16, ou seja durante 16,8% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Com MOP no valor de 30% não houve subprovisionamento nem superprovisionamento de memória.

Nesse caso, a utilização do valor de MOP igual a 10% seria mais vantajoso, pois não apresentou subprovisionamento, e a quantidade de tempo e de MB de superprovisionamento foram baixos. Se utilizássemos o valor de MOP igual a 30% o desperdício na quantidade de memória livre alocada seria maior, já que esse valor de MOP deixa sempre 30% a mais de memória livre para o sistema.

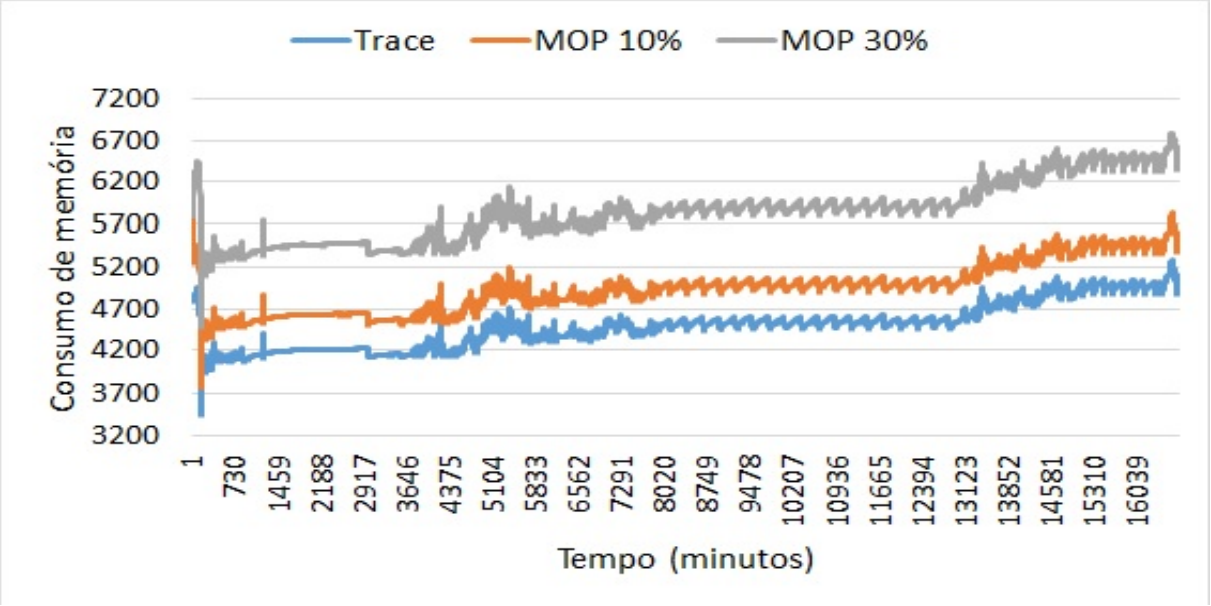

Figura 5.23: *Trace WS* Comparativo Moltó 10% e 30%.

A partir dos testes realizados com os dois valores de MOP é possível verificar que a utilização de um valor de MOP de 30% só é vantajoso nas *traces* que o valor de MOP igual a 10% sofreu subproviosnamento, do contrário, é mais vantajoso a utilização de um MOP de 10%, pois a quantidade de memória mantida livre com MOP igual a 30% é muito maior, mas depende também da necessidade especifica de cada usuário.

5.5 Oliveira/Dawoud/Heo

O método implementado por esses três autores propõem um atuador que calcula a próxima alocação de memória considerando um SLA de 90%, ou seja, a quantidade mínima de memória livre seria de 10% nos momentos de aumento ou diminuição no consumo de memória. O cálculo da previsão da próxima alocação de memória é dada pela fórmula abaixo.

$$u_{mem}(k+1) = u_{mem}(k) + \lambda \times v_{mem}(k) \times (u_{ref}^{mem} - r_{mem}(k))/u_{ref}^{mem} \tag{5.2}$$

Onde,

$$r_{mem} = \frac{v_{mem}(k)}{u_{mem}(k)} \tag{5.3}$$

Na fórmula, $u_{mem}(k+1)$ retorna o valor da próxima alocação de memória, baseado na última alocação de memória $u_{mem}(k)$ e na memória consumida $v_{mem}(k)$. O parâmetro λ é a constante de incremento, que determina a agressividade do atuador, ou seja, define se a quantidade de memória alocada será maior ou menor. O fator $r_{mem}(k)$ é uma média entre a memória usada e a memória alocada no tempo $k-1$. A desvantagem da abordagem é a necessidade da definição manual do valor de referência sobre a taxa de utilização de memória u_{ref}^{mem}, necessária para manter um determinado SLA.

Com esse método foram realizados testes com diferentes valores de referência sobre a taxa de utilização de memória, com valores de 90% e 70%, e para ambos, foram realizados testes com valores de λ iguais a 0,8, 1 e 1,2.

Considerando os valores de referência u_{ref}^{mem} de 70% e 90% com diferentes valores de λ, não houve alterações significativas nos gráficos de resultados, a linha desses valores de λ ficaram praticamente iguais.

Para o valor de referência u_{ref}^{mem} de 90% e λ igual a 1 e para o valor de referência u_{ref}^{mem} de 70% e λ igual a 1, nenhuma *trace* sintética apresentou problema de subprovisionamento ou superprovisionamento.

A Figura 5.24 mostra um comparativo da alocação de memória para esses valores com a *trace crescente*.

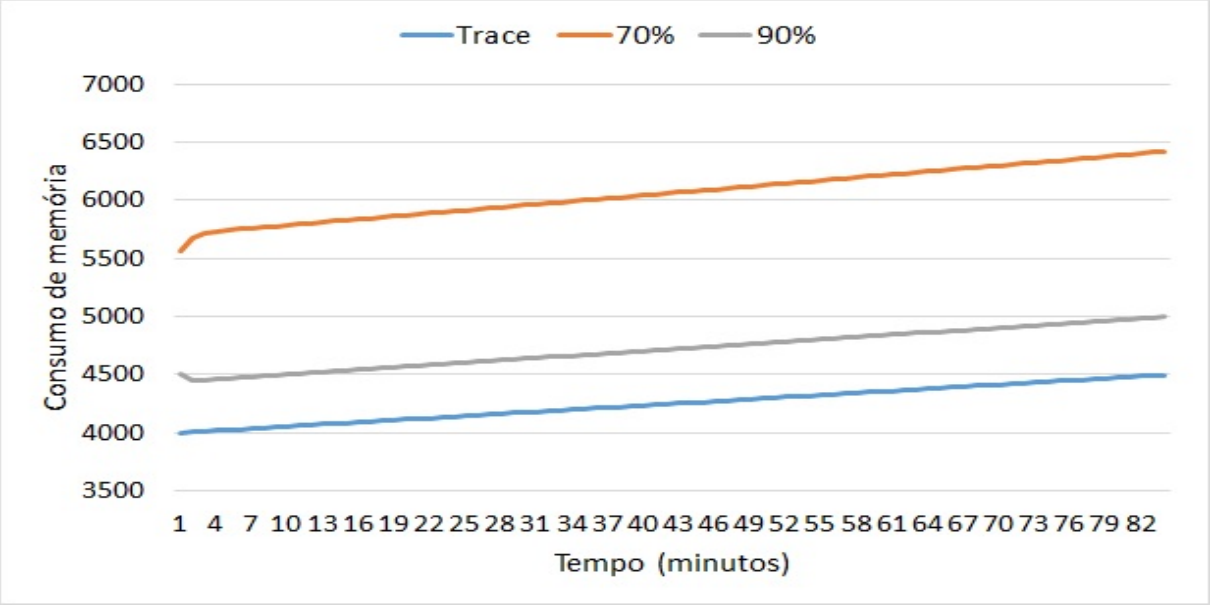

Figura 5.24: *Trace crescente* - Oliveira 90% e 70% com λ 1.

A Figura 5.25 mostra um comparativo da alocação de memória para esses valores com a *trace decrescente* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1.

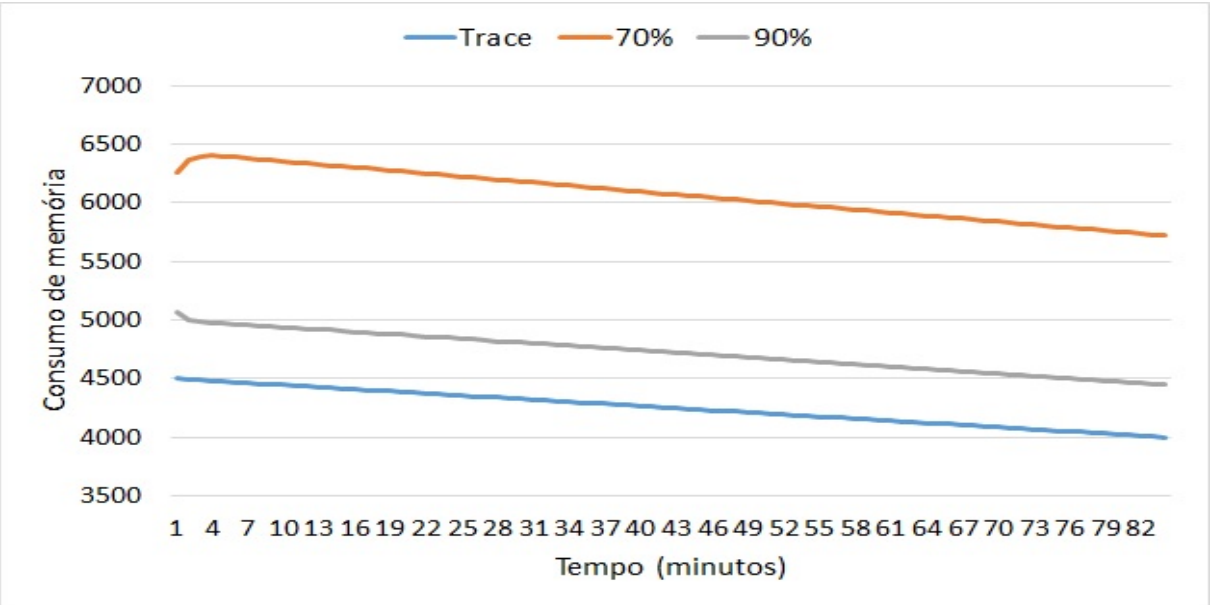

Figura 5.25: *Trace decrescente* - Oliveira 90% e 70% com λ 1.

A Figura 5.26 mostra um comparativo da alocação de memória para esses valores com a *trace picos* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1.

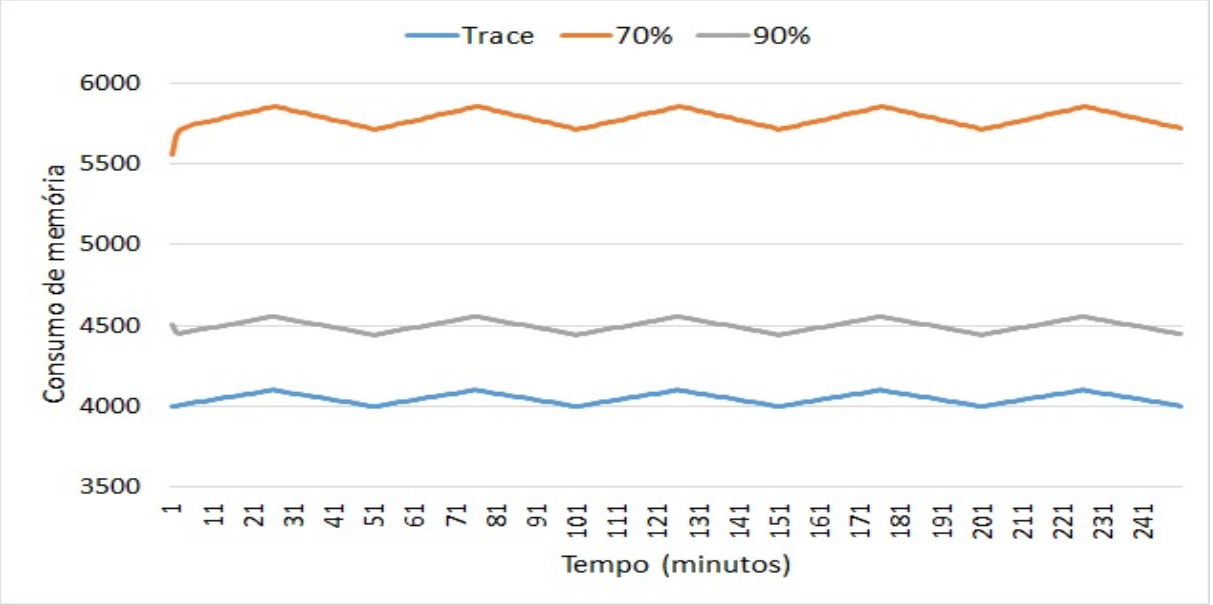

Figura 5.26: *Trace picos* - Oliveira 90% e 70% com λ 1.

A Figura 5.27 mostra um comparativo da alocação de memória para esses valores com a *trace piconst* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1.

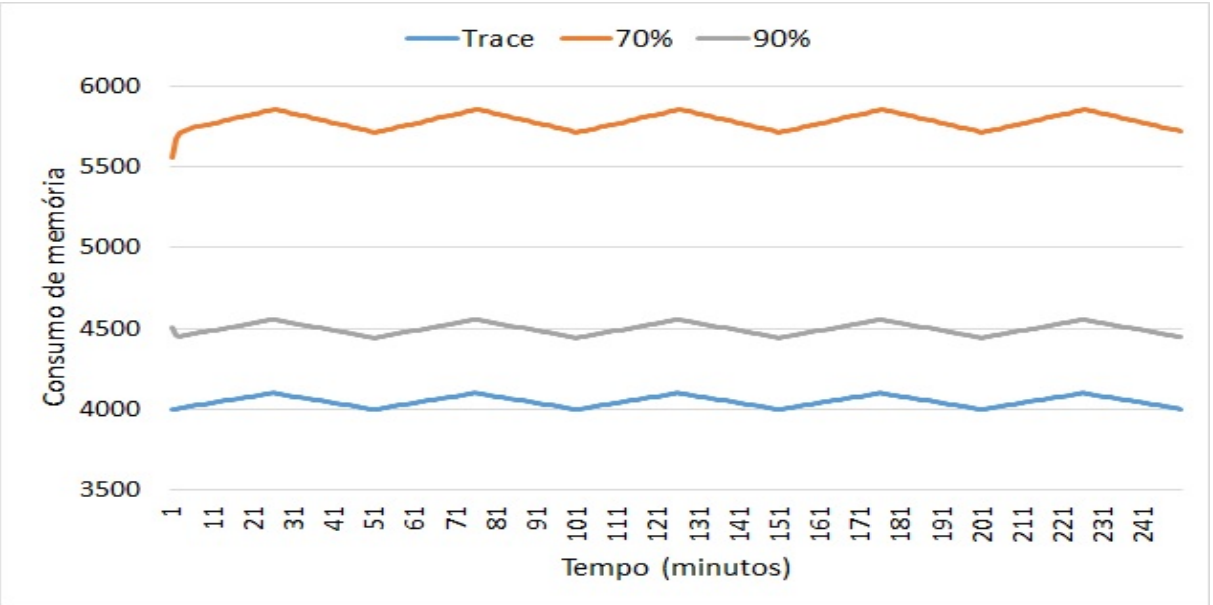

Figura 5.27: *Trace piconst* - Oliveira 90% e 70% com λ 1.

A Figura 5.28 mostra um comparativo da alocação de memória para esses valores com a *trace const* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1.

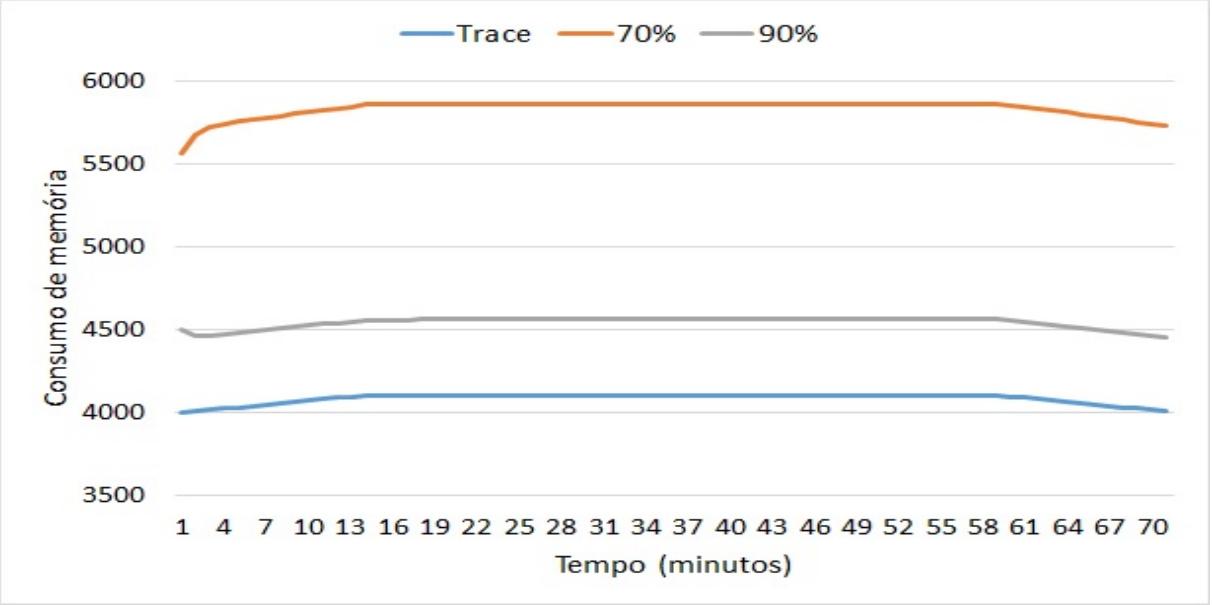

Figura 5.28: *Trace const* - Oliveira 90% e 70% com λ 1.

Nesses casos, é possível notar que o valor de referência u_{ref}^{mem} de 70%, deixa uma quantidade de memória livre maior do que o u_{ref}^{mem} de 90%.

Para as *traces* de consumo real o valor de referência u_{ref}^{mem} de 90% e λ igual a 1 e para o valor de referência u_{ref}^{mem} de 70% e λ igual a 1 não apresentaram problemas de subprovisionamento ou superprovisionamento.

A Figura 5.29 mostra um comparativo da alocação de memória para esses valores com a *trace fipS*.

