Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Provisionamento vertical de recursos em ambientes de nuvens

Armstrong Mardilson da Silva Goes

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas Distribuídos

Andrey Elísio Monteiro Brito
(Orientador)

Campina Grande, Paraíba, Brasil

G598p Goes, Armstrong Mardilson da Silva.
Provisionamento vertical de recursos em ambientes de nuvens / Armstrong Mardilson da Silva Goes. – Campina Grande, 2018.
80 f. : il. color.

Dissertação (Mestrado em Ciências da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2018.
"Orientação: Prof. Dr. Andrey Elísio Monteiro Brito".
Referências.

1. Sistemas de Processamento Distribuído. 2. Computação nas Nuvens. 3. Provisionamento Vertical – Ambiente de Nuvens. I. Brito, Andrey Elísio Monteiro. II. Título.

CDU 004.75(043)

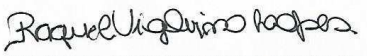
FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA SEVERINA SUELI DA SILVA OLIVEIRA CRB-15/225

"PROVISIONAMENTO VERTICAL DE RECURSOS EM AMBIENTES DE NUVENS"

ARMSTRONG MARDILSON DA SILVA GOES

DISSERTAÇÃO APROVADA EM 20/08/2018

ANDREY ELÍSIO MONTEIRO BRITO, Dr., UFCG
Orientador(a)



RAQUEL VIGOLVINO LOPES, Dra., UFCG
Examinador(a)

PRISCILA AMÉRICA SOLÍS MENDEZ BARRETO, Dra., UnB
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Na última década observou-se o rápido crescimento na utilização de infraestruturas virtualizadas para implantação de aplicações. Buscando flexibilidade de alocação de recursos e redução de custos de gerência, diversas organizações optaram por adotar nuvens computacionais como ambientes onde partes vitais de suas atividades são executadas. No contexto de aplicações executando em nuvens, observa-se o problema da garantia de qualidade de serviço. Aplicações normalmente apresentam uma demanda por recursos variável no tempo. No caso de aplicações interativas, como *sites* de *e-commerce*, por exemplo, picos de número de requisições são comuns e é necessário adicionar recursos com o intuito de garantir qualidade de serviço na forma de um tempo de resposta tolerável. No contexto de aplicações *batch*, em geral o interesse é garantir um prazo de execução, relacionado com a prioridade da aplicação. Estimar a quantidade de recursos necessários para garantir término dentro do prazo é complexo e sujeito a erros na forma de sub ou superalocações. Uma solução complementar é alterar a quantidade de recursos alocados caso o progresso da aplicação não seja satisfatório. Considerando este último raciocínio, este trabalho propõe uma solução de provisionamento baseada em controladores PID e provisionamento vertical de recursos. Com o intuito de avaliar a solução de provisionamento proposta no contexto de aplicações *batch*, foram executados dois *microbenchmarks* e uma aplicação utilizada no contexto de qualidade de dados em *Big Data*, em um ambiente de nuvem realista, com recursos provisionados por uma implementação da solução proposta. Foram coletados o tempo de execução e o uso de recursos dos *microbenchmarks* e da aplicação. Resultados mostraram que a solução de provisionamento é viável e eficaz quanto a garantir qualidade de serviço, apesar de configuração dependente da aplicação ser necessária.

Abstract

In the last decade, a large growth on the usage of virtualized infrastructures for application deployment has been observed. Seeking flexibility on resources allocation and lower management cost, many companies adopted computing clouds as environments to perform some of their crucial activities. A problem observed in such environments is the assurance of quality of service in the execution of applications. The amount of resources required by applications varies with the time. Peaks on the amount of requests in executions of interactive applications, such as e-commerce websites, are common and is necessary to add resources in order to keep a reasonable response time. In executions of batch applications, normally we want to ensure a deadline, related to the application's priority. Estimating the required amount of resources to assure execution deadline is difficult and prone to underallocations or superallocations. A different solution is to modify the amount of allocated resources in case the progress is not satisfactory. Following this strategy, this work proposes a provisioning method based on PID controllers and vertical scaling. In order to evaluate the proposed provisioning method on assuring quality of service of batch applications, we executed two microbenchmarks and one application used in Big Data, on a real cloud infrastructure where the provisioning was controlled by our method. We collected the execution times and resources usage data from the microbenchmarks and the test application. Results show that the provisioning method is feasible and effective on ensuring quality of service, although it needs application-dependent configuration.

Agradecimentos

Agradeço a minha família, pelo apoio incondicional nesses anos de estudo e trabalho; a Andrey, pela orientação e ensinamentos; ao CNPq e ao povo brasileiro, que financiaram a pesquisa realizada.

Conteúdo

Lista de Símbolos

API - *Application Programming Interface*

CPU - *Central Processing Unit*

KVM - *Kernel-based Virtual Machine*

vCPU - *Virtual* CPU

SSH - *Secure Shell*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

1.1 Contextualização

Computação na nuvem é um modelo de computação bem estabelecido atualmente, alimentado pela necessidade das empresas de reduzir custos em um contexto de crescente complexidade da gerência de recursos de tecnologia da informação e facilitar acesso a recursos e funcionalidades pelos usuários.

O NIST [2], *National Institute of Standards and Technology* (Instituto Nacional de Padrões e Tecnologia), define computação na nuvem mais formalmente como "um modelo para habilitar acesso conveniente, sob demanda e através da rede, a um conjunto compartilhado de recursos computacionais configuráveis que podem ser rapidamente provisionados e liberados com mínimo esforço gerencial ou interação do provedor do serviço."

Através desse modelo computacional empresas podem reduzir custos com gerência de recursos de tecnologia da informação, como administração de infraestruturas e software, transferindo a responsabilidade total ou parcialmente para provedores de serviços, através das diferentes modalidades que surgiram.

No contexto de uma das modalidades de Computação na nuvem, o IaaS (*Infrastructure as a Service*), onde máquinas virtuais são provisionadas sob demanda para os usuários do serviço, observou-se o desenvolvimento de diversas plataformas, como Amazon AWS[1],

[1] https://aws.amazon.com

Microsoft Azure[2], Rackspace[3] e Google Cloud[4], estimuladas pela crescente tendência de migração de infraestruturas de empresas para serviços que oferecem flexibilidade de alocação de recursos. Neste cenário, pode-se citar como caso de sucesso o Netflix [4].

Outro aspecto a ser considerado nos últimos anos é o destaque a softwares de código aberto para gerenciamento de nuvens, como o OpenStack[5], que passaram a ser utilizados por várias empresas para gerenciar seus próprios recursos.

1.2 Definição do problema

Aplicações executando em ambientes na nuvem normalmente necessitam que a quantidade de recursos alocados seja ajustada ao longo do tempo de forma a manter seu desempenho dentro de limites aceitáveis. Em aplicações interativas, como sites de *e-commerce*, em geral o interesse é manter o tempo de resposta a requisições abaixo de um certo limite. Em aplicações não-interativas (*batch*), deseja-se que a aplicação termine sua execução dentro de um prazo preestabelecido.

Em aplicações interativas, uma causa comum de variações no tempo de resposta é a variação na carga de trabalho. Cargas de aplicações desse tipo em geral apresentam certo nível de sazonalidade quanto ao número de requisições e as aplicações demandam diferentes quantidades de recursos ao longo do tempo.

Aplicações não-interativas podem ser difíceis de prever e de realizar um planejamento adequado de infraestrutura. Um exemplo são aplicações não-periódicas, que são executadas poucas vezes, ou mesmo uma vez apenas. Mesmo aplicações que são executadas periodicamente podem ser difíceis de prever, uma vez que os dados não são repetidos. Além disso, aplicações podem sofrer influência de perturbações do ambiente. Neste último caso, a quantidade de recursos planejada pode não ser o suficiente para cumprir metas de tempo de execução e é necessário adicionar recursos.

Considerando o contexto de aplicações não-interativas, tem-se então o problema de controlar ao longo do tempo a quantidade de recursos alocados à aplicação de forma a cumprir

[2]https://azure.microsoft.com

[3]https://www.rackspace.com

[4]https://cloud.google.com

[5]https://www.openstack.org

prazos de execução. Quando o progresso da aplicação não for satisfatório, deve-se ajustar essa quantidade de recursos.

Neste trabalho foi investigada uma alternativa de provisionamento de recursos baseada em controladores PID e provisionamento vertical, para aplicações não-interativas. Um *microbenchmark* focado em uso de CPU, um *microbenchmark* focado em I/O e uma aplicação utilizada no contexto de qualidade de dados em *Big Data* foram executados em um ambiente de nuvem realista com provisionamento de recursos coordenado por uma implementação da alternativa investigada. Com o objetivo de avaliar a qualidade do provisionamento, foram coletados o tempo de execução e o uso de recursos dos *microbenchmarks* e da aplicação. Resultados mostraram que a alternativa de provisionamento é viável e eficaz quanto a garantir qualidade de serviço, apesar de configuração dependente da aplicação ser necessária.

1.3 Estrutura da dissertação

Esta dissertação é estruturado em 8 capítulos. Um resumo do conteúdo dos próximos capítulos é apresentado nesta seção.

No Capítulo 2 são apresentados o estado da arte e da prática em provisionamento de recursos computacionais no contexto de computação na nuvem e do uso de controladores PID em provisionamento.

No Capítulo 3 são discutidos os principais desafios relacionados ao problema tratado nesta pesquisa e a contribuição do presente trabalho.

No Capítulo 4 é apresentado o contexto no qual a alternativa proposta foi implementada: o projeto BigSea e o *framework* Asperathos. São discutidas também as tecnologias envolvidas na implementação.

No Capítulo 5 é explicado o funcionamento dos controladores e métodos de atuação considerados na pesquisa. Adicionalmente, é apresentada a configuração utilizada.

No Capítulo 6 é detalhado o processo de avaliação da alternativa de provisionamento proposta. Detalhes como padrões de progresso e uso de recurso dos *microbenchmarks* e aplicações, processo de definição de prazos e características da infraestrutura são discutidos neste capítulo.

No Capítulo 7 são apresentados os resultados obtidos a partir do processo de avaliação descrito no Capítulo 6. Cumprimento de prazos e qualidade de gerenciamento da alternativa proposta são discutidos neste capítulo.

No Capítulo 8 são apresentadas as considerações finais quanto ao problema e aos resultados discutidos.

Capítulo 2

Trabalhos relacionados

Neste capítulo são discutidos os trabalhos relevantes ao problema tratado nesta pesquisa. O capítulo está estruturado em duas subseções, que tratam das temáticas principais consideradas: controladores PID e provisionamento de recursos em computação na nuvem.

2.1 Provisionamento

Em provisionamento, dois aspectos importantes a considerar são o método utilizado para decidir a quantidade de recursos a alocar e como esses recursos estão organizados.

Pode-se agrupar os métodos de decisão em reativos e proativos. Métodos reativos decidem ações a tomar baseados nos últimos estados da infraestrutura ou da aplicação, muitas vezes funcionando com base em gatilhos. Quando o valor de determinada métrica não é satisfatório, ele executa determinada ação na infraestrutura. Métodos proativos tentam estimar a demanda por recursos no futuro, geralmente utilizando históricos de utilização da infraestrutura.

Quanto à organização dos recursos, pode-se classificar o provisionamento como horizontal e vertical. Provisionamento vertical é a adição ou remoção de recursos através de aumento ou redução da capacidade de um ou mais nós de processamento, sem alterações no nível de paralelismo da aplicação. Exemplos são adição de uma CPU ou de memória a uma máquina virtual e modificação de limitadores de uso de CPU. Considerando um contexto de uma aplicação que executa em uma máquina virtual e recebe requisições encaminhadas por um balanceador de carga, um exemplo é apresentado no Figura 2.1.

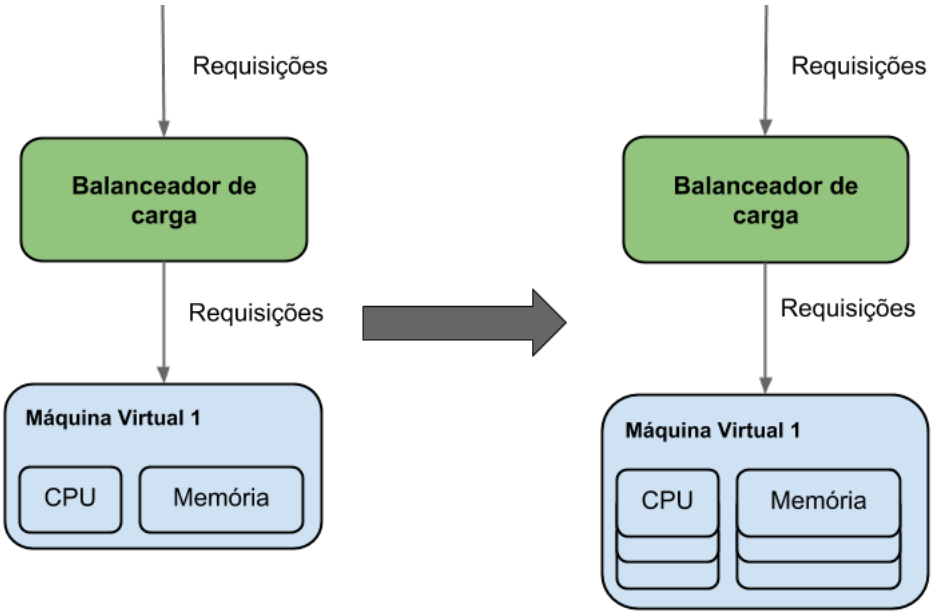


Figura 2.1: Exemplo de provisionamento vertical

Provisionamento horizontal, por sua vez, é a adição ou remoção de recursos através de adição ou remoção de nós de processamento, com alterações no nível de paralelismo da aplicação. Um exemplo é a adição de uma máquina virtual executando uma cópia da aplicação e compartilhamento da carga de trabalho utilizando um balanceador de carga. Um exemplo é apresentado na Figura 2.2.

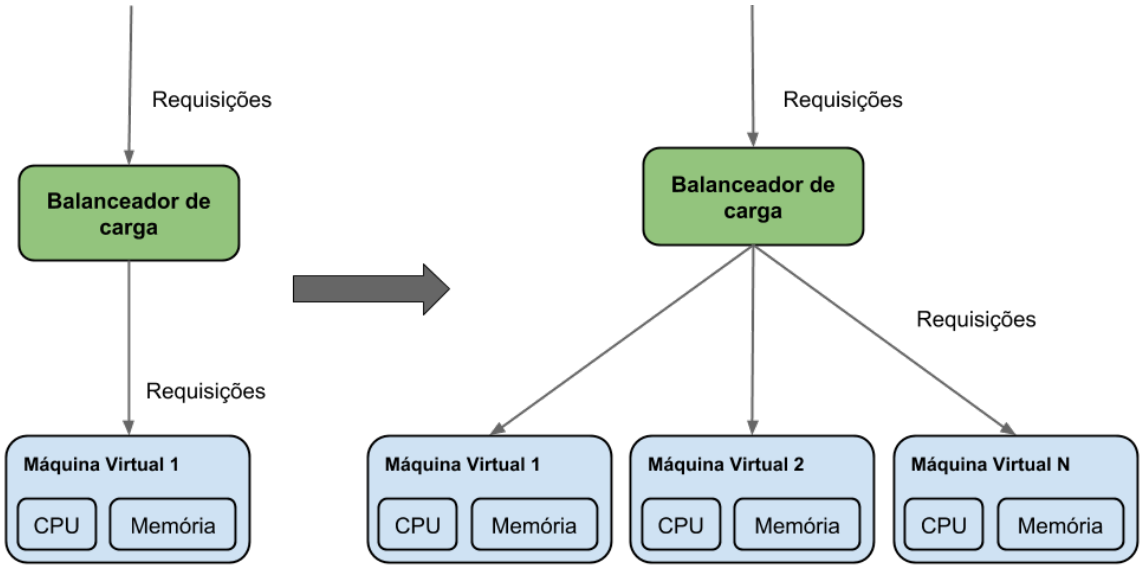


Figura 2.2: Exemplo de provisionamento horizontal

Provisionamento vertical, em geral, é executado rapidamente, independente de aplicação e com efeitos imediatos no ambiente. No entanto, esse tipo de provisionamento é limitado à capacidade dos servidores utilizados. Provisionamento horizontal, por sua vez, pode ser demorado e sua efetividade em termos de ganho de desempenho da aplicação depende do grau de paralelismo da mesma. Contudo, este tipo de provisionamento pode ser aplicado indefinidamente.

Uma forma de superar algumas das limitações dos dois tipos de provisionamento é combiná-los. Um método possível é utilizar provisionamento vertical para compensar oscilações no desempenho da aplicação e apenas utilizar provisionamento horizontal quando não for mais possível escalar verticalmente. Outra possibilidade é escalar verticalmente, mantendo a aplicação em um nível de desempenho razoável, enquanto o provisionamento horizontal é realizado.

Quanto a exemplos de uso de métodos reativos, pode-se citar os serviços de provisionamento horizontal automático do Google Cloud [7] e do AWS [6]. Ambos os serviços compartilham o mesmo princípio básico. O usuário define metas para uma ou mais métricas de desempenho, normalmente uso de CPU ou solicitações por segundo. Essas metas servem como gatilhos para ações do serviço, que monitora a infraestrutura e pode modificar a quantidade de máquinas virtuais ou contêineres alocados.

No mesmo contexto, de serviços de computação na nuvem, pode-se citar o emprego de provisionamento vertical no AWS. Um dos modelos de serviço disponíveis [4] é baseado em créditos de CPU. Neste modelo, máquinas virtuais acumulam créditos periodicamente quando estão ociosas e consomem esses créditos quando utilizam CPU. Ao executar aplicações como servidores *web*, que possuem picos esporádicos de uso de CPU, em estruturas que seguem este modelo, pode-se compensar os picos de utilização de CPU a partir do acúmulo de créditos. Este modelo, por outro lado, não é recomendado para aplicações do tipo *batch*.

Um segundo exemplo de trabalho em provisionamento vertical é o CloudVAMP [14]. O CloudVAMP é um sistema que permite o gerenciamento de memória ociosa de máquinas virtuais em uma infraestrutura de nuvem. Esse sistema reserva a memória ociosa das máquinas da nuvem para suprir as necessidades da infraestrutura, seja em provisionar mais memória para máquinas existentes ou criar novas. Se a memória é necessária na máquina original, ela é devolvida.

Pode-se citar também os trabalhos de Yazdanov et al. [30], que avalia provisionamento vertical de CPUs utilizando uso de CPU como métrica, e de Mao et al. [25], que avalia provisionamento horizontal de máquinas virtuais baseado em prazos de execução.

Quanto a abordagens híbridas, o trabalho de Sedaghat et al. [29] propõe uma abordagem que utiliza ambos provisionamento vertical e horizontal. Ela considera a heterogeneidade das máquinas virtuais da infraestrutura, assim como possíveis custos de migração, ao definir o conjunto de máquinas que apresentam o melhor custo-benefício para a execução da aplicação. Um aspecto importante deste trabalho é a importância dada às diferenças de resposta das aplicações a mudanças no tamanho das máquinas utilizadas ou, em outras palavras, diferenças nas reações das aplicações ao provisionamento. Este trabalho foca em provisionamento sem alterar máquinas virtuais em uso, técnica que pode ser explorada em outras pesquisas.

O trabalho de Netto et al. [28] aborda o problema de manter a utilização de recursos por uma determinada aplicação dentro de limites aceitáveis ao longo do tempo, comparando diferentes técnicas de provisionamento quanto à eficácia na resolução do problema. O trabalho propõe uma métrica de qualidade de provisionamento e avaliação de métodos de escolha de parâmetros para mecanismos de provisionamento.

Quanto a trabalhos com métodos proativos, o trabalho de Morais et al. [27] propõe uma abordagem híbrida, proativo e reativa, utilizando preditores de carga de trabalho. As predições são feitas com base em um histórico de utilização da infraestrutura. São propostos um mecanismo que escolhe o melhor preditor para o contexto e um método de correção que usa o histórico de erros dos preditores para reduzir a probabilidade de subprovisionamento de recursos. Utilizando o método proposto foi alcançada uma redução no custo de operação de até 37% e uma probabilidade de violação de SLOs menor que 0,036%.

O trabalho de Lorido-Botrán et al. [24] apresenta uma revisão de técnicas matemáticas que podem ser utilizadas para implementação tanto de estratégias preditivas de provisionamento como de estratégias reativas. Controladores PID são citados como opção reativa. A próxima seção discorre brevemente sobre Teoria de Controle e sobre trabalhos relacionados na área.

2.2 Controladores

Ao longo dos anos controladores foram utilizados na resolução de diversos problemas de engenharia que envolviam manter determinadas características de um sistema em condições satisfatórias. Um exemplo são os termostatos, onde o interesse é manter a temperatura de um determinado ambiente dentro de um certo intervalo. Em Computação, exemplos de utilização são balanceamento de carga e *caching* [15] em sistemas distribuídos.

Um dos modelos de controle mais amplamente utilizados é o controle de ciclo fechado, cujo esquema é apresentado de forma simplificada na Figura 2.3.

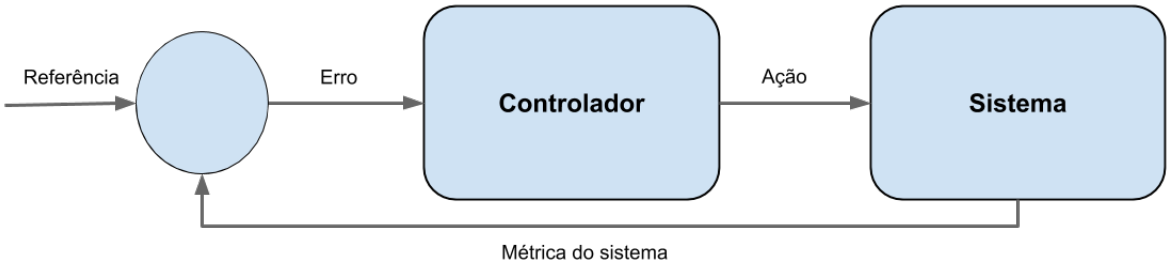


Figura 2.3: Diagrama de controle simplificado

Neste modelo de controle, os principais elementos são:

- Sistema: ambiente ou entidade considerada.
- Controlador: responsável pelo ajuste da característica do Sistema.
- Valor de referência: valor desejado para a característica do Sistema que deseja-se controlar.
- Métrica do sistema: valor coletado da característica do Sistema.
- Erro: diferença entre o valor de referência e o valor medido.
- Ação: valor que afeta o funcionamento do Sistema, decidido pelo Controlador.

Neste modelo, o Controlador recebe um valor de referência, que orienta suas ações. Periodicamente, o Controlador recebe um valor de erro, calculado a partir da métrica do sistema coletada e do valor de referência. A partir desse valor de erro, o Controlador decide a ação necessária sobre o Sistema.

Quanto a trabalhos sobre controladores em Computação, Hellestein et al. [21], em sua obra sobre controladores, tratam em detalhes dos diferentes modelos, implementações e de seu uso em sistemas computacionais.

Um dos problemas resolvidos com controladores PID é de detecção de falhas em sistemas distribuídos. O trabalho de Sá et al. [17] demonstra a utilização de controladores PID na configuração dinâmica de parâmetros de detectores de falha como período de monitoramento e *time-out*. O processo de configuração proposto considera mudanças no ambiente computacional e restrições de qualidade de serviço definidas pelo usuário.

Hollot et al. [22] demonstraram a utilização de controladores no contexto de redução de congestionamento de redes. Uma solução comum para redução da sobrecarga é o RED (*random early detection*), que consiste em descartar pacotes quando houver expectativa de sobrecarga, não apenas quando a sobrecarga já estiver presente. A partir do uso de controladores e da modelagem do RED como um sistema de controle, pode-se atacar o problema de sua configuração.

Um exemplo no contexto de provisionamento horizontal é o trabalho de Abranches et al. [16]. Este trabalho propõe um mecanismo de provisionamento para sistemas baseados em contêineres. O algoritmo de provisionamento, que utiliza tempo de resposta como principal métrica e é baseado em controladores PID, é avaliado sob uma carga de trabalho real e comparado com a opção do Google. Os resultados apontam que o mecanismo é mais eficiente na alocação de recursos.

Em provisionamento vertical, o trabalho de Farokhi et al. [19] propõe uma abordagem híbrida que considera ambos desempenho da aplicação e a utilização de recursos e realiza provisionamento vertical de memória a partir de um controlador. Resultados revelaram que a estratégia de provisionamento baseada em controladores atinge uma utilização de memória alta, com alocação mínima de memória e alta estabilidade.

Capítulo 3

Motivação e proposta

Este capítulo apresenta uma discussão sobre os principais desafios envolvidos na resolução do problema descrito no Capítulo 1, assim como a contribuição deste trabalho em termos de resolver o problema e suprir deficiências dos trabalhos citados no Capítulo 2.

3.1 Desafios

O caso de uso estudado neste trabalho, a execução de aplicações não-interativas (*batch*), em geral apresenta cumprimento do prazo de término como principal meta de qualidade de serviço. No contexto de uma aplicação de processamento paralelo que executa em um conjunto de máquinas virtuais, a implementação de um mecanismo que garanta término dentro do prazo a partir do controle da quantidade de recursos alocados exige que diversos aspectos sejam considerados. Alguns destes aspectos são detalhados nesta seção.

3.1.1 Diferentes tipos de progresso da aplicação

Linearidade é uma propriedade importante do progresso de uma aplicação. Aplicações com progresso linear são mais fáceis de modelar. Neste tipo de aplicação, o progresso, em termos de tarefas, por exemplo, ocorre em um ritmo semelhante ao progresso do tempo (n unidades de tempo correspondem quase sempre a um mesmo valor proporcional de unidades de progresso). Consequentemente, é menos provável que atuações bruscas no ambiente sejam necessárias, facilitando também o planejamento da infraestrutura.

