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

Escalonamento Adaptativo para Sistemas de Processamento Contínuo de Eventos

Rodrigo Duarte Sousa

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 de Computação

Andrey Elísio Monteiro Brito (Orientador)
Raquel Vigolvino Lopes (Orientadora)

Campina Grande, Paraíba, Brasil
©Rodrigo Duarte Sousa, 04/08/2014

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S725e Sousa, Rodrigo Duarte.
Escalonamento adaptativo para sistemas de processamento contínuo de eventos / Rodrigo Duarte Sousa. – Campina Grande, 2014.
73 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2014.

"Orientação: Prof. Dr. Andrey Elísio Monteiro Brito, Prof.ª Dr.ª Raquel Vigolvino Lopes".
Referências.

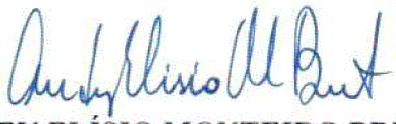
1. Sistemas Distribuídos. 2. Sistemas de Processamento de Fluxo de Eventos. 3. Desempenho. 4. Escalonamento. I. Brito, Andrey Elísio Monteiro. II. Lopes, Raquel Vigolvino. III. Título.

CDU 004.75(043)

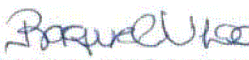
"ESCALONAMENTO ADAPTATIVO PARA SISTEMAS DE PROCESSAMENTO CONTÍNUO DE EVENTOS"

RODRIGO DUARTE SOUSA

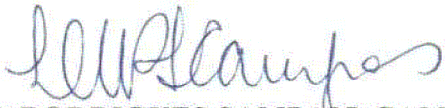
DISSERTAÇÃO APROVADA EM 04/09/2014



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



RAQUEL VIGOLVINO LOPES, D.Sc, UFCG
Orientador(a)



LÍVIA MARIA RODRIGUES SAMPAIO CAMPOS , D.Sc, UFCG
Examinador(a)

KEIKO VERÔNICA ONO FONSECA, D.Sc., UTFPR
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Sistemas de processamento contínuo de eventos vêm sendo utilizados em aplicações que necessitam de um processamento quase em tempo real. Essa necessidade, junto da quantidade elevada de dados processados nessas aplicações, provocam que tais sistemas possuam fortes requisitos de desempenho e tolerância a falhas. Sendo assim, escalonadores geralmente fazem uso de informações de utilização dos recursos das máquinas do sistema (como utilização de CPU, memória RAM, rede e disco), na tentativa de reagir a possíveis sobrecargas que possam aumentar a utilização dos recursos, provocando uma piora no desempenho da aplicação. Entretanto, devido aos diferentes perfis de aplicações e componentes, a complexidade de se decidir, de forma flexível e genérica, o que deve ser monitorado e a diferença entre o que torna um recurso mais importante que outro em um dado momento, podem provocar escolhas não adequadas por parte do escalonador. O trabalho apresentado nesta dissertação propõe um algoritmo de escalonamento que, através de uma abordagem reativa, se adapta a diferentes perfis de aplicações e de carga, tomando decisões baseadas no monitoramento da variação do desempenho de seus operadores. Periodicamente, o escalonador realiza uma avaliação de quais operadores apresentaram uma piora em seu desempenho e, posteriormente, tenta migrar tais operadores para nós menos sobrecarregados. Foram executados experimentos onde um protótipo do algoritmo foi avaliado e os resultados demonstraram uma melhora no desempenho do sistema, a partir da diminuição da latência de processamento e da manutenção da quantidade de eventos processados. Em execuções com variações bruscas da carga de trabalho, a latência média de processamento dos operadores foi reduzida em mais de 84%, enquanto que a quantidade de eventos processados diminuiu apenas 1,18%.

Abstract

The usage of event stream processing systems is growing lately, mainly at applications that have a near real-time processing as a requirement. That need, combined with the high amount of data processed by these applications, increases the dependency on performance and fault tolerance of such systems. Therefore, to handle these requirements, schedulers usually make use of the resources utilization (like CPU, RAM, disk and network bandwidth) in an attempt to react to potential overloads that may further increase their utilization, causing the application's performance to deteriorate. However, due to different application profiles and components, the complexity of deciding, in a flexible and generic way, what resources should be monitored and the difference between what makes a resource utilization more important than another in a given time, can provoke the scheduler to perform wrong actions. In this work, we propose a scheduling algorithm that, via a reactive approach, adapts to different applications profiles and load, taking decisions based at the latency variation from its operators. Periodically, the system scheduler performs an evaluation of which operators are giving evidence of being in an overloaded state, then, the scheduler tries to migrate those operators to a machine with less utilization. The experiments showed an improvement in the system performance, in scenarios with a bursty workload, the operators' average processing latency was reduced by more than 84%, while the number of processed events decreased by only 1.18%.

Agradecimentos

Este trabalho de dissertação foi orientado pelos professores Andrey Brito e Raquel Lopes, os quais agradeço pelos diversos ensinamentos, horas e horas de reunião, cobranças e enorme paciência. Agradeço também aos colegas do Laboratório de Sistemas Distribuídos e do CGHackspace, pelas incontáveis conversas e sábios conselhos que foram essenciais para a concepção desse trabalho.

Agradeço profundamente a minha família, em especial os meus pais Severina Duarte da Costa (Dona Nina) e Severino José de Sousa (Seu Sousa).