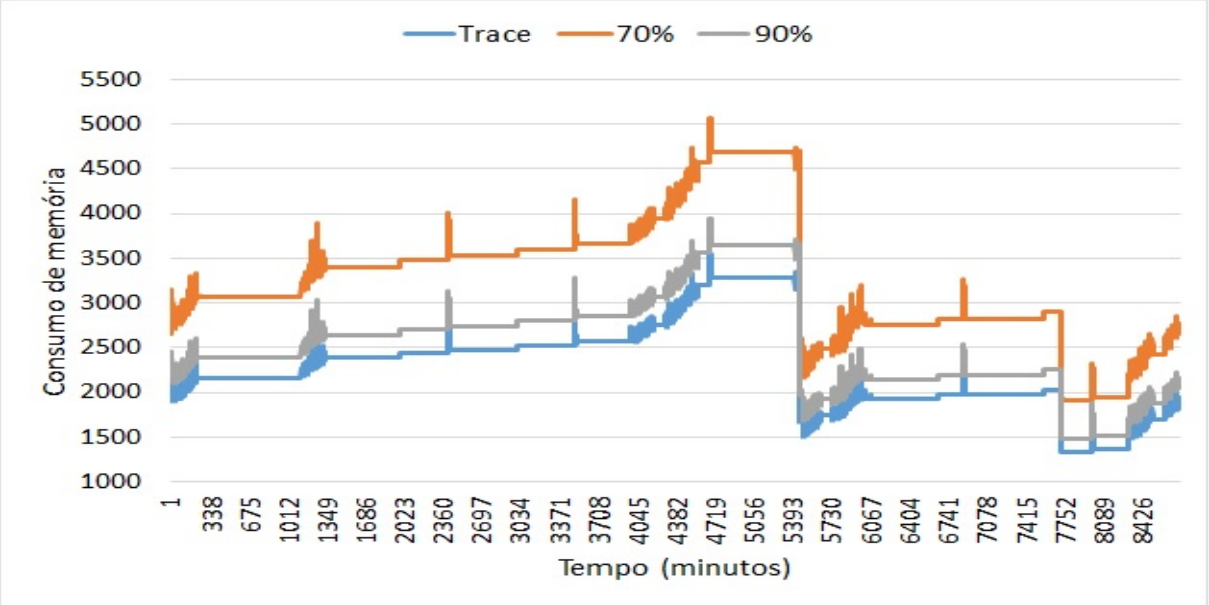

Figura 5.29: *Trace fipS* - Oliveira 90% e 70% com λ 1.

A Figura 5.30 mostra um comparativo da alocação de memória para esses valores com a *trace dalDC1* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1.

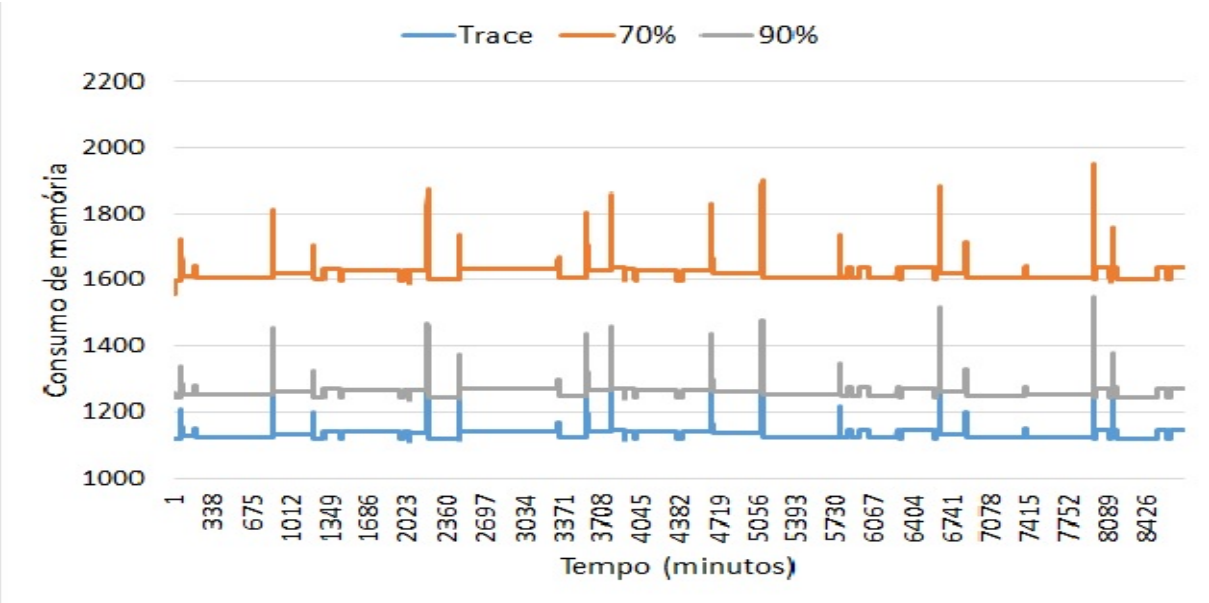

Figura 5.30: *Trace dalDC1* - Oliveira 90% e 70% com λ 1.

A Figura 5.31 mostra um comparativo da alocação de memória para esses valores com a *trace fipapp* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1.

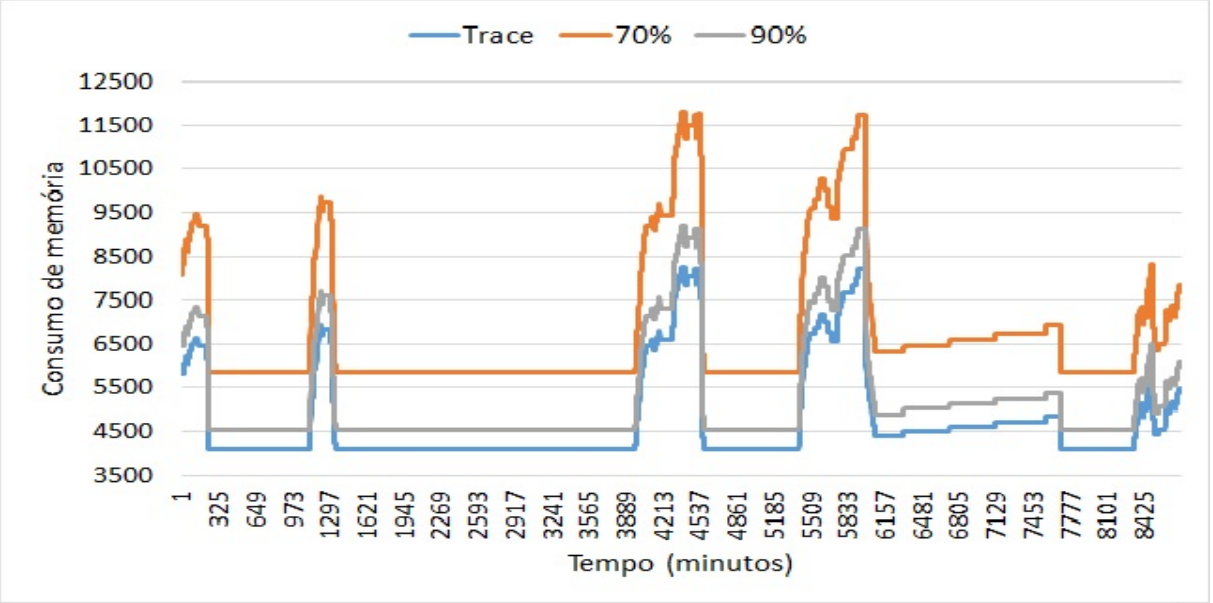

Figura 5.31: *Trace fipapp* - Oliveira 90% e 70% com λ 1.

A Figura 5.32 mostra um comparativo da alocação de memória para esses valores com a *trace fipnfe* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1.

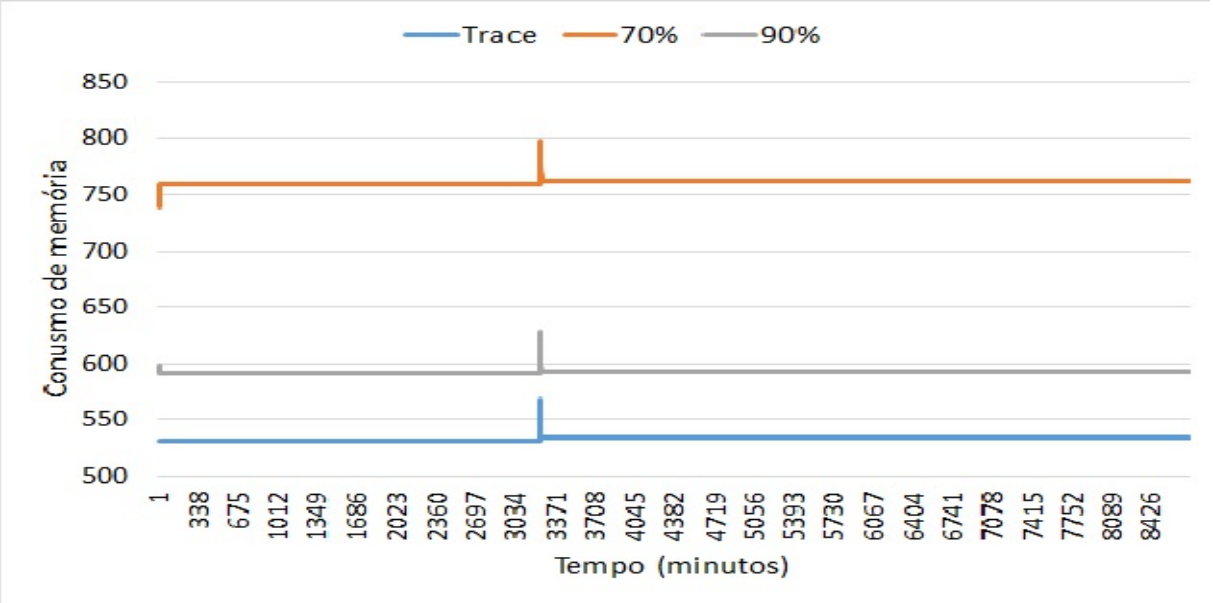

Figura 5.32: *Trace fipnfe* - Oliveira 90% e 70% com λ 1.

A Figura 5.33 mostra um comparativo da alocação de memória para esses valores com a *trace irWTS* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1.

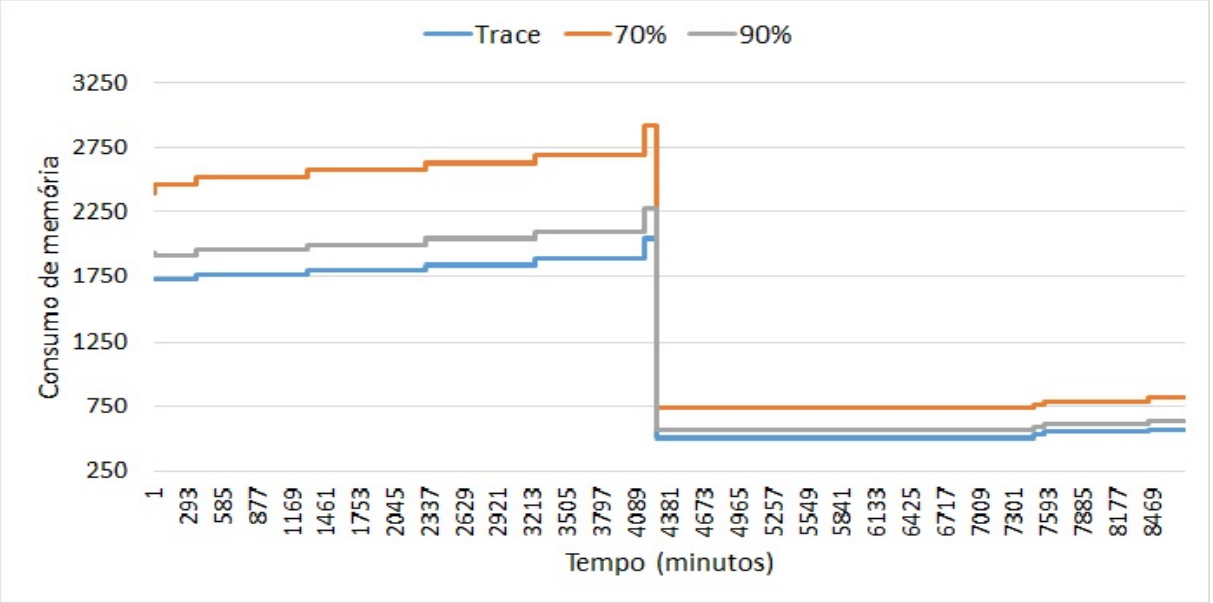

Figura 5.33: *Trace irWTS* - Oliveira 90% e 70% com λ 1.

A Figura 5.34 mostra um comparativo da alocação de memória para esses valores com a *trace TVL* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1.

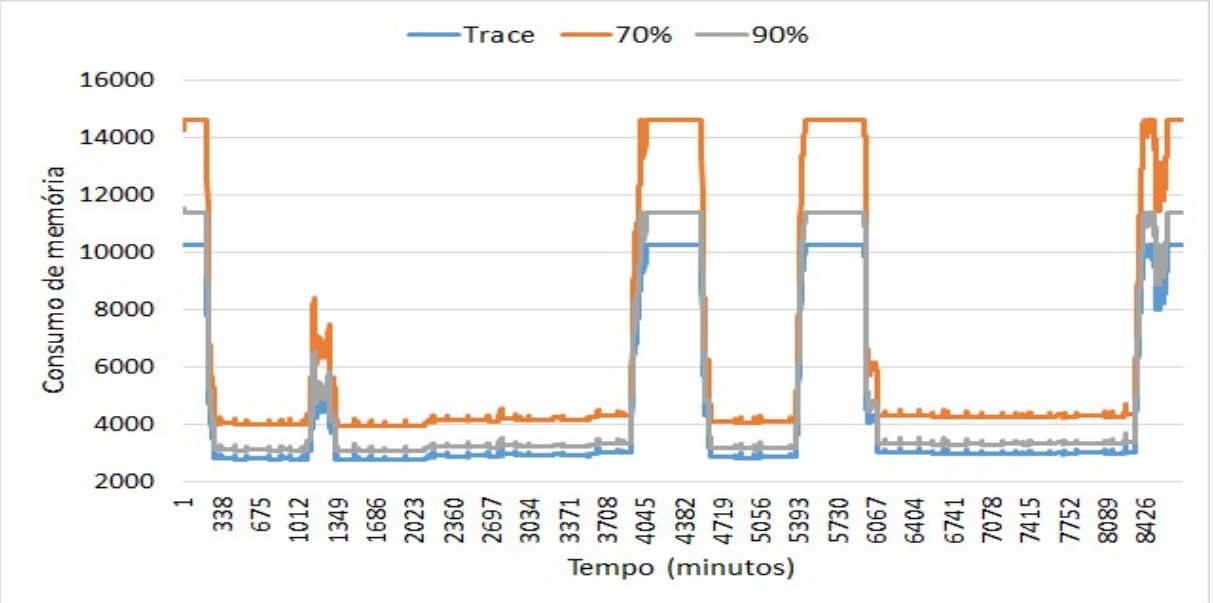

Figura 5.34: *Trace TVL* - Oliveira 90% e 70% com λ 1.

A Figura 5.35 mostra um comparativo da alocação de memória para esses valores com a *trace WS* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1.

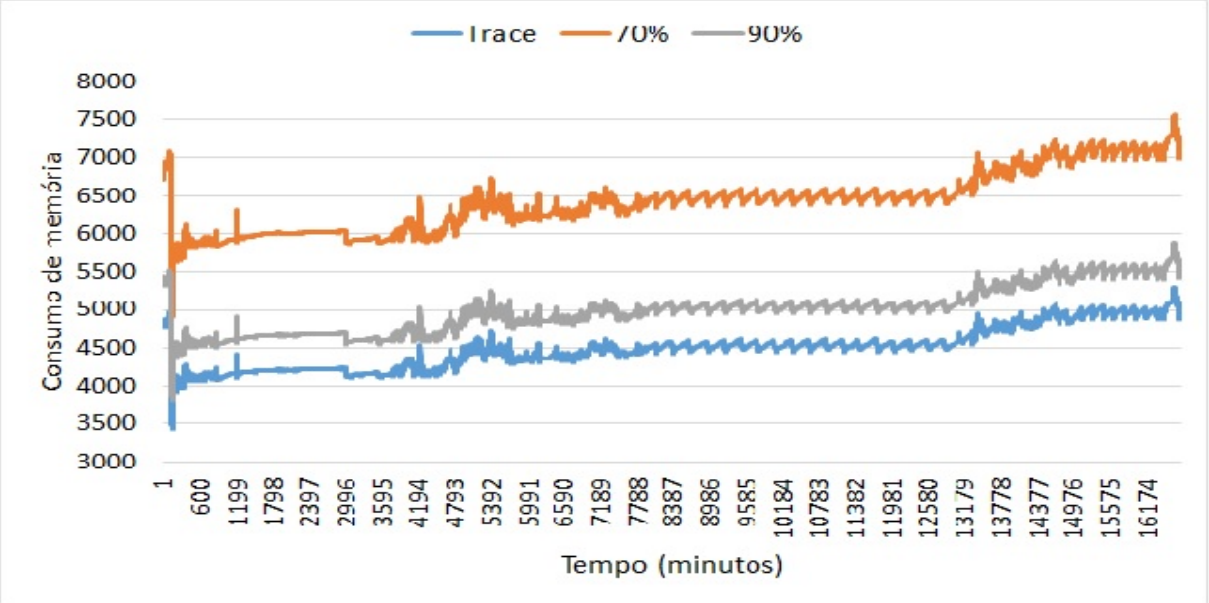

Figura 5.35: *Trace WS* - Oliveira 90% e 70% com λ 1.

Nesses casos, é possível notar que o valor de referência u_{ref}^{mem} de 70%, deixa uma quantidade de memória livre maior do que o u_{ref}^{mem} de 90%.

O valor de P_u, P_d são todos iguais a 0 nas *traces* acima, ou seja, a precisão do método com esses valores é ótima.

A métrica $timeshareU$ e $timeshareO$ também apresentam valores iguais a 0, pois não há subprovisionamento e nem superprovisionamento.

A definição de qual desses valores são melhores em cada caso vai depender da quantidade de memória livre que o usuário ou sistema irá precisar.

Para o valor de referência u_{ref}^{mem} de 90% e λ igual a 0,8 houve problemas de superprovisionamento de memória em 7 *traces*, todas de consumo real.

Já para o valor de referência u_{ref}^{mem} de 70% e λ igual a 0,8 nenhuma *trace* sintética apresentou problema de subprovisionamento ou superprovisionamento.

A Figura 5.36 mostra um comparativo da alocação de memória para esses valores com a *trace crescente*, essa *trace* não apresentou subprovisionamento e nem superprovisionamento.

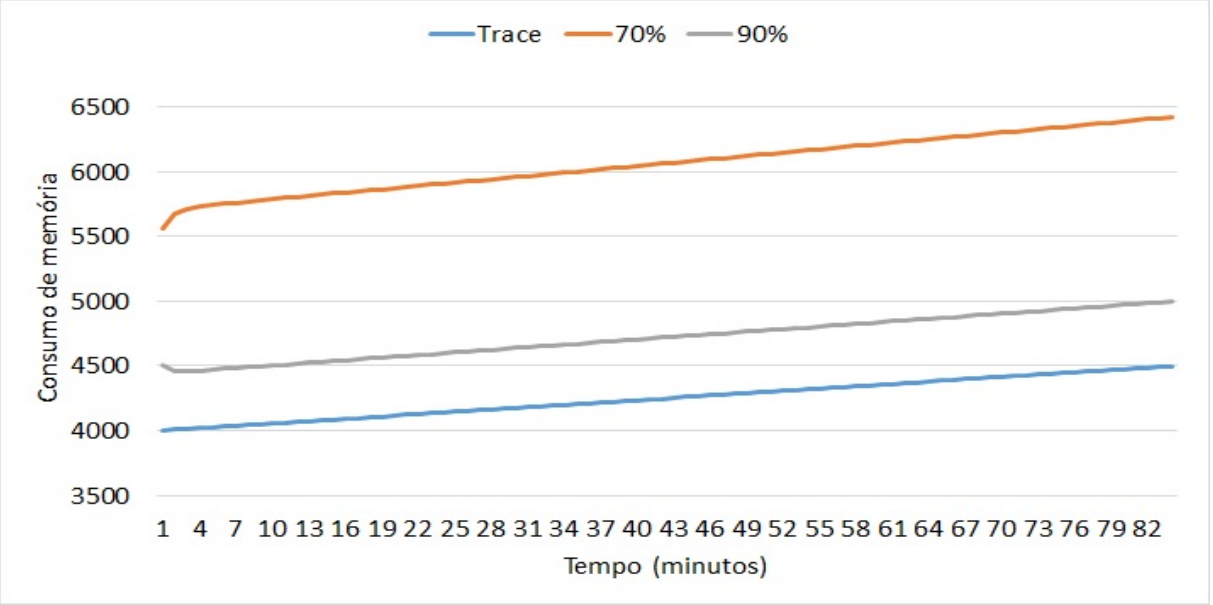

Figura 5.36: *Trace crescente* - Oliveira 90% e 70% com λ 0,8.