Aplicações com progresso não-linear, que apresentam fases muito rápidas e outras fases muito lentas, por exemplo, são mais difíceis de modelar. Dependendo do padrão do progresso, pode ser necessário realizar alterações mais bruscas e severas na quantidade de recursos alocados. Essas ações exigem mecanismos de ação mais rápida.

3.1.2 Escolha do método de atuação

O acesso a um método de alteração da quantidade de recursos alocados às aplicações é essencial ao funcionamento de um mecanismo que garante término dentro do prazo de execução. Neste trabalho, tal método é chamado de método de atuação. A escolha do método adequado é de extrema importância para o sucesso do provisionamento. Agir corretamente sobre os tipos de recursos que são gargalo é essencial para melhorar o desempenho da aplicação.

Dois aspectos devem ser considerados ao escolher o método de atuação: os tipos de recursos em questão, CPU, memória, I/O de disco e I/O de rede, por exemplo; e o tipo de provisionamento, vertical ou horizontal. É necessário também considerar as limitações das plataformas utilizadas, como hipervisores e gerentes de nuvem, que podem não oferecer certas opções de provisionamento [5].

Em um cenário ideal, a escolha do método de atuação é guiado pelas características da aplicação. Por exemplo, uma aplicação com alto grau de paralelismo deve se beneficiar mais de métodos de provisionamento horizontal do que outras com baixo grau de paralelismo. No caso das últimas, deve-se considerar provisionamento vertical. O provisionamento de aplicações que fazem uso intensivo de CPU, como simuladores, deve ser feito considerando métodos de atuação em CPU. Por sua vez, aplicações que fazem uso intensivo de disco, como aplicações MapReduce, devem considerar métodos de atuação em disco.

Exemplos de provisionamento vertical, considerando uma aplicação executando em uma máquina virtual, são alteração na cota de tempo de CPU, alteração na quantidade de memória alocada e alteração na cota de operações no disco. Um exemplo de provisionamento horizontal é a adição de um balanceador de carga e de mais máquinas virtuais executando cópias da aplicação.

3.1.3 Resultado da ação sobre a aplicação

Aplicações podem apresentar diferentes fases, que utilizam diferentes tipos de recursos e reagem de formas diferentes ao provisionamento. Exemplos são aplicações que apresentam fases de leitura dos dados do disco e, em sequência, realizam algum tipo de transformação desses dados. Essas aplicações, portanto, apresentam fases limitadas por disco e outras fases limitadas por CPU.

É necessário saber como a alteração nos recursos alocados afeta o desempenho e o progresso da aplicação. A dificuldade em prever o efeito da atuação aumenta as chances do mecanismo de provisionamento tomar decisões erradas e pode influenciar o quanto o mecanismo deve ser conservador.

3.1.4 Efeito da modificação sobre o ambiente

Deslocar mais recursos para uma aplicação pode prejudicar o desempenho de outras que estão executando no mesmo ambiente. Mudanças agressivas podem causar instabilidade na infraestrutura.

3.1.5 Perturbação do ambiente

Perturbações do ambiente podem provocar variações bruscas no progresso da aplicação. Um exemplo é um pico de utilização de disco pelas aplicações executando no mesmo nó de computação. A solução de provisionamento deve compensar esse tipo de variação.

3.1.6 Grande quantidade de parâmetros possíveis

Existe uma grande quantidade de parâmetros relacionados à forma como o mecanismo de provisionamento age. É necessário definir faixas de valores adequados para eles. Alguns exemplos são listados a seguir.

- Quantidade inicial de recursos alocados à aplicação: este parâmetro é mais importante no contexto de aplicações de execução mais curta. Uma vez que o provisionamento de recursos pode ser custoso (o tempo de adição de máquinas virtuais a um *cluster* é significativo [26], por exemplo), ele pode não ser concluído a tempo de cumprir prazos.

Aplicações de execução mais longa, por outro lado, são menos sensíveis ao tempo de provisionamento.

- Periodicidade de atuação: o quão frequente será a atuação na infraestrutura. Dependendo da implementação do mecanismo de provisionamento, atuar muito frequentemente pode ser problemático.

- Grão de atuação: a quantidade de recursos que será adicionada ou removida. Pode ser um valor fixo ou ser função do quão atrasada ou adiantada a aplicação está. Valores altos permitem compensar atrasos de forma mais imediata, no entanto podem causar maior perturbação na infraestrutura.

- Periodicidade de monitoramento: O quão frequente o monitoramento do estado da aplicação será feito. Uma maior quantidade de informação sobre o desempenho da aplicação permite ao mecanismo de provisionamento observar mais rapidamente o efeito das modificações na infraestrutura sobre o desempenho. Essa observação mais rápida favorece a implementação de mecanismos menos conservadores.

3.1.7 Coleta de progresso pode ser problemática

A coleta de informações sobre o desempenho de aplicações *batch* pode ser mais complexa que em aplicações contínuas. No caso de aplicações contínuas, onde deseja-se manter métricas como tempo de resposta dentro de determinados intervalos, é comum estimar o valor de métricas de serviço a partir de métricas de infraestrutura, como uso de CPU. O provisionamento é, então, executado independentemente de aplicação. No caso de aplicações *batch*, a utilização desse método não é possível.

É comum que aplicações *batch* utilizem boa parte ou mesmo a totalidade dos recursos disponibilizados. Por exemplo, uma aplicação de processamento de dados pode permanecer durante longos períodos de tempo utilizando todos os recursos de CPU disponíveis, e essa alta utilização não é indício de que o desempenho da aplicação seja insuficiente ou de que a aplicação esteja atrasada. Desta forma, o provisionamento exige a consulta da aplicação ou da plataforma utilizada para executá-la, adicionando mais complexidade à implementação do mecanismo de provisionamento.

Adicionalmente, aplicações podem não apresentar informações suficientes sobre o progresso ao longo da sua execução, dificultando o provisionamento de recursos. Exemplos são aplicações que possuem poucas etapas e cujo progresso ocorre em grandes passos. Nesse tipo de situação, informações sobre progresso são mais escassas, o que faz ser mais provável que o mecanismo de provisionamento tome decisões erradas sobre a quantidade de recursos necessários.

Outra questão é a necessidade de compensar o atraso adicionado pelos serviços utilizados para coletar e publicar informações de progresso. Nos casos em que as informações recebidas pelo mecanismo de provisionamento são muito desatualizadas, é necessário estimar o estado atual da aplicação antes de tomar decisões.

3.2 Contribuição

Considerando o problema apresentado e os desafios relacionados, foram analisados a viabilidade e os benefícios da utilização de provisionamento vertical baseado em controladores PID com o objetivo de garantir qualidade de serviço de aplicações não-interativas executando em ambientes de nuvem.

A investigação executada consistiu da avaliação de três aspectos principais do método de provisionamento citado:

- Garantia de qualidade de serviço, em termos do término das aplicações no prazo;
- Uso inteligente dos recursos disponíveis, gerando baixa perturbação no ambiente e alocando apenas os recursos necessários. Subestimar a quantidade de recursos necessários provoca atrasos, enquanto que superalocar recursos provoca rejeição desnecessária de aplicações;
- Simplicidade de configuração.

Os métodos de provisionamento dos trabalhos relacionados tendem a tentar prever a carga à qual a infraestrutura será submetida ou a considerar aplicações interativas. O método de provisionamento proposto neste trabalho apresenta-se como uma complementação às técnicas de previsão, compensando a irregularidade das aplicações. Além disso, a atuação

na infraestrutura é realizada de forma vertical, com efeito imediato, independente do grau de paralelismo da aplicação e sem migração de máquinas virtuais.

3.3 Escopo

Este trabalho trata de alguns dos problemas citados neste capítulo. Foram realizadas execuções com aplicações em que o progresso é não-linear. Dois casos de não-linearidade foram observados e sugestões sobre como lidar com o problema foram testadas. Quanto à escolha de método de atuação, foi observada a eficácia de atuação em dois tipos de recursos, CPU e I/O de disco, no contexto de provisionamento vertical. Quanto à parametrização, foi investigada a eficácia da decisão do grão de atuação por controladores durante a execução. Quanto ao efeito da modificação sobre o ambiente, a análise da solução de provisionamento proposta considerou a perturbação que a solução pode causar no ambiente de execução.

Este trabalho considera aplicações que são executadas diversas vezes. Um exemplo é uma aplicação de processamento de dados no contexto de *Big Data*, que realiza periodicamente alguma transformação destinada a melhorar a qualidade dos dados. Tais aplicações podem ser mais facilmente modeladas e o provisionamento destas aplicações pode se beneficiar com maior facilidade do conteúdo deste trabalho.

Além disso, um controle ótimo para aplicações com diversas fases exigiria modelagem e parametrização individual do controlador para cada fase. Embora em alguns casos de aplicações que são computacionalmente custosas e executadas muitas vezes isso seja vantajoso, neste trabalho considera-se apenas uma configuração de controlador para toda a execução da aplicação.

Capítulo 4

Serviços de infraestrutura BigSea

A pesquisa descrita nesta dissertação foi desenvolvida no contexto do projeto BigSea. Neste capítulo, este projeto será apresentado brevemente, assim como os componentes desenvolvidos que interagem com o objeto de estudo, o Controlador.

4.1 Arquitetura

O projeto BigSea[1] visa o desenvolvimento de arquiteturas e serviços com o intuito de facilitar a implantação de aplicações de análise de dados no contexto de *Big Data* em grandes cidades, com qualidade de serviço e segurança. Participam do projeto equipes de universidades brasileiras e europeias, divididas em 7 módulos de trabalho, os WPs (*Work-Packages*). Os focos dos módulos de trabalhos variam de desenvolvimento de modelos de programação à serviços de infraestrutura. Este último é o foco do WP3, subprojeto no qual esta pesquisa foi desenvolvida.

Os trabalhos do WP3 se concentram em desenvolver componentes que garantam qualidade de serviço através de ajustes da infraestrutura virtualizada sobre a qual as aplicações de análise de dados executam. Atualmente esses componentes encontram-se agrupados no *framework* Asperathos.

A Figura 4.1 apresenta o fluxo de submissão e execução de uma aplicação na plataforma BigSea. Os principais módulos envolvidos em uma submissão são os seguintes:

[1] http://www.eubra-bigsea.eu

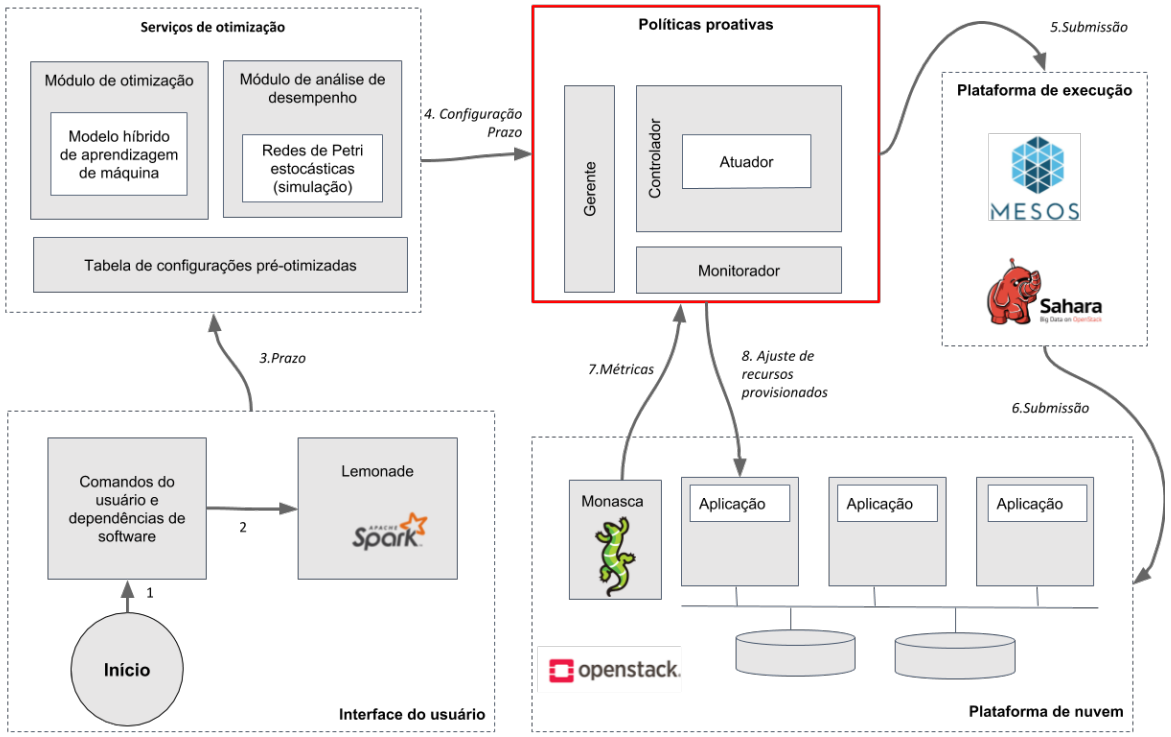


Figura 4.1: Fluxo de execução nos componentes BigSea

1. Interface do usuário: módulo de acesso à plataforma BigSea, ela também é responsável por facilitar o desenvolvimento de aplicações paralelas. A interface BigSea utiliza *Lemonade (Live Exploration and Mining Of a Non-trivial Amount of Data from Everywhere)*, plataforma de análise de dados desenvolvida pela UFMG que suporta definição de tarefas de exploração, mineração e aprendizagem de forma intuitiva. Através de uma aplicação *web*, o usuário define fluxos de tarefas visualmente, manipulando operações e conjuntos de dados. Os comandos são exportados em uma aplicação que pode ser executada em uma plataforma de processamento paralelo.

2. Serviços de otimização: os serviços contidos neste módulo estimam a quantidade de recursos necessários para a execução da aplicação dentro do prazo utilizando técnicas de aprendizagem de máquina e Redes de Petri, e apresentam essa estimativa na forma de configuração para os serviços de políticas proativas.

3. Políticas proativas: módulo responsável por ajustar a quantidade de recursos alocados à aplicação durante sua execução, baseado em métricas fornecidas pelos sistemas de monitoramento e da configuração apresentada pelos serviços de otimização.

4. Plataforma de execução: responsável por coordenar a alocação dos recursos destinados à execução da aplicação e por disparar essa execução no ambiente de nuvem.

5. Plataforma de nuvem: onde a aplicação e os sistemas de monitoramento de recursos são executados.

Este trabalho tem como foco o módulo de políticas proativas. Mais especificamente, o trabalho considera certas tecnologias, por exemplo, OpenStack como gerente de nuvem, KVM como provisionador de máquinas virtuais, e Spark como plataforma de execução de aplicações *batch*. Essas tecnologias são detalhadas abaixo. Maiores informações sobre outras plataformas e tecnologias suportadas pelo BigSea podem ser encontradas na página do projeto[2].

4.2 Tecnologias

A implementação dos componentes do Asperathos considerou a utilização de algumas tecnologias na implementação das aplicações, no ambiente de execução e na infraestrutura virtualizada. Essas tecnologias são apresentadas nesta seção.

4.2.1 KVM

KVM[3] (*Kernel-based Virtual Machine*) é uma solução de virtualização para Linux, contendo extensões de virtualização (Intel VT or AMD-V). KVM é composto por um módulo do kernel, kvm.ko, que fornece a infraestrutura de virtualização e módulos específicos para processadores, kvm-intel.ko e kvm-amd.ko.

4.2.2 OpenStack

OpenStack[4] é uma das principais plataformas de gerência de infraestruturas de nuvem disponíveis no mercado atualmente. A plataforma é composta de um grande número de serviços

[2]http://www.eubra-bigsea.eu

[3]https://www.linux-kvm.org/page/Main_Page

[4]https://www.openstack.org

que gerenciam cada aspecto da infraestrutura de nuvem individualmente, permitindo a utilização eficiente e fácil de recursos computacionais. OpenStack é software de código-aberto e utilizado por diversas organizações, de institutos de pesquisa, como o CERN [12], Organização Europeia para a Pesquisa Nuclear, a redes varejistas, como o Walmart [13].

No contexto do Asperathos, os componentes Openstack de maior destaque são o Nova, do qual são utilizadas as funcionalidades de criação e remoção de máquinas virtuais, o Sahara, na criação e remoção de *clusters* e o Monasca[5], como serviço de disponibilização de métricas.

4.2.3 Spark

Em um contexto de complexidade de desenvolvimento e implantação de aplicações paralelas, o modelo MapReduce [18] surgiu como uma alternativa em termos de simplicidade e praticidade. O modelo consiste no desenvolvimento de aplicações baseadas em duas funções: *map* (tipicamente associada a transformação) e *reduce* (associada a agregação). Essas duas funções devem conter a lógica de negócio da aplicação. Preocupações de execução, como transmissão de dados e tolerância a falhas, são deixadas para a implementação do modelo.

No contexto de MapReduce, Hadoop[6] despontou como uma das principais implementações disponíveis no mercado. Nos últimos anos, Spark[7], projeto derivado do Hadoop, tornou-se mais popular, devido ao melhor uso de memória disponível durante a execução e consequentes melhores resultados [11].

4.3 Asperathos

O objetivo do Asperathos é disponibilizar componentes configuráveis que permitam o gerenciamento de recursos de uma nuvem, com garantia de qualidade de serviço. O *framework* segue uma arquitetura de microsserviços, em que seus componentes se comunicam através de APIs REST. Os três componentes do *framework*, Gerente, Monitorador e Controlador, podem ser configurados e adaptados a partir da criação de *plugins*.

[5]https://wiki.openstack.org/wiki/Monasca

[6]http://hadoop.apache.org

[7]https://spark.apache.org

O fluxo de execução de uma submissão típica de uma aplicação é apresentado na Figura 4.2. No Asperathos, a submissão é recebida pelo Gerente, que inicia a aplicação na infraestrutura. O Monitorador, que coleta métricas, e o Controlador, que atua na infraestrutura, são iniciados com o objetivo de garantir o prazo de término da aplicação. A descrição mais detalhada do funcionamento dos componentes é apresentada nas seções a seguir.

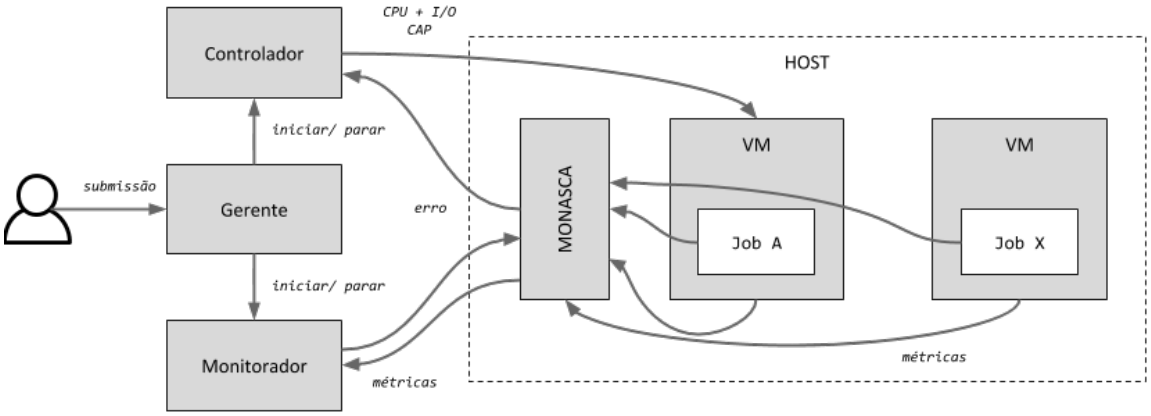


Figura 4.2: Fluxo de execução no Asperathos

4.3.1 Gerente

O Gerente é responsável por coordenar a execução de aplicações utilizando a infraestrutura e o funcionamento dos outros componentes do *framework*. É seu papel comunicar-se com o *software* de gerência de nuvem, como o OpenStack, para coordenar a criação do ambiente no qual a aplicação será executada e tornar os outros componentes cientes da aplicação. O fluxo de funcionamento de uma submissão típica ao Gerente é descrito a seguir e apresentado na Figura 4.3.

1. Gerente recebe a submissão da aplicação contendo sua configuração, do monitorador e do controlador utilizados.

2. Gerente cria o *cluster* ou as máquinas virtuais utilizadas pela aplicação.

3. Gerente solicita ao controlador que ajuste a quantidade de recursos alocados.

4. Gerente inicia a aplicação na máquina virtual ou *cluster*.

5. Gerente solicita ao controlador que inicie suas ações sobre a infraestrutura.

6. Gerente solicita ao monitorador que inicie o monitoramento da aplicação.

7. Gerente espera o fim da execução da aplicação. A forma como o Gerente espera o fim da execução depende do tipo da aplicação.

    - Exemplo 1: Aplicação executando em um *cluster* Spark. Neste caso, o Gerente consulta o mestre do *cluster* utilizado.
    - Exemplo 2: Aplicação executando em uma máquina virtual. Neste segundo caso, o Gerente supõe que a aplicação desliga a máquina ao terminar. Supondo isso, o Gerente consulta a API do OpenStack sobre o estado da máquina virtual.

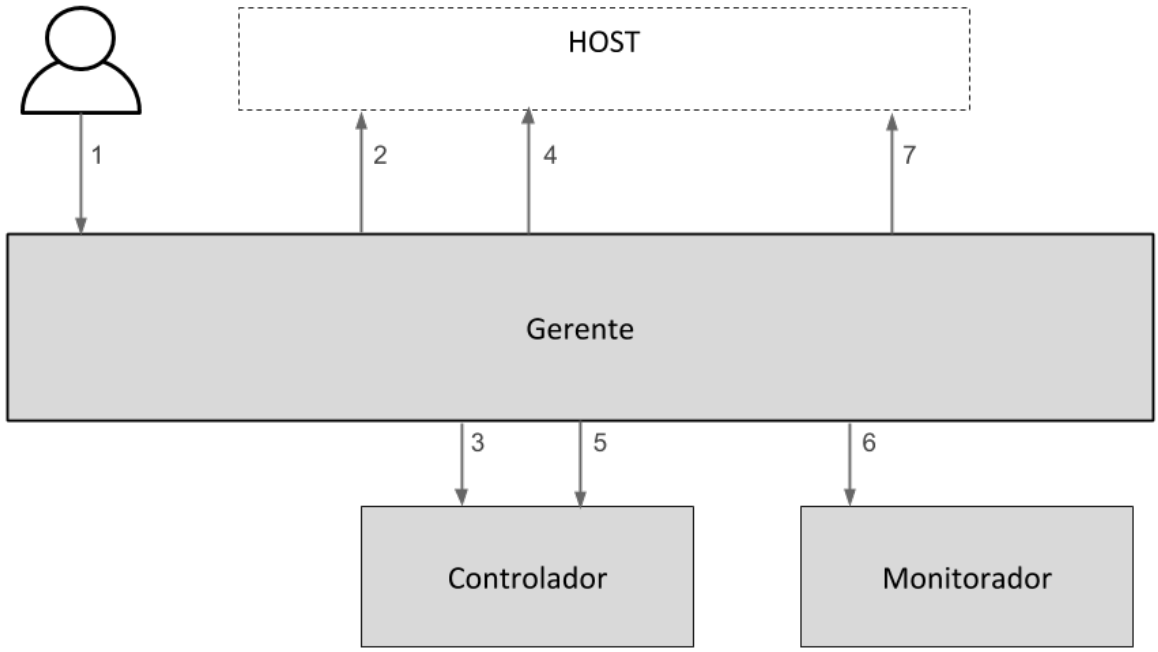


Figura 4.3: Fluxo de execução no Asperathos considerando as ações do Gerente

O Gerente do Asperathos é *software* de código-aberto disponível no GitHub [9].

4.3.2 Monitorador

O Monitorador é responsável por coletar métricas de desempenho das aplicações e fornecer às entidades interessadas um diagnóstico sobre o estado das aplicações baseado em valores

de referência. O diagnóstico, na implementação considerada neste trabalho, é apresentado como um valor de erro, ou a diferença entre o valor ideal e o valor medido. O fluxo de um monitoramento típico é descrito a seguir e apresentado na Figura 4.4.

1. Monitorador recebe requisição de monitoramento de aplicação do gerente, com credenciais e informações específicas da aplicação monitorada.

2. Periodicamente, o Monitorador recalcula o progresso da aplicação a partir de alguma métrica exportada por ela. O método de coleta do progresso depende do tipo da aplicação.

   - Exemplo 1: Aplicação executando em um *cluster* Spark. Neste caso, o Monitorador pode utilizar a API do Spark para obter informações sobre o progresso da aplicação. Em Python, um exemplo de requisição é apresentado no Código 4.1.

   Código Fonte 4.1: Exemplo de requisição de coleta de progresso

```
1        job_request = requests.get(spark_master
2                          + ':4040/api/v1/applications/'
3                          + id_aplicacao + '/jobs')
```

   - Exemplo 2: Aplicação executando em uma máquina virtual. Neste segundo caso, supondo-se que a aplicação registra o progresso em arquivo, o Monitorador acessa a máquina virtual onde a aplicação executa e obtém o progresso a partir de informações no arquivo.

3. Monitorador calcula o erro. O cálculo do erro depende do tipo da aplicação. Nas implementações de Monitorador consideradas nesta pesquisa, o valor do erro é calculado utilizando um progresso linear como referência.