Agradeço aos meus amigos, mesmo que a ajuda não tenha sido diretamente relacionada ao conteúdo científico desse trabalho. Sempre estiveram comigo, comemorando os momentos de vitória e também dando suporte nos momentos difíceis e de tristeza.

Esse trabalho é dedicado a minha mãe, que despertou em mim a ambição que resultou na minha busca por uma pós-graduação e foi capaz de prover os meus estudos através de muito amor e trabalho árduo. Ela é a maior responsável por quem sou e pela pessoa que sonho um dia me tornar.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

O armazenamento e processamento de grandes contingentes de dados é um problema recorrente na Ciência da Computação. Entretanto, com o surgimento de novas aplicações como motores (*engines*) de buscas, redes sociais e mais recentemente com a chegada das chamadas sociedades inteligentes (*smart societies*), esse problema passou a ter um maior destaque [12; 14; 21].

Por um certo tempo, foram desenvolvidas soluções específicas para cada situação onde se necessitava processar um grande volume de dados. Naturalmente, para que sua computação possa ser realizada, tal processamento precisa ser distribuído em várias máquinas ou até mesmo *clusters*. Porém, devido à necessidade de distribuição/paralelização e consequentemente, tolerância a falhas, o desenvolvimento de tais soluções se torna uma tarefa complexa [18].

Em reação às dificuldades de desenvolvimento e utilização desses sistemas, surgiu o modelo de programação *MapReduce* [18], que recebeu boa aceitação no meio acadêmico e na indústria [17]. Nesse modelo de programação, ao se construir um novo sistema há apenas a obrigação de seguir algumas regras do modelo adotado e o arcabouço *MapReduce* lida com toda complexidade da distribuição do processamento, como também das falhas e erros que possam ocorrer durante toda execução. Dessa forma, há a completa separação de preocupações entre arcabouço e aplicação.

Entretanto, o modelo de programação *MapReduce* possui características que limitam o

perfil da aplicação a ser executada. Os dados são processados em lotes e geralmente não existe um prazo para que a saída seja fornecida. Além disso, fatores relacionados ao custo de sua inicialização e tratamentos da ordenação dos dados o tornam lento em certos casos [26]. Por exemplo, existem aplicações com requisitos de qualidade de serviço (*quality of service (QoS)*) bem mais restritos, a informação deve ser processada continuamente (a medida em que é gerada) e o tempo máximo para que respostas sejam produzidas não ultrapassa a faixa dos milissegundos, sendo menor até que o tempo que seria necessário para que a informação fosse persistida [17]. Redes de sensores [11; 15], detecção de fraudes em operações de cartões de crédito [12] e análise de transações no mercado financeiro [33] são exemplos de tais aplicações.

Processamento Contínuo de Eventos (*ESP*, do inglês *Event Stream Processing*) [6; 8; 12; 21] é um conjunto de técnicas criadas para solucionar os problemas citados acima e, atualmente, vêm sendo desenvolvidas diversas ferramentas que fornecem um arcabouço para a construção de tais sistemas: Borealis [3], Apache Storm[1], Apache S4 [31], Apache Samza[2], IBM InfoSphere [32] e StreamMine3G[3] [12; 29]. Esses sistemas são expressos como um grafo direcionado onde os vértices são chamados de *operadores* e as arestas representam conexões, nas quais os dados são processadas em fluxo. Conceitualmente, o processamento em sistemas *ESP* nunca termina, isto é, o fluxo de dados é *infinito*. Cada dado individual recebido e processado pelo sistema é chamado de *evento*.

Ao se processar informação continuamente, é necessário que sistemas *ESP* mantenham o fluxo de dados sempre estável, mesmo na presença de falhas: a perda de apenas um operador pode interromper a execução de todo o sistema e causar cenários indesejáveis [29].

1.2 Definição do Problema

Várias são as técnicas utilizadas em sistemas distribuídos no contexto de tolerância a falhas e entre as mais comuns, podemos citar replicação ativa e replicação passiva. Como objetivo principal, essas técnicas tentam evitar que a execução do sistema pare devido a falhas de *software* e/ou *hardware*. Em especial, o uso de replicação ativa é preferível em vários cená-

[1] https://storm.incubator.apache.org

[2] http://samza.incubator.apache.org

[3] https://streammine3g.inf.tu-dresden.de

rios, pois permite que o sistema continue sua execução sem interrupções, caso as falhas não afetem todas as réplicas de um mesmo processo distribuído. Outra vantagem do uso de replicação ativa é a possibilidade de melhora de desempenho ao se ter uma réplica mais rápida que outras (em uma máquina menos sobrecarregada, por exemplo), já que nesse modelo de mascaramento de falhas, os operadores que recebem as mensagens levam em consideração, na maioria dos casos, o primeiro evento, não importando a réplica de origem.

As vantagens descritas fazem com que a utilização de replicação ativa seja interessante em sistemas *ESP*. Outra prática comum é a atribuição de vários operadores diferentes a um mesmo nó físico para reduzir o desperdício de recursos. Naturalmente, essa prática deve seguir algumas regras, entre elas, a de se separar réplicas de um mesmo operador em nós diferentes. Além disso, cenários com variações bruscas de carga e perda de recursos podem acontecer [29]. Ao se processar mais eventos, o sistema pode ficar sobrecarregado e o seu desempenho ser depreciado, correndo o risco de não cumprir com os requisitos de desempenho da aplicação ou até causando a perda de dados.