A Figura 5.37 mostra um comparativo da alocação de memória para esses valores com a *trace decrescente* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 0,8.

Essa *trace* não apresentou subprovisionamento e nem superprovisionamento.

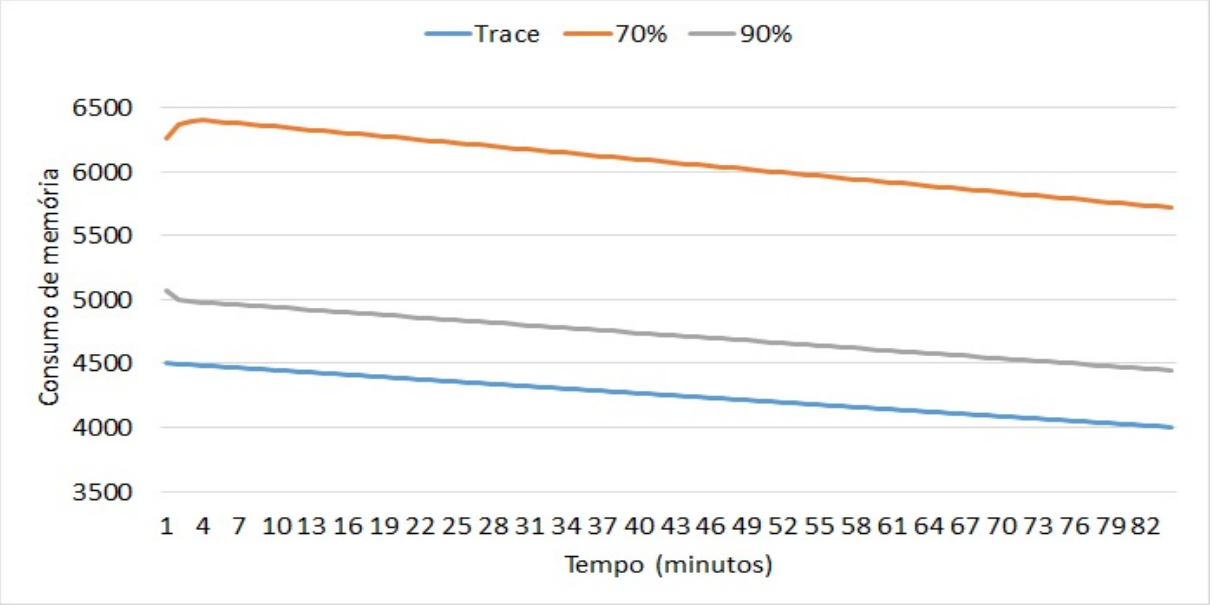

Figura 5.37: *Trace decrescente* - Oliveira 90% e 70% com λ 0,8.

A Figura 5.38 mostra um comparativo da alocação de memória para esses valores com a *trace const* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 0,8. Essa

trace não apresentou subprovisionamento e nem superprovisionamento.

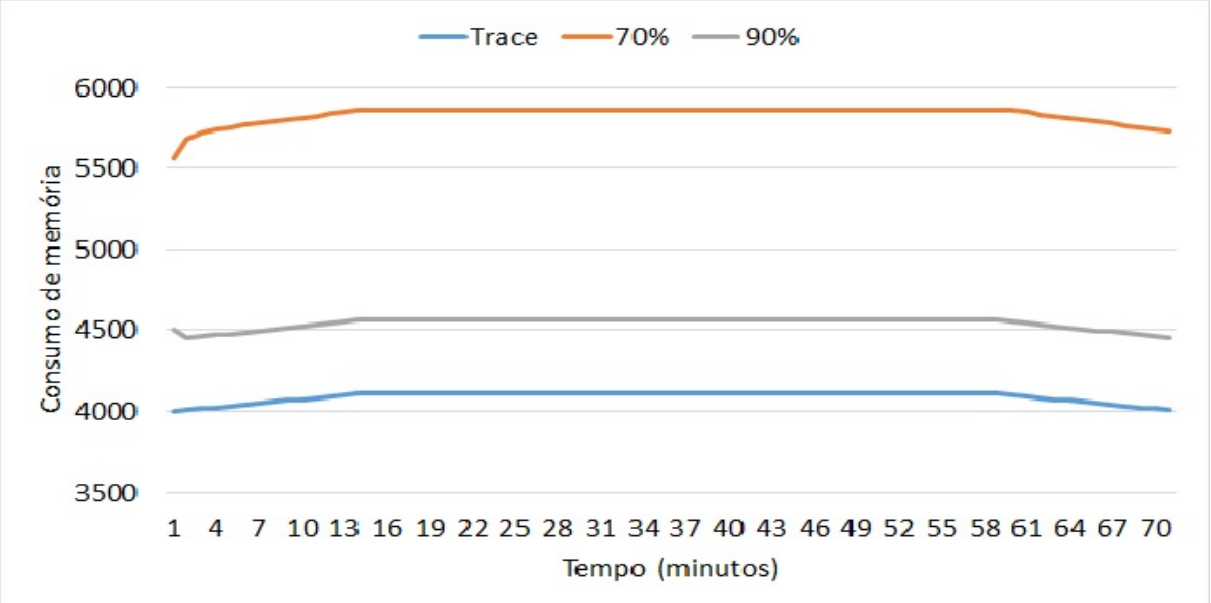

Figura 5.38: *Trace const* - Oliveira 90% e 70% com λ 0,8.

A Figura 5.39 mostra um comparativo da alocação de memória para esses valores com a *trace piconst* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 0,8. Essa *trace* não apresentou subprovisionamento e nem superprovisionamento.

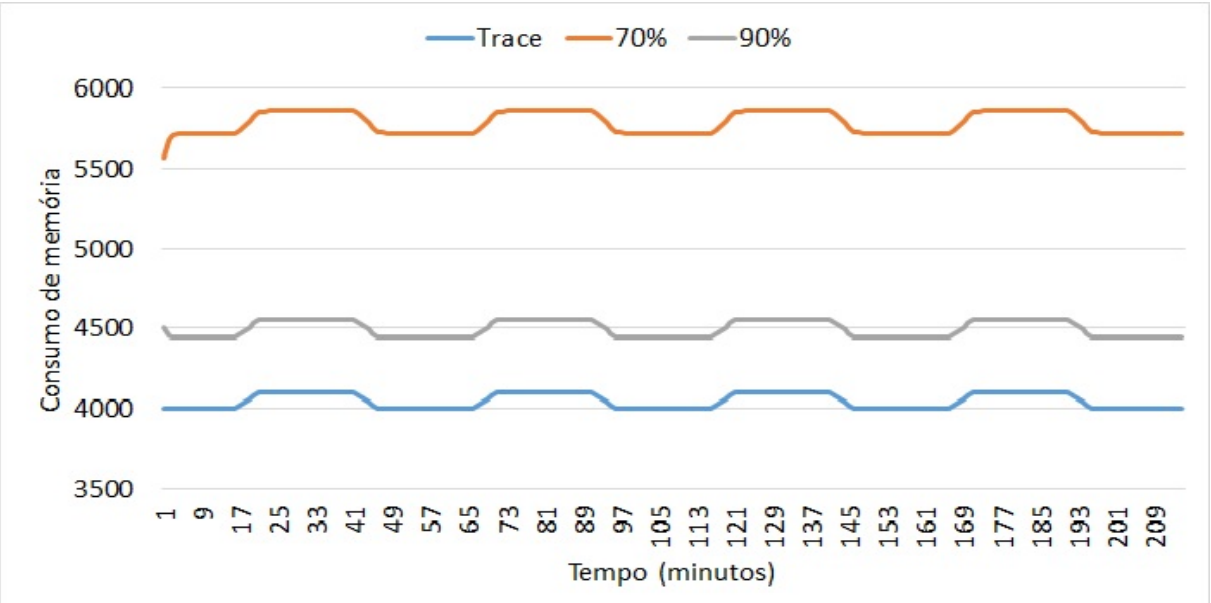

Figura 5.39: *Trace piconst* - Oliveira 90% e 70% com λ 0,8.

A Figura 5.40 mostra um comparativo da alocação de memória para esses valores com a

trace picos considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 0,8. Essa *trace* não apresentou subprovisionamento e nem superprovisionamento.

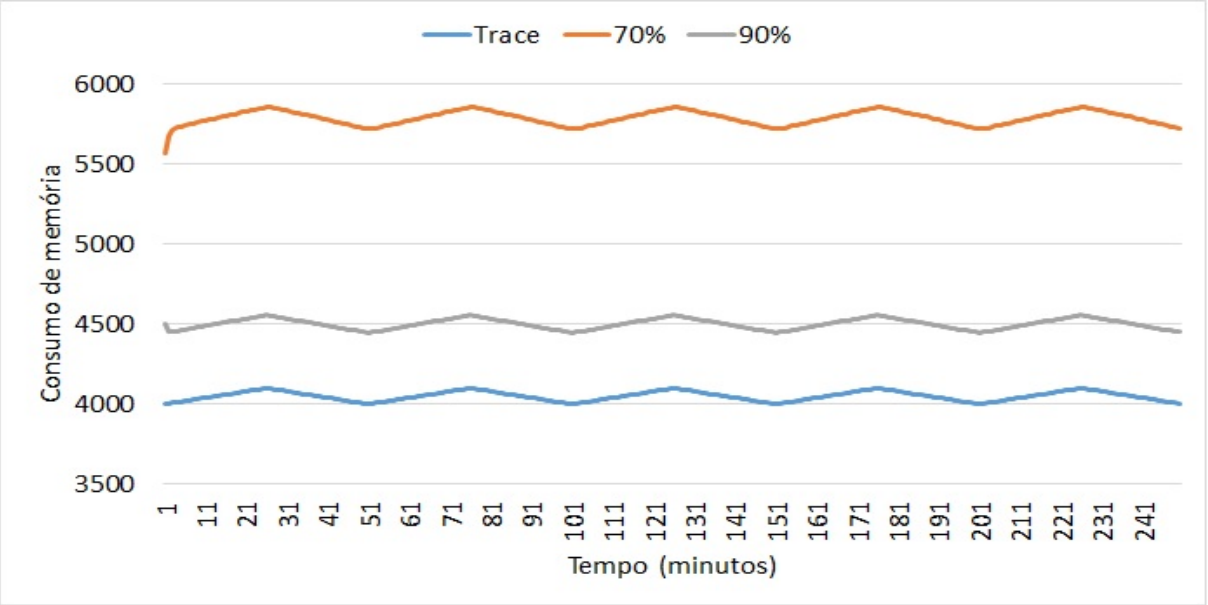

Figura 5.40: *Trace picos* - Oliveira 90% e 70% com λ 0,8.

Nesses casos, é possível notar que o valor de referência u_{ref}^{mem} de 70%, deixa uma quantidade de memória livre maior do que o u_{ref}^{mem} de 90%.

A Figura 5.41 mostra um comparativo da alocação de memória para esses valores com a *trace fipS*.

Para esta *trace*, com valor de referência u_{ref}^{mem} de 90% houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 8610 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 226071 MB. Apresentando um valor de P_d igual a 25, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um péssimo resultado, pois quanto maior o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,98, ou seja durante 98% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

Para essa *trace* durante quase todo o tempo de coleta houve superprovisionamento de memória.

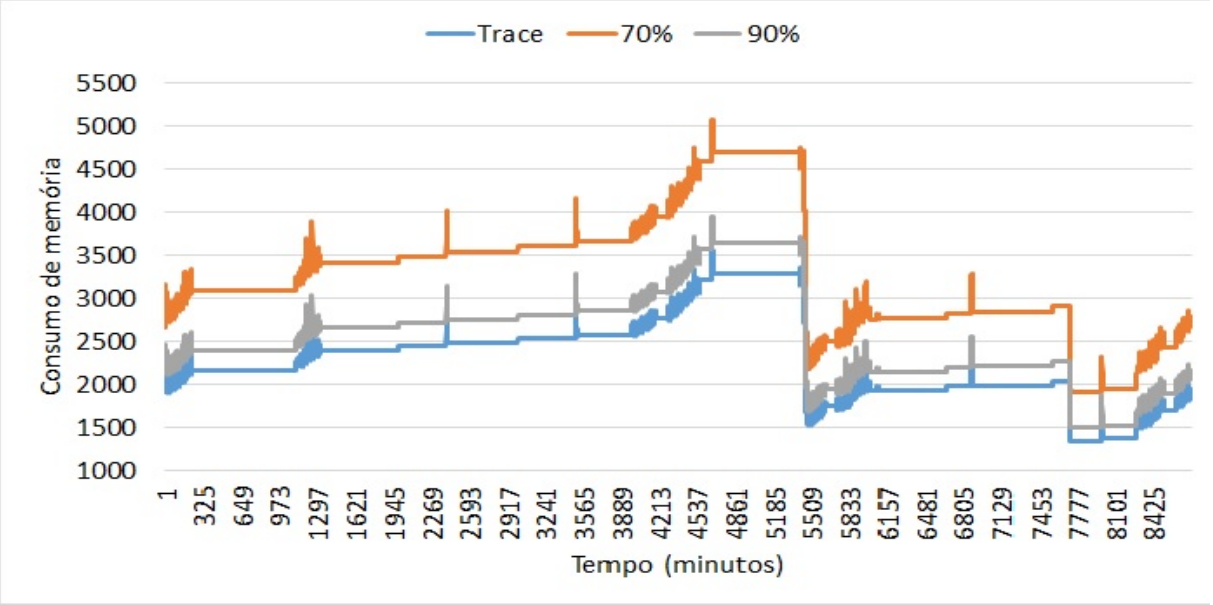

Figura 5.41: *Trace fipS* - Oliveira 90% e 70% com λ 0,8.

A Figura 5.42 mostra um comparativo da alocação de memória para esses valores com a *trace dalDC1* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 0,8.

Para esta *trace*, com valor de referência u_{ref}^{mem} de 90% houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 8121 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 103516 MB.

Apresentando um valor de P_d igual a 11, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,92, ou seja durante 92% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

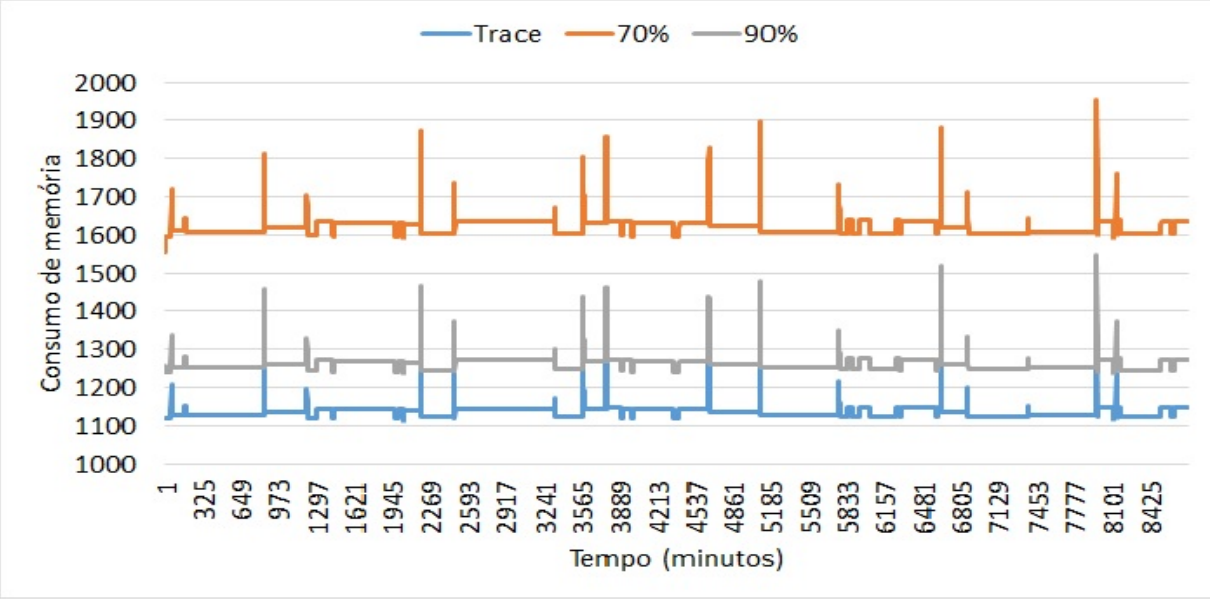

Figura 5.42: *Trace dalDC1* - Oliveira 90% e 70% com λ 0,8.

A Figura 5.43 mostra um comparativo da alocação de memória para esses valores com a *trace fipapp* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 0,8.

Para esta *trace*, com valor de referência u_{ref}^{mem} de 90% houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 8338 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 445069 MB.

Apresentando um valor de P_d igual a 50, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,95, ou seja durante 95% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

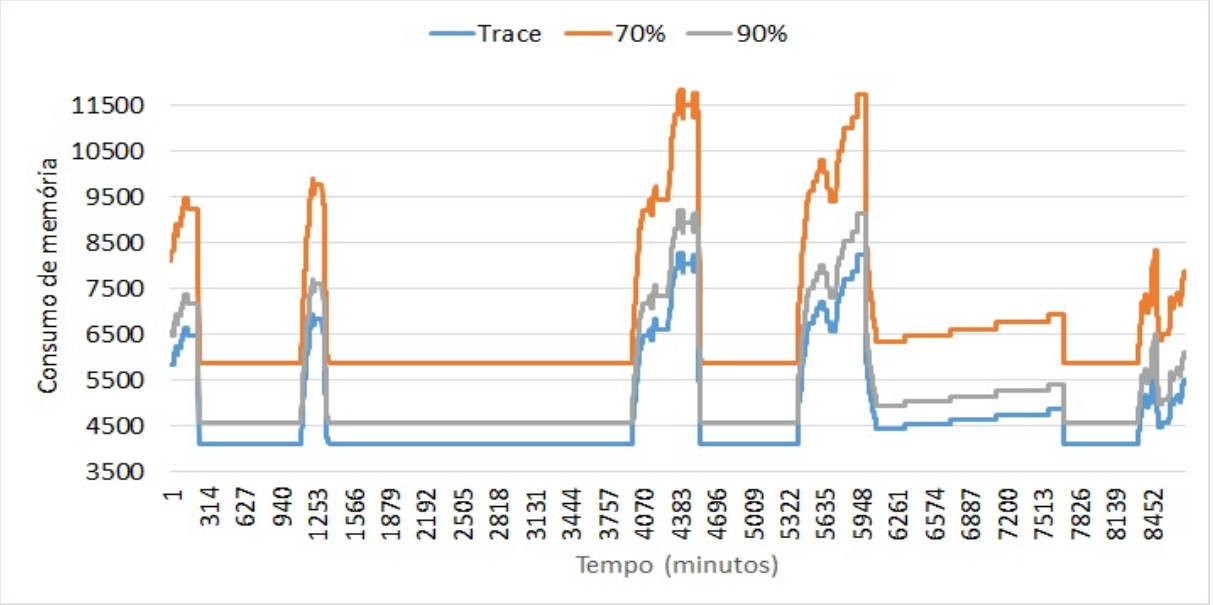

Figura 5.43: *Trace fipapp* - Oliveira 90% e 70% com λ 0,8.