   - Exemplo 1: No caso de uma aplicação Spark, o erro é calculado a partir das informações coletadas da API do Spark. Sejam n_c o número de tarefas completadas, n_t o número total de tarefas, p_r o progresso real da aplicação, p_e o progresso esperado, d o prazo de execução e t_e o tempo real de execução, o erro e é calculado da seguinte maneira.

$$p_r = n_c/n_t \tag{4.1}$$

$$p_e = t_e/d \tag{4.2}$$

$$e = p_r - p_e \tag{4.3}$$

Os valores de n_c, n_t e t_e podem ser obtidos através de consulta à API RESTful do Spark.

- Exemplo 2: No caso de uma aplicação em uma máquina virtual, o erro é calculado a partir das informações coletadas de um arquivo de log. Sejam p o progresso real da aplicação, p_e o progresso esperado, d o prazo de execução e t_e o tempo real de execução, o erro e é calculado da seguinte maneira.

$$p_e = t_e/d \tag{4.4}$$

$$e = p - p_e \tag{4.5}$$

Uma vez que considera-se um padrão de progresso linear como referência, o valor do progresso esperado nos dois exemplos é calculado como a fração já transcorrida do prazo de execução. Por exemplo, caso o tempo real de execução seja 30% do valor do prazo de execução, espera-se que o valor do progresso seja de 30%.

4. Monitorador publica métrica no Monasca.

O Monitorador do Asperathos é *software* de código-aberto disponível no GitHub [10].

4.3.3 Controlador

O Controlador é responsável por gerenciar o desempenho das aplicações, ajustando a quantidade de recursos alocados à infraestrutura virtual sob a qual a aplicação é executada, com o objetivo de garantir o cumprimento de metas de qualidade de serviço, tais como prazo de execução e tempo de resposta. Esse serviço inclui os componentes Controlador e Atuador. O Controlador, baseado no diagnóstico apresentado pelo Monitorador, decide a quantidade

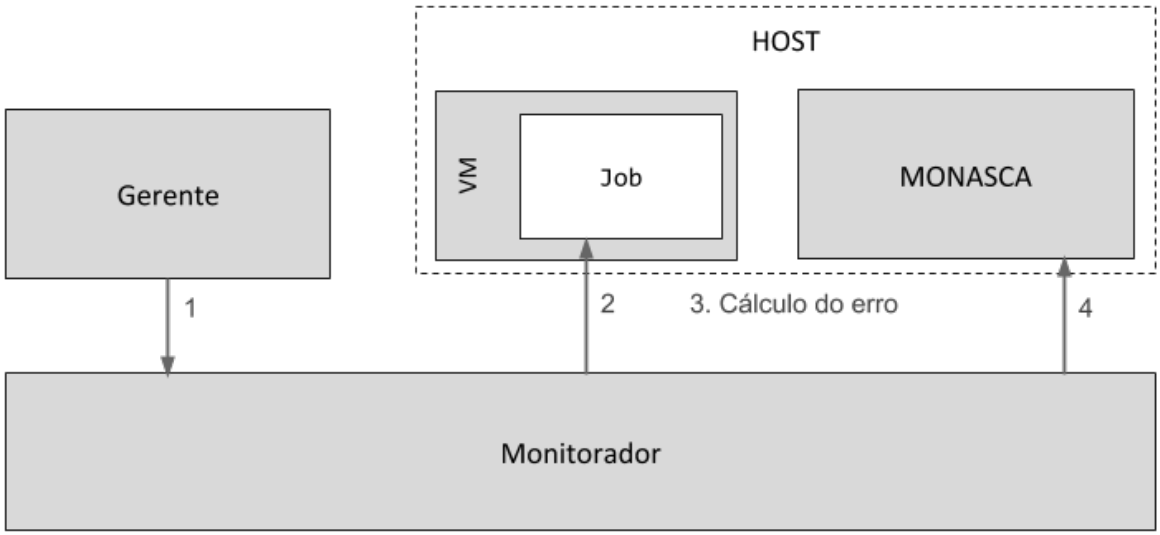


Figura 4.4: Fluxo de execução no Asperathos considerando as ações do Monitorador

de recursos a alocar para as aplicações. O Atuador é responsável por acessar a infraestrutura e disparar os comandos que alocam ou desalocam recursos, baseado nos pedidos do Controlador. Neste trabalho foram investigadas opções de atuação em CPU e disco. O fluxo de controle típico é descrito a seguir e apresentado na Figura 4.5.

1. Controlador recebe requisição de controle de aplicação do gerente, com credenciais e informações específicas da aplicação.

2. Controlador recebe do Monasca o último valor de erro calculado pelo Monitorador.

3. Controlador decide uma ação a realizar no ambiente de execução utilizando um algoritmo de controle.

4. Controlador acessa a infraestrutura e aloca ou desaloca recursos, de acordo com a decisão tomada.

O Controlador do Asperathos é *software* de código-aberto disponível no GitHub [8]. Os algoritmos de controle e atuação utilizados no Asperathos e nesta pesquisa são detalhados no Capítulo 5.

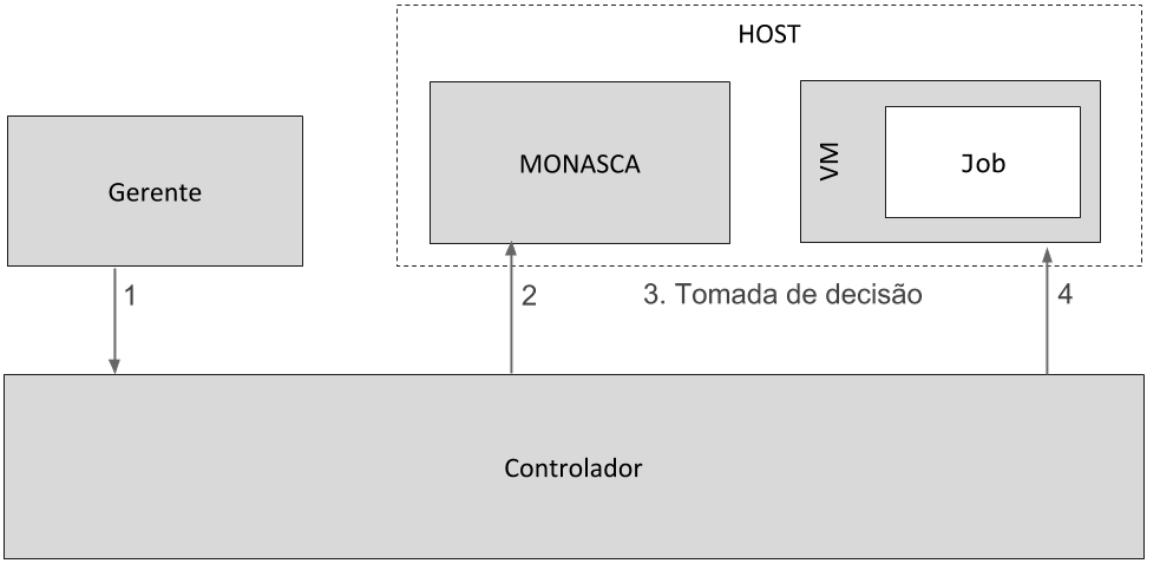


Figura 4.5: Fluxo de execução no Asperathos considerando as ações do Controlador

Capítulo 5

Controladores

Neste capítulo é detalhado o mecanismo de provisionamento proposto neste trabalho. Na Seção 5.1, o funcionamento dos algoritmos de controle é explicado; na Seção 5.2, descreve-se os métodos de atuação; por fim, na Seção 5.3, discute-se aspectos importantes para configuração e execução do mecanismo de provisionamento proposto.

5.1 Algoritmos

Os algoritmos de controle recebem como argumento um valor de erro associado a uma métrica. O valor de erro é a diferença entre o valor esperado e o valor medido. Os algoritmos Proporcional, Proporcional-Derivativo e PID retornam o tamanho da atuação. O valor do tamanho da atuação é utilizado pelo componente Controlador para calcular o novo valor de cota utilizado. No contexto do controlador, cota é uma abstração da quantidade de recursos alocados à execução da aplicação. O funcionamento do controlador não considera os tipos de recursos provisionados, nem a organização destes recursos. A conversão do valor de cota calculado pelos controladores em uma quantidade real de recursos a alocar para a aplicação é realizada pelo método de atuação. O valor da cota máxima é 100%, que representa o total de recursos disponíveis para a execução. A cota mínima é um parâmetro de funcionamento dos controladores e varia entre 0 e 100%. Sejam q_t o valor da cota no tempo t e a o último valor de tamanho de atuação retornado pelo algoritmo de controle, o novo valor de cota q_i é calculado como mostrado a seguir:

$$q_i = q_{i-1} + a \tag{5.1}$$

Esse cálculo não é realizado pelo componente Controlador no caso do algoritmo Mínimo-máximo, que retorna o valor de cota. O funcionamento dos algoritmos de controle, assim como o cálculo do tamanho de atuação e dos valores de cota, são apresentados e discutidos nas seções seguintes.

5.1.1 Mínimo-máximo

Existem dois valores de cota possíveis: um valor base e um valor máximo. O algoritmo modifica a cota com base em valores-gatilho de erro, `gatilho-baixo` e `gatilho-cima`. Quando o erro é negativo (progresso menor que o esperado, por exemplo) e sua magnitude maior que `gatilho-cima`, o valor da cota é modificado para o valor máximo. Quando o erro é positivo (progresso maior que o esperado, por exemplo) e sua magnitude maior que `gatilho-baixo`, o valor da cota é modificado para o valor base. Um exemplo de funcionamento do Controlador Mínimo-Máximo é apresentado na Figura 5.1.

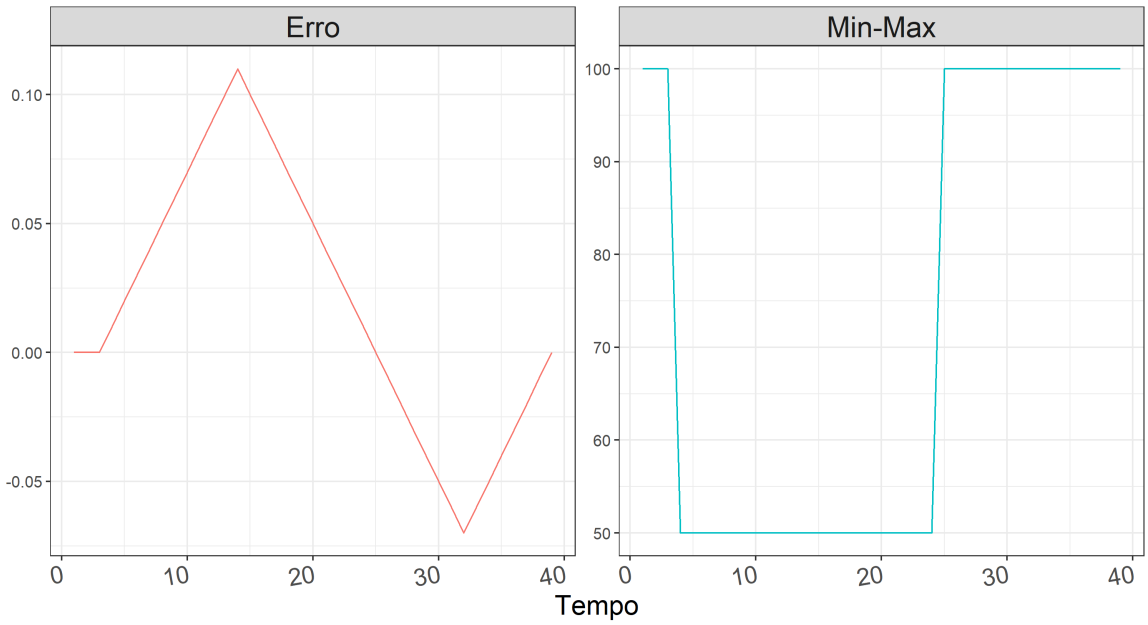


Figura 5.1: Exemplo do comportamento do Controlador Mínimo-Máximo

5.1.2 Proporcional

A quantidade de recursos adicionada ou removida é uma função do erro e do ganho proporcional. O tamanho da atuação é calculado como mostrado a seguir:

$$a = -1 * g_p * e \tag{5.2}$$

Onde g_p é o ganho proporcional, e é o valor do erro e a é o tamanho da atuação. Um exemplo de funcionamento do Controlador Proporcional é apresentado na Figura 5.2. Pode-se observar como a curva da quantidade de recursos a alocar do Controlador Proporcional é simplesmente um espelhamento do gráfico do erro, escalada de acordo com o ganho proporcional.

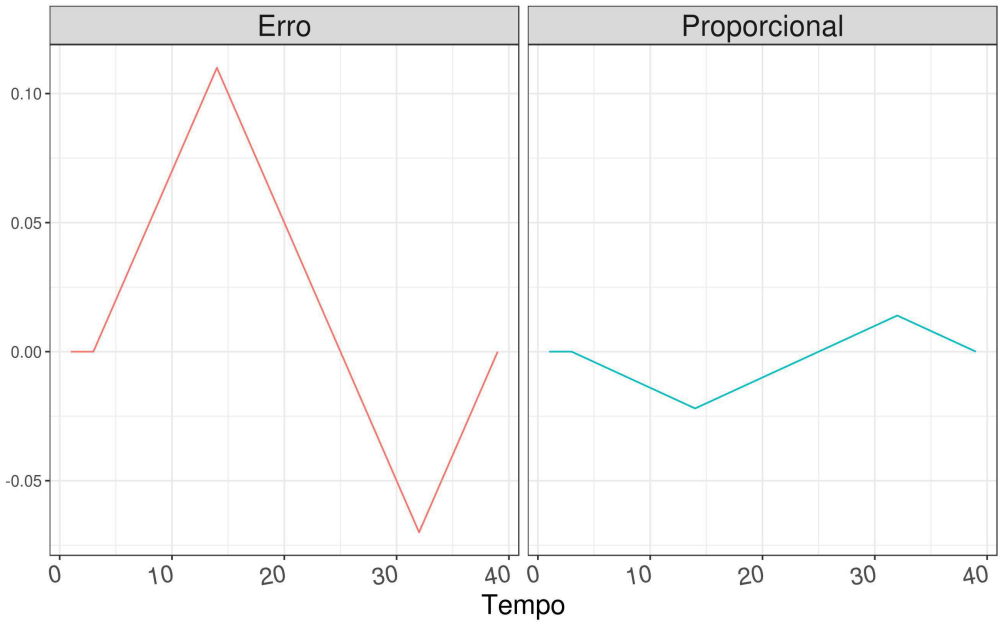


Figura 5.2: Exemplo do comportamento do Controlador Proporcional

Um exemplo de como o tamanho de atuação calculado pelo algoritmo afeta a cota é apresentado na Tabela 5.1. O valor inicial da cota é de 70% e o ganho proporcional é igual a 2. Inicialmente o erro é positivo, o tamanho de atuação calculado é negativo e a cota é reduzida. Em seguida, o erro apresentado ao controlador é negativo, o tamanho de atuação calculado é positivo e a cota é aumentada.

Tabela 5.1: Efeito do tamanho de atuação do controlador Proporcional na cota

Tempo	Erro	Tamanho de atuação	Próxima Cota (%)
1	0	0	70
2	0,01	-0,02	68
3	0,05	-0,1	58
4	0	0	58
5	-0,03	0,06	64

5.1.3 Proporcional-derivativo

A quantidade de recursos adicionada ou removida é uma função dos seguintes valores:

- Erro atual
- Variação do erro (a diferença entre o valor atual e o último recebido)
- Ganho proporcional
- Ganho derivativo

O tamanho da atuação é calculado como mostrado a seguir:

$$c_p = -1 * g_p * e_i \tag{5.3}$$

$$c_d = -1 * g_d * (e_i - e_{i-1}) \tag{5.4}$$

$$a = c_p + c_d \tag{5.5}$$

Onde, c_p é o componente proporcional, c_d é o componente derivativo e a é o tamanho da atuação. g_p é o ganho proporcional, g_d é o ganho derivativo e e_t é o valor do erro no tempo t. Um exemplo de funcionamento do Controlador Proporcional-Derivativo é apresentado na Figura 5.3. Ao utilizar ambos o valor do erro e sua variação, o Controlador Proporcional-Derivativo consegue corrigir o erro observado levando em conta tendências de atraso. Pode-se observar na Figura 5.3 como o Controlador Proporcional-Derivativo reage de forma mais rápida à tendência de atraso do que o Controlador Proporcional.

Um exemplo de como o tamanho de atuação calculado pelo algoritmo afeta a cota é apresentado na Tabela 5.2. O valor inicial da cota é de 70%, o ganho proporcional é igual

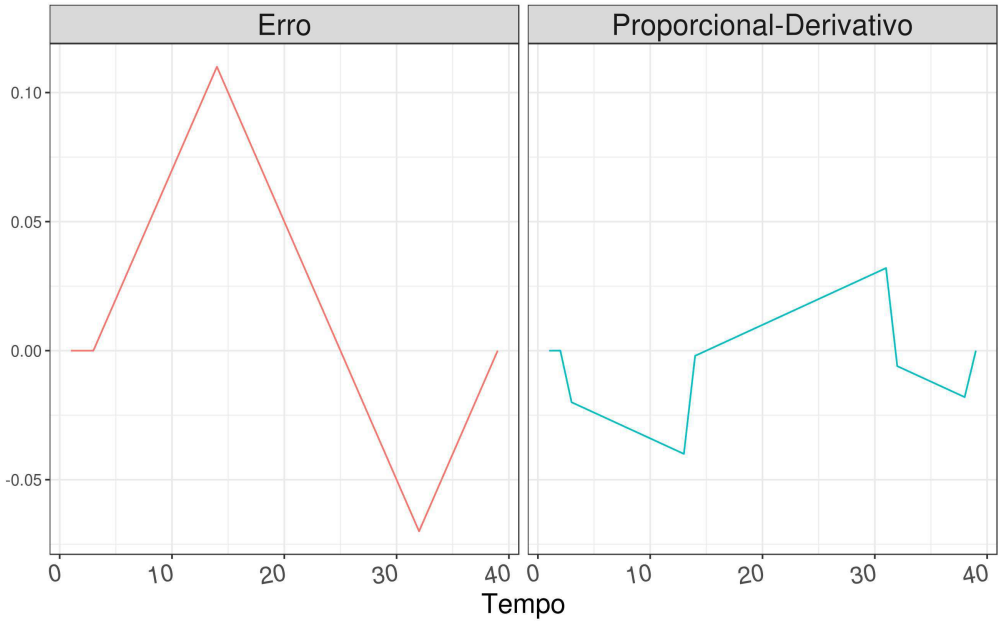


Figura 5.3: Exemplo do comportamento do controlador Proporcional-Derivativo

a 2 e o ganho derivativo é igual a 5. Neste exemplo, o erro inicialmente é positivo, torna-se negativo e no final volta a ser positivo. O valor do componente proporcional segue um comportamento semelhante ao do caso do controlador Proporcional. O valor do componente derivativo acompanha as tendências de erro, independente do valor do erro atual. Inicialmente, a tendência do erro é de assumir valores positivos cada vez maiores. Dessa forma, o valor do componente é negativo. A redução do valor absoluto do erro, que ocorre na transição entre tempo = 3 e tempo = 4, representa uma tendência a atrasar, e o componente derivativo responde com um tamanho de atuação positivo.

No caso deste controlador, o tamanho de atuação final é a soma dos valores dos componentes proporcional e derivativo. Dessa forma, dependendo do comportamento do erro ao longo do tempo, um componente pode compensar a ação do outro componente. Um exemplo é o que ocorre entre tempo = 3 e tempo = 5, em que o componente proporcional continua a reduzir a cota, baseado no último valor de erro observado, mas o componente derivativo, baseado na variação negativa, compensa a redução com um aumento proporcional à tendência de atraso.

Tabela 5.2: Efeito do tamanho de atuação do controlador Proporcional-Derivativo na cota

Tempo	Erro	Proporcional	Derivativo	Tamanho de atuação	Próxima Cota (%)
1	0	0	0	0	70
2	0,01	-0,02	-0,05	-0,07	63
3	0,02	-0,04	-0,05	-0,09	54
4	0,01	-0,02	0,05	0,03	57
5	0	0	0,05	0,05	62
6	-0,01	0,02	0,05	0,07	69
7	0,01	-0,02	-0,1	-0,12	57

5.1.4 Proporcional-Integrativo-Derivativo (PID)

A quantidade de recursos adicionada ou removida é uma função dos seguintes valores:

- Erro atual
- Variação do erro (a diferença entre o valor atual e o último recebido)
- Erro acumulado
- Ganho proporcional
- Ganho derivativo
- Ganho integrativo

O tamanho da atuação é calculado como mostrado a seguir:

$$c_p = -1 * g_p * e_i \tag{5.6}$$

$$c_d = -1 * g_d * (e_i - e_{i-1}) \tag{5.7}$$

$$c_i = -1 * g_i * (e_1 + e_2 + ... + e_{i-1} + e_i) \tag{5.8}$$

$$a = c_p + c_d + c_i \tag{5.9}$$

Onde, c_p é o componente proporcional, c_d é o componente derivativo, c_i é o componente integrativo e a é o tamanho da atuação. g_p é o ganho proporcional, g_d é o ganho derivativo,

g_i é o ganho integrativo e e_t é o erro no tempo t. Um exemplo de funcionamento do Controlador PID é apresentado na Figura 5.4. Pode-se observar como a adição do componente integrativo, a soma dos erros ao longo do tempo, fez com que a reação do Controlador PID aos valores negativos de erro fosse tardia, em comparação aos outros controladores. O valor do componente integrativo acumulado durante a fase positiva do erro é alto o suficiente para garantir que o Controlador PID continue tentando remover recursos, mesmo observando os valores negativos da segunda fase.

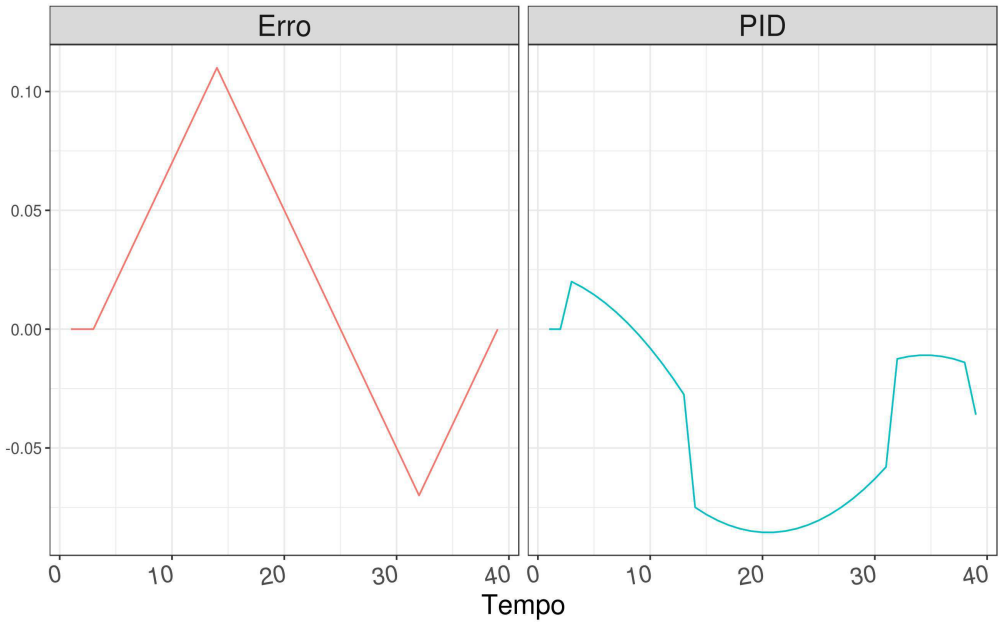


Figura 5.4: Exemplo do comportamento do Controlador PID

Um exemplo de como o tamanho de atuação calculado pelo algoritmo afeta a cota é apresentado na Tabela 5.3. O valor inicial da cota é de 70%, o ganho proporcional é igual a 2, o ganho derivativo é igual a 5 e o ganho integrativo é igual a 1. Os valores de erro neste exemplo são iguais aos valores de erro no exemplo do controlador Proporcional-Derivativo. Pode-se observar como o componente integrativo, influenciado pelos valores de erro positivos iniciais, permanece com valores negativos até o final, fazendo com que os valores de cota sejam menores que os do caso Proporcional-Derivativo.

Baseado na análise apresentada, há indícios de que, em um contexto em que os valores de erro podem persistir positivos ou negativos por um tempo significativo, a utilização do componente integrativo e, consequentemente, do Controlador PID, não é recomendada.

Tabela 5.3: Efeito do tamanho de atuação do controlador PID na cota

Tempo	Erro	Proporcional	Derivativo	Integrativo	Atuação	Próxima Cota (%)
1	0	0	0	0	0	70
2	0,01	-0,02	-0,05	-0,01	-0,08	62
3	0,02	-0,04	-0,05	-0,03	-0,12	50
4	0,01	-0,02	0,05	-0,04	-0,01	49
5	0	0	0,05	-0,04	0,01	50
6	-0,01	0,02	0,05	-0,03	0,04	54
7	0,01	-0,02	-0,1	-0,04	-0,16	38

5.2 Métodos de atuação

Como discutido anteriormente, há uma grande quantidade de métodos possíveis. Considerando provisionamento vertical de recursos, atuação em CPU e I/O de disco são as opções mais utilizadas.