Uma opção em cenários de sobrecarga, é o uso de migração: nós sobrecarregados podem ter processos migrados para outro nó que disponha de mais recursos livres, e assim, evitar que o desempenho seja afetado pela sobrecarga. Operadores em sistemas *ESP* possuem algumas características que tornam interessante o uso de migração. Quando o operador não possui estado, os resultados do processamento são baseados apenas no evento atual (eventos anteriores são desconsiderados) e por causa disso podem ser migrados frequentemente. Já para operadores com estado, a computação é realizada em janelas, onde apenas uma quantidade fixa de eventos é considerada por vez. Sendo assim, o final de uma janela de eventos proporciona um momento ideal para a migração do operador. Nesse instante, o operador envia o resultado do processamento para o operador da próxima etapa e inicia uma nova janela, se uma migração ocorrer antes que um novo processamento comece, não há a necessidade de que os dados sejam transferidos para o novo nó.

Dessa maneira, fica clara a necessidade de um mecanismo que possibilite a tolerância falhas na presença de mudanças de carga e considerando também o uso eficiente de recursos físicos, para que assim, o sistema evite que determinadas situações de sobrecarga afetem o desempenho ou que em cenários extremos, causem a perda de eventos.

1.3 Objetivo

Esta dissertação tem por objetivo analisar os efeitos da sobrecarga em sistemas *ESP* e assim propor estratégias de escalonamento que se adaptem em tempo de execução a possíveis variações de carga. Nesses sistemas, o gerente pode realizar o monitoramento da utilização dos recursos físicos e assim reagir caso algum nó ultrapasse certo limiar de utilização. Entretanto, esta pode não ser uma boa decisão se levarmos em consideração que: (i) o sistema pode estar em um estado de equilíbrio (sem variações de carga); (ii) aplicações diferentes podem ser afetadas por sobrecargas em recursos diferentes (CPU, memória RAM, disco, rede); (iii) limiares de sobrecarga podem variar de aplicação para aplicação. Identificar qual recurso pode estar afetando o desempenho e dinamicamente escolher um limiar diferente para cada operador é uma tarefa complexa e propensa a erros.

Nessa dissertação, é considerado que as aplicações estão executando em uma infraestrutura onde há disponível um número fixo de máquinas virtuais, aqui chamadas de nós. O arcabouço escolhido como caso de uso foi o *StreamMine3G*, o mesmo oferece a funcionalidade de migração de operadores e monitoramento de métricas de utilização e desempenho do sistema. Além disso, vem sendo desenvolvido por parceiros de pesquisa [29].

1.4 Resultados e Contribuições

Nesta dissertação é proposto um algoritmo de escalonamento adaptativo para sistemas *ESP*. A solução proposta foi motivada por um estudo inicial onde foi avaliado o impacto que a sobrecarga dos recursos de *hardware* dos nós possuem no desempenho do sistema. Nesse estudo, foram executados experimentos onde um sistema *ESP* era submetido a cenários de sobrecarga de diferentes recursos físicos, como CPU, memória RAM, rede e disco. Os resultados indicaram que todos os recursos avaliados afetaram negativamente o desempenho do sistema e, dessa forma, métricas que indiquem o estado atual da utilização desses recursos podem ser levados em consideração por um escalonador que se proponha a controlar o desempenho de tais sistemas.

Em particular, a solução proposta nessa dissertação leva em consideração apenas métricas de desempenho na escolha de quais operadores serão candidatos à migração. Além

disso, métricas de utilização de recursos físicos são observadas durante a escolha do nó que receberá o operador a ser migrado, assumindo que, ao migrar um operador para um nó com algum recurso mais sobrecarregado, seu problema de desempenho não seria solucionado. Os resultados mostram que o algoritmo proposto ocasionou um ganho na latência média dos operadores maior que 84% em cenários com carga de trabalho variada.

1.5 Organização do Documento

O conteúdo seguinte desta dissertação está dividido em mais quatro capítulos:

- **Capítulo 2 - Fundamentação Teórica e Estado da Arte.** Nesse capítulo, serão apresentados os conceitos principais da área de processamento contínuo de eventos. Bem como o estado da arte das pesquisas da área.

- **Capítulo 3 - Escalonamento Adaptativo.** O algoritmo de escalonamento adaptativo proposto neste trabalho de dissertação é apresentado nesse capítulo. Primeiramente é feito um estudo onde é avaliado o impacto que a sobrecarga dos recursos físicos possuem no desempenho de um sistema de processamento contínuo de eventos. Em seguida, os mecanismos de monitoramento, migração e detalhes do algoritmo proposto são detalhados.

- **Capítulo 4 - Avaliação e Resultados.** Os métodos utilizados na avaliação do algoritmo de escalonamento adaptativo e de seus parâmetros são descritos nesse capítulo. Em seguida, são apresentados e analisados os resultados.

- **Capítulo 5 - Conclusão.** Nesse capítulo são apresentadas as conclusões, as contribuições deste trabalho de dissertação para a área de escalonamento em sistemas de processamento contínuo de eventos, as limitações e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica e Estado da Arte

Neste capítulo são apresentados os conceitos básicos relativos a sistemas *ESP*. Inicialmente, são introduzidos os conceitos arquiteturais básicos, como os diferentes componentes e as mensagens trocadas entre eles. Em seguida, é detalhado o conceito de operador e seus tipos de janelas de processamento: de salto e deslizante. No final do capítulo, é feito um levantamento da literatura sobre estratégias de escalonamento em sistemas de processamento contínuo de eventos.