A Figura 5.44 mostra um comparativo da alocação de memória para esses valores com a *trace fipnfe* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 0,8.

Para esta *trace*, com valor de referência u_{ref}^{mem} de 90% houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 3245 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 19490 MB. Apresentando um valor de P_d igual a 2, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,37, ou seja durante 37% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

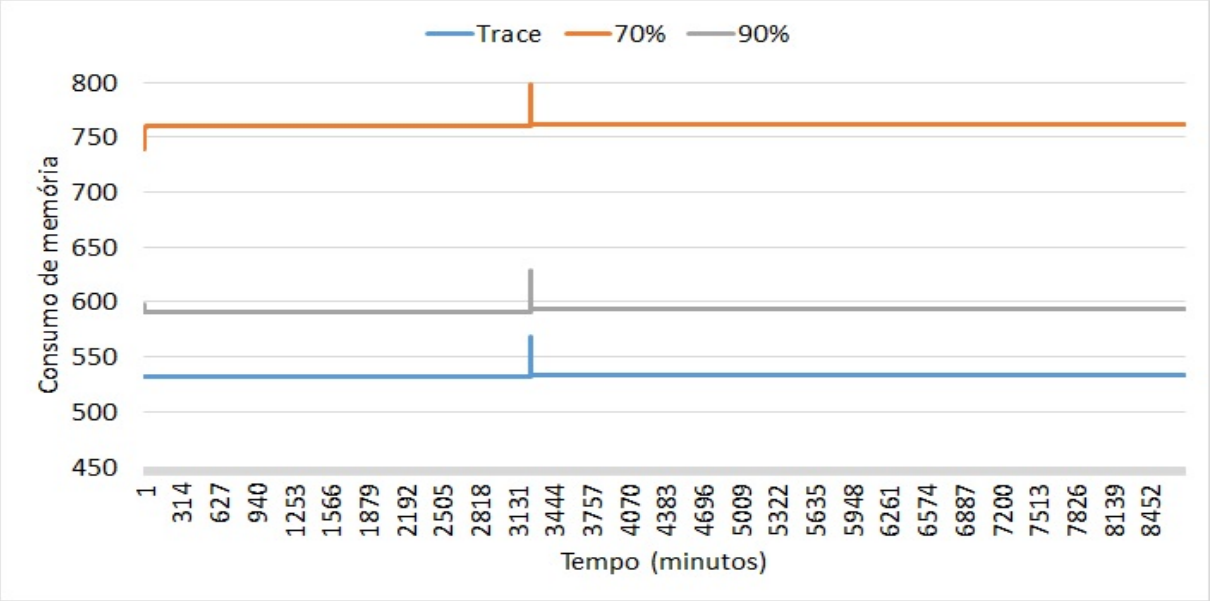

Figura 5.44: *Trace fipnfe* - Oliveira 90% e 70% com λ 0,8.

A Figura 5.45 mostra um comparativo da alocação de memória para esses valores com a *trace irWTS* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 0,8.

Para esta *trace*, com valor de referência u_{ref}^{mem} de 90% houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 8438 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 112627 MB. Apresentando um valor de P_d igual a 12, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,96, ou seja durante 96% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

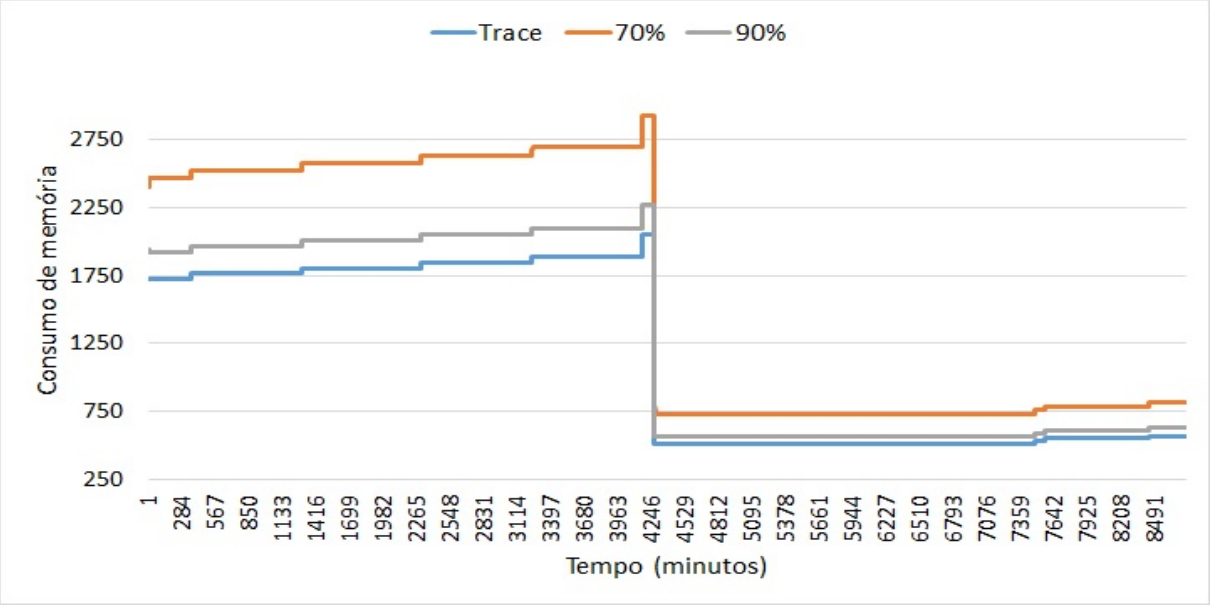

Figura 5.45: *Trace irWTS* - Oliveira 90% e 70% com λ 0,8.

A Figura 5.46 mostra um comparativo da alocação de memória para esses valores com a *trace TVL* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 0,8.

Para esta *trace*, com valor de referência u_{ref}^{mem} de 90% houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 8515 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 427068 MB. Apresentando um valor de P_d igual a 48, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,97, ou seja durante 97% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

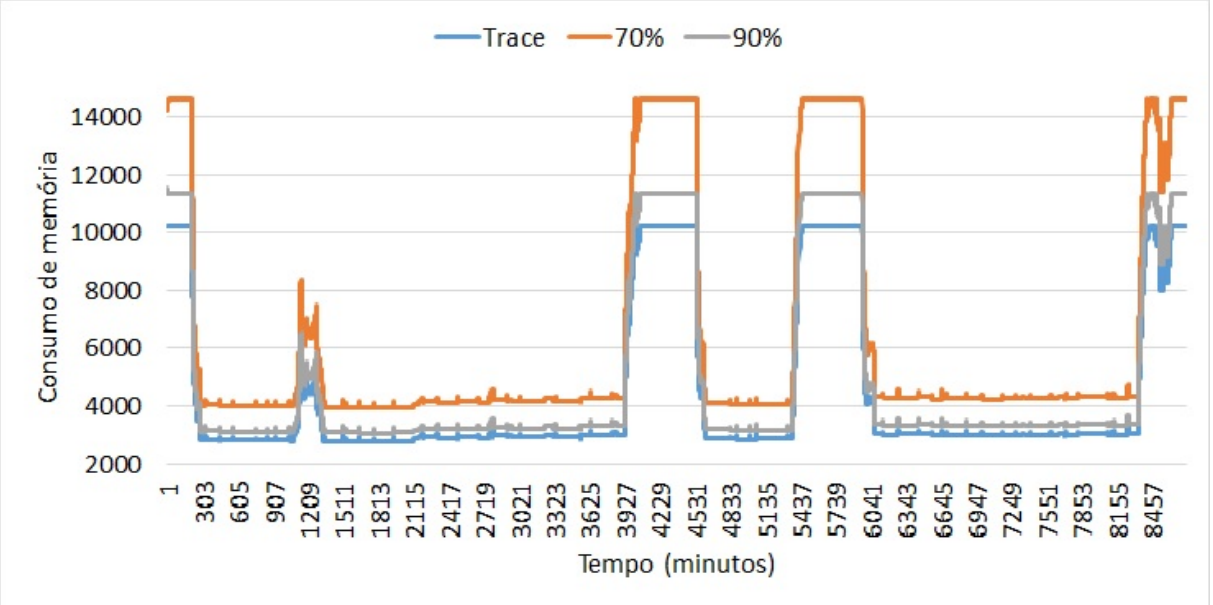

Figura 5.46: *Trace TVL* - Oliveira 90% e 70% com λ 0,8.

A Figura 5.47 mostra um comparativo da alocação de memória para esses valores com a *trace WS* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 0,8.

Para esta *trace*, com valor de referência u_{ref}^{mem} de 90% houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 5700 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 269474 MB. Apresentando um valor de P_d igual a 16, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,65, ou seja durante 65% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

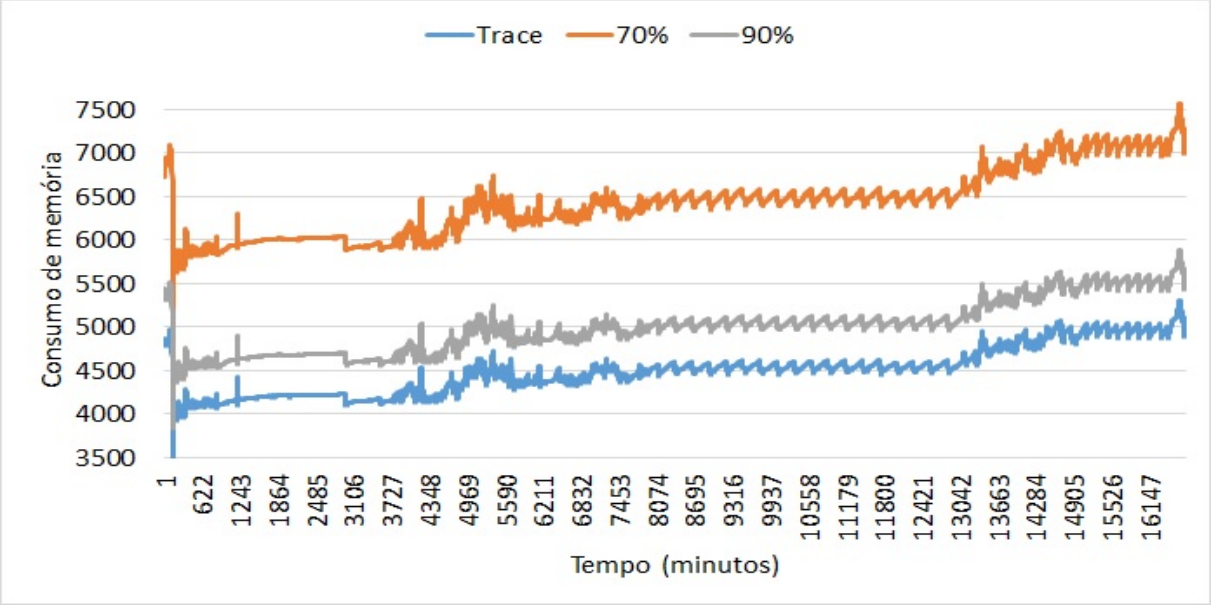

Figura 5.47: *Trace WS* - Oliveira 90% e 70% com λ 0,8.

Para o valor de referência u_{ref}^{mem} de 90% e λ igual a 1,2 e para o valor de referência u_{ref}^{mem} de 70% e λ igual a 1,2, nenhuma *trace* sintética apresentou problema de subprovisionamento ou superprovisionamento.

A Figura 5.48 mostra um comparativo da alocação de memória para esses valores com a *trace crescente*.

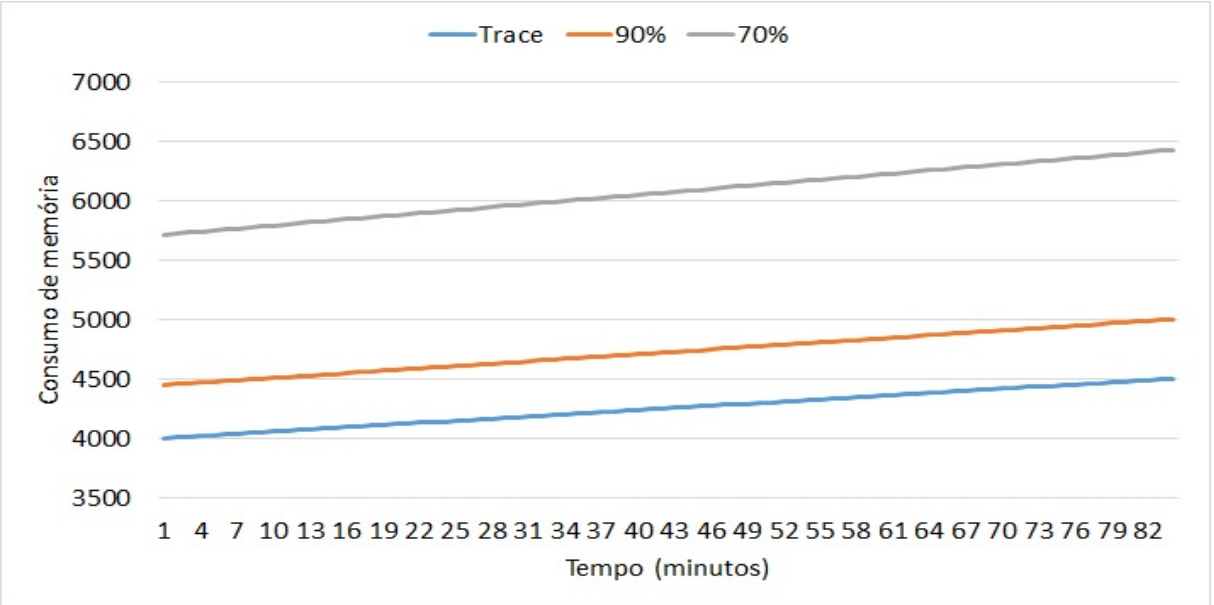

Figura 5.48: *Trace crescente* - Oliveira 90% e 70% com λ 1,2.

A Figura 5.49 mostra um comparativo da alocação de memória para esses valores com a *trace decrescente* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1,2.

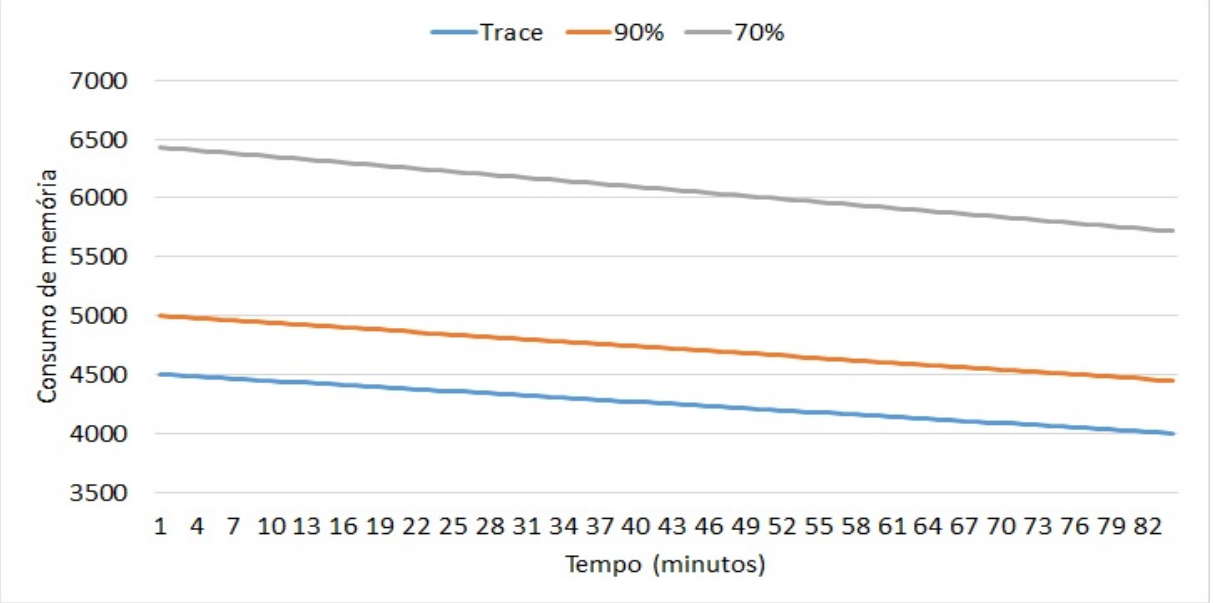

Figura 5.49: *Trace decrescente* - Oliveira 90% e 70% com λ 1,2.

A Figura 5.50 mostra um comparativo da alocação de memória para esses valores com a *trace picos* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1,2.

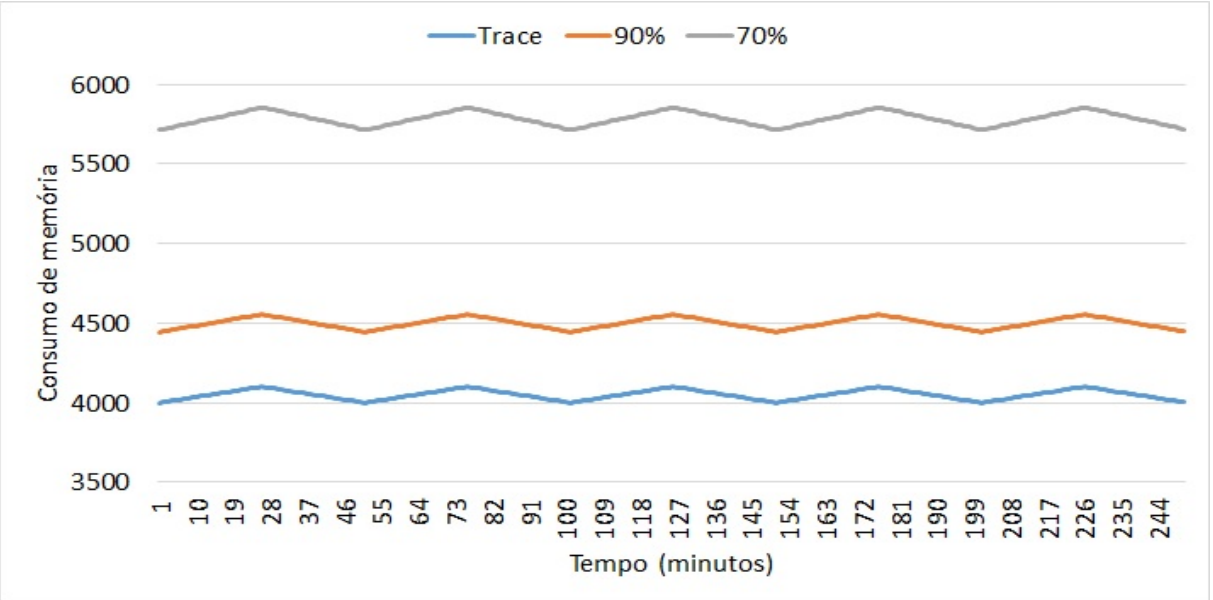

Figura 5.50: *Trace picos* - Oliveira 90% e 70% com λ 1,2.

A Figura 5.51 mostra um comparativo da alocação de memória para esses valores com a

trace piconst considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1,2.