Em plataformas de computação na nuvem é comum haver dependência entre as quantidades de recursos de CPU e I/O de disco fornecidas [1][3]. Ou seja, uma maior quantidade de recursos de CPU alocados corresponde a uma maior quantidade de recursos de I/O de disco alocados. Esse tipo de balanceamento é utilizado com o objetivo de manter equilibradas a capacidade de manipulação de dados, na forma de recursos de I/O de disco, e a capacidade de processar dados, na forma de recursos de CPU. Para certos tipos de aplicação este tipo de balanceamento é benéfico. Adicionalmente, este balanceamento facilita o gerenciamento da infraestrutura, evitando que recursos fiquem ociosos.

Utilizando como referência essa correspondência de recursos, foram escolhidas duas opções de atuação para análise. A primeira é um caso base, atuação apenas em CPU. A segunda é um caso mais realista, em que há atuação em ambos CPU e I/O de disco. A implementação de atuador considerada neste trabalho atua em infraestruturas cujo hipervisor é KVM.

5.2.1 CPU

Neste trabalho, o atuador em CPU fixa a cota de I/O em um determinado nível no início da execução. Durante a execução ele modifica apenas a quantidade de tempo de CPU alocado.

A cota de CPU para a máquina virtual é a fração da quantidade máxima de tempo de CPU alocado para a máquina virtual que será disponibilizada na execução.

Em KVM, o controle da cota é realizado a partir do ajuste de dois parâmetros: `vcpu_quota` e `vcpu_period`. A cota de CPU da máquina virtual é a razão entre o tempo de CPU disponibilizado (`vcpu_quota`) em um período considerado (`vcpu_period`). Dessa forma, fixado um valor de `vcpu_period`, pode-se controlar a cota alterando o valor de `vcpu_quota`. Exemplo: se a cota de CPU desejada é 50% e o `vcpu_period` é de 100000 milissegundos, deve-se alterar o valor de `vcpu_quota` para 50000 milissegundos, uma vez que $50000/100000 = 50\%$. Interpreta-se que, de cada 100000 milissegundos de CPU disponibilizados para a máquina virtual, apenas 50000 serão utilizados.

5.2.2 CPU + I/O

Modifica cota de ambos CPU e I/O durante a execução. A atuação em CPU é feita de forma semelhante ao do atuador de CPU. Quanto a I/O, em KVM, há as seguintes opções de atuação disponíveis:

- Cota de iops: para leitura e escrita. Em KVM, a cota de iops é controlada a partir da manipulação de dois parâmetros: `iops_reference` e `total_iops_sec`. A cota é a razão entre o valor alocado de iops (`total_iops_sec`) e o valor máximo (`iops_reference`).

- Cota de bytes por segundo: para leitura e escrita. Em KVM, a cota de bytes por segundo é controlada a partir da manipulação de dois parâmetros: `bs_reference` e `total_bs_sec`. A cota é a razão entre o valor alocado de bytes por segundo (`total_bs_sec`) e o valor máximo (`bs_reference`).

Fixados valores de `iops_reference` e `bs_reference`, pode-se controlar a cota alterando o valor de `total_iops_sec` e `total_bs_sec`, de forma análoga à atuação em CPU.

No restante do texto, o termo "CAP"será utilizado como sinônimo de cota. Por exemplo, CAP de 50% de CPU é o mesmo que uma cota de 50% de CPU.

5.3 Configuração

5.3.1 Referências para atuação em I/O

Um disco rígido típico de servidor tem vazão de 75 a 200 IOPS e de 100 a 200 MB/s. Considerando uma configuração com RAID 10, com particionamento para aumentar a velocidade e replicação para garantir disponibilidade, obtemos os valores de vazão de 150 a 400 IOPS e 200 a 400 MB/s.

Os servidores utilizados têm capacidade para 8 máquinas virtuais, em média. Obtemos, então, 19 a 50 IOPS e 25 a 50 MB/s por máquina virtual. A utilização de várias máquinas virtuais compartilhando discos diminui a banda útil (escritas curtas e concorrentes). No entanto, as máquinas não estão constantemente usando toda a capacidade disponível, sendo comum a superalocação, ou *overcommitting*, de recursos. Neste caso, a soma da quantidade de recursos alocados às máquinas virtuais executando em um nó de processamento é maior do que a quantidade real de recursos.

Seguindo este raciocínio foram utilizados os valores de referência de 30 MB/s e 50 IOPS, adotados pelo Laboratório de Sistemas Distribuídos[1] (LSD) da Universidade Federal de Campina Grande (UFCG), onde a pesquisa foi realizada, como os valores de `bs_reference` e `iops_reference`, respectivamente.

5.3.2 CAPs mínimo e máximo

A seguinte investigação foi realizada com o objetivo de entender como o valor do nível de recursos afeta o desempenho de aplicações e obter conhecimento sobre como definir limites razoáveis de utilização.

Inicialmente, investigou-se o efeito do CAP de CPU. Executou-se uma aplicação limitada por CPU com diferentes níveis de CAP de CPU e coletou-se o tempo de execução. A aplicação utilizada foi uma implementação de fatorial de 100. Cada ensaio consistiu da execução da aplicação em uma máquina virtual cujo uso de CPU era limitado pelo CAP associado ao ensaio e da medição do tempo total de execução. Foram realizadas 30 repetições de ensaios com cada nível considerado. Os resultados são apresentados na Figura 5.5.

[1] https://www.lsd.ufcg.edu.br

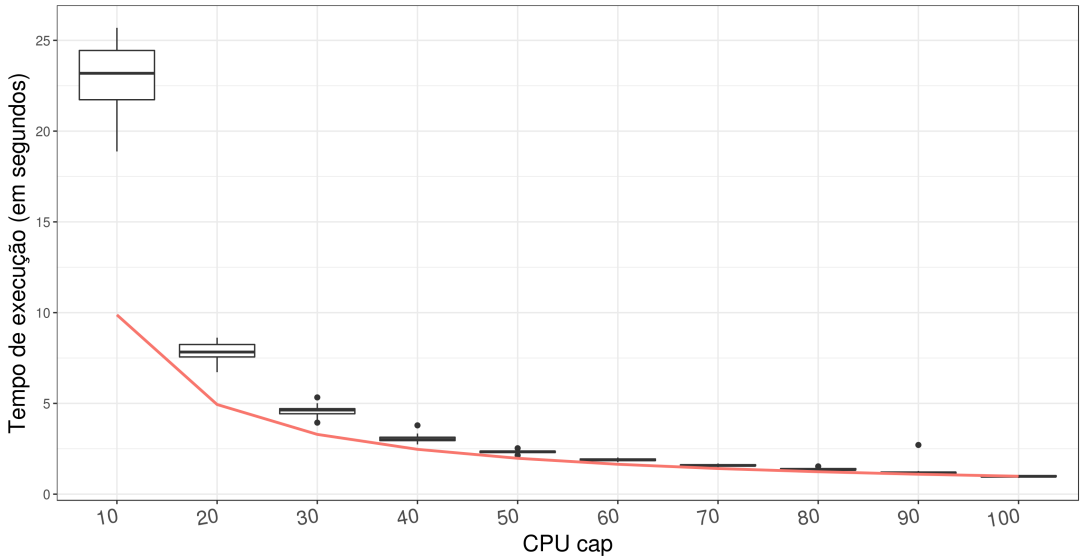


Figura 5.5: CPU CAP $\times$ Tempo ajustado

Em uma situação ideal, o efeito de uma redução do CAP seria um aumento proporcional no tempo de execução da aplicação. Por exemplo, o efeito de uma redução de 100% para 50% no nível de recursos seria dobrar o tempo de execução da aplicação. Deste modo, estabelecido o melhor tempo de execução possível (CAP = 100%), pode-se determinar o tempo esperado para os outros valores de CAP. Sejam t_{e_x} o tempo esperado para o valor de CAP = x e t_{r_x} o tempo real para o valor de CAP = x, o valor do tempo esperado para um CAP x pode ser calculado como apresentado a seguir.

$$t_{e_x} = t_{r_{100}} * (100/x) \tag{5.10}$$

Os valores esperados para os CAPs são apresentados na Figura 5.5 como a linha vermelha. Observa-se que para valores de CAP mais baixos, os valores de tempo de execução começam a ser maiores que o esperado. O valor de CAP que é o limiar das perdas maiores que o esperado pode ser determinado a partir de uma análise das perdas obtidas. Considera-se nesta análise duas variáveis: perda esperada e perda real.

Sejam p_{e_x} a perda esperada para o valor de CAP = x, p_{r_x} a perda real para o valor de CAP = x, t_{e_x} o tempo esperado para o valor de CAP = x e t_{r_x} o tempo real para o valor de CAP = x, podemos determinar perda esperada e perda real para os valores de CAP através do cálculo apresentado a seguir.

$$p_{e_x} = (t_{e_x} - t_{r_{100}})/t_{r_{100}} \tag{5.11}$$

$$p_{r_x} = (t_{r_x} - t_{r_{100}})/t_{r_{100}} \tag{5.12}$$

A Figura 5.6 apresenta a perda percentual esperada e a perda percentual real, para valores de CAP maiores ou iguais a 30%. Neste caso, ocorre o padrão de crescimento da diferença entre a perda real em relação à perda esperada, mais notável para valores de CAP menores que 50%.

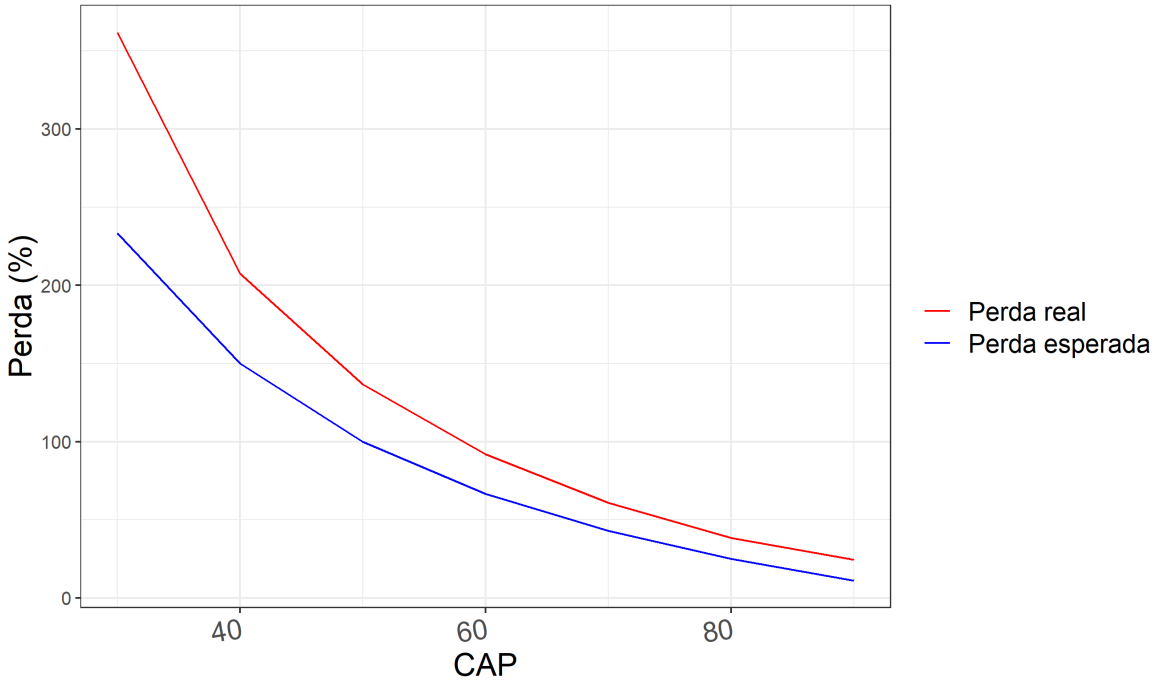


Figura 5.6: Perda esperada e perda real

Os valores da diferença entre a perda real e a perda esperada são apresentados na Figura 5.7. Pode-se observar que entre 90% e 80% o crescimento é nulo, que entre 80% e 50% o padrão de crescimento é aproximadamente linear e que para valores menores que 50% há uma quebra do padrão linear de crescimento. Ou seja, ocorre uma perda desproporcional de desempenho. Baseado na perda de desempenho observada, nesta pesquisa estabeleceu-se 50% como valor mínimo de CAP de CPU.

Uma investigação equivalente foi realizada para CAP de I/O. Executou-se uma aplicação limitada por I/O com diferentes níveis de CAP de I/O e coletou-se o tempo de execução.

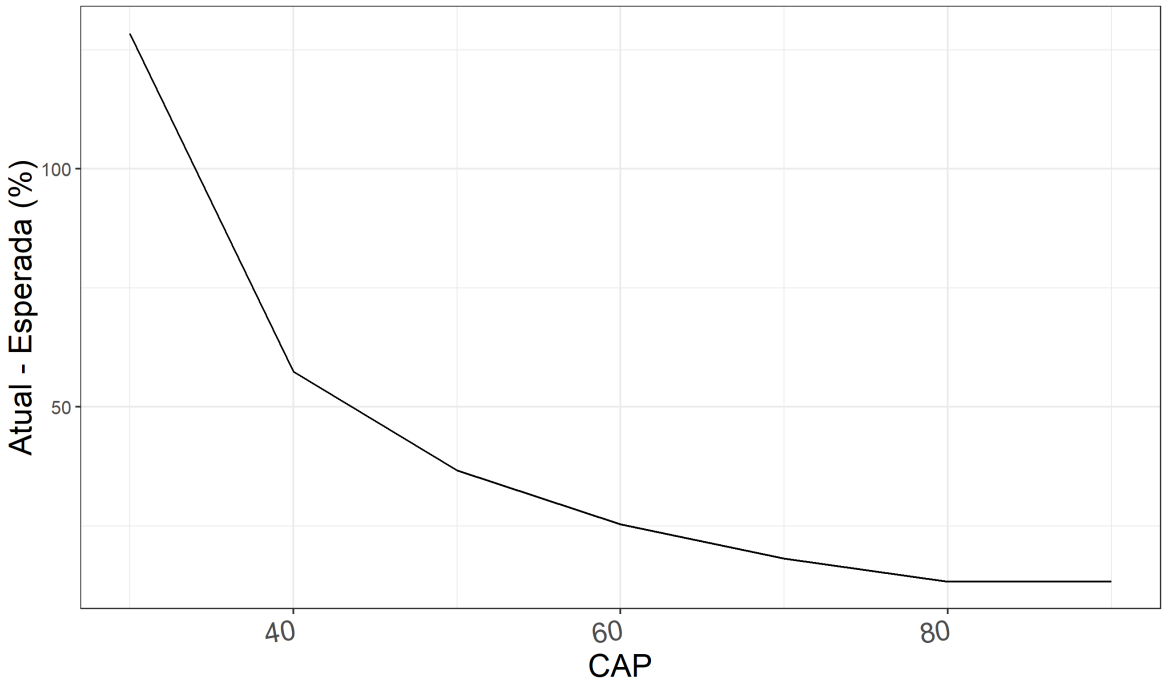


Figura 5.7: Diferença entre a perda esperada e a perda real

Cada ensaio consistiu da execução da aplicação em uma máquina virtual cujo uso de I/O era limitado pelo CAP associado ao ensaio e da medição do tempo total de execução. Para todos os níveis apresentados foram realizadas 10 repetições, exceto o caso em que CAP = 10%, em que foram realizadas 2 repetições. Os resultados são apresentados na Figura 5.8.

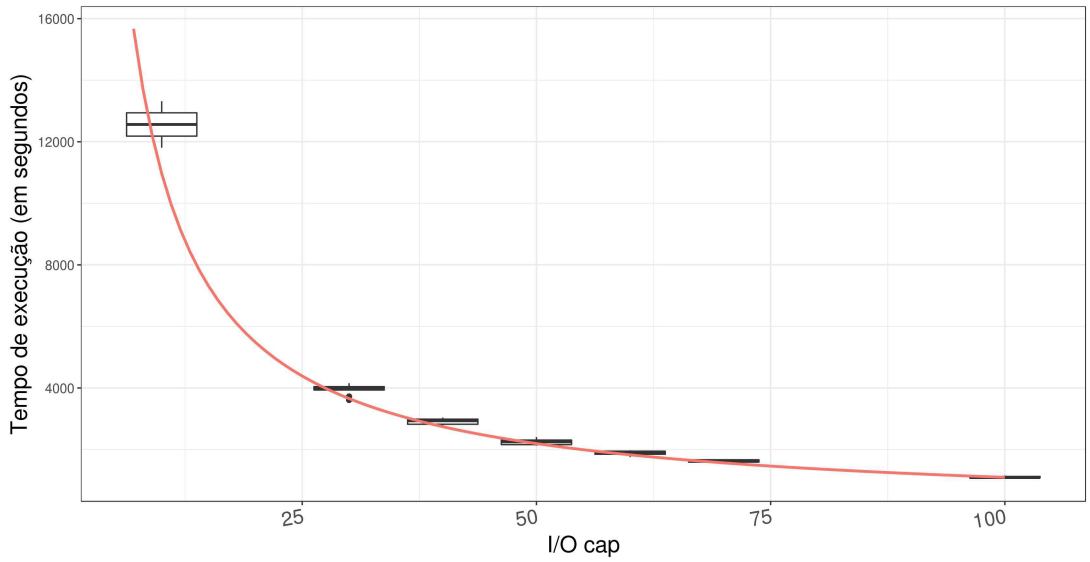


Figura 5.8: I/O CAP × Tempo ajustado

Novamente, a linha vermelha na Figura 5.8 indica o tempo esperado. No caso de CAP de I/O, as perdas são menores do que no caso do CAP de CPU. No entanto, uma vez que nesta pesquisa a atuação em I/O de disco ocorre de forma que a cota de disco é igual à cota de CPU, o CAP mínimo de I/O também é 50%.

Capítulo 6

Metodologia de Avaliação

6.1 Introdução

O objetivo do estudo apresentado neste capítulo é avaliar a viabilidade da utilização dos algoritmos de controle e dos métodos de atuação apresentados nos capítulos anteriores. Deseja-se que os controladores e atuadores sejam capazes de garantir qualidade de serviço em termos de cumprimento de prazos de execução de aplicações executando em ambientes de nuvem realistas.

A principal métrica observada na avaliação foi tempo de execução. A qualidade dos controladores e atuadores está relacionada à sua capacidade de fazer a aplicação terminar dentro do prazo estabelecido. Outras propriedades desejáveis são baixa dispersão no tempo de execução e suavidade na alocação de recursos. A baixa dispersão indica que a combinação de controlador e atuador é mais confiável, uma vez que é mais fácil prever o resultado de suas ações. Uma alocação suave de recursos causa menor perturbação no ambiente sobre o qual as aplicações executam.

A metodologia utilizada consistiu da execução de aplicações e *microbenchmarks* utilizando o Asperathos sobre uma infraestrutura de nuvem, com diferentes opções de algoritmos de controle e métodos de atuação. Durante a execução foram coletados dados quanto ao uso de CPU, escrita, leitura e aos níveis de alocação de recursos. Após o término da execução, o tempo de execução era coletado.

6.2 *Design* experimental

6.2.1 Fatores e níveis

O experimento segue o *design* fatorial completo [23]. Neste *design*, são observadas todas as combinações de níveis dos fatores considerados. No caso deste experimento, são considerados dois fatores: o algoritmo de controle utilizado para decidir a quantidade de recursos alocados à aplicação ao longo da execução; e o método de atuação, utilizado para ajustar a quantidade de recursos alocados à aplicação. Os níveis do fator Algoritmo de controle são apresentados na Tabela 6.1 e os níveis do fator Método de atuação são apresentados na Tabela 6.2.

Tabela 6.1: Níveis considerados para o fator Algoritmo de controle

Identificador do nível	Nome do algoritmo de controle
1	Mínimo-máximo
2	Proporcional
3	Proporcional-derivativo
4	PID

Tabela 6.2: Níveis considerados para o fator Método de atuação

Identificador do nível	Nome do método de atuação
1	Atuação em CPU apenas
2	Atuação em CPU e I/O

Especificamente no caso dos *microbenchmarks*, em que apenas um tipo de recurso é utilizado, apenas o método de atuação correspondente ao recurso é utilizado. Nas observações com *microbenchmark* CPU, apenas o método de atuação em CPU foi utilizado. Nas observações com o *microbenchmark* I/O, apenas o método de atuação em CPU e I/O foi utilizado. No caso da aplicação, ambos os níveis foram considerados.

6.2.2 Fluxo de execução

O fluxo de execução de cada ensaio é apresentado na Figura 6.1. Inicialmente são escolhidos a aplicação e o tratamento a utilizar. Neste contexto, um tratamento é uma combinação de um controlador e de um atuador. Em seguida, a aplicação é submetida ao gerente com a configuração de provisionamento. Realiza-se a verificação periódica de término da aplicação. Enquanto ela ainda estiver executando, informações quanto ao seu uso de recursos são coletadas. A coleta do tempo de execução da aplicação, disparada quando detecta-se seu término, encerra a coleta de dados.

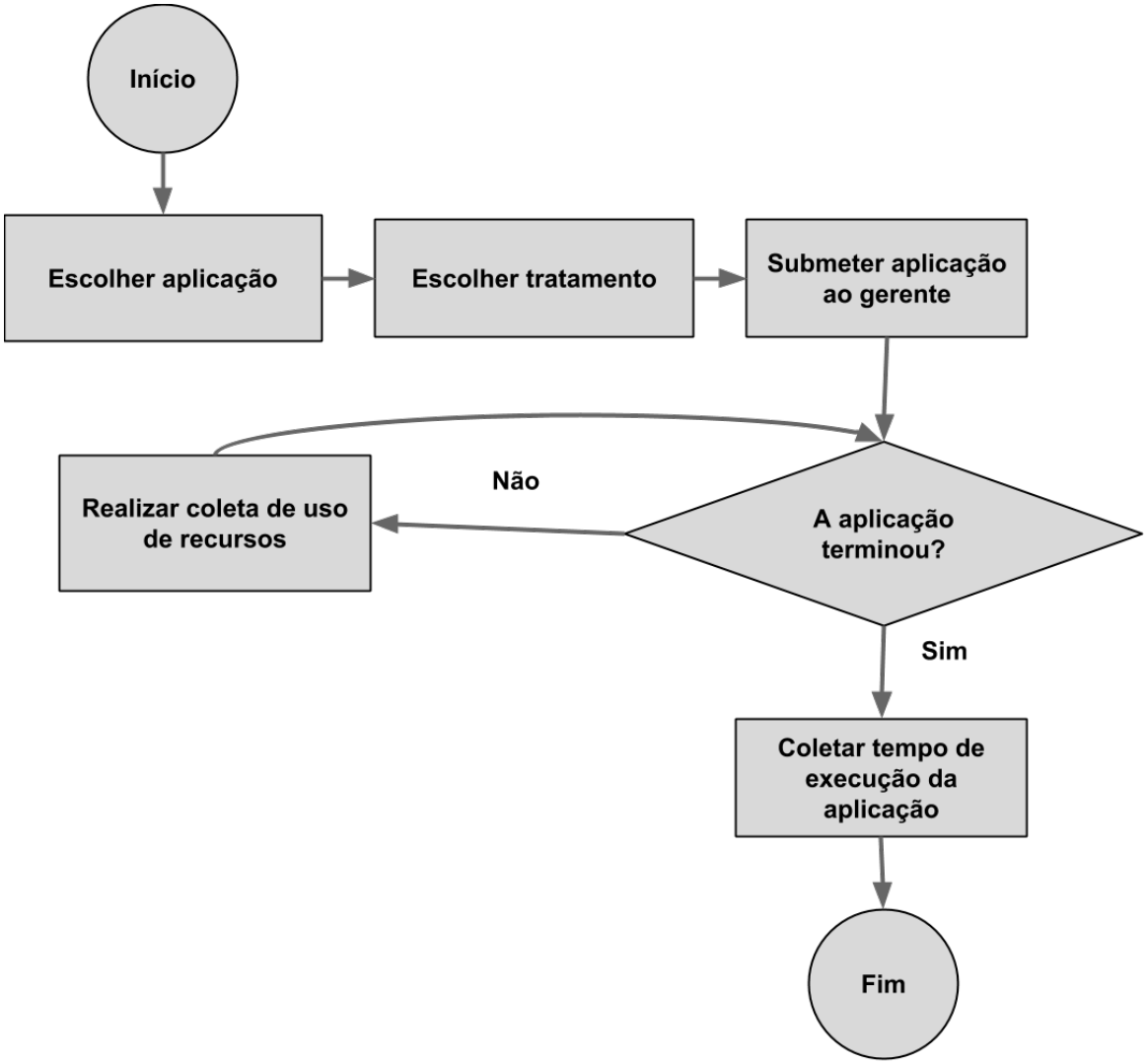


Figura 6.1: Fluxo de execução do experimento

6.2.3 Definição de prazos de execução

A definição do prazo de execução para os *microbenchmarks* e para a aplicação foi realizada através de suas execuções e da coleta de seus tempos de execução. Foram realizadas 10 execuções para cada caso, com os mesmos dados utilizados no experimento. Uma vez que o objetivo do experimento era avaliar a capacidade dos controladores de garantir prazo de execução, foi utilizado um valor menor (90% da média das execuções) que o esperado como prazo. Utilizou-se como nível de recursos o valor de 70%, durante toda a execução. Uma vez que o valor mínimo do cap é de 50%, como definido anteriormente, e o valor máximo é de 100%, 70% é um valor que fornece ao controlador uma faixa da valores para adicionar e remover recursos.

6.2.4 Configuração do controlador e do atuador

Como cap inicial para as execuções foi utilizado o valor de 70%, igual ao valor utilizado na definição dos prazos de execução.