2.1 Sistemas de Processamento Contínuo de Eventos

Sistemas *ESP* possuem uma ou mais fontes que geram dados a serem processados de forma contínua. A sequência de eventos produzidos é chamado de fluxo de dados. Uma característica marcante de tais sistemas, é o forte requisito de latência (ou tempo de resposta). A rapidez com que a saída é fornecida é de extrema importância para esses sistemas. Adicionalmente, a taxa de chegada de eventos pode aumentar ou diminuir rapidamente durante a execução e gerar cenários de sobrecarga, causando uma piora do desempenho e quebras de requisitos de *QoS*.

A arquitetura utilizada por diversos sistemas *ESP*, entre eles, o StreamMine3G está ilustrada na Figura 2.1. Maiores detalhes sobre o StreamMine3G podem ser encontrados em [7; 12; 29].

Nessa arquitetura, o processamento é realizado em uma sequência de etapas. Uma etapa é representada por um componente chamado de *operador*, que por sua vez, é *replicado* entre

os diferentes nós da infraestrutura. Existe também a presença de um componente chamado *gerente*, que é o responsável pela distribuição inicial da topologia da aplicação, monitoramento de métricas do sistema e escalonamento das réplicas após o início da execução.

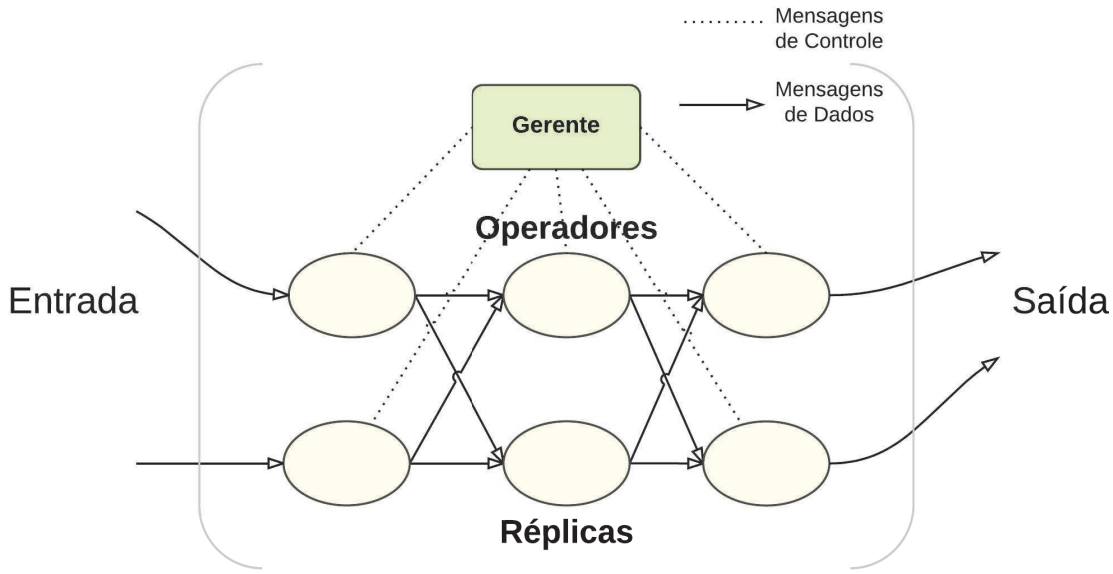


Figura 2.1: Arquitetura dos componentes da ferramenta StreamMine3G.

Existem dois tipos principais de mensagens trocadas entre os componentes do sistema:

1. **Mensagens de Controle**: enviadas periodicamente pelos operadores para o gerente informando estatísticas de desempenho e de utilização de recursos do nó em que estão localizados.

2. **Mensagens de Dados**: possuem os eventos originados das fontes de dados ou do resultado de alguma operação realizada pelos operadores. Possuem um número de sequência único que as identifica em todo o sistema. Outras informações de controle também podem ser inseridas nessas mensagens para serem utilizadas pelo arcabouço.

2.2 Operadores

Operadores são as menores unidades de processamento em um sistema *ESP* e representam uma etapa no fluxo das operações realizadas sobre o fluxo de dados. Existem diversos tipos

de operações e entre as mais comuns podemos citar: filtragem (seleção de eventos que podem avançar para a próxima etapa), agregação (cálculo de médias móveis ou casamento de padrões), transformação (conversão entre unidades) e até algumas simplificações ou adaptações de algoritmos de mineração de dados (*frequent items*, elementos únicos, histogramas, etc) [23].

Em algumas operações, como a filtragem, não há necessidade de se guardar estado, eventos anteriores não fazem parte do processamento do evento atual (operadores sem estado). Entretanto, no cálculo da média móvel, por exemplo, eventos processados anteriormente são considerados na operação e um estado deve ser mantido (operadores com estado).

Nos operadores com estado, o processamento geralmente se dá em janelas. Gráficos que ilustram o funcionamento de janelas estão exibidos nas Figuras 2.2 e 2.3.

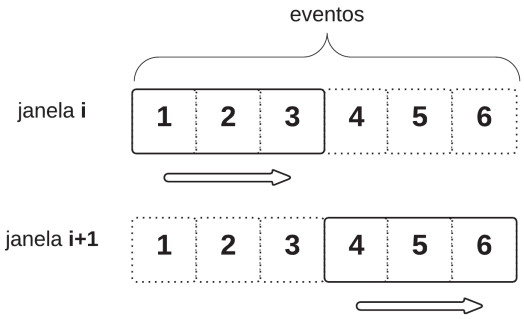


Figura 2.2: Processamento de um operador a partir de janelas de salto.