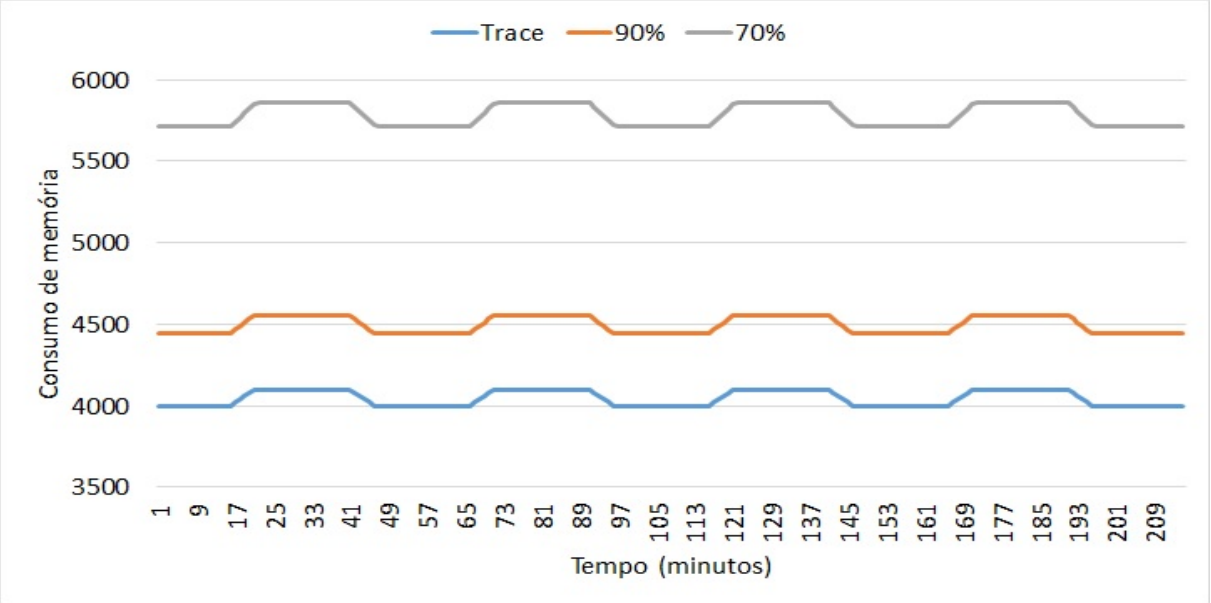

Figura 5.51: *Trace piconst* - Oliveira 90% e 70% com λ 1,2.

A Figura 5.52 mostra um comparativo da alocação de memória para esses valores com a *trace const* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1,2.

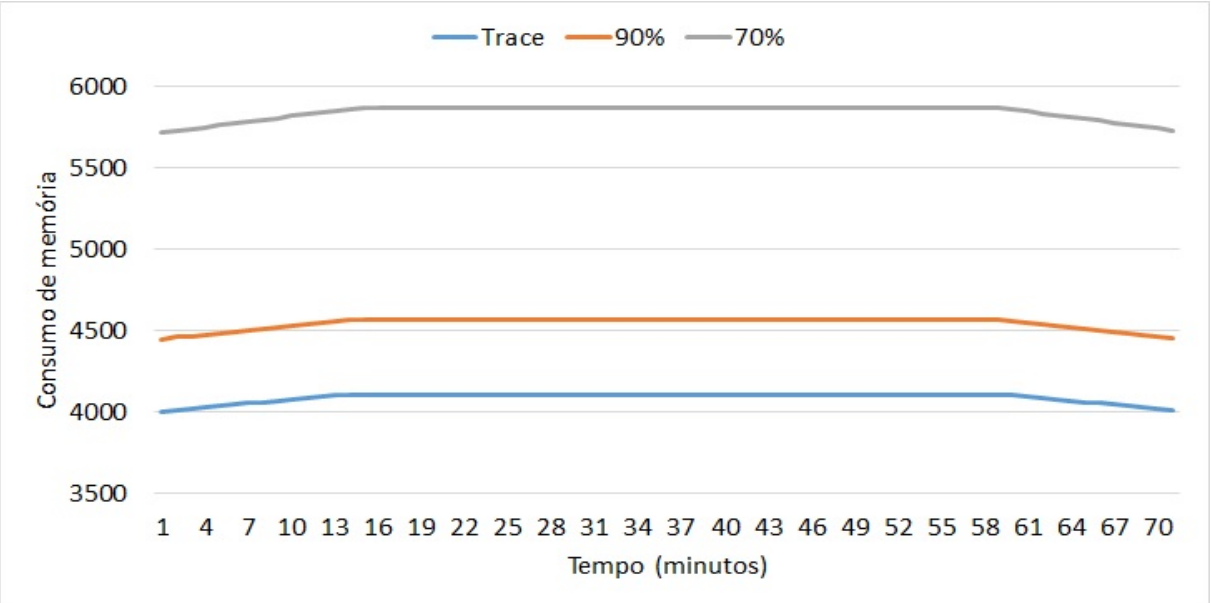

Figura 5.52: *Trace const* - Oliveira 90% e 70% com λ 1,2.

Nesses casos, é possível notar que o valor de referência u_{ref}^{mem} de 70%, deixa uma quantidade de memória livre maior do que o u_{ref}^{mem} de 90%.

Para as *traces* de consumo real o valor de referência u_{ref}^{mem} de 90% e λ igual a 1,2 e para o valor de referência u_{ref}^{mem} de 70% e λ igual a 1,2 não apresentaram problemas de subprovisionamento ou superprovisionamento.

A Figura 5.53 mostra um comparativo da alocação de memória para esses valores com a *trace fipS*.

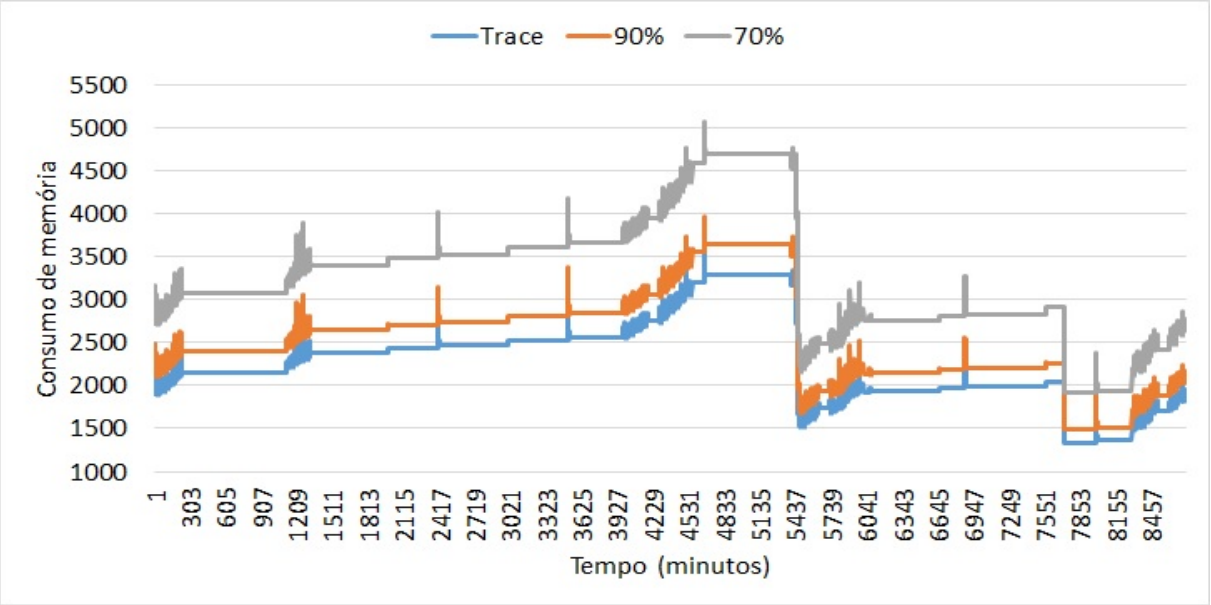

Figura 5.53: *Trace fipS* - Oliveira 90% e 70% com λ 1,2.

A Figura 5.54 mostra um comparativo da alocação de memória para esses valores com a *trace dalDC1* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1,2.

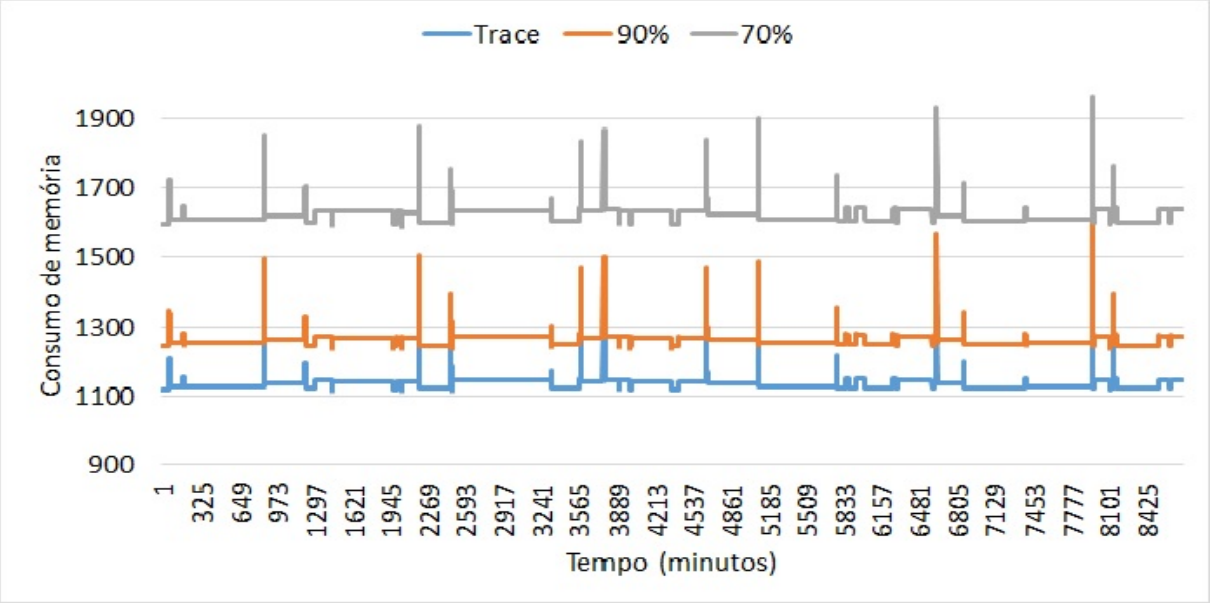

Figura 5.54: *Trace dalDC1* - Oliveira 90% e 70% com λ 1,2.

A Figura 5.55 mostra um comparativo da alocação de memória para esses valores com a *trace fipapp* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1,2.

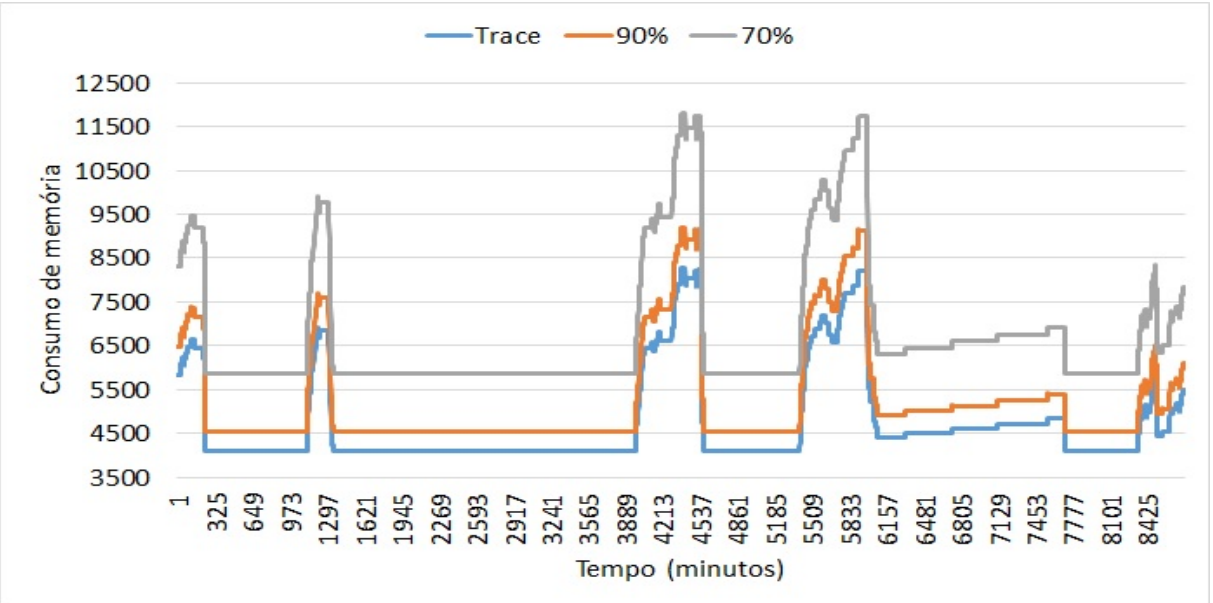

Figura 5.55: *Trace fipapp* - Oliveira 90% e 70% com λ 1,2.

A Figura 5.56 mostra um comparativo da alocação de memória para esses valores com a *trace fipnfe* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1,2.

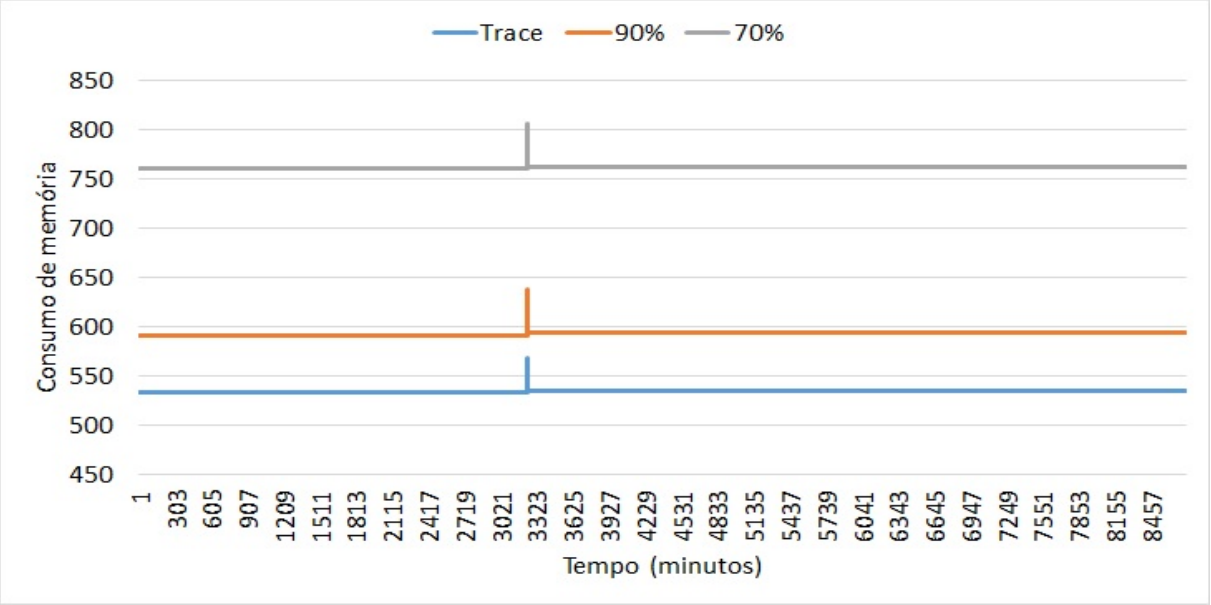

Figura 5.56: *Trace fipnfe* - Oliveira 90% e 70% com λ 1,2.

A Figura 5.57 mostra um comparativo da alocação de memória para esses valores com a *trace irWTS* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1,2.

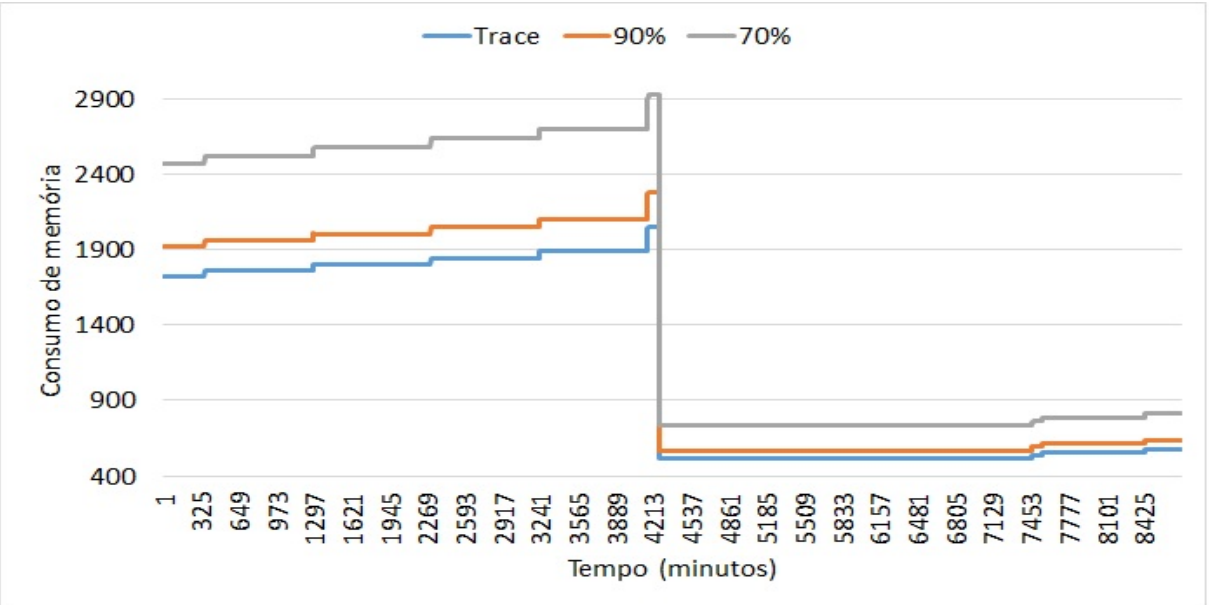

Figura 5.57: *Trace irWTS* - Oliveira 90% e 70% com λ 1,2.

A Figura 5.58 mostra um comparativo da alocação de memória para esses valores com a *trace TVL* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1,2.

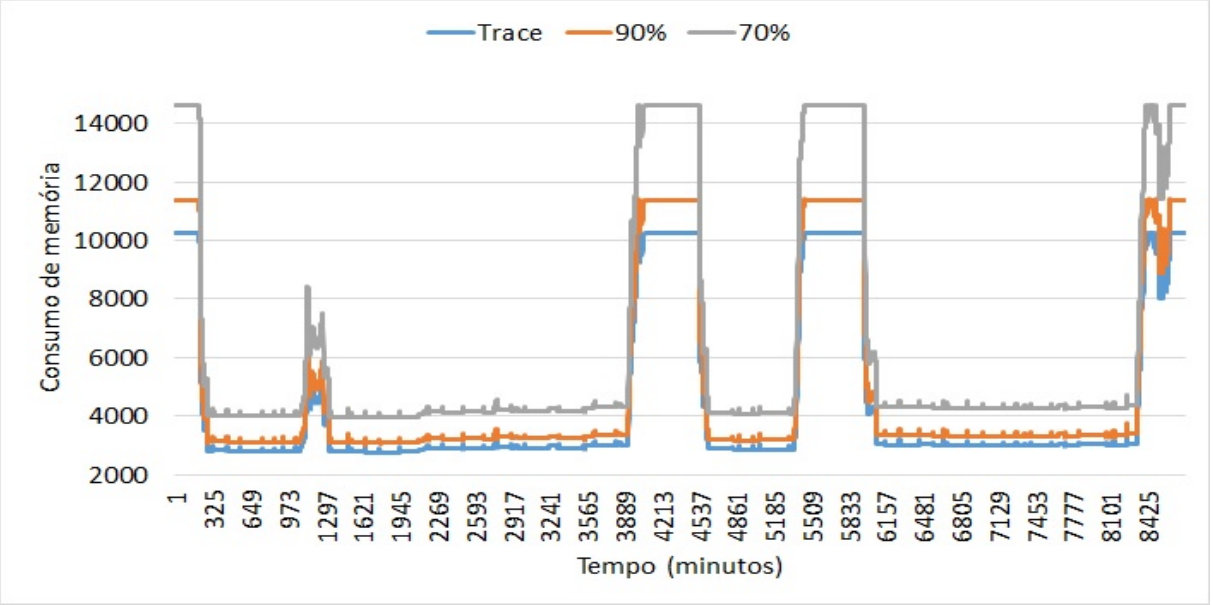

Figura 5.58: *Trace TVL* - Oliveira 90% e 70% com λ 1,2.