Quanto ao intervalo de monitoramento, caso o tempo de execução esperado para a aplicação seja baixo, utilizar um valor alto de intervalo pode causar a perda de informações importantes sobre o progresso. Nesses casos, é possível que o controlador não fique ciente de mudanças bruscas no progresso da aplicação. Para minimizar a chance de perdas, foi utilizado o valor mínimo de 10 segundos, que é alto o suficiente para não sobrecarregar o serviço de divulgação de métricas. Na implementação de controlador utilizada, este é o valor mínimo de espera entre verificações de erro de progresso das aplicações.

Como valor para os ganhos que fazem parte do cálculo do tamanho da atuação, foram utilizados inicialmente valores triviais, como apresentado na Tabela 6.3.

Tabela 6.3: Valores dos ganhos para cada opção de controlador

	Ganho Proporcional	Ganho Derivativo	Ganho Integrativo
Proporcional	1	Não aplicável	Não aplicável
Proporcional-Derivativo	1	1	Não aplicável
PID	1	1	1

No caso do controlador Min-Max, foram utilizados valores de `gatilho-baixo` e

`gatilho-cima` iguais a zero. Desta forma, quando o erro é diferente de zero, o controlador age na infraestrutura.

6.2.5 Coleta de dados

O tempo de execução das aplicações foi coletado utilizando a API fornecida pelo Gerente. Essa API oferece informações como tempo de início da execução, tempo de execução e estado da aplicação.

A coleta do uso de recursos, como uso de CPU, dados escritos e lidos, foi realizada durante toda a execução das aplicações. Essa coleta foi realizada através do acesso aos nós de computação onde as máquinas virtuais estavam alocadas, utilizando SSH, e da consulta ao hipervisor, neste caso, KVM.

6.3 Infraestrutura

Os experimentos foram realizados na nuvem de produção do Laboratório de Sistemas Distribuídos (LSD) na Universidade Federal de Campina Grande (UFCG). Foram utilizados 3 servidores, dois Dell PowerEdge R410 e um Dell PowerEdge R420, conectados através de uma rede gigabit e utilizando Ceph[1] para armazenamento compartilhado. Todos os servidores executavam Ubuntu Linux. A configuração desses nós é apresentada na Tabela 6.4.

Tabela 6.4: Configuração dos servidores utilizados

Servidor	Modelo	Processador (Intel Xeon)	Núcleos	Memória (GB)	Kernel
1	R410	X5675 3.07 GHz	6	32	4.4.0-112
2	R410	X5675 3.07 GHz	6	32	4.4.0-98
3	R420	E5-2407 2.20 GHz	4	20	4.4.0-98

Os servidores 1 e 2 foram utilizados para alocar máquinas virtuais para execução das aplicações e o servidor 3 para alocação dos componentes do Asperathos e para gerenciamento dos experimentos. As máquinas virtuais criadas durante os experimentos possuíam 2 vCPUs e 4 GB de memória. Esses dois valores eram fixos durante toda a execução. O *cluster* Spark utilizado possuía 1 nó mestre e 12 nós escravos.

[1] https://ceph.com

6.4 *Microbenchmarks*

No processo de avaliação dos algoritmos de controle foram utilizados dois *microbenchmarks* e uma aplicação. Os *microbenchmarks* são apresentados nesta seção e a aplicação é apresentada na seção seguinte. Os *microbenchmarks* e a aplicação foram executados 10 vezes e foram coletadas informações sobre o progresso, o uso de CPU, a leitura e escrita de dados.

6.4.1 CPU

O *microbenchmark* de CPU calcula fatoriais seguindo o roteiro de execução recebido como argumento. O roteiro de execução contém o argumento da função fatorial a ser calculada e a quantidade de execuções da função a realizar. Um exemplo de comando de execução é apresentado no Código 6.1. Neste exemplo, o *microbenchmark* realiza 10 execuções de fatorial de 1000 e 2 execuções de fatorial de 10000.

Código Fonte 6.1: Exemplo de execução do *microbenchmark* de CPU

```
1 $ python cpu_benchmark.py 10 1000 2 10000
```

Quanto ao monitoramento do progresso, cada execução de fatorial corresponde a uma tarefa a ser realizada. O progresso da execução é a razão entre a quantidade de tarefas já terminadas e o número total de tarefas. Por exemplo, uma execução iniciada a partir do comando apresentado no Código 6.1 apresentaria 12 tarefas. Nesta execução, 3 tarefas completadas corresponderiam a um progresso de 25%.

O *microbenchmark* calcula o progresso e registra seu valor em um arquivo de *log* após o término de cada tarefa. Esse arquivo é, então, consultado pelo Monitorador, que disponibiliza essa informação para o Controlador. Através de modificações no roteiro de execução é possível alterar o padrão de progresso do *microbenchmark*, gerando variações de progresso que oferecem oportunidades de atuação para o Controlador.

Progresso

Um exemplo de progresso do *microbenchmark*, utilizando como entrada apenas um argumento de função fatorial, é apresentado na Figura 6.2. Pode-se observar na Figura que o progresso do *microbenchmark*, neste caso, é completamente linear.

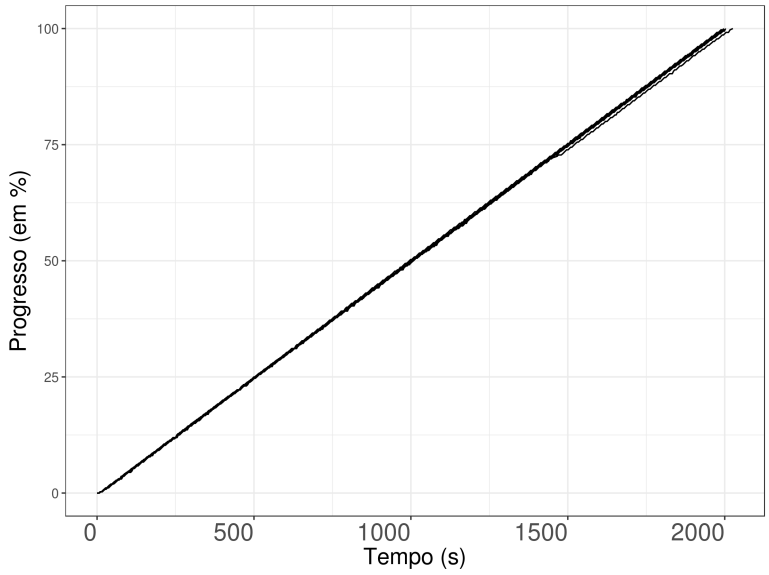


Figura 6.2: Progresso do benchmark CPU, em percentual de tarefas, ao longo da execução

O progresso do *microbenchmark* CPU pode apresentar padrão não-linear dependendo da carga de trabalho, ou a sequência de argumentos de função fatorial, passados como entrada.

Uso de CPU

O uso de CPU do *microbenchmark* ao longo do tempo é apresentado na Figura 6.3. O *microbenchmark*, como esperado, utiliza todos os recursos de CPU durante toda a execução.

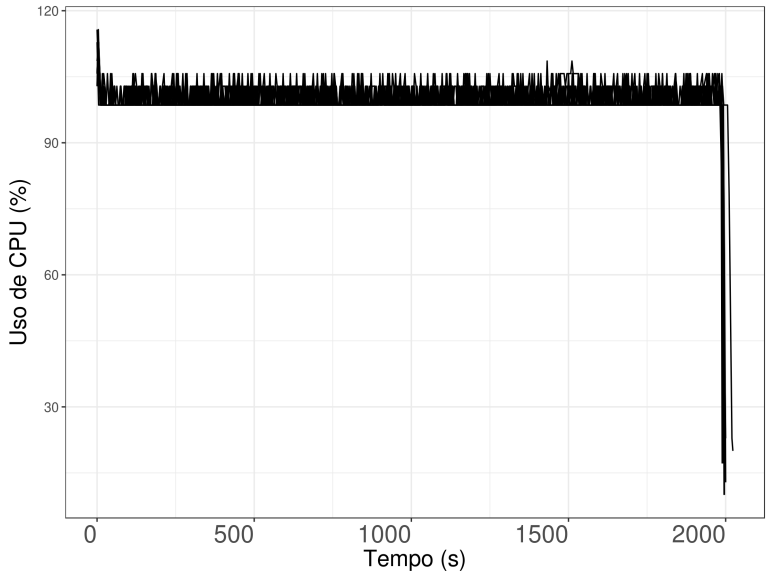


Figura 6.3: Uso de CPU do *microbenchmark* CPU ao longo da execução

6.4.2 I/O

O *microbenchmark* de I/O escreve e lê arquivos seguindo o roteiro de execução recebido como argumento. O roteiro de execução contém o tamanho e a quantidade de arquivos considerados. Na primeira fase o *microbenchmark* cria os arquivos e escreve os dados. Na segunda fase, o *microbenchmark* lê os dados escritos na primeira fase. Um exemplo de comando de execução é apresentado no Código 6.2. Neste exemplo, o *microbenchmark* escreve e lê 30 arquivos de 100000 KB e 3 arquivos de 300000 KB.

Código Fonte 6.2: Exemplo de execução do *microbenchmark* de I/O

```
1 $ python io_benchmark.py 100000 30 300000 3
```

Quanto ao acompanhamento do progresso, cada escrita ou leitura de arquivo corresponde a uma tarefa a ser realizada. De forma análoga ao progresso no *microbenchmark* CPU, o progresso de uma execução do *microbenchmark* I/O é a razão entre a quantidade de tarefas já terminadas e o número total de tarefas. Uma execução iniciada a partir do comando apresentado no Código 6.2 apresentaria 66 tarefas, 33 tarefas de leitura e 33 tarefas de escrita. Nesta execução, 5 tarefas completadas corresponderiam a um progresso de aproximadamente 8%. O *microbenchmark* registra o progresso em um arquivo de *log* após o término de cada tarefa, assim como no *microbenchmark* de CPU.

Progresso

O progresso da aplicação ao longo do tempo é apresentado na Figura 6.4. Na Figura é fácil observar as duas fases do *microbenchmark*: primeiro a fase de escrita, mais rápida, depois a fase de leitura, mais lenta. Em ambas as fases o progresso é linear.

Leitura de dados

A quantidade de dados lidos pela aplicação ao longo do tempo é apresentada na Figura 6.5. Durante toda a primeira fase, praticamente não ocorre leitura, que é concentrada na segunda fase.

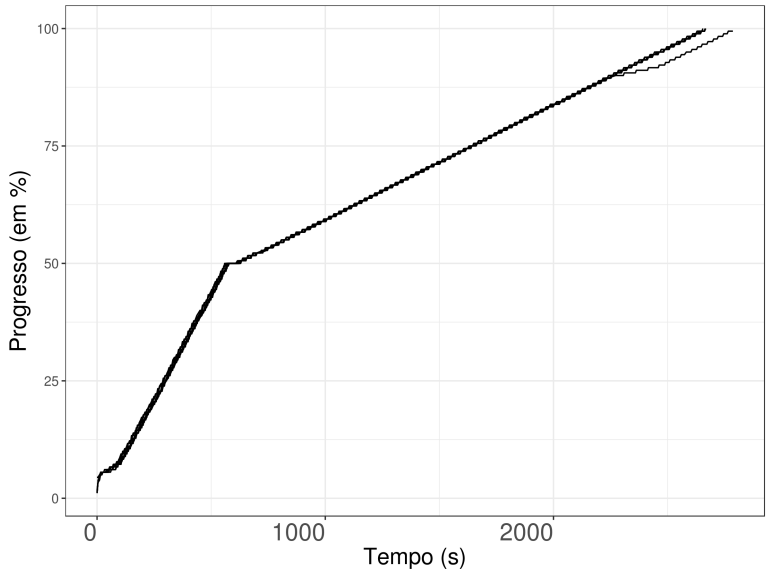


Figura 6.4: Progresso do *microbenchmark* I/O ao longo da execução

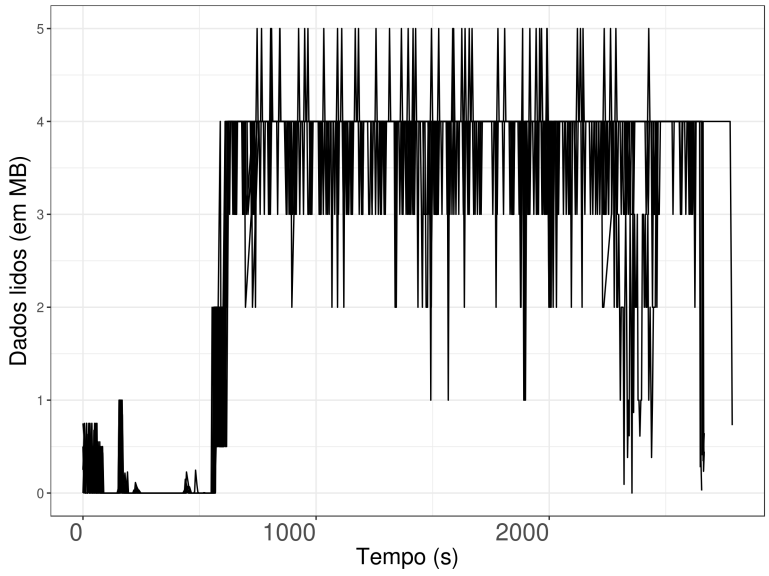


Figura 6.5: Leitura de dados do *microbenchmark* I/O ao longo da execução

Escrita de dados

A quantidade de dados escritos pela aplicação ao longo do tempo é apresentada na Figura 6.6. Toda a escrita é concentrada no começo da execução.

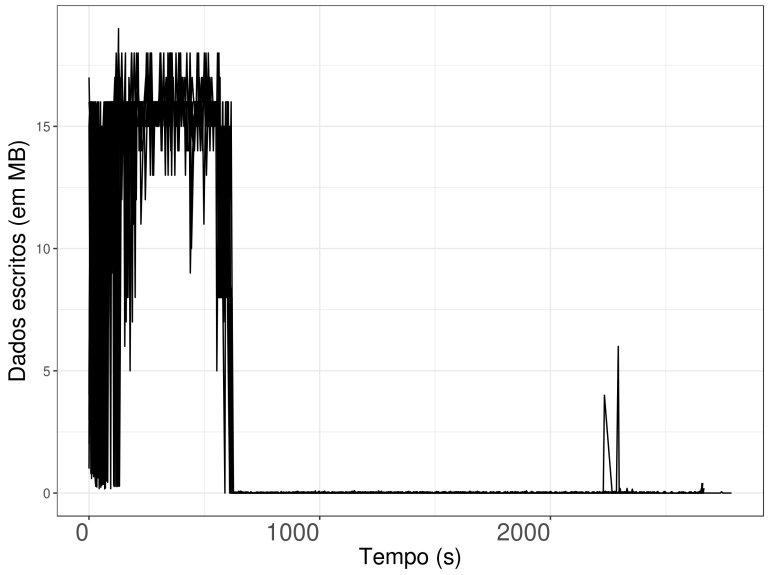


Figura 6.6: Escrita de dados do *microbenchmark* I/O ao longo da execução

6.5 Aplicação avaliada: EMaaS

O EMaaS [20](*Entity Matching as a Service*) implementa um algoritmo de comparação de entidades, utilizado no contexto de garantia de qualidade de dados em *Big Data*, para execução em infraestruturas Spark.

Para a apresentação do uso de recursos pela aplicação foram selecionados dados de 3 máquinas virtuais onde o EMaaS foi executado.

Progresso

A execução de aplicações Spark é estruturada em unidades de trabalho de diferentes níveis hierárquicos, sendo as *tasks*, ou tarefas, as unidades de nível mais baixo. Neste trabalho considera-se o progresso da execução de uma aplicação Spark como a razão entre o número de tarefas Spark completadas e o número total de tarefas Spark. Por exemplo, uma aplicação que possui 20 tarefas, das quais 7 já foram completadas, apresenta progresso de 35%. O progresso do EMaaS ao longo do tempo é apresentado no Gráfico 6.7. Pode-se observar como a aplicação apresenta progresso alternado entre etapas rápidas e etapas lentas.

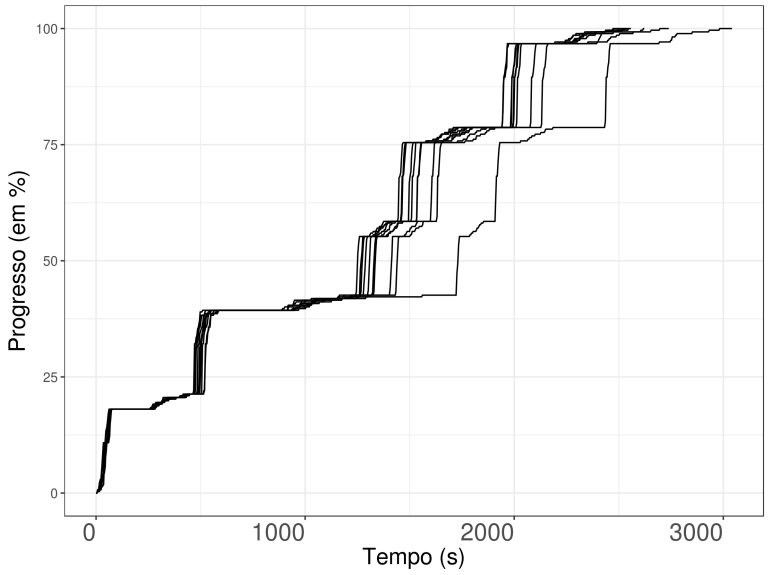


Figura 6.7: Progresso do EMaaS, em percentual de tarefas, ao longo da execução

Uso de CPU

O uso de CPU da aplicação ao longo do tempo é apresentado na Figura 6.8. Nas máquinas virtuais investigadas, o uso de CPU alternou entre 50% e 100%. Esse efeito provavelmente é causado pela variação na quantidade de tarefas executando nas máquinas virtuais, que possuíam 2 CPUs. Como o Spark aloca uma CPU para cada tarefa, nos momentos em que o uso de CPU era igual a 100%, existiam duas tarefas executando. Nos momentos em que o uso era igual a 50%, existia apenas uma tarefa executando.

Leitura de dados

A quantidade de dados lidos pela aplicação ao longo do tempo é apresentada na Figura 6.9. Nas máquinas virtuais investigadas, foram observadas leituras esparsas durante a execução, concentradas em alguns momentos. Uma vez que essas etapas de leitura parecem coincidir com os momentos de progresso mais lento, é possível que atuar em I/O durante essas leituras seja importante para o cumprimento de prazos de execução.

Escrita de dados

A quantidade de dados escritos pela aplicação ao longo do tempo é apresentada no Gráfico 6.10. As escritas, por sua vez, se apresentam mais espalhadas e acontecem durante toda a

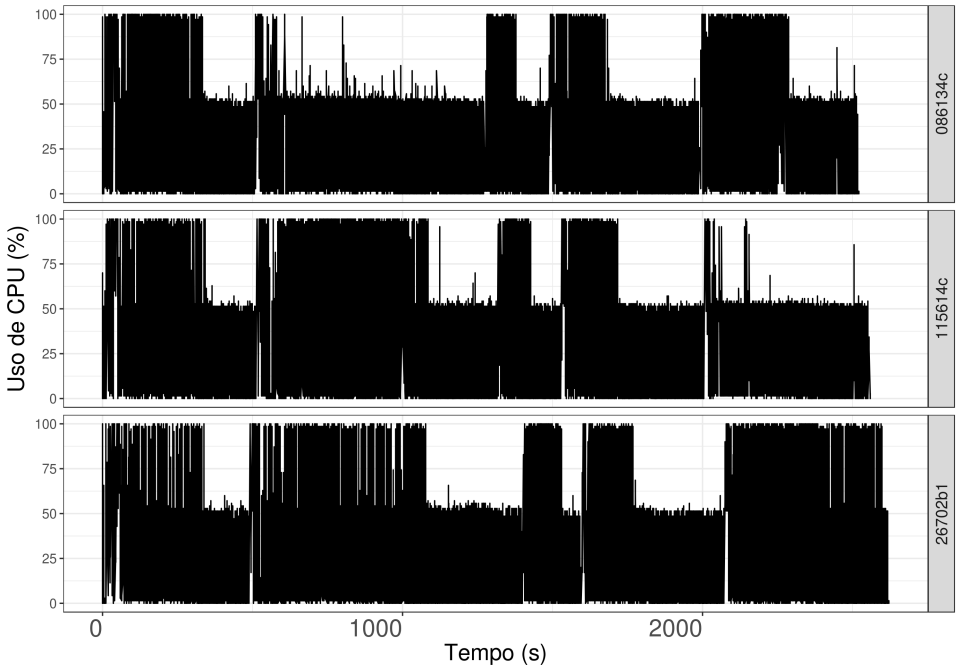


Figura 6.8: Uso de CPU do EMaaS ao longo da execução

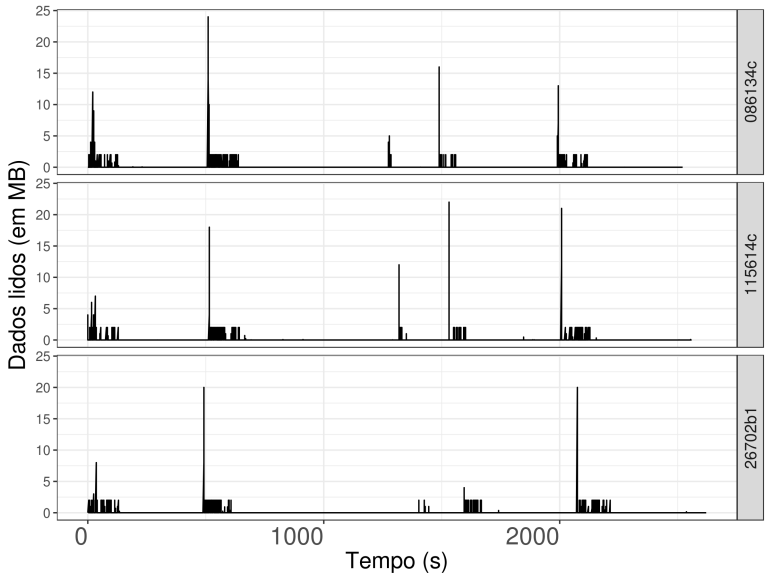


Figura 6.9: Leitura de dados do EMaaS ao longo da execução

execução. Provavelmente essas escritas são resultados parciais da execução do EMaaS.

A partir destas observações conclui-se que o EMaaS é uma aplicação muito limitada por CPU e pouco limitada por I/O de disco. Uma possível hipótese é de que a combinação de atuação em CPU e I/O de disco não resulte em melhoria de desempenho do controlador em comparação à opção que utiliza atuação em CPU apenas.

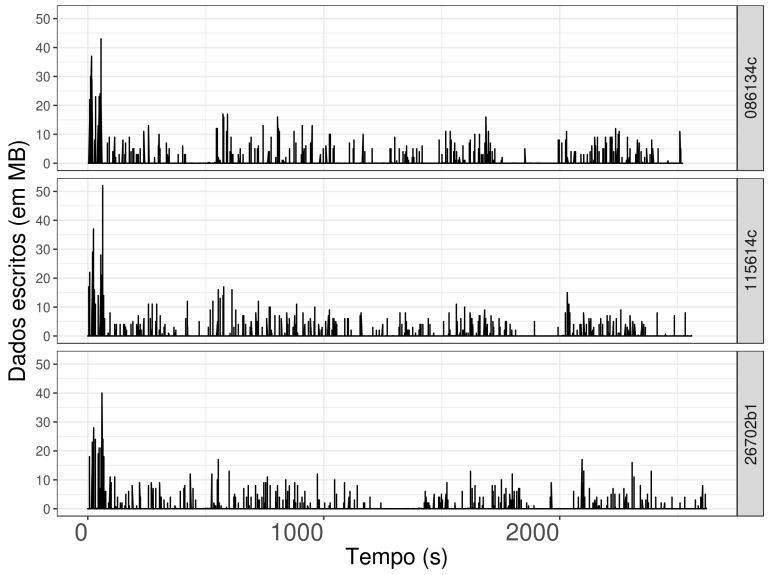


Figura 6.10: Escrita de dados do EMaaS ao longo da execução

6.6 Cargas de trabalho

Na execução dos *microbenchmarks* foram utilizadas cargas de trabalho diferentes das usadas para definição dos prazos de execução. O objetivo era avaliar a capacidade dos controladores de ajustar a quantidade de recursos alocados em caso de variação no progresso. Portanto, à configuração inicial dos *microbenchmarks*, foram adicionadas tarefas mais pesadas. A adição das tarefas mais pesadas seguiu as seguintes orientações:

- As tarefas devem aumentar o tempo de execução do *microbenchmark* de forma notável. No entanto, o aumento não pode ser tão alto que não permita o cumprimento do prazo de execução original a partir da adição de recursos.
- As tarefas devem alterar levemente o padrão de progresso do *microbenchmark*.

O *microbenchmark* CPU, na definição dos prazos de execução, calculava 360 vezes a função fatorial com argumento igual a 70000. O mesmo *microbenchmark*, nas observações com os controladores, além dos cálculos mencionados, calculava 10 vezes a função fatorial com argumento igual a 100000 em dois momentos da execução: após o cálculo de um terço das tarefas com função fatorial com argumento igual a 70000 e após o cálculo de dois terços das tarefas.

O padrão de progresso, com a adição das perturbações no *microbenchmark* CPU, é apresentado na Figura 6.11. Observa-se como a adição de tarefas modificou o padrão de progresso do *microbenchmark*. Inicialmente linear, como apresentado na Figura 6.2, o progresso apresenta fases mais lentas. A média do tempo de execução do *microbenchmark* CPU sem perturbações é de 2060 segundos e, com perturbações, é de 2296 segundos. Portanto, a adição das perturbações aumentou o tempo de execução esperado em aproximadamente 11,5%.