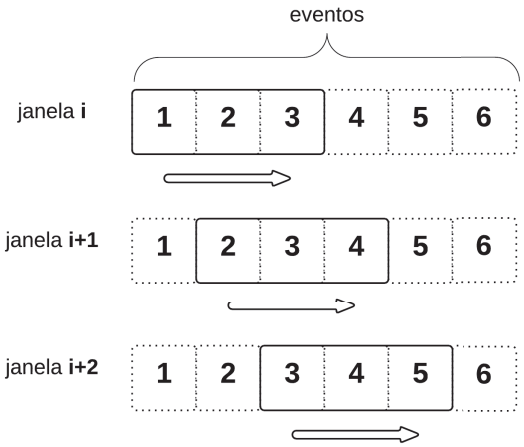


Figura 2.3: Processamento de um operador a partir de janelas deslizantes.

Nas janelas de salto, não há sobreposição de eventos: o estado do operador é inteiramente descartado no final do processamento e uma nova janela começa a ser produzia a partir do próximo evento. Já nas deslizantes, há um parâmetro de sobreposição, a próxima janela possuirá um ou mais (mas não todos) eventos da janela atual.

2.3 Estado da Arte

O escalonamento em sistemas *ESP* consiste no gerenciamento da topologia da aplicação e de acordo com o momento em que as ações são realizadas pelo escalonador, pode ser essencialmente classificado de duas maneiras: (i) estático, quando é realizado antes do início da execução do sistema; (ii) dinâmico, quando é realizado enquanto a aplicação está executando. Em escalonadores dinâmicos geralmente é realizado o monitoramento de informações que indicam o estado do sistema em um determinado momento. De posse dessas informações, o escalonador pode tomar medidas que minimizem os efeitos negativos sempre que um estado indesejável é detectado. Ações essas, que podem variar desde a migração de operadores entre os nós do sistema, ou até o descarte dos eventos recebidos pelo sistema. Com relação ao projeto de algoritmos de escalonamento, um fator a ser considerado é a simplicidade da solução, em [19], foi mostrado que algoritmos de escalonamento simples podem gerar um ganho representativo no desempenho do sistema e a medida que a complexidade dos algoritmos aumenta, o ganho passa a se estabilizar, chegando a diminuir em certos casos.

Uma estratégia de detecção de sobrecarga utilizada em escalonadores dinâmicos é através do uso de limiares de utilização dos recursos físicos ou tamanho de filas de entrada ou saída dos componentes do sistema. Sempre que uma determinada métrica ultrapassar um limiar preestabelecido, o escalonador deve tomar providências para que a possível situação de sobrecarga seja resolvida. Martin et al. [29], utiliza uma estratégia onde cada operador possui duas réplicas ativas: uma primária e outra secundária. Nessa estratégia, uma réplica primária de um operador e uma réplica secundária de um outro operador são atribuídas ao mesmo nó, quando a utilização de CPU do nó ultrapassa um certo valor, a réplica secundária é desativada e passa apenas a persistir os eventos recebidos, como se fosse uma réplica passiva. Em outros trabalhos, como [27], é iniciado um balanceamento da carga dos operadores quando é detectado que a taxa de chegada de eventos ultrapassou um limiar pré-configurado,

os eventos excedentes são enviados para um processador externo com o intuito de manter a vazão do sistema dentro dos valores aceitáveis. Entretanto, o uso de limiares de utilização como regra para detecção de sobrecarga possui algumas limitações. Escalonadores que fazem uso dessa estratégia podem disparar ações desnecessárias porque uma métrica monitorada ultrapassou um certo valor, mesmo quando há uma estabilidade no sistema e nada precisa ser feito.

De forma semelhante a [29], a ferramenta Borealis [3; 16; 25] utiliza replicação ativa. No Borealis, o processamento é baseado em consultas e dessa forma, os operadores são entidades genéricas que se adaptam dinamicamente à consulta realizada e ao enfrentar situações de sobrecarga, o sistema pode realizar o descarte de mensagens (*load shedding*) na tentativa de manter o desempenho do sistema em um nível aceitável. Esse controle é feito a partir de uma abordagem, conhecida como *pull*, onde as mensagens são enviadas apenas após a autorização do operador destino. Em outros trabalhos, como [12] e [31], é proposto um mecanismo de balanceamento de carga através de um esquema de atribuição de chaves, semelhante ao que é feito no paradigma *MapReduce* [18]. Todos os eventos do sistema possuem uma chave atribuída a eles e os operadores, por sua vez, são responsáveis pelo processamento de uma coleção limitada dessas chaves. Através dessa abordagem, o sistema pode ser facilmente escalado: quando alguma sobrecarga é detectada a partir do uso de limiares de utilização de CPU ou de taxa de chegada de eventos, os operadores podem ser divididos, reduzindo a quantidade de chaves que os mesmos são responsáveis. Adicionalmente, o S4 também realiza o descarte de mensagens.

Para várias aplicações onde erros podem gerar prejuízos, como de compra e venda de ações no mercado financeiro, pode ser que o descarte de mensagens não seja uma opção, impedindo assim, o uso de arcabouços que utilizam tal estratégia. Com relação ao uso de chaves para o controle da aplicação, existem algumas limitações. Muitas vezes, a divisão de um operador pode ser desnecessária, a migração do operador para outro nó menos sobrecarregado poderia resolver o problema naquele instante. Além disso, essa abordagem pode provocar com que a carga de trabalho fique desbalanceada entre as diferentes máquinas do sistema, causando assim, que certos operadores tenham seu desempenho afetado negativamente.