A Figura 5.59 mostra um comparativo da alocação de memória para esses valores com a *trace WS* considerando os valores de referência u_{ref}^{mem} de 90% e 70% e λ igual a 1,2.

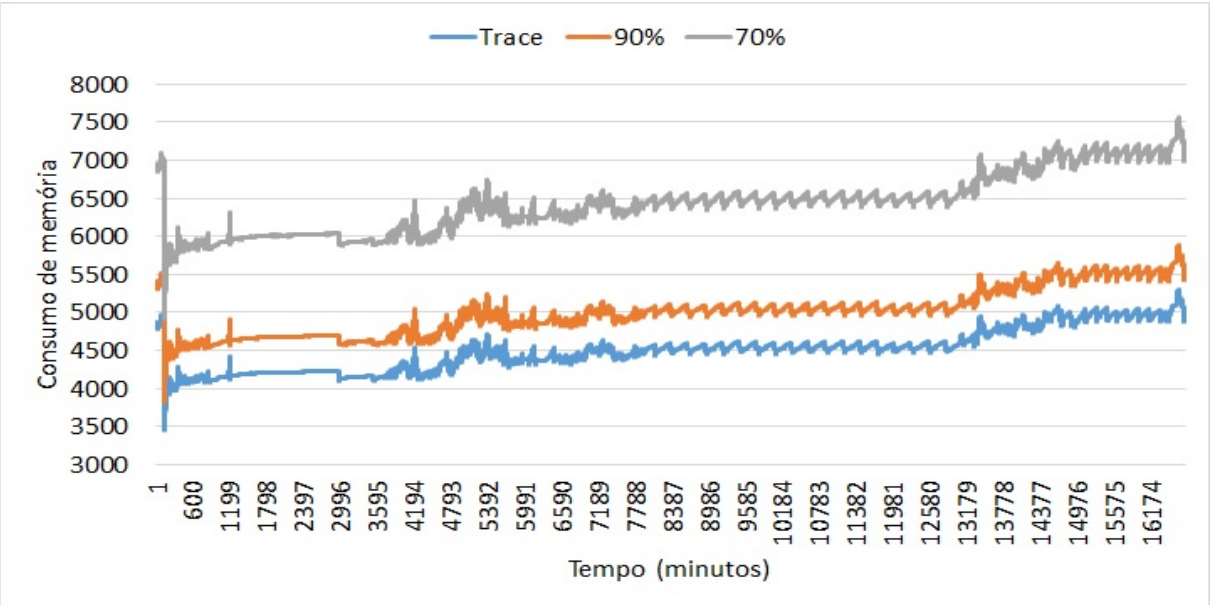

Figura 5.59: *Trace WS* - Oliveira 90% e 70% com λ 1,2.

Nesses casos, é possível notar que o valor de referência u_{ref}^{mem} de 70%, deixa uma quantidade de memória livre maior do que o u_{ref}^{mem} de 90%.

O valor de P_u, P_d são todos iguais a 0 nas *traces* acima, ou seja a precisão do método com esses valores é ótima.

A métrica $timeshareU$ e $timeshareO$ também apresentam valores iguais a 0, pois não há subprovisionamento e nem superprovisionamento.

A definição de qual desses valores são melhores em cada caso vai depender da quantidade de memória livre que o usuário ou sistema irá precisar.

5.6 Jenitha

O método descrito por Jenitha et al. 2014 [16] também foi implementado, ele propõem um método de alocação elástica de memória que utiliza MME, as previsões de alocação de memória são obtidas a partir da fórmula a seguir.

$$E(t) = \alpha \times E(t-1) + (1-\alpha) \times O(t-1) \tag{5.4}$$

Onde: E(t) é a memória estimada no tempo t

E(t - 1) é a memória estimada no tempo t-1

O(t-1) é a memória usada no tempo t-1

E alfa é obtido através da média da memória que foi utilizada no tempo t-1 pela memória estimada no tempo t-1

$$\alpha = \frac{O(t-1)}{E(t-1)} \tag{5.5}$$

Neste método foi necessário realizar um ajuste no fator $(1-\alpha)$, porque durante os testes verificamos que era necessário utilizar o valor absoluto desse termo, pois ele estava retornando resultados negativos o que sugeria a alocação de valores negativos de memória em momentos que não havia desalocação. Utilizando o valor absoluto do termo, os resultados melhoraram.

Nas *traces* sintéticas o algoritmo se comporta bem, porém no início da execução ele demora para que a previsão de alocação se aproxime do consumo real para evitar desperdício de memória.

Esse método não obteve bons resultados, pois de acordo com a métrica utilizada, nas *traces* de consumo sintético ele apresentou problemas em 5 das 6 *traces*, 3 delas deram problema de subprovisionamento e superprovisionamento, e em 2 problema de superprovisionamento.

Na *trace crescente* o algoritmo apresentou superprovisionamento de memória, a Figura 5.60 mostra a alocação realizada pelo algoritmo.

Para esta *trace*, houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 5 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 930 MB. Apresentando um valor de P_d igual a 11, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior for o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,059, ou seja durante 5,9% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

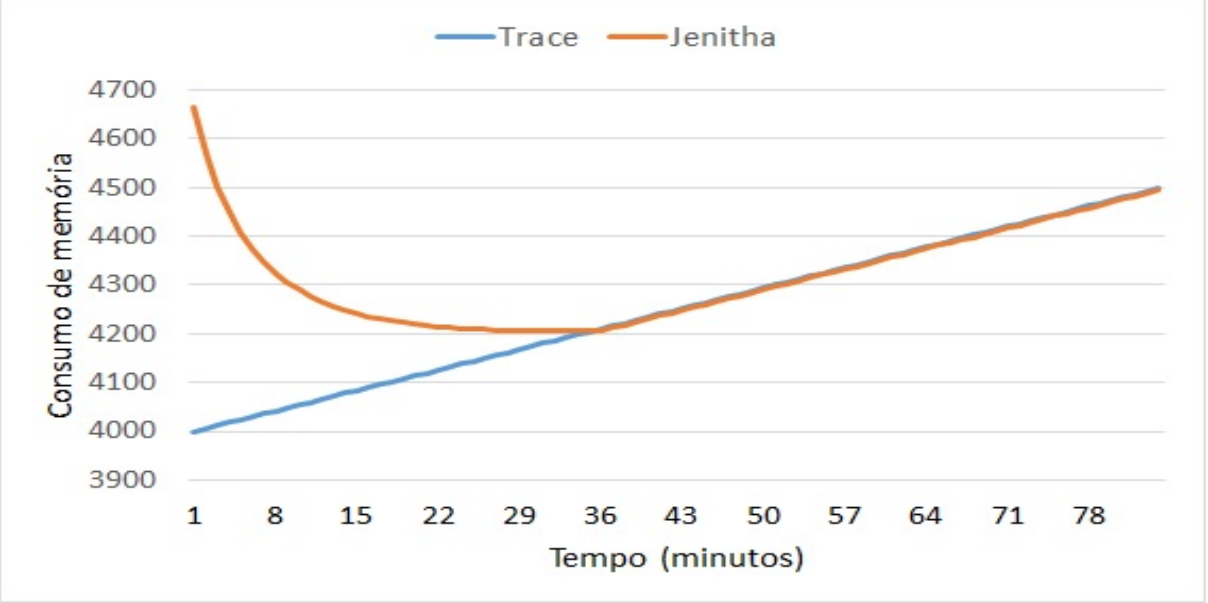

Figura 5.60: *Trace Crescente* - Jenitha.

Na *trace decrescente* o algoritmo apresentou superprovisionamento de memória, a Figura 5.61 mostra a alocação realizada pelo algoritmo.

Para esta *trace* a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 6 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 1181 MB. Apresentando um valor de P_d igual a 14, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior for o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,07, ou seja durante 7% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

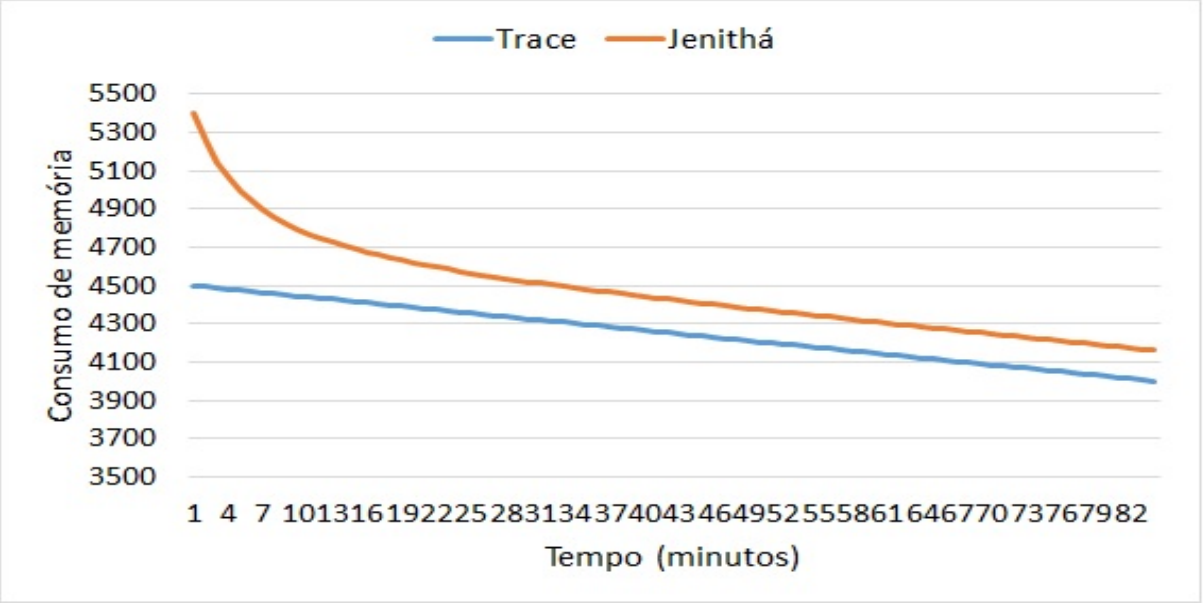

Figura 5.61: *Trace Decrescente* - Jenitha.

Na *trace const* o algoritmo apresentou superprovisionamento de memória, a Figura 5.62 mostra a alocação realizada pelo algoritmo.

Para esta *trace*, houve superprovisionamento, a avaliação feita pela métrica apresentou o seguinte resultado: O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 5 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 912 MB. Apresentando um valor de P_d igual a 12, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior for o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,07, ou seja durante 7% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

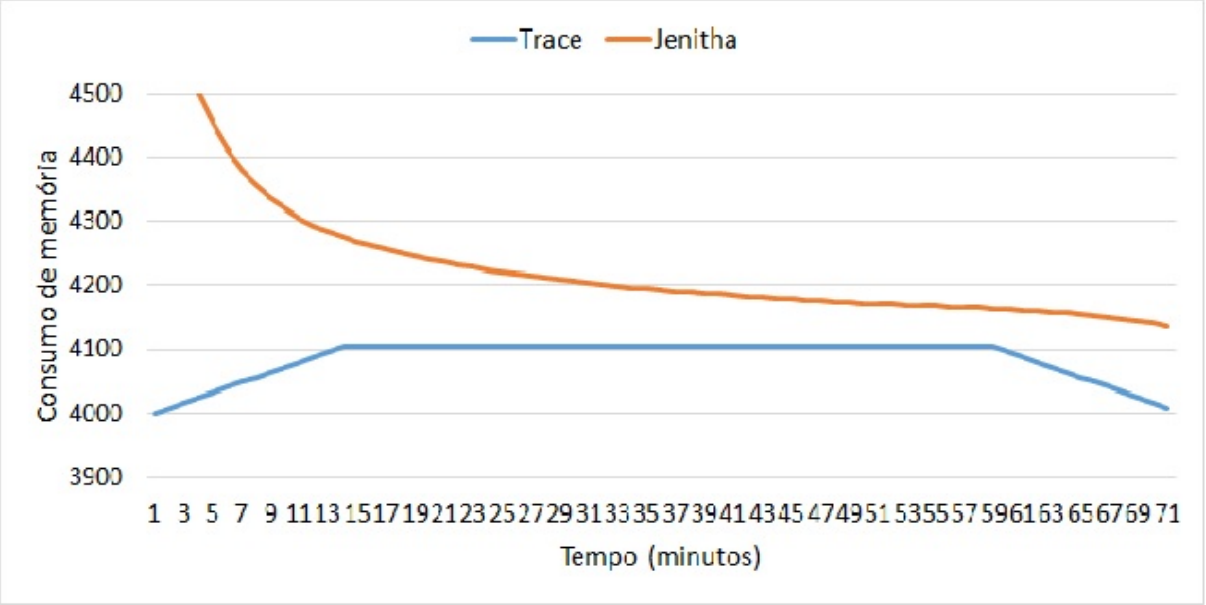

Figura 5.62: *Trace const* - Jenitha.

Na *trace picos* o algoritmo apresentou subprovisionamento e superprovisionamento de memória, a Figura 5.63 mostra a alocação realizada pelo algoritmo.

Para esta *trace* a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 23 minutos, que é o tempo total de subprovisionamento; $\sum \overline{U}$ é de 56 MB que é a quantidade total de memória subprovisionada; P_u próximo a 0 que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor próximo a 0 significa que, apesar de ter ocorrido subprovisionamento, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,09, ou seja durante 9% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 5 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 947 MB. Apresentando um valor de P_d igual a 3, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior for o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,019, ou seja durante 2% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

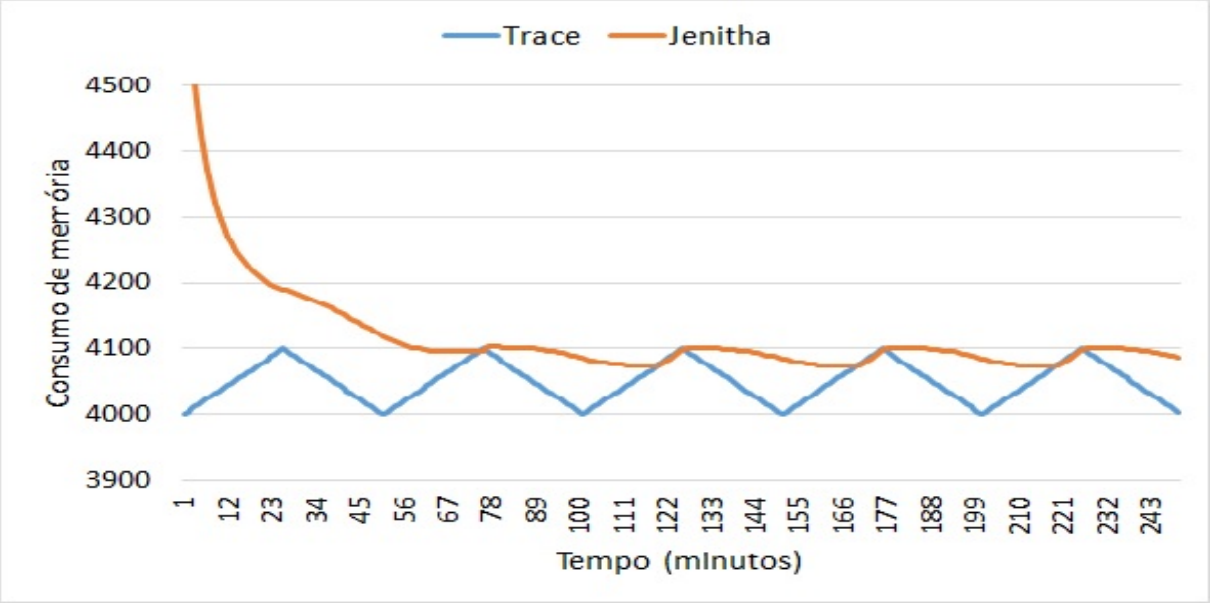

Figura 5.63: *Trace Picos* - Jenitha.

Na *trace piconst* o algoritmo apresentou subprovisionamento e superprovisionamento de memória, a Figura 5.64 mostra a alocação realizada pelo algoritmo.

Para esta *trace* a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 5 minutos, que é o tempo total de subprovisionamento; $\sum \overline{U}$ é de 62 MB que é a quantidade total de memória subprovisionada; P_u próximo a 0 que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor próximo a 0 significa que, apesar de ter ocorrido subprovisionamento, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,023, ou seja durante 2,3% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 5 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 980 MB. Apresentando um valor de P_d igual a 4, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior for o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,023, ou seja durante 2,3% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

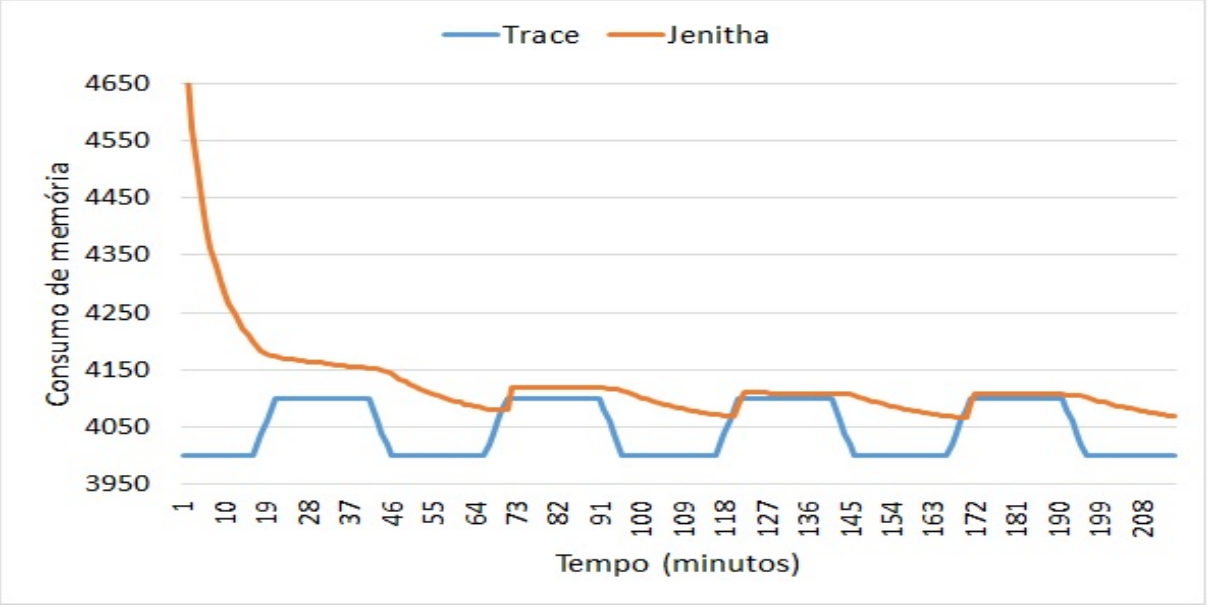

Figura 5.64: *Trace Piconst* - Jenitha.

Já nas *traces* de consumo real, em 6 das 7 *traces* o método apresentou subprovisionamento e superprovisionamento.

Na *trace fipS* o algoritmo apresentou subprovisionamento e superprovisionamento de memória, a Figura 5.65 mostra a alocação realizada pelo algoritmo.