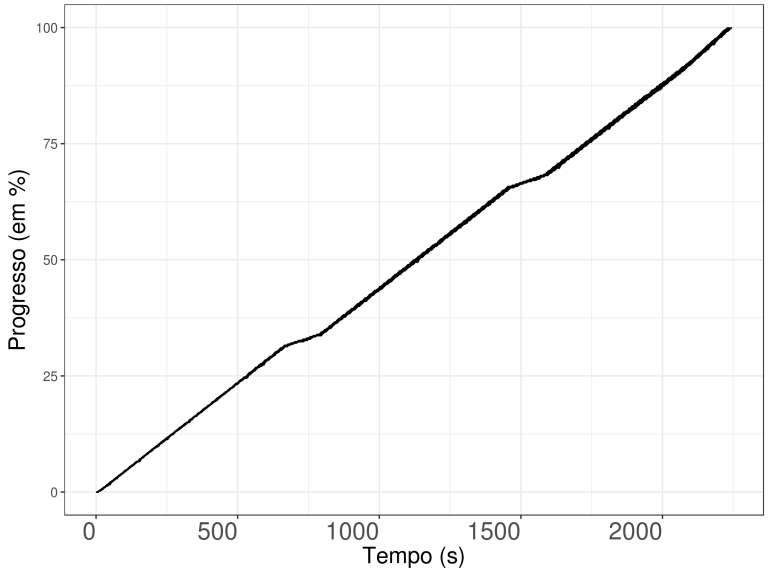


Figura 6.11: Progresso do *microbenchmark* CPU com a carga de trabalho utilizada na avaliação

O *microbenchmark* I/O, na definição dos prazos de execução, escrevia 90 arquivos de 100000 KB e depois lia esses 90 arquivos. O mesmo *microbenchmark*, nas observações com os controladores, além das operações citadas, realizava escrita e leitura de 3 arquivos de 300000 KB, inseridas em quatro momentos da execução. As novas tarefas de escrita eram disparadas após escrita de um terço e dois terços dos arquivos de 100000 KB. As novas tarefas de leitura eram disparadas após a leitura de um terço e dois terços dos arquivos de 100000 KB.

O padrão de progresso, com a adição das perturbações no *microbenchmark* I/O, é apresentado na Figura 6.12. Observa-se como a adição de tarefas modificou o padrão de progresso de ambas as fases de leitura e escrita do *microbenchmark*. A média do tempo de execução do *microbenchmark* I/O sem perturbações é de 2733 segundos e, com perturbações,

é de 3257 segundos. Portanto, a adição das perturbações aumentou o tempo de execução esperado em aproximadamente 19,2%.

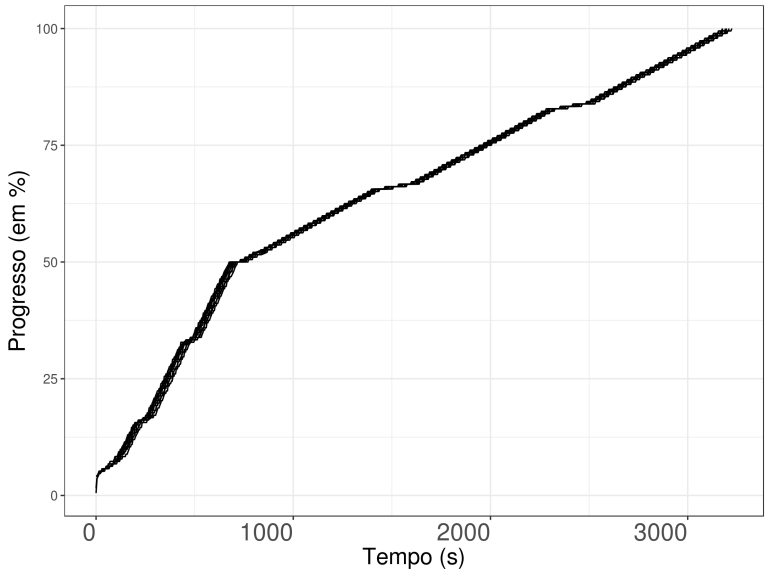


Figura 6.12: Progresso do *microbenchmark* I/O com a carga de trabalho utilizada na avaliação

Capítulo 7

Análise

Neste capítulo são discutidos os resultados obtidos com as execuções dos *microbenchmarks* e da aplicação, seguindo a metodologia apresentada no Capítulo 6. O principal aspecto considerado nesta análise é a capacidade dos mecanismos de provisionamento, na forma da combinação de um controlador e de um atuador, de garantir cumprimento de prazos de execução com uso eficiente de recursos e baixa perturbação no ambiente. A análise de cada caso é estruturada em 2 seções. Na primeira seção discute-se os resultados relacionados a tempos de execução. Na segunda discute-se os resultados relacionados a uso de recursos.

7.1 *Microbenchmark* CPU

7.1.1 Tempo de execução

Nas execuções do *microbenchmark* de CPU foi utilizado apenas atuador de CPU. Os tempos de execução do *microbenchmark* de CPU, utilizando as diferentes opções de controlador, são apresentados na Figura 7.1. Quanto ao cumprimento do prazo de execução estabelecido, pode-se concluir, a partir dos valores de tempo de execução coletados, que os controladores são eficientes em alocar recursos de CPU de forma a fazer o *microbenchmark* terminar no tempo. Adicionalmente, os valores de tempo de execução no caso do controlador Proporcional-Derivativo são mais próximos do prazo. Ou seja, o controlador Proporcional-Derivativo conseguiu alocar recursos de forma mais precisa. Além disso, a dispersão de valores no caso do controlador Proporcional-Derivativo é menor que em todos os outros

casos. Consequentemente, os resultados de suas ações sobre o nível de recursos são mais previsíveis e ele é, portanto, mais confiável.

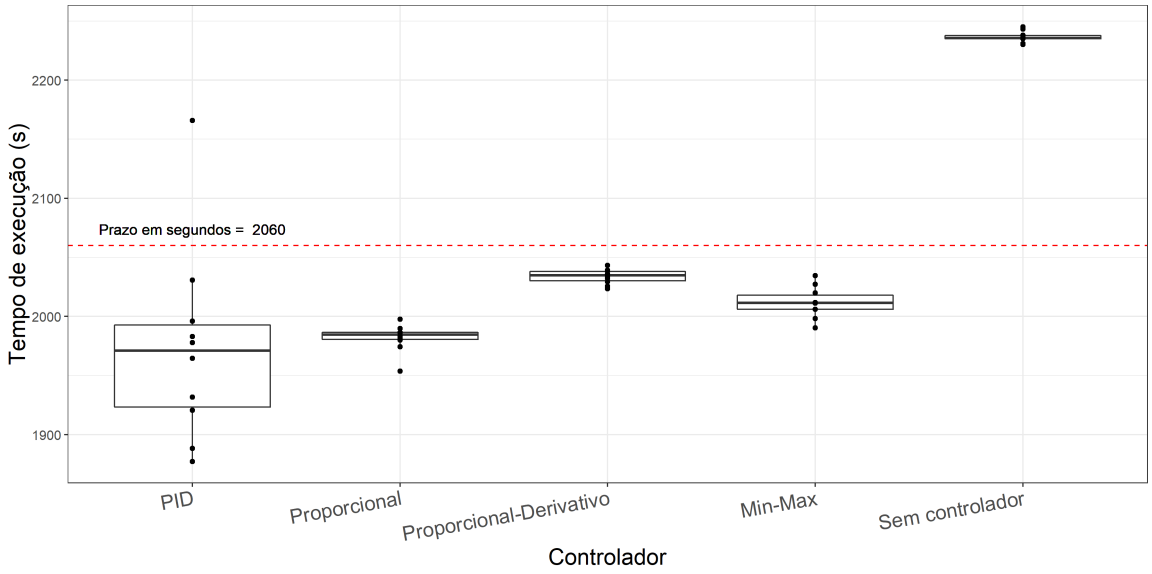


Figura 7.1: Tempo de execução utilizando diferentes controladores

7.1.2 Uso de recursos

O uso de CPU do *microbenchmark* de CPU, utilizando as diferentes opções de controlador, são apresentados na Figura 7.2. Inicialmente, nos casos dos controladores PID, Proporcional-Derivativo e Proporcional, pode-se observar picos na utilização de CPU que coincidem com os momentos de perturbação na execução, apresentados na Figura 6.11. Esses picos são evidência da ação dos controladores, tentando compensar o atraso causado pela execução de tarefas mais pesadas.

Pode-se observar que as variações no uso de CPU no caso do controlador Min-Max são muito mais bruscas do que nos casos dos controladores Proporcional-Derivativo e Proporcional. Estes conseguem modificar a quantidade de recursos alocados de maneira mais suave. Esse padrão de variação no caso Min-Max faz com que a quantidade de recursos utilizados pela aplicação seja mais imprevisível, tornando balanceamento da infraestrutura mais complicado.

A alocação de CPU ao longo do tempo, utilizando as diferentes opções de controlador, é apresentada na Figura 7.3. A mudança no nível de recursos é bastante brusca, como espe-

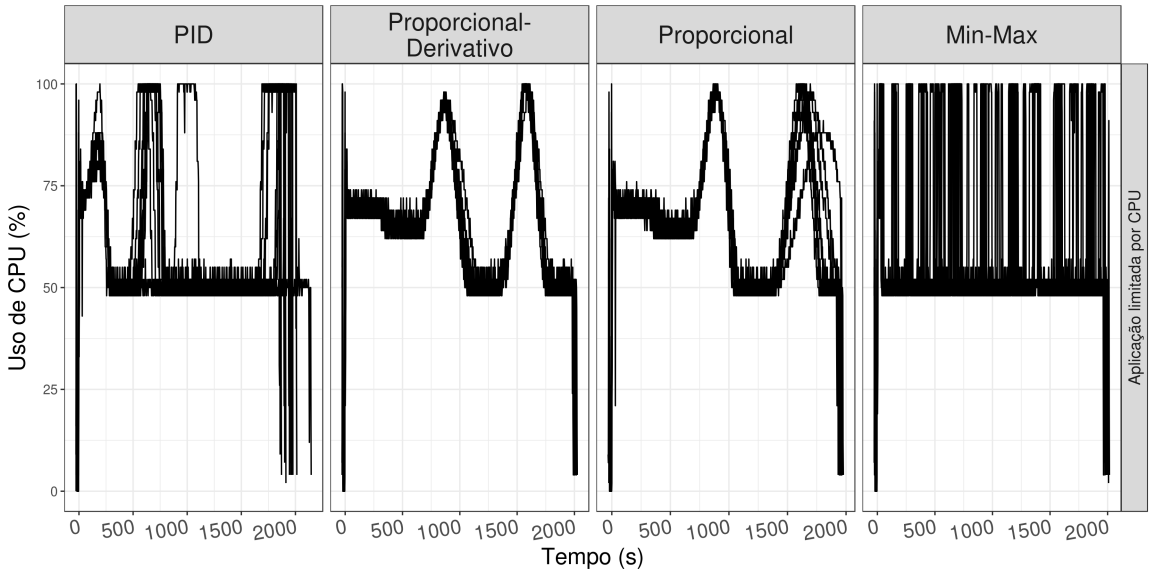


Figura 7.2: Uso de CPU utilizando diferentes controladores

rado, no caso Min-Max, e mais suave nos casos proporcional e proporcional-derivativo. A adição do componente integrativo tornou a adição e remoção de recursos mais agressiva em comparação aos casos proporcional e proporcional-derivativo.

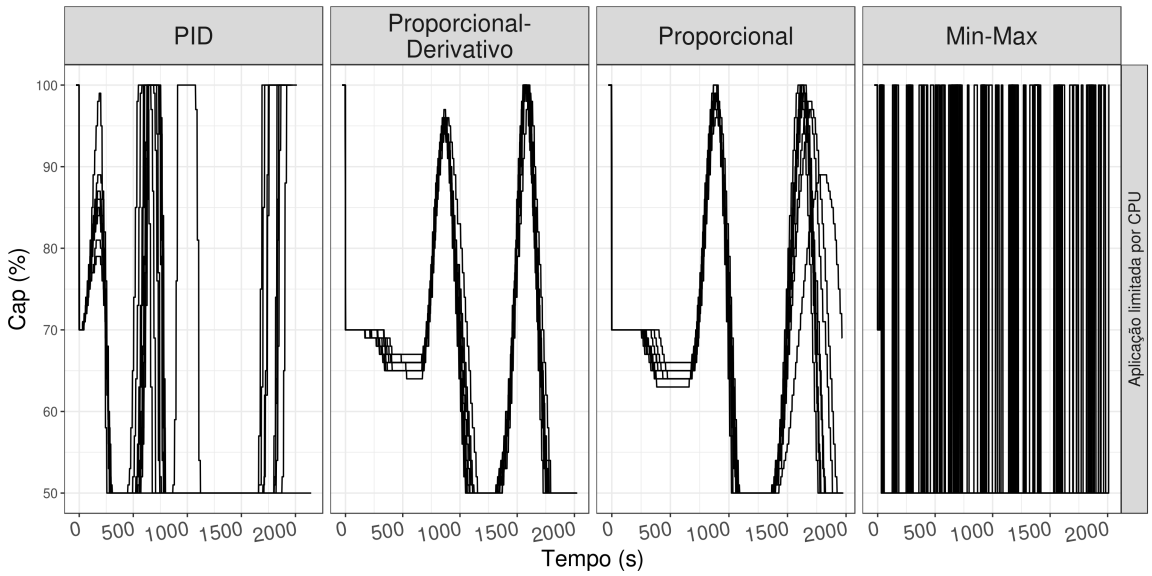


Figura 7.3: Modificação no nível de recursos alocados utilizando diferentes controladores

7.2 *Microbenchmark* I/O

7.2.1 Tempo de execução

Nas execuções do *microbenchmark* de I/O foi utilizado apenas o atuador de CPU e I/O. Os tempos de execução do *microbenchmark* de I/O, utilizando as diferentes opções de controlador, são apresentados na Figura 7.4.

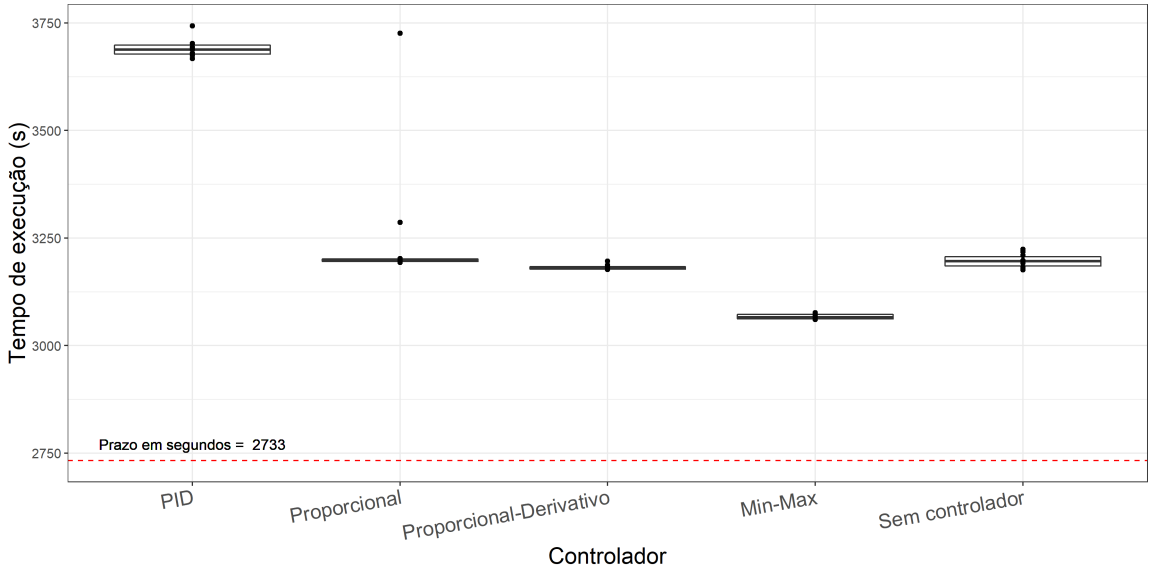


Figura 7.4: Tempo de execução utilizando diferentes controladores

Os controladores estudados foram ineficazes em fazer o *microbenchmark* I/O terminar no prazo estabelecido. Nos casos dos controladores Proporcional e Proporcional-Derivativo, os resultados de tempo de execução foram muito semelhantes aos das execuções sem controlador, e, no caso do controlador PID, chegaram a ser piores. As execuções com controlador Min-Max apresentaram os melhores resultados em termos de tempo de execução e ficaram mais próximas de terminar no prazo. Uma discussão sobre as causas destes resultados negativos é apresentada na seção seguinte.

7.2.2 Uso de recursos

A alocação de I/O ao longo da execução, utilizando as diferentes opções de controlador, é apresentada na Figura 7.5. Observou-se o mesmo comportamento no nível de recursos em todos os casos estudados. Primeiramente ocorre uma queda na alocação de recur-

sos, seguida por um aumento. Esse comportamento é causado pelo padrão de progresso do *microbenchmark*, que apresenta uma primeira fase rápida e uma segunda fase lenta. Deste modo, o erro de progresso apresentado na primeira fase é positivo, uma vez que o *microbenchmark* está adiantado, e os controladores removem recursos. Na segunda fase, o *microbenchmark* é mais lento, a quantidade reduzida de recursos provoca o atraso do *microbenchmark* e os controladores adicionam recursos de volta.

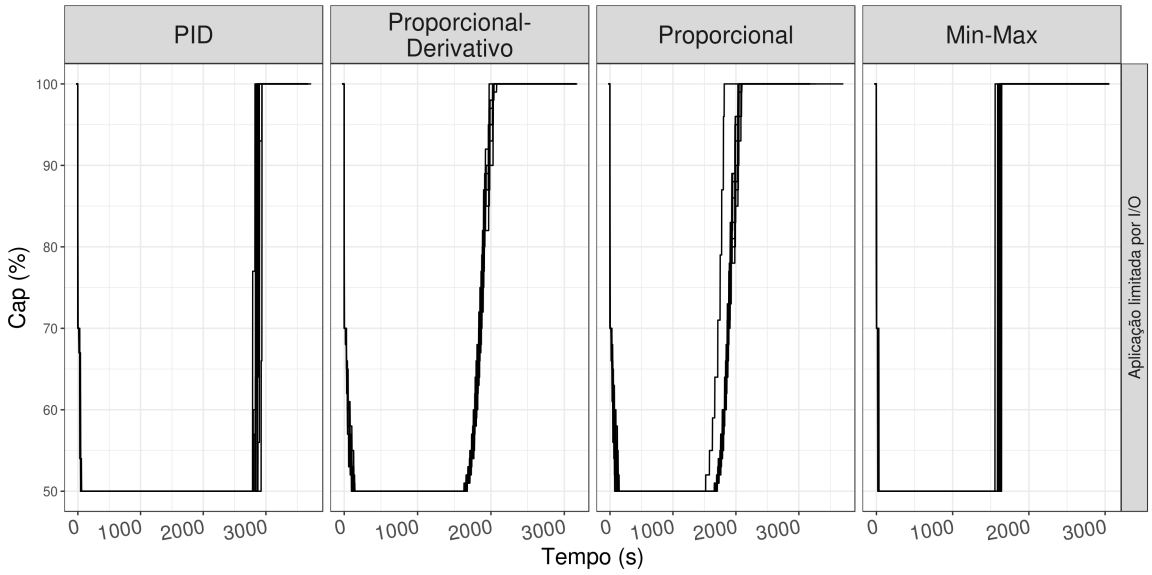


Figura 7.5: Modificação no nível de recursos alocados utilizando diferentes controladores

Pode-se observar na Figura 7.5 como o controlador Min-Max, que obteve os melhores resultados, aumenta a alocação de I/O de forma muita rápida, comparado com os controladores Proporcional e Proporcional-Derivativo. Baseado na análise de comportamento dos controladores e nos resultados de tempo de execução, formula-se a hipótese de que o cumprimento dos prazos de execução pelo *microbenchmark* depende da capacidade de adicionar recursos cedo o suficiente, de forma a compensar o atraso na segunda fase do *microbenchmark*.

A quantidade de dados lidos ao longo da execução, utilizando as diferentes opções de controlador, é apresentada na Figura 7.6.

Pode-se observar como o ritmo de leitura nas execuções utilizando controlador Min-Max chega a seu máximo mais cedo. Nas execuções com os controladores Proporcional e Proporcional-Derivativo, a aumento é bem mais gradual. O aumento do ritmo no caso PID é bem mais tardio.

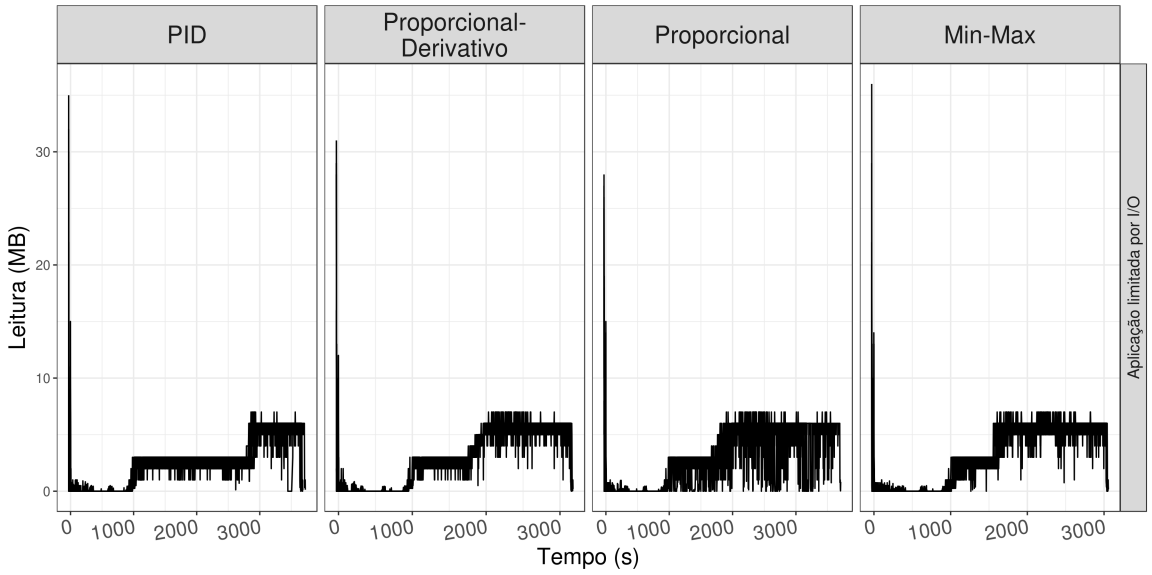


Figura 7.6: Leitura utilizando diferentes controladores

A quantidade de dados escritos ao longo da execução, utilizando as diferentes opções de controlador, é apresentada na Figura 7.7. Nas execuções com os controladores Proporcional e Min-Max foram observadas anomalias pontuais na forma de aumento brusco na quantidade de dados escritos e um ritmo de escrita muito mais rápido que o normal. No entanto, o padrão de escrita observado é o mesmo para todos os casos.

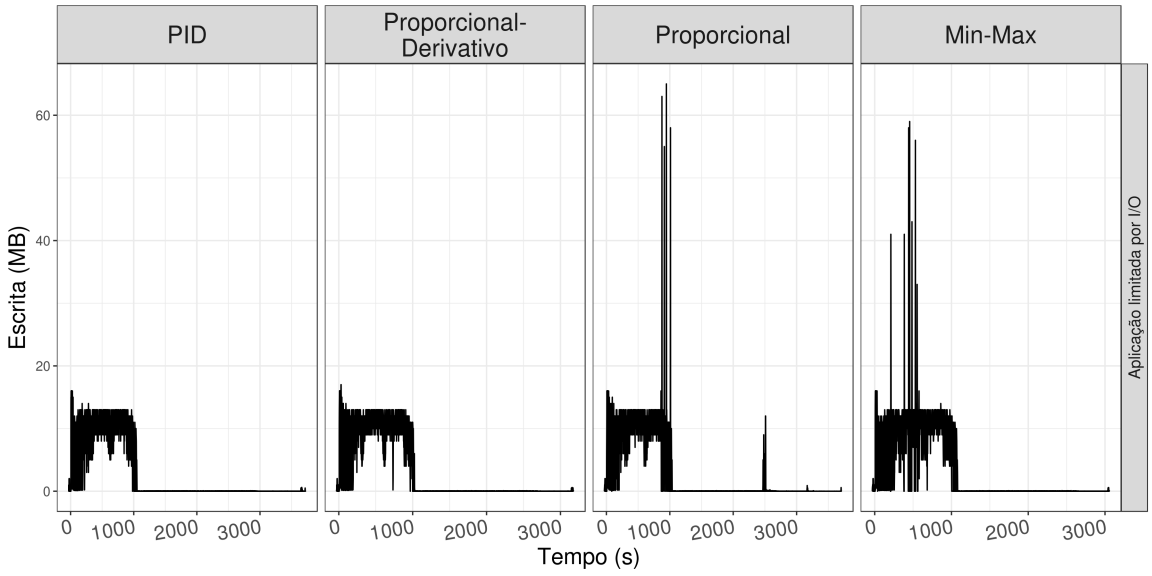


Figura 7.7: Escrita utilizando diferentes controladores

Baseado nos dados de alocação e uso de recursos pelo *microbenchmark*, pode-se levantar

a hipótese de que os atrasos podem ser evitados utilizando uma configuração adequada. Uma opção de controlador Proporcional ou Proporcional-Derivativo, utilizando uma configuração ajustada, deve ser capaz de reconhecer tendência a atrasos na segunda fase da aplicação cedo o suficiente e garantir o término no prazo.