Em arcabouços como o Apache Storm [2] e Apache Samza [1], existe a garantia de

que as mensagens, uma vez no sistema, serão processadas. No Storm, essa garantia é dada a partir de reenvios de mensagens que não receberam uma confirmação de processamento dentro de um prazo pré-configurado. No Samza, é realizado um esquema de recuperação de mensagens no caso falhas, através da persistência das mesmas em um sistema de arquivos e, para isso, a ferramenta Kafka [28] é utilizada. Nenhuma das duas ferramentas disponibilizam mecanismos de balanceamento dinâmico de carga e de forma semelhante ao Borealis, o controle é realizado através de uma abordagem *pull*.

O problema de balanceamento de carga e manutenção de desempenho também é comum para outros tipos de aplicações distribuídas. Como, por exemplo, atribuição de máquinas virtuais a recursos físicos em ambientes de *computação na nuvem*. Nesse contexto, não é possível que o escalonador monitore métricas de desempenho, como latência, das aplicações que estão em execução na infraestrutura e normalmente, também não é possível monitorar algumas métricas de utilização dos recursos físicos. Por isso, geralmente são propostas soluções alternativas para o escalonamento de máquinas virtuais. Em [5], é proposto um algoritmo que realiza migrações de máquinas virtuais entre um conjunto fixo de máquinas físicas. O algoritmo não faz de uso limiares estáticos de utilização dos recursos físicos, mas de seu histórico de utilização que, a partir de alguns cálculos que identificam quais nós da infraestrutura possuíram um aumento significativo e permanente da sua carga de trabalho, máquinas virtuais são migradas para nós menos sobrecarregados. Já em [20], as métricas de utilização dos recursos físicos das máquinas virtuais são utilizadas para prever a latência da aplicação que está sendo executada. Caso a latência esteja acima dos valores definidos nos acordos de *QoS*, mais recursos podem ser alocados para a execução da aplicação. A ideia de utilizar a latência se encaixa bem no escalonamento de sistemas *ESP*, principalmente pela sensibilidade à variação de carga que tais sistemas possuem.

Tendo em vista as limitações apresentadas pelas abordagens de escalonamento encontradas na literatura, principalmente pela falta de mecanismos que proporcionem uma manutenção do desempenho das aplicações em execução, como também, um melhor balanceamento da utilização dos recursos da infraestrutura, sobretudo na presença de sobrecarga. A solução proposta nesse trabalho visa preencher essa lacuna, propondo um algoritmo de escalonamento que se adapta a diferentes tipos de aplicações e de carga do sistema. Uma indício de que algum operador possa estar em um estado de sobrecarga é justamente um aumento

na sua latência. Adicionalmente, o atraso em um operador específico, pode gerar um atraso geral no sistema.

2.4 Considerações Finais

Neste capítulo foram introduzidos os conceitos fundamentais para o entendimento de sistemas ESP. Foi apresentada a arquitetura utilizada pelo arcabouço StreamMine3G e os diferentes tipos de mensagens trocadas no sistema: de controle e de dados. Em seguida, foi detalhado o conceito de operador e algumas de suas caraterísticas, como a presença ou não de um estado e a abordagem de processamento em janelas.

Na Seção 2.3 foi feita uma análise do estado da arte do problema investigado neste trabalho. Primeiramente foi feito um levantamento dos arcabouços utilizados na construção de sistemas *ESP*, onde foi observado algumas lacunas nas estratégias utilizadas, principalmente no que diz respeito ao desempenho da aplicação em execução. Também foram analisadas algumas estratégias de escalonamento utilizadas na literatura, não só de sistemas *ESP*, como também no contexto de escalonamento de máquinas virtuais em ambientes de *computação na nuvem*.

No próximo capítulo será apresentado o algoritmo de escalonamento proposto neste trabalho de dissertação. Inicialmente, os experimentos de avaliação do impacto da utilização dos recursos são detalhados, para que em seguida, a estratégia de escalonamento adaptativo proposta seja detalhada.

Capítulo 3

Escalonamento Adaptativo

Neste capítulo será apresentado o algoritmo de escalonamento adaptativo proposto neste trabalho de dissertação. Primeiramente é apresentado um estudo que motiva as decisões de projeto do algoritmo e que está relacionado à avaliação do impacto da sobrecarga de diferentes recursos no desempenho do sistema. Posteriormente as etapas do algoritmo são detalhadas.

3.1 Impacto da Utilização dos Recursos no Desempenho

Como citado no Capítulo 2, os operadores em sistemas *ESP* podem realizar diversos tipos diferentes de operações. Um exemplo é uma aplicação que realiza análise de sinais coletados por sensores em redes elétricas [10]. Tal aplicação pode, por exemplo, ter operadores responsáveis pela sumarização dos dados coletados, análise da presença de situações extremas que podem disparar algum tipo de alarme e a persistência desses dados para análise posterior. Em consequência disso, é possível que esses operadores utilizem de forma diferente os recursos físicos dos nós em que estão localizados.

Para melhorar o entendimento dessa relação entre sobrecarga em diferentes recursos físicos e o efeito negativo no desempenho dos operadores, foram realizados alguns experimentos onde eram executadas operações com o objetivo principal de sobrecarregar um recurso por vez e, consequentemente, avaliar o impacto no desempenho dos operadores. Formalmente, temos como variáveis *independentes*, ou de controle, as utilizações dos recursos do sistema. Como variável *dependente*, a latência do processamento dos operadores.