Para esta *trace* a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 150 minutos, que é o tempo total de subprovisionamento; $\sum \overline{U}$ é de 6584 MB que é a quantidade total de memória subprovisionada; P_u igual a 1 que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor próximo a 0 significa que, apesar de ter ocorrido subprovisionamento, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,017, ou seja durante 1,7% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 133 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 11693 MB. Apresentando um valor de P_d próximo de 0, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado bom, pois quanto mais próximo de 0 for o valor de P_d melhor é o desempenho do método.

A métrica $timeshareO$ obteve 0,015, ou seja durante 1,5% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

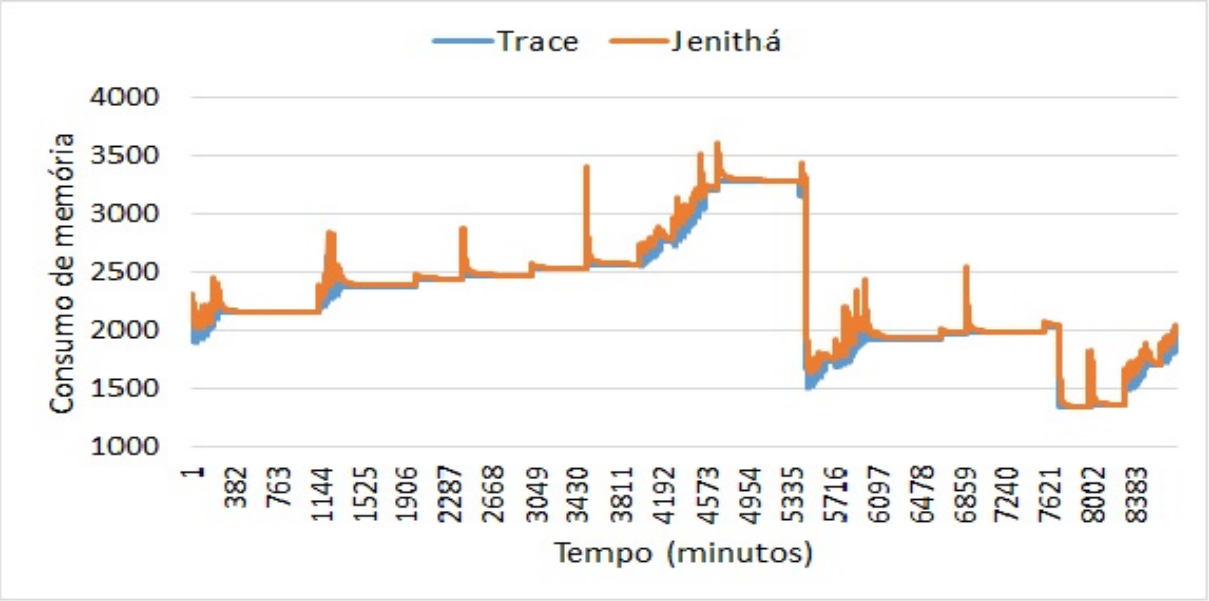

Figura 5.65: *Trace fipS* - Jenitha.

Na *trace dalDC1* o algoritmo apresentou subprovisionamento e superprovisionamento de memória, a Figura 5.66 mostra a alocação realizada pelo algoritmo.

Para esta *trace* a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 31 minutos, que é o tempo total de subprovisionamento; $\sum \overline{U}$ é de 1936 MB que é a quantidade total de memória subprovisionada; P_u próximo de 0, que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor próximo a 0 significa que, apesar de ter ocorrido subprovisionamento, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,00035, ou seja durante 0,35% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 82 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 5678 MB. Apresentando um valor de P_d próximo de 0, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado bom, pois quanto mais próximo de 0 for o valor de P_d melhor é o desempenho do método.

A métrica $timeshareO$ obteve 0,00093, ou seja durante 0,93% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

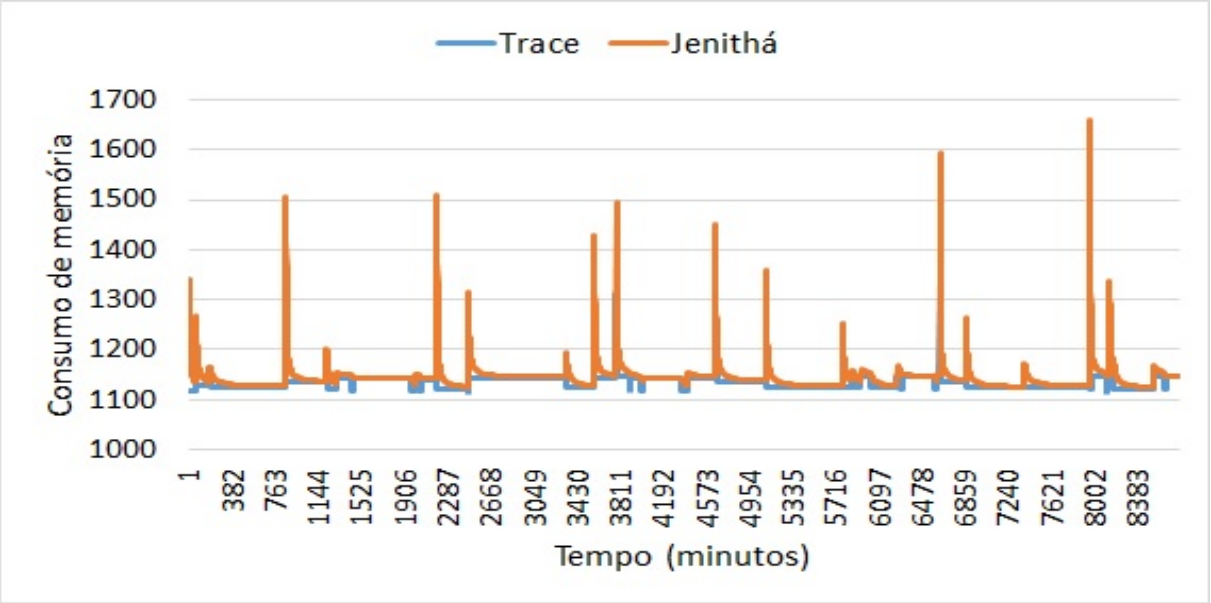

Figura 5.66: *Trace dalDC1* - Jenitha.

Na *trace fipenfe* o algoritmo apresentou subprovisionamento e superprovisionamento de memória, a Figura 5.67 mostra a alocação realizada pelo algoritmo.

Para esta *trace* a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 1 minuto, que é o tempo total de subprovisionamento; $\sum \overline{U}$ é de 36 MB que é a quantidade total de memória subprovisionada; P_u próximo de 0, que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor próximo a 0 significa que, apesar de ter ocorrido subprovisionamento, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,0001, ou seja durante 0,01% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 8 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 162 MB. Apresentando um valor de P_d próximo de 0, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado bom, pois quanto mais próximo de 0 for o valor de P_d melhor é o desempenho do método.

A métrica $timeshareO$ obteve 0,00091, ou seja durante 0,09% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

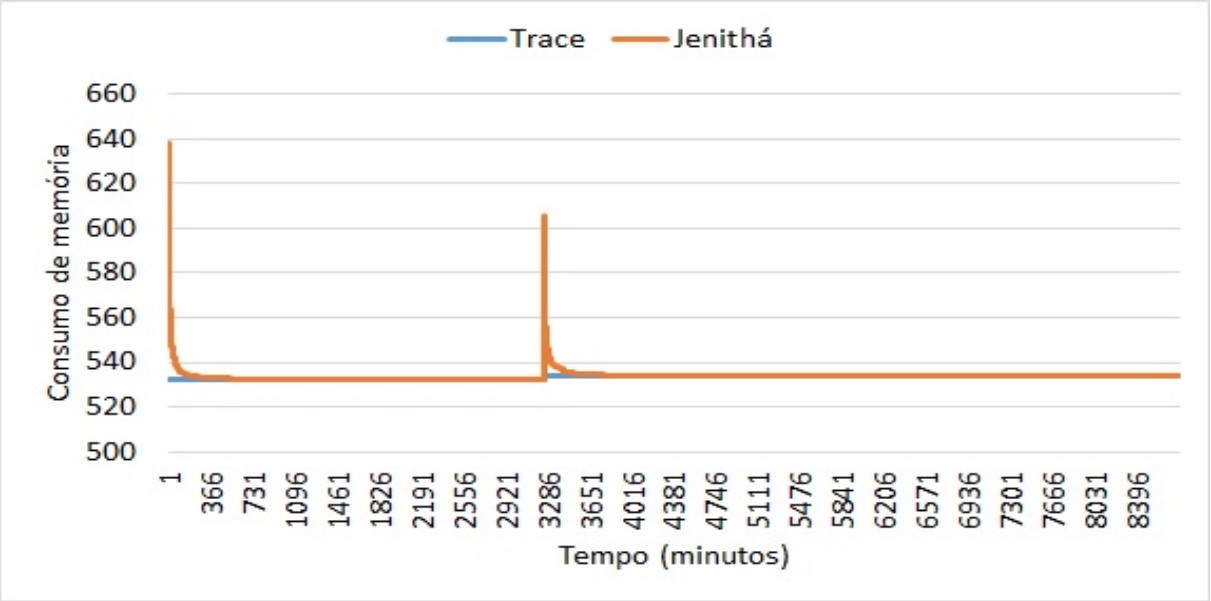

Figura 5.67: *Trace fipnfe* - Jenitha.

Na *trace fipapp* o algoritmo apresentou subprovisionamento e superprovisionamento de memória, a Figura 5.68 mostra a alocação realizada pelo algoritmo.

Para esta *trace* a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 119 minutos, que é o tempo total de subprovisionamento; $\sum \overline{U}$ é de 9050 MB que é a quantidade total de memória subprovisionada; P_u igual a 1, que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor próximo a 0 significa que, apesar de ter ocorrido subprovisionamento, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,013, ou seja durante 1,36% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 93 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 28439 MB. Apresentando um valor de P_d igual a 3, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior for o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,01, ou seja durante 1% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

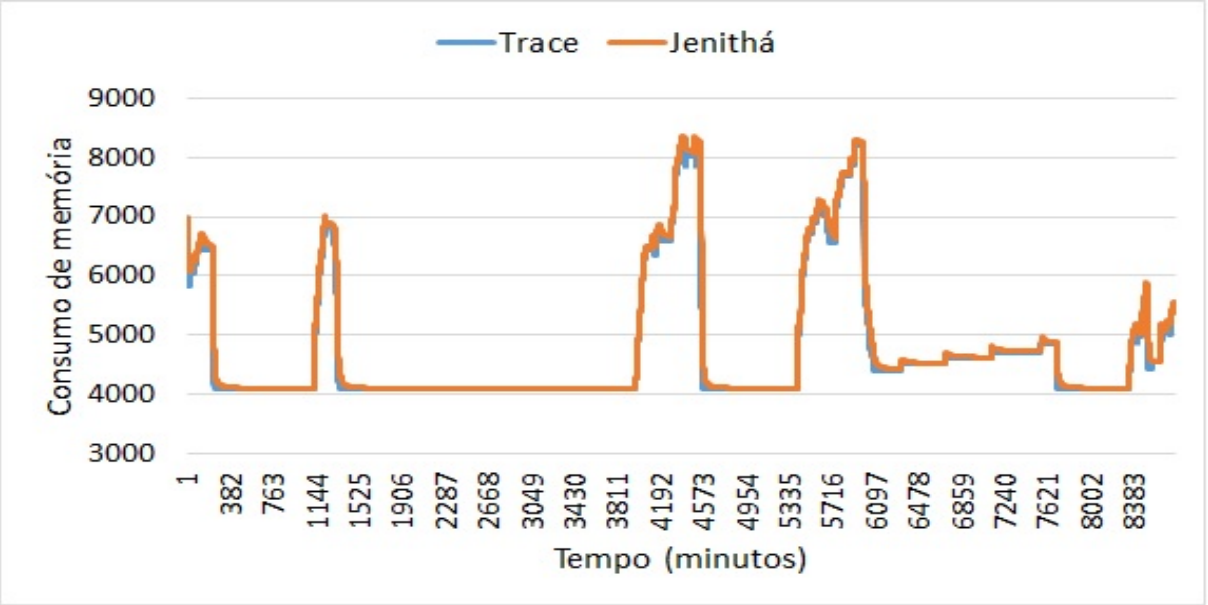

Figura 5.68: *Trace fipapp* - Jenitha.

Na *trace irWTS* o algoritmo apresentou subprovisionamento e superprovisionamento de memória, a Figura 5.69 mostra a alocação realizada pelo algoritmo.

Para esta *trace* a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 8 minutos, que é o tempo total de subprovisionamento; $\sum \overline{U}$ é de 298 MB que é a quantidade total de memória subprovisionada; P_u próximo de 0, que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor próximo a 0 significa que, apesar de ter ocorrido subprovisionamento, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,00009, ou seja durante 0,09% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 15 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 2694 MB. Apresentando um valor de P_d próximo de 0, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado bom, pois quanto mais próximo de 0 for o valor de P_d melhor é o desempenho do método.

A métrica $timeshareO$ obteve 0,00017, ou seja durante 0,17% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

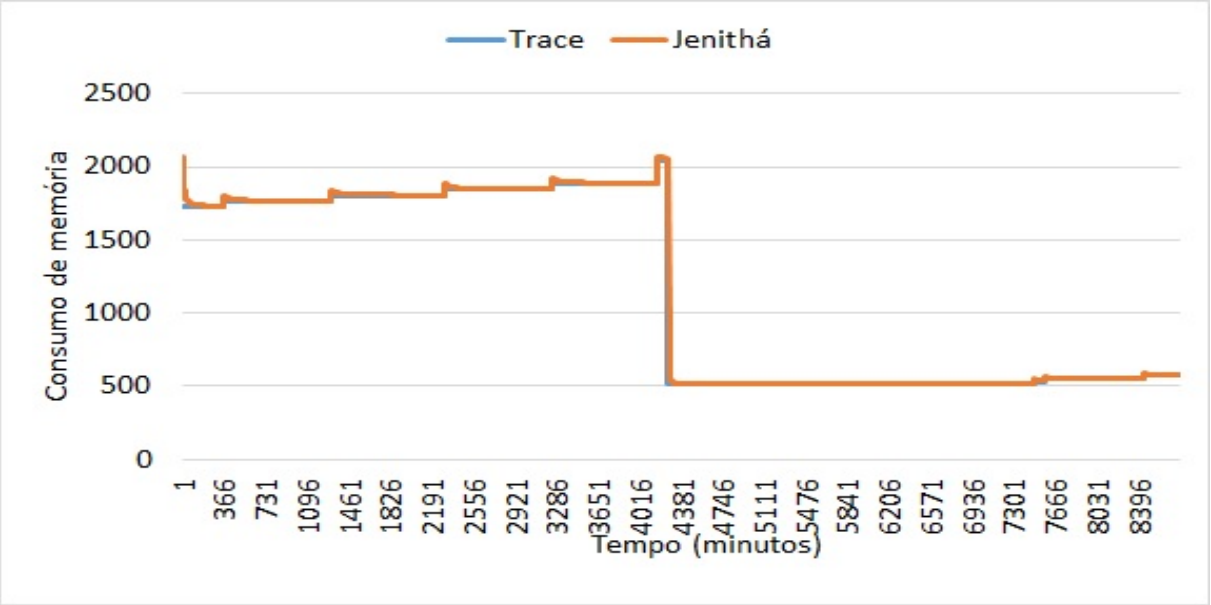

Figura 5.69: *Trace irWTS* - Jenitha.

Na *trace TVL* o algoritmo apresentou subprovisionamento e superprovisionamento de memória, a Figura 5.70 mostra a alocação realizada pelo algoritmo.

Para esta *trace* a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 163 minutos, que é o tempo total de subprovisionamento; $\sum \overline{U}$ é de 19949 MB que é a quantidade total de memória subprovisionada; P_u igual a 2, que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor maior do que 0 significa que, o método não obteve um resultado bom, pois quanto maior for o valor de P_u, pior é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,018, ou seja durante 1,8% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 177 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 65255 MB. Apresentando um valor de P_d igual a 7, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado ruim, pois quanto maior for o valor de P_d pior é o desempenho do método.

A métrica $timeshareO$ obteve 0,02, ou seja durante 2% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

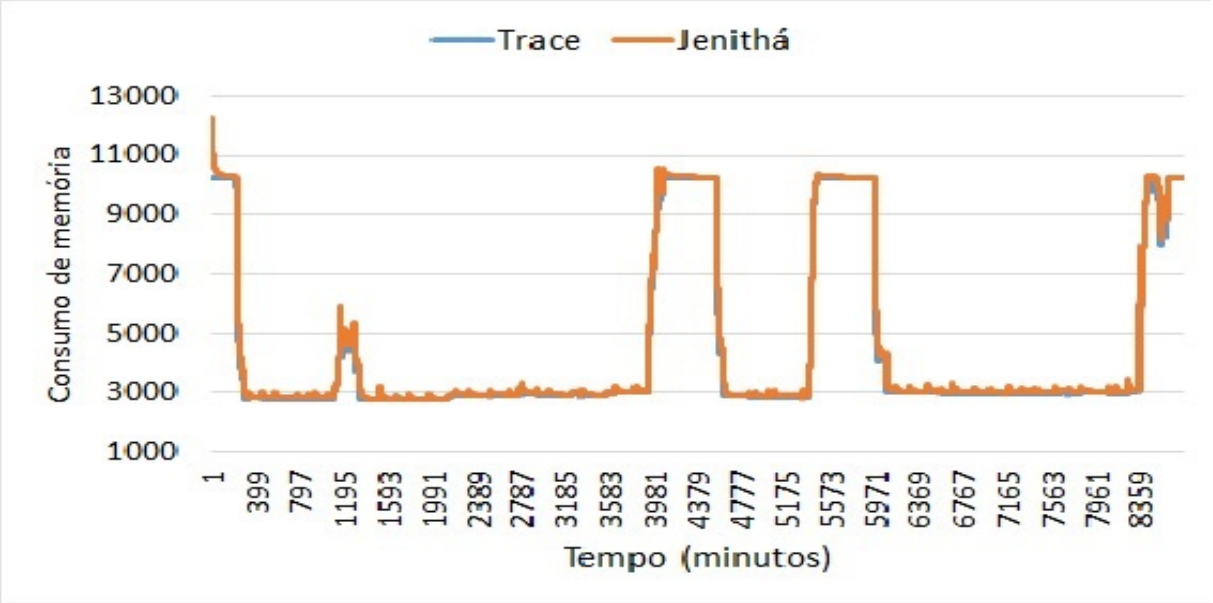

Figura 5.70: *Trace TVL* - Jenitha.

Na *trace WS* o algoritmo apresentou subprovisionamento e superprovisionamento de memória, a Figura 5.71 mostra a alocação realizada pelo algoritmo.

Para esta *trace* a avaliação feita pela métrica apresentou o seguinte resultado:$\sum \overline{A}$ 1760 minutos, que é o tempo total de subprovisionamento; $\sum \overline{U}$ é de 5597 MB que é a quantidade total de memória subprovisionada; P_u próximo de 0, que mede a precisão do método para alocar mais memória quando encontra-se em um estado de subprovisionamento, esse valor maior do que 0 significa que, o método não obteve um resultado ruim, pois quanto mais próximo de 0 for o valor de P_u, melhor é a precisão do método para alocar memória.

A métrica $timeshareU$ obteve 0,105, ou seja durante 10,5% do tempo de execução da *trace* ocorreu subprovisionamento de memória.