7.3 *Microbenchmark* I/O – Ajustes

Com o objetivo de encontrar uma configuração que permita aos controladores cumprirem prazos de execução no caso do *microbenchmark* I/O, foram testados dois ajustes. O primeiro ajuste testado envolveu mudança nos valores dos ganhos utilizados pelo controlador Proporcional-Derivativo. Foi testada uma configuração com ganho derivativo mais alto que o ganho proporcional.

Ao utilizar as configurações triviais, com valores de ganhos proporcional e derivativo iguais a 1, os componentes proporcional e derivativo têm pesos similares no cálculo do tamanho da atuação, dependendo apenas dos valores dos erros. Devido à não-linearidade do progresso das execuções do *microbenchmark*, com uma primeira fase muito rápida, o erro tende a permanecer em valores positivos e de valor absoluto alto durante boa parte da execução.

Em uma situação com padrão de erro deste tipo, um controlador Proporcional-Derivativo, utilizando a configuração trivial, tende a apresentar um componente proporcional de valor negativo e de valor absoluto alto durante boa parte da execução. O componente derivativo reage à tendência de atraso futuro da execução. No entanto, uma vez que o valor absoluto da variação do erro aumenta de maneira suave, o valor do componente derivativo é muito pequeno em relação ao tamanho do componente proporcional e sua ação para compensar o atraso não é refletida no tamanho da atuação.

Estes dois aspectos causam o atraso da aplicação, uma vez que o controlador Proporcional-Derivativo, utilizando essa configuração trivial, demora a adicionar recursos quando necessário. Com a adoção de uma configuração cujo ganho derivativo é mais alto que o ganho proporcional, esperava-se que o componente derivativo conseguisse aumentar o tamanho da atuação de maneira mais rápida, adicionando recursos mais cedo. Neste ajuste foi utilizado um ganho proporcional igual a 0,1 e um ganho derivativo igual a 20.

O segundo ajuste envolveu uma modificação na forma como o progresso é fornecido pelo *microbenchmark*. Normalmente, cada tarefa completada, seja de escrita ou leitura, contribui o mesmo valor para o progresso total. Executando dessa maneira, obtemos um progresso não-linear, com uma segunda fase muito mais lenta, como mostrado na Figura 6.4.

O segundo ajuste consistiu de alterar a contribuição das tarefas de leitura utilizando um fator de ajuste t. Utilizando este ajuste, uma tarefa de leitura contribui t vezes mais para o progresso total do que uma tarefa de escrita. O valor de t foi determinado a partir de uma observação em que o *microbenchmark* era executado sem controladores e seu progresso era coletado. Foi observado que as execuções em que o valor de t utilizado era igual a 3 apresentavam progresso aproximadamente linear. Uma nova versão do *microbenchmark* com progresso linearizado foi executada utilizando controlador Proporcional-Derivativo com a mesma configuração da observação anterior.

Os resultados das execuções com os dois ajustes são apresentados a seguir.

7.3.1 Tempo de execução

Os tempos de execução do *microbenchmark* de I/O, utilizando as novas opções de controle, são apresentados na Figura 7.8.

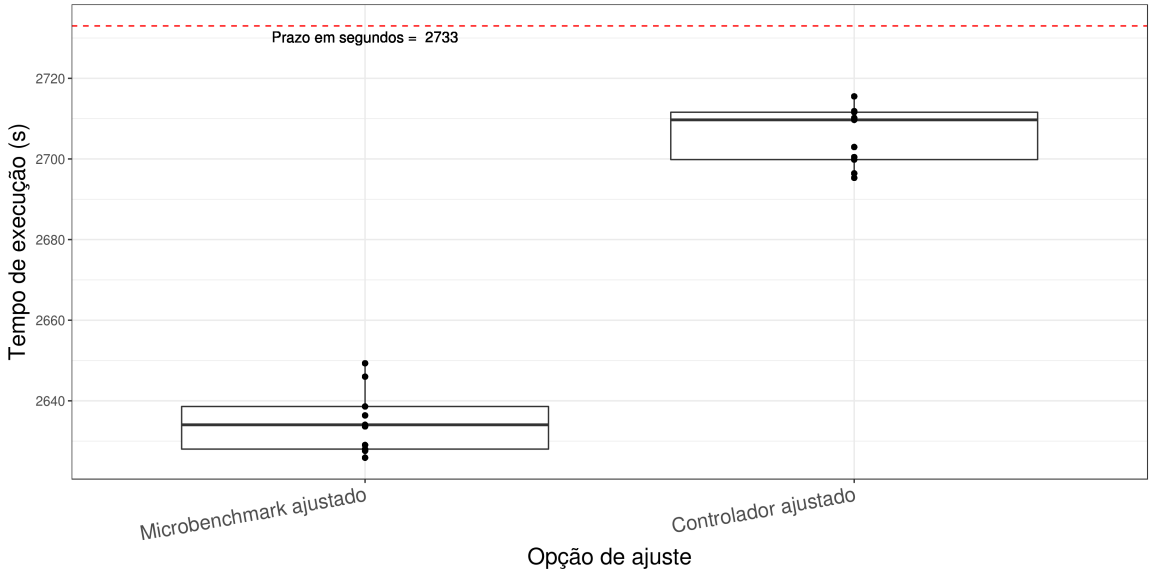


Figura 7.8: Tempo de execução utilizando as novas opções de controle

As duas opções foram eficazes em fazer o *microbenchmark* terminar no prazo estabe-

lecido. Os tempos de execução utilizando o controlador ajustado foram mais próximos do prazo, o que indica maior eficiência no uso de recursos.

7.3.2 Uso de recursos

A alocação de I/O ao longo da execução, utilizando as novas opções de controle, é apresentada na Figura 7.9

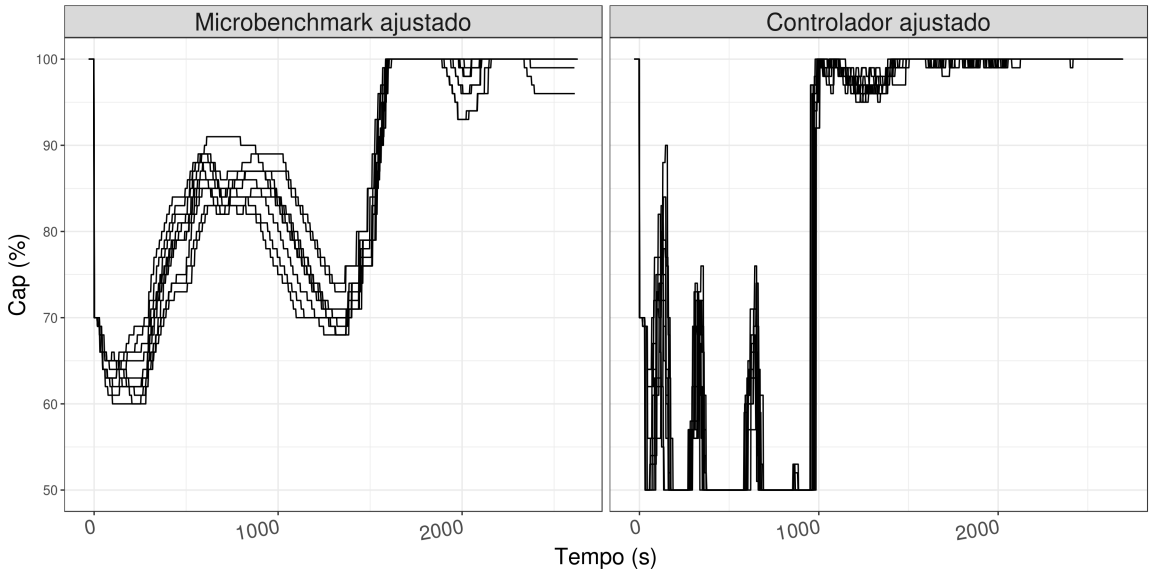


Figura 7.9: Modificação no nível de recursos alocados utilizando as novas opções de controle

Pode-se observar que, utilizando a nova configuração, o controlador Proporcional-Derivativo perdeu sua suavidade de ação, apresentando picos acentuados na alocação de recursos. Isso representa uma desvantagem na sua utilização. As execuções que utilizaram o *microbenchmark* ajustado apresentaram um ajuste de recursos mais suave.

A quantidade de dados lidos ao longo da execução, utilizando as novas opções de controle, é apresentada na Figura 7.10. Em ambas as opções de ajuste, a leitura chegou a seu ritmo máximo mais cedo que nos casos investigados anteriormente. No caso do controlador Proporcional-Derivativo ajustado o crescimento foi um pouco mais agressivo. No caso do *microbenchmark* ajustado, o crescimento no ritmo de leitura começou bem mais cedo.

A quantidade de dados escritos ao longo da execução, utilizando as novas opções de controle, é apresentada na Figura 7.11. Pode-se observar que o ritmo de escrita no caso do *microbenchmark* ajustado foi maior que no caso Proporcional-Derivativo ajustado, e mesmo

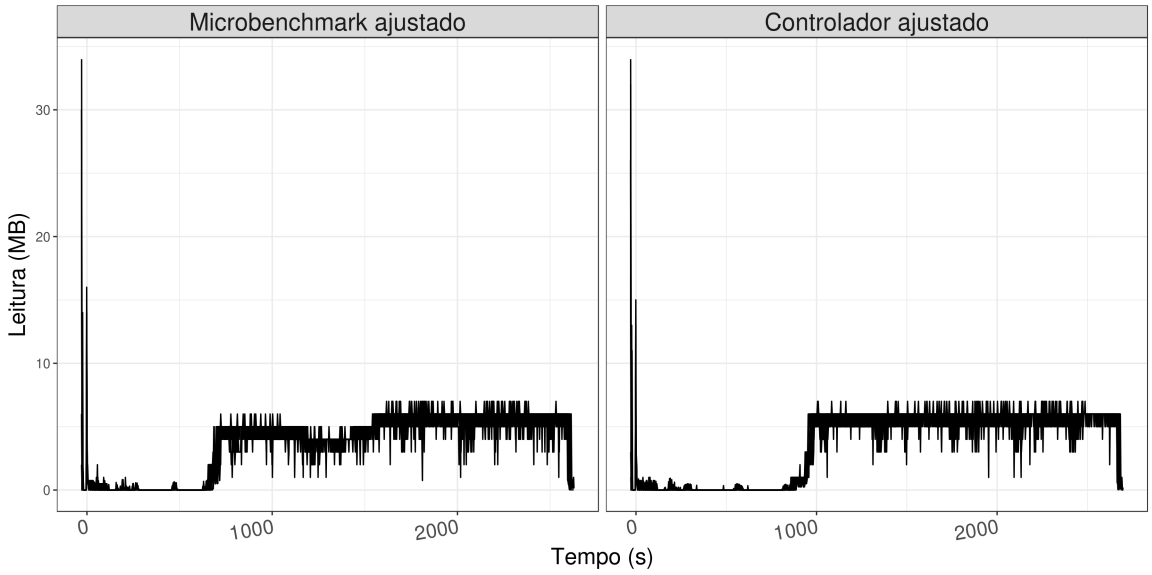


Figura 7.10: Leitura utilizando as novas opções de controle

maior que todos os casos investigados anteriormente. A escrita chega consistentemente a 20 MB/s. Esse maior ritmo de escrita explica o término da fase de escrita bem mais cedo neste caso e também a diferença de tempo de execução entre os casos Proporcional-Derivativo ajustado e *microbenchmark* ajustado.

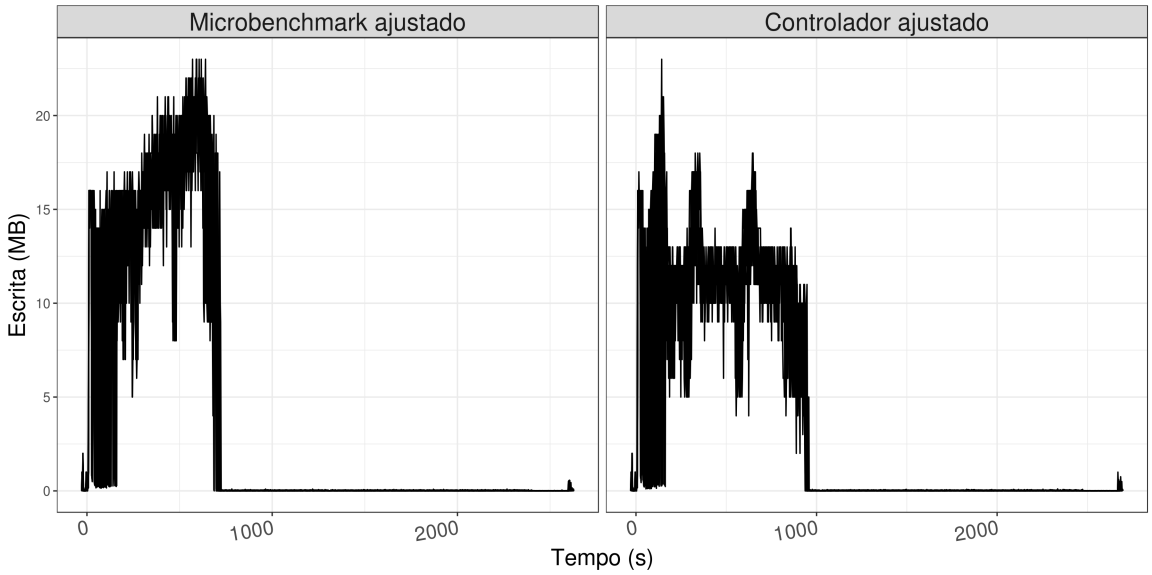


Figura 7.11: Escrita utilizando as novas opções de controle

Considerando as observações com o *microbenchmark* de I/O, pode-se concluir que os controladores estudados apresentam dificuldade em garantir prazo de execução do

microbenchmark. Os resultados observados na execução do *microbenchmark* ajustado, melhores que os resultados da execução do *microbenchmark* sem alterações, sugerem que o aspecto causador da dificuldade é a não-linearidade do progresso da aplicação. Adicionalmente, pode-se concluir que é possível obter resultados melhores na execução de aplicações de progresso não-linear a partir de ajustes na configuração do controlador.

7.4 EMaaS

7.4.1 Tempo de execução

As execuções dos *microbenchmarks* utilizando o controlador PID confirmaram a hipótese de que este controlador é inadequado para utilização no contexto da pesquisa. A dispersão de valores de tempo de execução e a variabilidade de níveis de alocação de recursos são altos no caso deste controlador. Portanto, essa opção foi excluída das observações com a aplicação.

Os tempos de execução do EMaaS, utilizando as diferentes opções de controlador e atuador, são apresentados na Figura 7.12. Quanto ao cumprimento do prazo de execução estabelecido, nenhuma combinação de controle e atuação foi eficiente em cumprir o prazo de execução. As execuções com os controladores Proporcional e Proporcional-Derivativo apresentaram tempos de execução piores do que as execuções sem controlador. O controlador Min-Max, por sua vez, conseguiu os melhores resultados, com tempos de execução mais próximos do prazo e com menor dispersão. A adição de atuação em I/O, através do Atuador CPU + I/O, para o EMaaS, não resultou em melhoria alguma em termos de cumprimento de prazos de execução. Pode-se chegar a conclusões sobre esses resultados a partir da análise do uso de recursos da aplicação, apresentada na seção seguinte.

7.4.2 Uso de recursos

A alocação de recursos ao longo da execução, utilizando as diferentes opções de controlador, é apresentada na Figura 7.13

Pode-se observar como a mudança no nível de recursos no caso do controlador Min-Max é agressiva, enquanto que nos casos dos controladores Proporcional e Proporcional-Derivativo, a adição de recursos foi mais suave. Observa-se como o nível de recursos, no caso

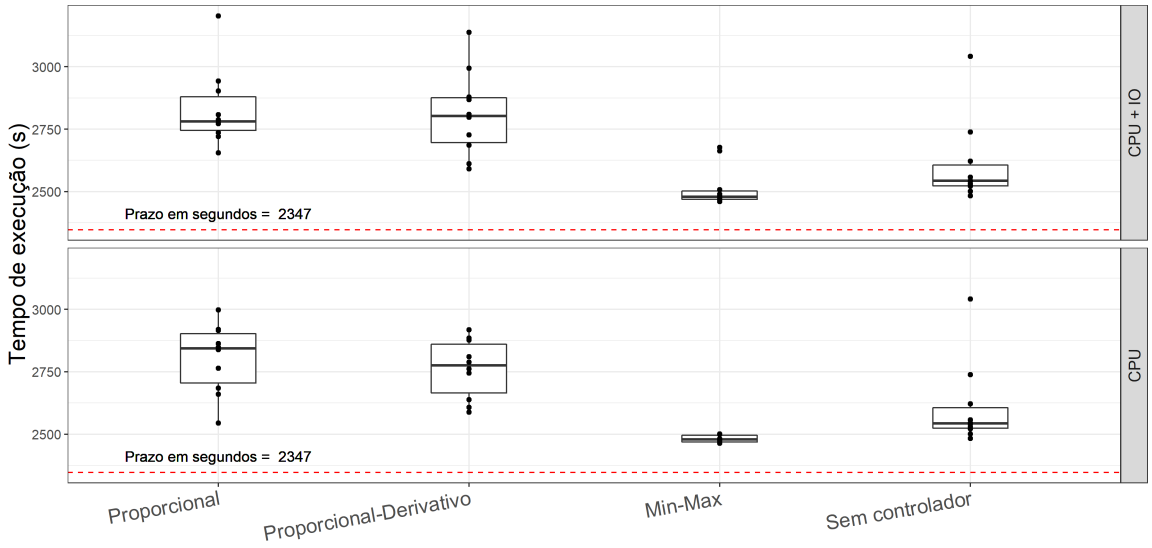


Figura 7.12: Tempo de execução utilizando diferentes controladores

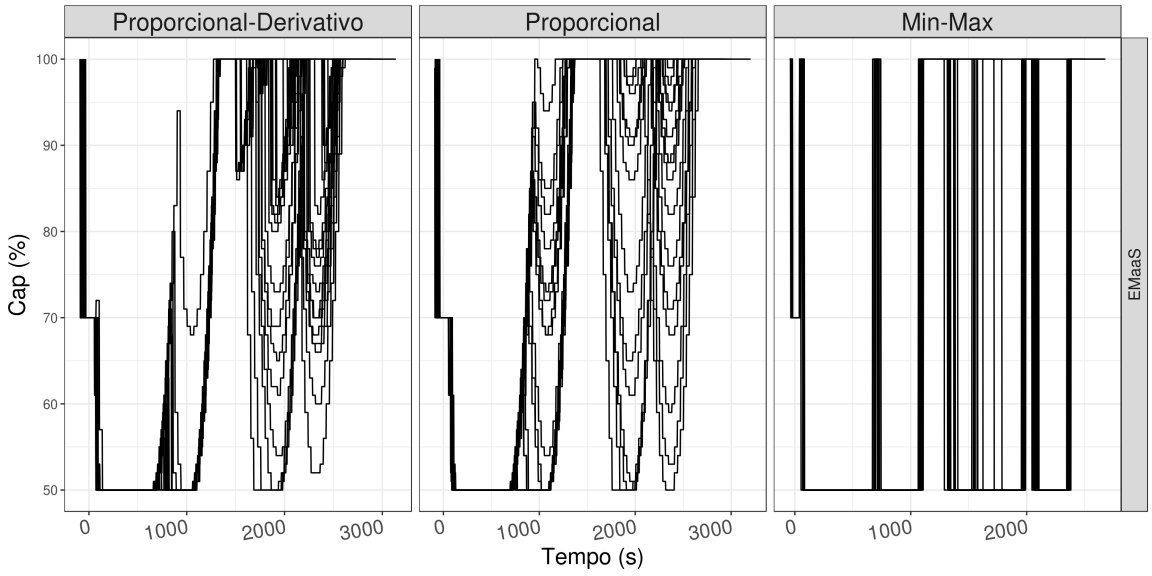


Figura 7.13: Modificação no nível de recursos alocados utilizando diferentes controladores

Min-Max, chega muito mais rapidamente a 100%. Nos casos Proporcional e Proporcional-Derivativo, o nível passa muito tempo em valores mais baixos, e, por vezes, não chega a 100%.

Os atrasos obtidos na utilização dos controladores e atuadores estudados levam a acreditar que a quantidade de recursos disponibilizados à aplicação foi insuficiente. Portanto, o nível de recursos deveria permanecer em valores mais altos por mais tempo. Uma opção é

fazer o controlador Proporcional-Derivativo reagir mais cedo a atrasos, através de uma nova configuração.

O uso de CPU ao longo da execução, utilizando as diferentes opções de controlador, é apresentada na Figura 7.14. Quanto ao uso de CPU, as execuções com controlador Min-Max são caracterizadas por uma variabilidade mais agressiva. A variação do uso de CPU nas execuções com controladores Proporcional e Proporcional-Derivativo é mais suave, apesar de apresentar quedas acentuadas.

Um aspecto a notar no uso de CPU no caso Min-Max é a diferença entre os padrões no fim da execução quando utilizados atuadores CPU e CPU + I/O. No caso CPU + I/O, o uso de CPU é bem maior. Uma hipótese é de que a adição de recursos de I/O nesse momento permitiu que as operações de leitura e escrita fossem completadas mais rapidamente, permitindo que a parte limitada por CPU da aplicação agisse mais ativamente.

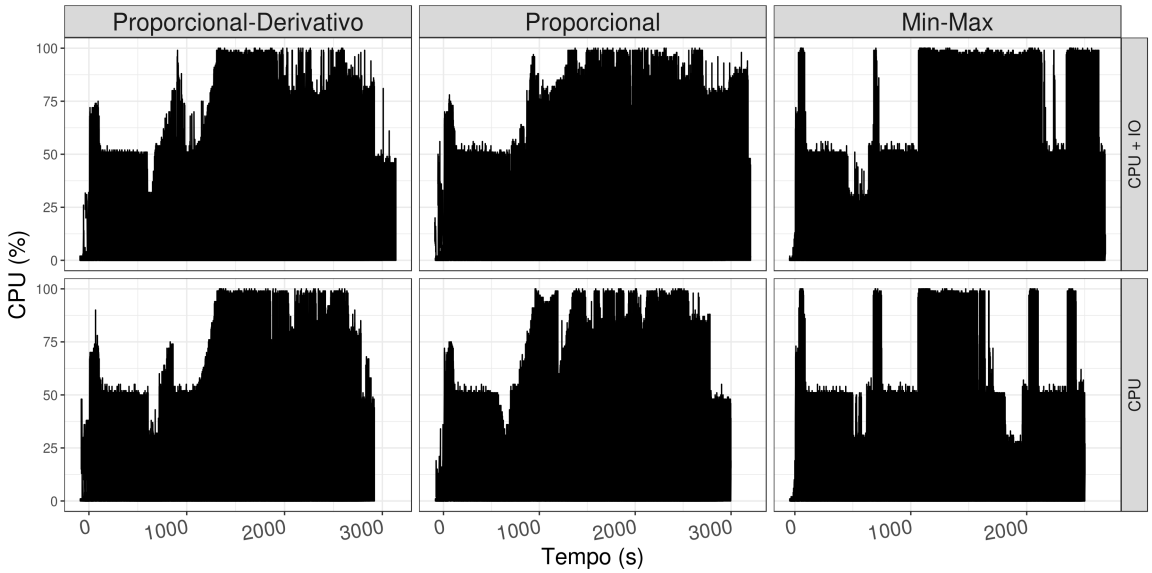


Figura 7.14: Uso de CPU utilizando diferentes controladores

A quantidade de dados lidos ao longo da execução, utilizando as diferentes opções de controlador, é apresentada na Figura 7.15. Quando à leitura, nenhuma diferença relevante foi notada entre os padrões das execuções com controladores Proporcional e Proporcional-Derivativo. No caso Min-Max, no entanto, pode-se notar leituras maiores, principalmente no final da execução. Esse padrão corrobora a hipótese feita na análise do uso de CPU. A variação entre os padrões de leitura é possivelmente causada pela diferença no ritmo de

adição de recursos nos diferentes controladores.

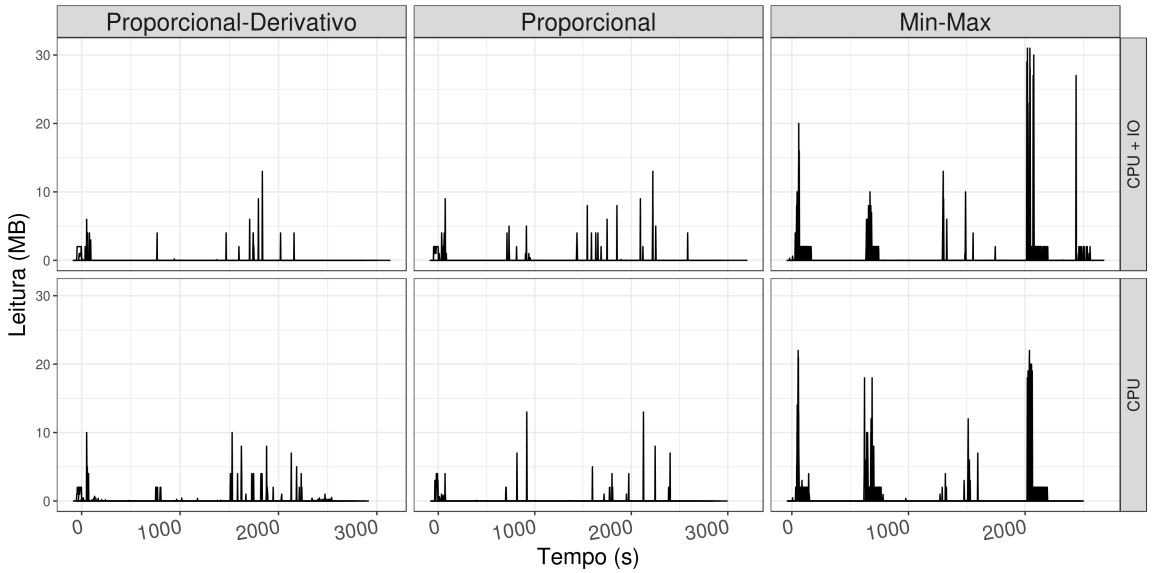


Figura 7.15: Leitura utilizando diferentes controladores

A quantidade de dados escritos ao longo da execução, utilizando as diferentes opções de controlador, é apresentada na Figura 7.16. Quanto à escrita, foi observada pouca diferença entre as opções de controle e atuação estudadas.