Os seguintes fatores foram considerados nos experimentos: taxa de chegada de eventos (λ), número de operadores (n_{op}) e tempo de duração de cada réplica do experimento (t). O tipo de processamento realizado pelos operadores também varia de acordo com o recurso mais usado na computação e que, consequentemente, fica sobrecarregado. Em cada cenário foram executadas 30 réplicas para que fossem obtidos os intervalos de confiança utilizados nas comparações entre os mesmos.

Adicionalmente, todos os experimentos foram executados em uma infraestrutura de *computação na nuvem privada*. Na Figura 3.1 está detalhada a topologia dos experimentos:

- Fonte: gerador de eventos.
- Worker: responsável pelo processamento dos eventos que chegam da *fonte* e é o operador observado durante a coleta de métricas de utilização e desempenho. Após o processamento, envia os eventos para o *sorvedouro*.
- Sorvedouro: recebe os eventos já processados pelo *worker*. Responsável apenas pelo cálculo da latência. Para um cálculo preciso da latência, as máquinas virtuais foram sincronizadas via *Network Time Protocol* (NTP).

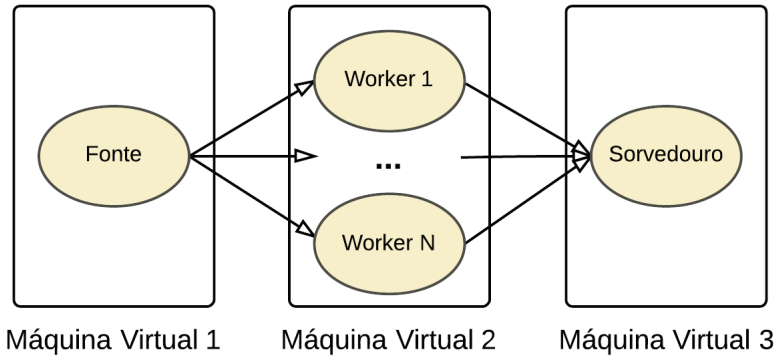


Figura 3.1: Topologia dos experimentos de sobrecarga de recursos.

Os protótipos para avaliação foram desenvolvidos na linguagem Java 1.6 e as três máquinas virtuais utilizadas executavam Ubuntu Server 12.04 LTS (Precise Pangolin), com as seguintes configurações:

- Máquina Virtual 1: 2 VCPUs, 2 GB de memória RAM e 100 GB de disco (magnético).
- Máquina Virtual 2: 1 VCPU, 1 GB de memória RAM e 50 GB de disco (magnético).
- Máquina Virtual 3: 2 VCPUs, 2 GB de memória RAM e 100 GB de disco (magnético).

Não há a presença do gerente. Os dados das *mensagens de controle* são logados pelos próprios operadores.

3.1.1 Sobrecarga de CPU

A Utilização de CPU é a métrica mais explorada em diferentes trabalhos [4; 9; 13; 29] e muitas vezes é considerada como única métrica na detecção de sobrecarga. Quando a utilização está muito alta, a fila de eventos a serem processados pelos operadores pode aumentar e consequentemente gerar um aumento no tempo de processamento dos mesmos.

Para variar apenas a utilização de CPU durante o experimento, a única operação realizada pelo operador ao receber um evento é uma espera ocupada de 1 *ms*. Como o tempo de serviço do operador é praticamente constante, a variação da utilização de CPU ocorre quando a quantidade de eventos a serem processados aumenta. Os cenários estão sumarizados na Tabela 3.1.

Cenário	λ *(eventos/s)*	n_{op}	*t (s)*
1	100	1	120
2	250	1	120
3	500	1	120
4	900	1	120

Tabela 3.1: Cenários do experimento de sobrecarga de CPU

Como mostra a Figura 3.2, o crescimento da taxa de chegada de eventos gerou um aumento considerável na utilização de CPU: sendo menor que 20% no Cenário 1 e partindo para mais de 90% no Cenário 4. Já a utilização de memória RAM, exibido pela Figura 3.3, permaneceu estável dado que a operação utilizada não realizava alocação de memória RAM.

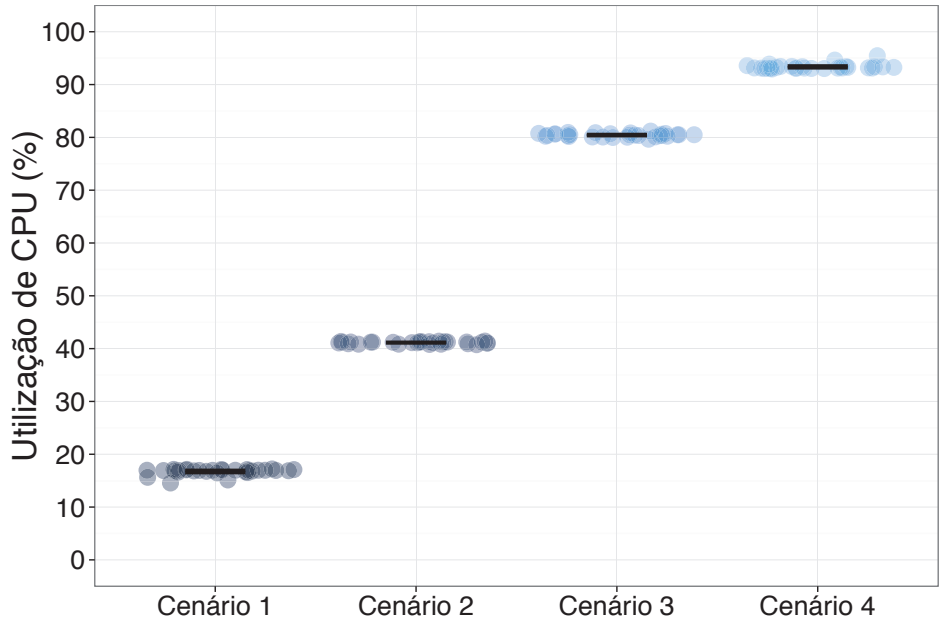


Figura 3.2: Utilização média de CPU para as réplicas dos cenários do experimento de sobrecarga de CPU.