O total de tempo de superprovisionamento $\sum \overline{B}$ foi de 17 minutos, e a quantidade de memória superprovisionada $\sum \overline{O}$ foi de 4162 MB. Apresentando um valor de P_d próximo de 0, que é a métrica que calcula a precisão de liberação de memória do método, que nesse caso obteve um resultado bom, pois quanto mais próximo de 0 for o valor de P_d melhor é o desempenho do método.

A métrica $timeshareO$ obteve 0,0001, ou seja durante 0,1% do tempo de execução da *trace* ocorreu superprovisionamento de memória.

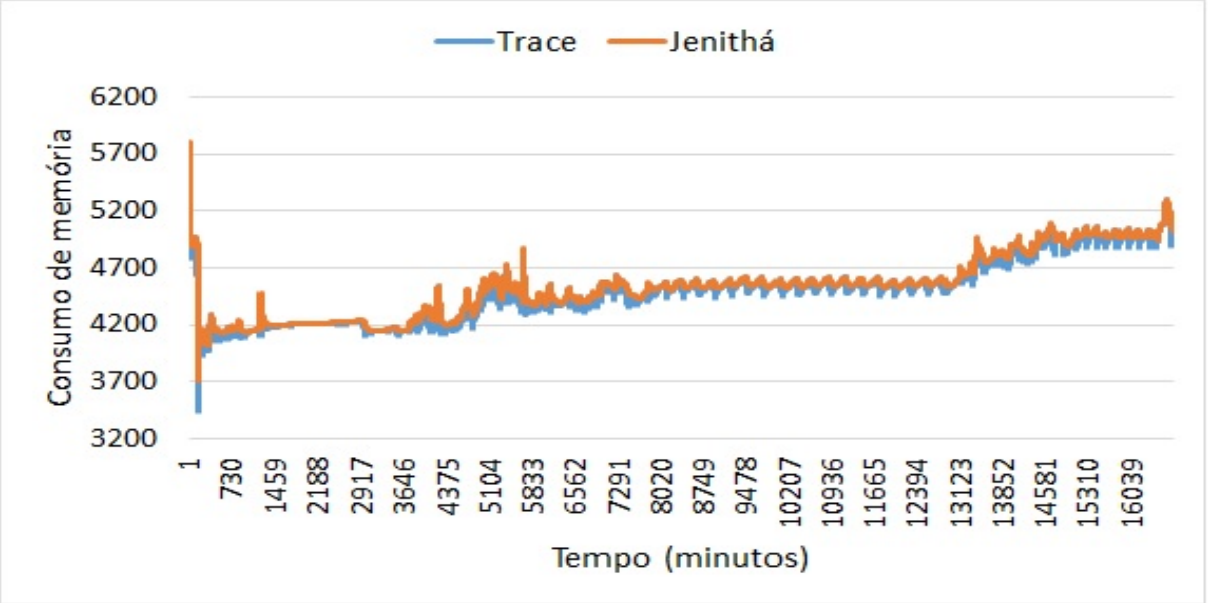

Figura 5.71: *Trace WS* - Jenitha.

5.7 Baruchi

Foi implementado também o método apresentado no trabalho Baruchi et al. 2010 [9], o qual consiste em um mecanismo de alocação de memória dinâmica em MV's utilizando o método média móvel exponencial (MME) que define quando a memória deve ser alocada e quando deve ser removida. Nessa técnica, as amostras de utilização de memória mais recentes têm maior peso que as amostras mais antigas.

São utilizados dois períodos de amostragem que tem por objetivo melhorar a identificação de tendências de alta e baixa demanda de memória e suavizar momentos de picos de utilização. O cálculo da MME pode ser observado na seguinte equação:

$$MME_{atual} = MME_{atual-1} + \alpha \times (MEM_{atual} - MME_{atual-1}) \tag{5.6}$$

onde:

$$\alpha = 2 \div (QtdeAmostras + 1); \tag{5.7}$$

e

$$MEM_{atual} = QuantidadeDeMemoriaUsadaNaUltimaMedida \tag{5.8}$$

No trabalho de Baruchi são utilizados dois períodos de amostragem: 5 segundos e 25 segundos. A memória é alocada quando a amostra de 5 segundos cruza o gráfico com a amostra de 25 segundos no sentido crescente, e é liberada quando a amostra de 5 segundos cruza com a de 25 segundos no sentido decrescente, quando acontece o cruzamento dessas duas médias a memória é aumentada ou diminuída em 10%.

Neste trabalho utilizamos também dois períodos de amostragem, 1 minuto e 5 minutos, foram escolhidos esses períodos pelo fato de a coleta das MV's reais serem feitas a cada minuto.

Foram realizados testes com todas as 12 *traces*, porém o comportamento do algoritmo não foi satisfatório, o fato de o método proposto sempre alocar ou desalocar 10% de memória cada vez que o requisito de cruzamento das médias fosse cumprido, fez com que as alocações e desalocações tivessem valores altos/baixos demais.

Foram realizados testes alocando e desalocando uma quantidade de memória menor do que 10%, com valores de 5%, porém os resultados continuaram insatisfatórios, o algoritmo alocava e desalocava memória onde não havia necessidade. O artigo original não mostrava detalhes da implementação e nem os resultados para a comprovação do método.

5.8 Comparação dos metódos

A Tabela 5.1 abaixo apresenta um comparativo dos métodos apresentados e testados, o símbolo $\surd$ significa que para aquele método com aqueles parâmetros não houve subprovisionamento de memória e nem superprovisionamento de memória; o símbolo *P* significa que o método com aqueles parâmetros apresentou apenas superprovionamento de memória; o símbolo $\times$ significa que o método apresentou subprovisionamento de memória e superprovisionamento de memória.

Trace	**Oliveira 70%** $lambda$ **0,8**	**Oliveira 70%** $lambda$ **1**	**Oliveira 70%** $lambda$ **1,2**	**Oliveira 90%** $lambda$ **0,8**	**Oliveira 90%** $lambda$ **1**	**Oliveira 90%** $lambda$ **1,2**	**Moltó 10%**	**Moltó 30%**	**Jenitha**	**Baruchi**
crescente	√	√	√	√	√	√	*P*	√	*P*	×
decrescente	√	√	√	√	√	√	√	√	*P*	×
const	√	√	√	√	√	√	*P*	√	*P*	×
picos	√	√	√	√	√	√	*P*	√	×	×
piconst	√	√	√	√	√	√	*P*	√	×	×
fipS	√	√	√	*P*	√	√	×	√	×	×
fipapp	√	√	√	*P*	√	√	*P*	√	×	×
fipnfe	√	√	√	*P*	√	√	*P*	√	×	×
irWTS	√	√	√	*P*	√	√	*P*	√	×	×
dalDC1	√	√	√	*P*	√	√	×	*P*	×	×
TVL	√	√	√	*P*	√	√	×	*P*	×	×
WS	√	√	√	*P*	√	√	*P*	√	×	×

Tabela 5.1: Comparação dos métodos de alocação de memória.

Capítulo 6

Conclusão

Este trabalho teve como objetivo a implementação e avaliação de técnicas reativas de alocação elástica de memória, com o intuito de avaliar a aplicação desses métodos em diferentes cenários, por meio de simulações e experimentos práticos com *traces* sintéticas e reais de consumo de memória.

Foi realizada uma revisão bibliográfica de métodos reativos e proativos. Nos métodos reativos os recursos podem ser adicionados e removidos do ambiente se determinada condição ocorrer. Já nos métodos proativos são utilizadas técnicas analítico-matemáticas e heurísticas, para obter uma previsão sobre o comportamento das cargas do sistema e, a partir desses resultados, tomar decisões de como e quando escalar os recursos para alocação de memória e demais recursos. Dos métodos reativos apresentados foram escolhidos 4 algoritmos para implementação e testes. Foram utilizadas 12 *traces* de consumo de memória, dentre as quais, 5 eram sintéticas com consumo bem específico; e 7 de consumo real, 1 de uma estação de trabalho e 6 de consumo de MV's reais, cedidas pela empresa Constel Tecnologia.

De acordo com o relatório da Comissão Europeia [14] sobre o futuro da computação em nuvem, a elasticidade vertical é uma das áreas pouco exploradas, embora a sua importância no fornecimento de um serviço de nuvem eficiente para qualquer aplicação seja reconhecida. As empresas, como é o caso a Constel Tecnologia, enfrentam um problema para o gerenciamento dinâmico de memória. Sem esse gerenciamento dinâmico, a maioria da MV's possuem o fornecimento de memória de forma estática, o que causa um desperdício de recursos para a empresa, que não pode alocar a memória reservada, mesmo esta estando livre, para outro cliente, e um gasto desnecessário para os clientes que pagam pelo máximo de memória que eles utilizariam mesmo fora dos momentos de picos.

Esse trabalho e os testes realizados com as *traces* cedidas pela empresa, mostram que existe a possibilidade de implementação desses métodos em cenários reais, e que com a possibilidade de resultados satisfatórios, mas um dos desafios seriam a utilização desses métodos com aplicações que não suportam a elasticidade vertical, como por exemplo o SGBD's.

Os testes descritos e as avaliações apresentadas no Capítulo 5 mostram que, os algoritmos de alocação elástica de memória que obtiveram melhores resultados, foram os apresentados por Oliveira et al. 2015 [2]/Dawoud et al. 2011 [17]/ Heo et al. 2009 [22], e o método de Moltó et al. 2011 [12]. Esses dois algoritmos tiveram resultados satisfatórios tanto nas *traces* sintéticas quanto nas *traces* reais, cumprindo com o SLA ou qualquer exigência do usuário. O método apresentado por Oliveira obteve melhores resultados com valor de referência u_{ref}^{mem} de 70% e a constante de incremento lambda λ igual a 1, 1,2 e 0,8; e de u_{ref}^{mem} de 90% com λ igual a 1 e λ igual a 1,2, deixando no mínimo, respectivamente, 30% e 10% de memória livre para a MV.

O método de Moltó apresentou melhores resultados com valor de MOP de 30%, que para os cenários apresentados tem um menor desperdício de memória. Já o algoritmo apresentado por Jenitha et al. 2014 [16], apresentou resultado insatisfatório na maioria das *traces* de consumo sintéticas e reais, apresentando bom desempenho apenas nas *traces* com comportamento decrescente e na *trace* de consumo constante. Já o algoritmo apresentado no trabalho de Baruchi et al. 2010 [9], os resultados descritos no artigo não foram alcançados apesar de todas as modificações realizadas na tentativa de melhorar o algoritmo.

Trabalhos futuros incluem a implementação e teste de algoritmos preditivos de alocação dinâmica de memória, análise de testes de algoritmos reativos por diferentes métricas para avaliação da elasticidade, análise de outras *traces*, implementação prática (sem simulação) dos algoritmos e análise real dos resultados.

Referências Bibliográficas

[1] GALANTE, G. *Explorando a Elasticidade em Nível de Programação no Desenvolvimento e na Execução de Aplicações CientÍficas*. Tese (Doutorado) — Universidade Federal do Paraná, 2014.

[2] OLIVEIRA, I. C. d. et al. Aprimorando a elasticidade de aplicações de banco de dados utilizando virtualização em nível de sistema operacional. Pontifícia Universidade Católica do Rio Grande do Sul, 2015.

[3] HERBST, N. et al. Ready for rain? a view from spec research on the future of cloud metrics. *arXiv preprint arXiv:1604.03470*, 2016.

[4] COUTINHO, E. et al. Elasticidade em computaçao na nuvem: Uma abordagem sistemática. *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2013)-Minicursos*, 2013.

[5] MELL, P. M.; GRANCE, T. The nist definition of cloud computing. National Institute of Standards & Technology, Special Publication 800-145, n. 800-145, 2011.

[6] SCHULZE, B. et al. Computação em nuvem. 2011.

[7] SANTOS, U. et al. Adoção de computação em nuvem e suas motivações. *CSABR – Cloud Security Alliance – Brazil*, 2012.

[8] MBAYI, D. M. Classificação dos métodos de tomada de decisão para gerenciamento autônomo da infraestrutura de computação em nuvem. Universidade Federal de Santa Catarina, 2015.

[9] BARUCHI, A.; MIDORIKAWA, E. T. Elasticidade de memória em máquinas virtuais utilizando média móvel exponencial. In: *VII Workshop de Sistemas Operacionais (WSO)*. [S.l.: s.n.], 2010.

[10] GALANTE, G.; BONA, L. C. E. D. Explorando a elasticidade em nível de programação. XV Simpósio em Sistemas Computacionais de Alto Desempenho, WSCAD, 2014.

[11] RIGHI, R. d. R. Elasticidade em cloud computing: conceito, estado da arte e novos desafios. *Revista Brasileira de Computação Aplicada*, v. 5, n. 2, p. 2–17, 2013.

[12] MOLTÓ, G. et al. Elastic memory management of virtualized infrastructures for applications with dynamic memory requirements. *Procedia Computer Science*, Elsevier, v. 18, p. 159–168, 2013.

[13] VERGARA, G. F.; MOURA, B.; ARAÁUJO, A. Elasticidade em uma infraestrutura de nuvens federadas bionimbuz. *Universidade de Brasília-UnB*, 2014.

[14] DAWOUD, W.; TAKOUNA, I.; MEINEL, C. Dynamic scalability and contention prediction in public infrastructure using internet application profiling. In: IEEE. *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. [S.l.], 2012. p. 208–216.

[15] BAUN, C. et al. *Cloud Computing: Web-based dynamic IT services*. [S.l.]: Springer Science & Business Media, 2011.

[16] JENITHA, V. H. A.; VEERAMANI, R. Dynamic memory allocation using ballooning and virtualization in cloud computing. *IOSR Journals (IOSR Journal of Computer Engineering)*, v. 1, n. 16, p. 19–23.

[17] DAWOUD, W.; TAKOUNA, I.; MEINEL, C. Elastic vm for cloud resources provisioning optimization. In: SPRINGER. *International Conference on Advances in Computing and Communications*. [S.l.], 2011. p. 431–445.

[18] COUTINHO, E. F.; GOMES, D. G.; SOUZA, J. N. de. Uma proposta de arquitetura autonômica para elasticidade em computação em nuvem. *Anais do 4º Workshop de Sistemas Distribuídos Autonômicos - WoSiDA*, 2014.

[19] PAIXÃO, C. A. *Gerenciamento de recursos computacionais em plataformas de Computação em Nuvem*. Tese (Doutorado) — Universidade de São Paulo, 2014.

[20] GONG, Z.; GU, X.; WILKES, J. Press: Predictive elastic resource scaling for cloud systems. In: IEEE. *2010 International Conference on Network and Service Management*. [S.l.], 2010. p. 9–16.

[21] HERBST, N. R.; KOUNEV, S.; REUSSNER, R. H. Elasticity in cloud computing: What it is, and what it is not. In: *ICAC*. [S.l.: s.n.], 2013. p. 23–27.

[22] HEO, J. et al. Memory overbooking and dynamic control of xen virtual machines in consolidated environments. In: IEEE. *2009 IFIP/IEEE International Symposium on Integrated Network Management*. [S.l.], 2009. p. 630–637.

[23] CARVALHO, H. E. T. *VOLTAIC: Um Sistema de Gerência Automatizada de Recursos de Nós Virtuais Para Computaçao em Nuvens*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2012.

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

[25] ARMBRUST, M. et al. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[26] VAQUERO, L. M. et al. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, ACM, v. 39, n. 1, p. 50–55, 2008.

[27] SOUSA, F. R.; MOREIRA, L. O.; MACHADO, J. C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. *II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)*, p. 150–175, 2009.

[28] SOBRAGI, C. G. Adoção de computação em nuvem: estudo de casos múltiplos. Universidade Federal do Rio Grande do Sul, 2012.

[29] BOLIN, J. S. *Use case analysis for adopting cloud computing in Army test and evaluation*. Tese (Doutorado) — Monterey, California. Naval Postgraduate School, 2010.

[30] MATHER, T.; KUMARASWAMY, S.; LATIF, S. *Cloud security and privacy: an enterprise perspective on risks and compliance*. [S.l.]: "O'Reilly Media, Inc.", 2009.

[31] POSSOBOM, C. C. Estudo de caso: Cloud computing-computação em nuvem. 2012.

[32] MARSTON, S. et al. Cloud computing—the business perspective. *Decision support systems*, Elsevier, v. 51, n. 1, p. 176–189, 2011.

[33] BADGER, L. et al. Cloud computing synopsis and recommendations (draft), nist special publication 800-146. *Recommendations of the National Institute of Standards and Technology, Tech. Rep*, 2011.

[34] RUCK, D. B.; OLIVEIRA, R. de; KOSLOVSKI, G. P. Comparação de algoritmos para alocação de infraestruturas virtuais. *Revista Brasileira de Computação Aplicada*, v. 6, n. 2, p. 98–112, 2014.

[35] CANAL Comstor - Blog de Negócios de TI. Consultado na INTERNET: http://blogbrasil.comstor.com/bid/282879/Cloud-Computing-Nuvens-H-bridas-Privadas-e-P-blicas, 2016.

[36] IBM developersWorks. Consultado na INTERNET: http://www.ibm.com/developerworks/br/websphere/techjournal/1206_dejesus/1206_dejesus.html.

[37] FELIPE, D. S. S. et al. Cloud computing: definições e análise para além das vantagens. In: *V Congresso Tecnológico de TI e Telecom InfoBrasil*. [S.l.: s.n.], 2012.

[38] RIGHI, R. R.; GALANTE, G. Explorando a elasticidade em nuvem para a computação de aplicações paralelas. XVI Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, ERAD, 2016.

[39] GALANTE, G.; BONA, L. C. E. de. A survey on cloud computing elasticity. In: IEEE. *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*. [S.l.], 2012. p. 263–270.

[40] MOORE, L. R.; BEAN, K.; ELLAHI, T. A coordinated reactive and predictive approach to cloud elasticity. Citeseer, 2013.

[41] SAWAMURA, R.; BOERES, C.; REBELLO, V. E. Evaluating the impact of memory allocation and swap for vertical memory elasticity in vms. In: IEEE. *Computer Architecture and High Performance Computing (SBAC-PAD), 2015 27th International Symposium on*. [S.l.], 2015. p. 186–193.

[42] ALVES, L. C. D. *O impacto da virtualização no desempenho de aplicações distribuídas baseadas em SOA e a sua influência nos modelos de desempenho*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, 2013.

[43] LIU, H. et al. Hotplug or ballooning: A comparative study on dynamic memory management techniques for virtual machines. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 26, n. 5, p. 1350–1363, 2015.

[44] DENNING, P. J. Working sets past and present. *IEEE Transactions on Software Engineering*, IEEE, n. 1, p. 64–84, 1980.

[45] FAROKHI, S. et al. A hybrid cloud controller for vertical memory elasticity: A control-theoretic approach. *Future Generation Computer Systems*, Elsevier, 2016.

[46] SPINNER, S. et al. Proactive memory scaling of virtualized applications. In: IEEE. *2015 IEEE 8th International Conference on Cloud Computing*. [S.l.], 2015. p. 277–284.

[47] HINES, M. R. et al. Applications know best: Performance-driven memory overcommit with ginkgo. In: IEEE. *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. [S.l.], 2011. p. 130–137.

[48] SALOMIE, T.-I. et al. Application level ballooning for efficient server consolidation. In: ACM. *Proceedings of the 8th ACM European Conference on Computer Systems*. [S.l.], 2013. p. 337–350.

[49] TAN, Y. et al. Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In: IEEE. *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. [S.l.], 2012. p. 285–294.

[50] COUTINHO, E. F. *FOLE: Um framework conceitual para avaliação de desempenho da elasticidade em ambientes de computação em nuvem*. Tese (Doutorado) — Universidade Federal do Ceará, 2014.

[51] ISLAM, S. et al. How a consumer can measure elasticity for cloud platforms. In: ACM. *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. [S.l.], 2012. p. 85–96.