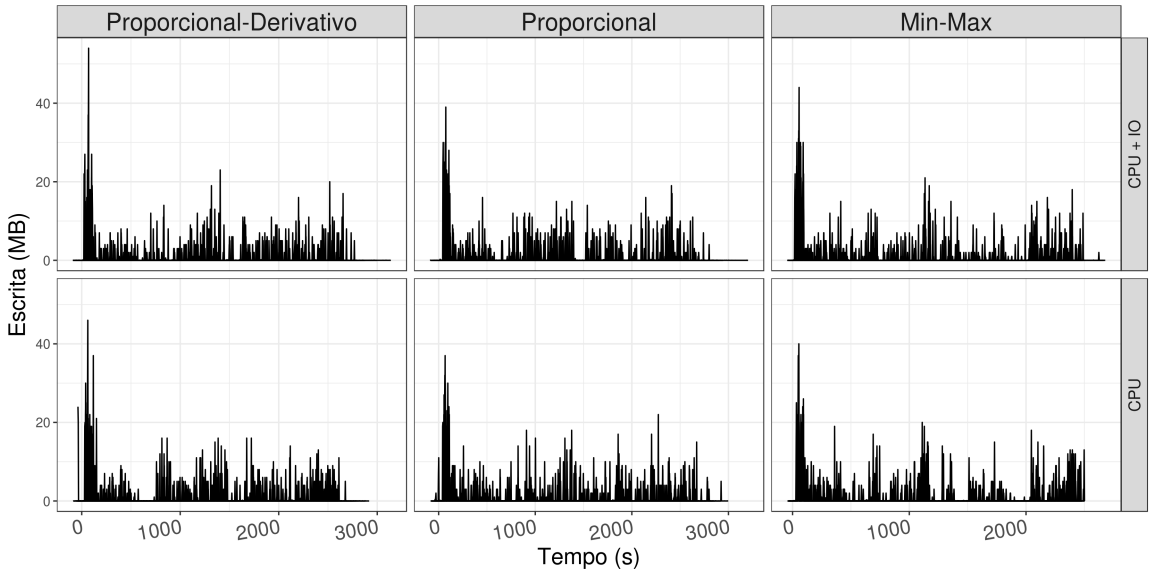


Figura 7.16: Escrita utilizando diferentes controladores

7.5 EMaaS – Ajustes

Com o objetivo de tornar suficiente a quantidade de recursos disponibilizados durante a execução, foi estudada uma nova configuração de controlador Proporcional-Derivativo. Essa configuração utilizava um fator derivativo mais alto, de forma a fazer o controlador reagir mais rapidamente a tendências de atrasos e, assim, adicionar recursos mais cedo.

7.5.1 Tempo de execução

Os tempos de execução do EMaaS, utilizando as configurações de controlador derivativo, são apresentados na Figura 7.17.

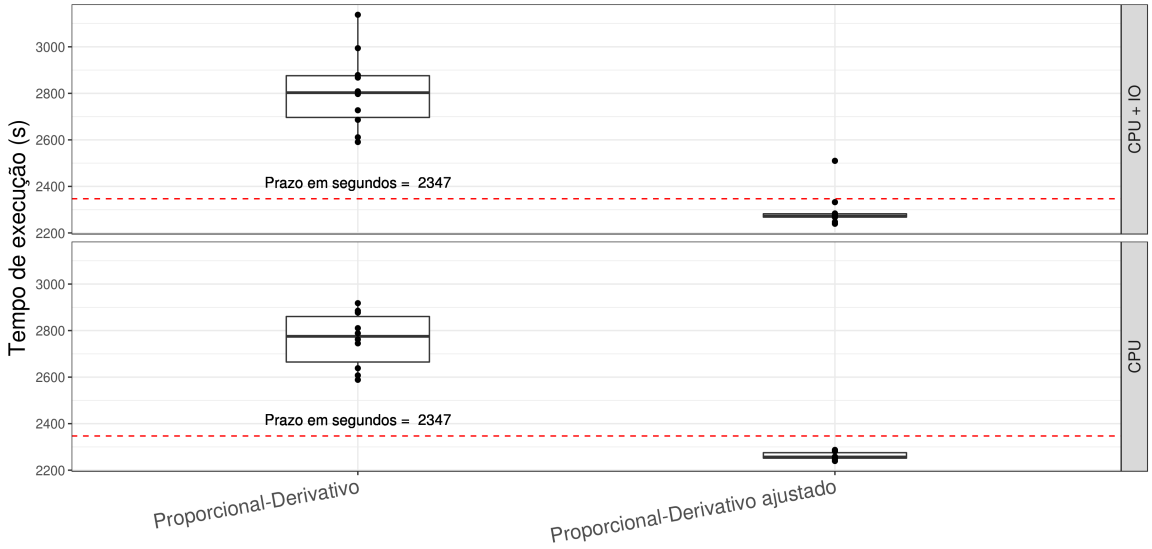


Figura 7.17: Comparação do tempo de execução utilizando as duas configurações do controlador derivativo

Pode-se observar que o controlador Proporcional-Derivativo, utilizando a nova configuração e considerando ambos os atuadores de CPU e CPU + I/O, é eficaz em termos de cumprimento de prazos de execução para o EMaaS. Observa-se também que não houve nenhuma melhoria associada à adição de atuação em I/O. A adição de atuação em I/O parece, inclusive, ter prejudicado a confiabilidade do controlador, uma vez que os valores de tempo de execução são um pouco mais dispersos.

7.5.2 Uso de recursos

A alocação de recursos ao longo da execução, utilizando as configurações de controlador derivativo, é apresentada na Figura 7.18. O controlador Proporcional-Derivativo, utilizando a nova configuração, começa a adicionar recursos mais cedo mas mantendo a suavidade na atuação. A remoção de recursos ainda é agressiva, sendo necessários ajustes adicionais visando suavizar a atuação nesses momentos.

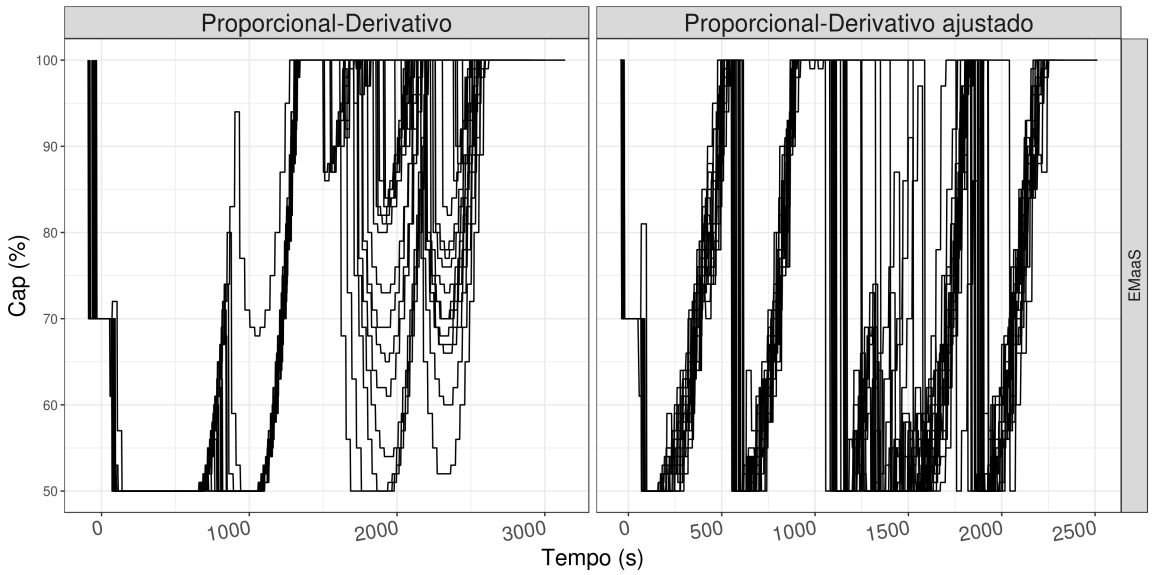


Figura 7.18: Comparação da mudança no nível de recursos utilizando as duas configurações do controlador derivativo

O uso de CPU ao longo da execução, utilizando as configurações de controlador derivativo, é apresentada na Figura 7.19. O controlador Proporcional-Derivativo, utilizando sua nova configuração, demonstrou ter variações mais agressivas de uso de CPU do que em sua configuração trivial. No entanto, essas variações tendem a ocorrer durante a remoção de recursos. Como apresentado anteriormente, nessa configuração a remoção de recursos é realizada de forma mais agressiva e o problema da variação no uso de CPU deve ser resolvido com novos ajustes.

A quantidade de dados lidos ao longo da execução, utilizando as configurações de controlador derivativo, é apresentada na Figura 7.20. As execuções utilizando a nova configuração do controlador Proporcional-Derivativo apresentaram leituras maiores, em um padrão semelhante ao do caso Min-Max, onde os resultados foram os melhores na primeira avaliação.

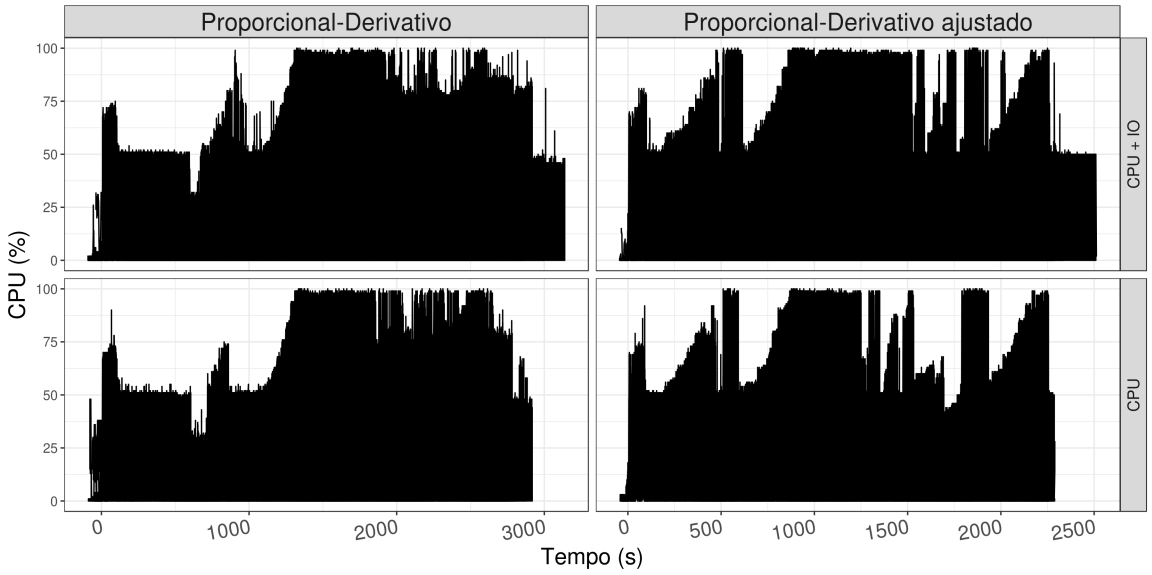


Figura 7.19: Comparação do uso de CPU utilizando as duas configurações do controlador derivativo

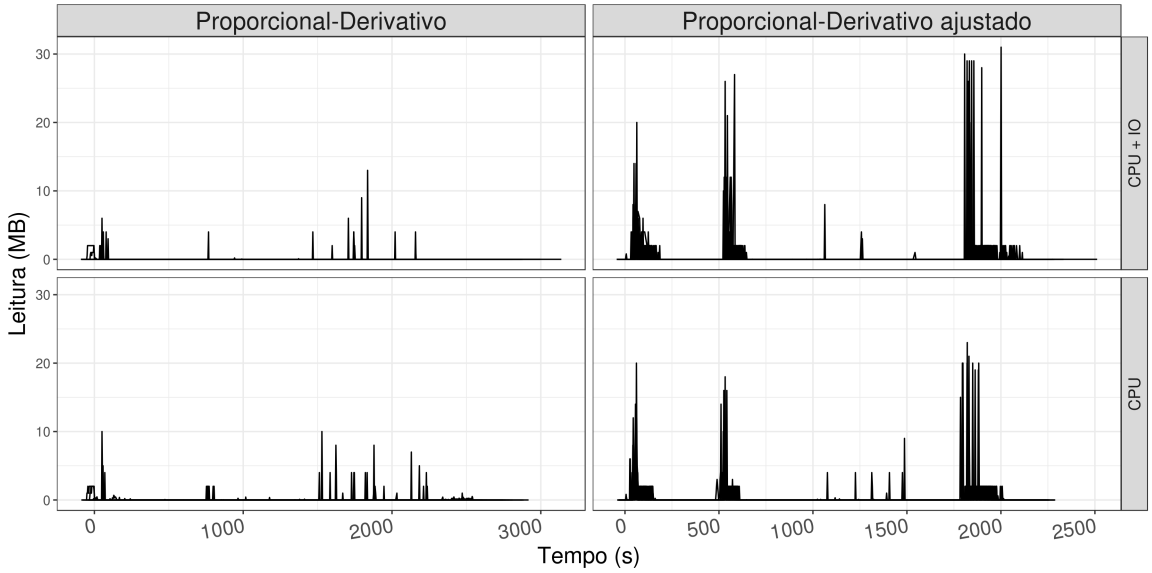


Figura 7.20: Comparação da quantidade de dados lidos utilizando as duas configurações do controlador derivativo

A quantidade de dados escritos ao longo da execução, utilizando as configurações de controlador derivativo, é apresentada na Figura 7.21. Quando à escrita, novamente, foi observada pouca diferença entre as opções de controle estudadas.

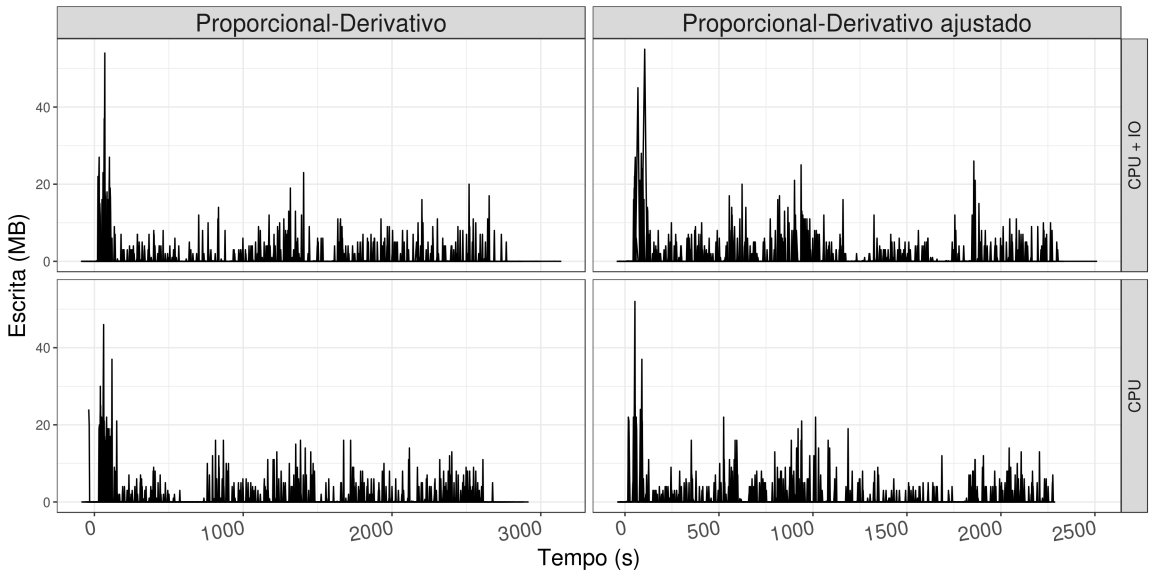


Figura 7.21: Comparação da quantidade de dados escritos utilizando as duas configurações do controlador derivativo

Capítulo 8

Conclusão

Foram avaliadas opções de controlador utilizando métodos de atuação vertical com o objetivo de garantir qualidade de serviço de aplicações na forma de cumprimento de prazos de execução. Os resultados obtidos mostram que a utilização de um mecanismo com essas características é viável. Resultados positivos foram obtidos na execução de dois *microbenchmarks* e uma aplicação realista.

Neste capítulo serão apresentadas as conclusões e considerações finais quanto aos resultados e sugestões de trabalhos futuros que podem complementar esta pesquisa.

8.1 Considerações finais

Resultados positivos foram obtidos quanto a garantir prazos de execução. No entanto, a opção de configuração mais simples, o controlador Min-Max, age de forma bastante agressiva, causando perturbação na infraestrutura, e em alguns casos não funciona. Os controladores que funcionaram e agiam de forma mais suave precisaram de ajustes que eram dependentes do comportamento do progresso e do uso de recursos da aplicação. O controlador PID, por sua vez, apresentou pouca confiabilidade, com alta dispersão de valores de tempo de execução, e grande variação na quantidade de recursos alocados. A utilização deste último controlador é, portanto, pouco recomendada.

Foi observado que os controladores estudados funcionam melhor com aplicações que apresentam progresso linear ou aproximadamente linear, como nos casos do *microbenchmark* de CPU e do *microbenchmark* I/O com progresso linearizado. No caso do

microbenchmark de CPU mesmo as configurações triviais mostraram-se eficazes. Aplicações com progresso irregular se mostraram mais problemáticas em termos de configuração. Aplicações que apresentam etapas muito rápidas seguidas de etapas muito lentas, como o EMaaS, podem precisar de reações mais rápidas ao atraso, obtidas utilizando o componente derivativo.

Na execução desta pesquisa, foram utilizadas certas tecnologias. Como gerente de nuvem foi utilizado o OpenStack, como provisionador de máquinas virtuais foi utilizado o KVM e como plataforma de execução de aplicações *batch* foi utilizado Spark. Essas tecnologias são amplamente utilizadas em ambientes de nuvem e a estratégia de provisionamento discutida neste trabalho deve ser válida para implantação em qualquer ambiente em que estas tecnologias estejam presentes. Adicionalmente, a estratégia de provisionamento pode ser utilizada no contexto de outras tecnologias, contanto que possuam características semelhantes. Por exemplo, ao usar um hipervisor diferente do KVM, é necessário que o hipervisor considerado ofereça opções de atuação e modificação de cota de CPU e operações de disco de forma semelhante ao KVM.

8.2 Trabalhos futuros

A solução de provisionamento apresentada neste trabalho foi desenvolvida no contexto do projeto BigSea. Desejava-se obter um mecanismo baseado em provisionamento vertical que garantisse qualidade de serviço para aplicações consideradas no projeto, como o EMaaS. O objetivo foi alcançado, uma vez que o mecanismo, com os devidos ajustes, consegue garantir prazo de término. No entanto, o mecanismo apresenta limitações quanto à facilidade de configuração. Este problema pode ser atacado com investigações relacionadas a autoajuste. Porém, tais investigações extrapolavam o escopo da pesquisa.

Devido ao escopo do projeto no qual a pesquisa estava inserida, o estudo de caso proposto era limitado a aplicações não-interativas, como aplicações *batch*. Essas aplicações apresentam uma complexidade maior em termos de provisionamento, uma vez que as métricas utilizadas para medir qualidade de serviço, como cumprimento de prazos, dificultam a avaliação rápida da qualidade do controle e consequentemente sua reconfiguração caso necessário. Um possível complemento às observações realizadas nesta pesquisa seria a ava-

liação dos controladores com aplicações interativas (como uma aplicação web). Neste caso, utilizando uma métrica de qualidade de serviço com maior granularidade, o provisionamento e a reconfiguração dos controladores seriam mais fáceis.

Bibliografia

[1] Configuração de instâncias no aws. https://aws.amazon.com/pt/ec2/instance-types/. Acessado: 17/06/2018.

[2] Definição de computação na nuvem segundo o nist. https://csrc.nist.gov/projects/cloud-computing. Acessado: 12/04/2018.

[3] Desempenho do disco no google cloud. https://cloud.google.com/compute/docs/disks/performance#ssd-pd-performance. Acessado: 17/06/2018.

[4] Experiência positiva da netflix com computação na nuvem. https://media.netflix.com/pt_br/company-blog/completing-the-netflix-cloud-migration. Acessado: 17/06/2018.

[5] Limitações do kvm quanto à alteração da quantidade de cpus em uma máquina virtual. https://www.linux-kvm.org/page/CPUHotPlug. Acessado: 12/09/2018.

[6] Provisionamento automático no aws. https://aws.amazon.com/pt/autoscaling/. Acessado: 10/05/2018.

[7] Provisionamento automático no google cloud. https://cloud.google.com/compute/docs/autoscaler/. Acessado: 10/05/2018.

[8] Repositório do componente controlador. https://github.com/bigsea-ufcg/bigsea-controller. Acessado: 05/06/2018.

[9] Repositório do componente gerente. https://github.com/bigsea-ufcg/bigsea-manager. Acessado: 05/06/2018.

[10] Repositório do componente monitorador. `https://github.com/bigsea-ufcg/bigsea-monitor`. Acessado: 05/06/2018.

[11] Resultados do spark na ordenação de 1 petabyte. `https://databricks.com/blog/2014/10/10/spark-petabyte-sort.html`. Acessado: 10/05/2018.

[12] Sobre uso de openstack no cern. `https://www.openstack.org/videos/tokio-2015/unveiling-cern-cloud-architecture`. Acessado: 08/05/2018.

[13] Sobre uso de openstack no walmart. `http://superuser.openstack.org/articles/inside-walmartlabs-and-its-openstack-core/`. Acessado: 08/05/2018.

[14] Website do cloudvamp. `http://www.grycap.upv.es/cloudvamp/index.php`. Acessado: 16/05/2018.

[15] Differentiated caching services; a control-theoretical approach. In *Proceedings of the The 21st International Conference on Distributed Computing Systems*, ICDCS '01, pages 615–, Washington, DC, USA, 2001. IEEE Computer Society.

[16] Marcelo Cerqueira Abranches and Priscila Solis. Um mecanismo de auto elasticidade com base no tempo de resposta para ambientes de computação em nuvem baseados em containers. In *Anais do XXXVI congresso da sociedade brasileira de computação*, pages 1959–1972, Porto Alegre, RS, Brazil, 2016. Sociedade Brasileira de Computação.

[17] Alirio Santos de Sá and Raimundo José de Araújo Macêdo. Qos self-configuring failure detectors for distributed systems. In Frank Eliassen and Rüdiger Kapitza, editors, *Distributed Applications and Interoperable Systems*, pages 126–140, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[18] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[19] Soodeh Farokhi, Pooyan Jamshidi, Ewnetu Bayuh Lakew, Ivona Brandic, and Erik Elmroth. A hybrid cloud controller for vertical memory elasticity: A control-theoretic

approach. *Future Generation Computer Systems*, 65:57 – 72, 2016. Special Issue on Big Data in the Cloud.

[20] Sandro Fiore, Donatello Elia, Cosimo Palazzo, Walter dos Santos Filho, Tulio Braga, Carlos Eduardo Pires, Demetrio Mestre, and Giovanni Aloisio. D4.4: Qos extensions for the big and fast data eco-system. Technical report, January 2018. `http://www.eubra-bigsea.eu/sites/default/files/D4.4_QoS%20extensions%20for%20the%20big%20and%20fast%20data%20eco-system_1.0.pdf`. Acessado: 17/06/2018.

[21] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, Inc., USA, 2004.

[22] C. V. Hollot, V. Misra, D. Towsley, and Wei-Bo Gong. A control theoretic analysis of red. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 3, pages 1510–1519 vol.3, 2001.

[23] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1991.

[24] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *J. Grid Comput.*, 12(4):559–592, December 2014.

[25] M. Mao, J. Li, and M. Humphrey. Cloud auto-scaling with deadline and budget constraints. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 41–48, Oct 2010.

[26] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, pages 423–430, Washington, DC, USA, 2012. IEEE Computer Society.

[27] Fabio Jorge Almeida Morais, Francisco Vilar Brasileiro, Raquel Vigolvino Lopes, Ricardo Araújo Santos, Wade Satterfield, and Leandro Rosa. Autoflex: Service agnostic

auto-scaling framework for iaas deployment models. In *CCGRID*, pages 42–49. IEEE Computer Society, 2013.

[28] Marco A. S. Netto, Carlos Cardonha, Renato L. F. Cunha, and Marcos D. Assuncao. Evaluating auto-scaling strategies for cloud computing environments. In *Proceedings of the 2014 IEEE 22Nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, MASCOTS '14, pages 187–196, Washington, DC, USA, 2014. IEEE Computer Society.

[29] Mina Sedaghat, Francisco Hernandez-Rodriguez, and Erik Elmroth. A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, pages 6:1–6:10, New York, NY, USA, 2013. ACM.

[30] L. Yazdanov and C. Fetzer. Vertical scaling for prioritized vms provisioning. In *2012 Second International Conference on Cloud and Green Computing*, pages 118–125, Nov 2012.