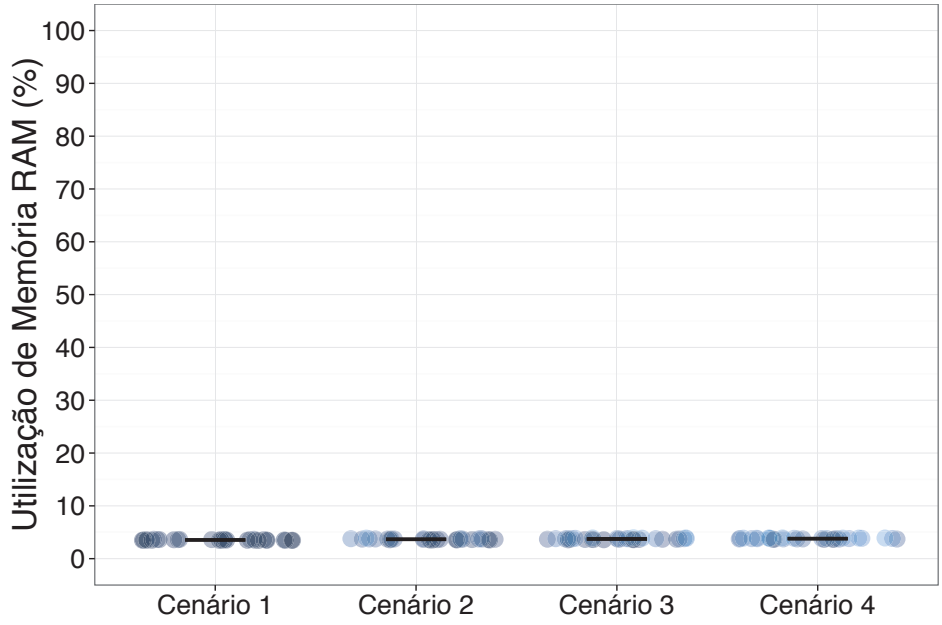


Figura 3.3: Utilização média de Memória RAM para as réplicas dos cenários do experimento de sobrecarga de CPU.

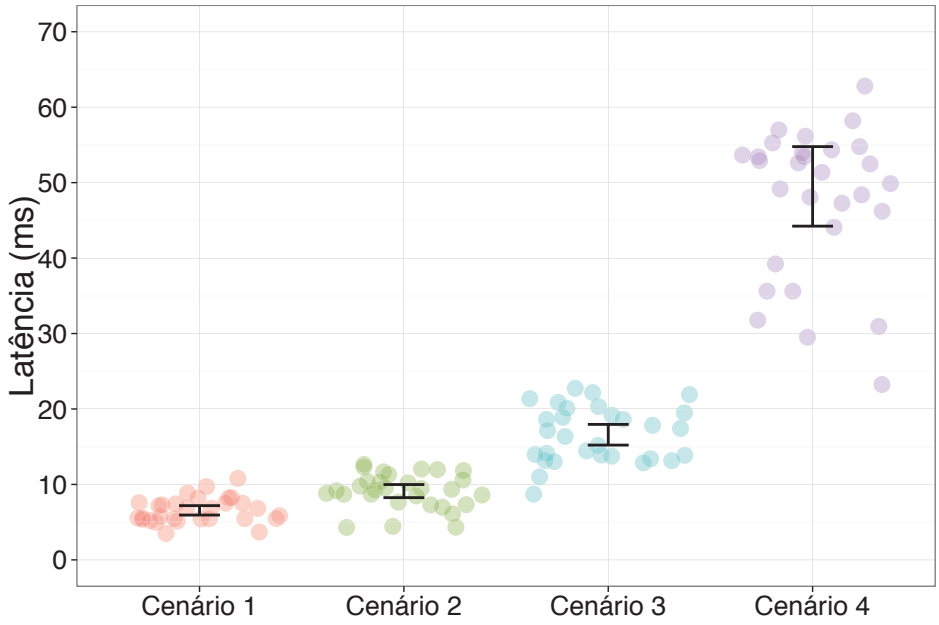


Figura 3.4: Intervalos de confiança da latência nos cenários do experimento de sobrecarga de CPU.

Como dito anteriormente, é esperado que à medida que a utilização de CPU aumente, também exista o crescimento da fila de eventos a ser processada pelo operador. Devido a uma maior espera na fila, há o aumento da latência de processamento dos eventos, o que justifica o crescimento da latência exibido na Figura 3.4.

3.1.2 Sobrecarga de memória RAM

A sobrecarga da memória RAM do sistema pode ocorrer de duas maneiras: a partir do aumento da quantidade de memória utilizada e através de diferentes processos competindo pela banda de memória (do inglês, *memory bandwidth*).

Quantidade de memória utilizada

Quanto maior é a ocupação da memória RAM, mais operações de paginação são realizadas pelo sistema operacional na prevenção de situações de escassez total de memória. O maior *overhead* acontece quando a aplicação tenta acessar uma página que foi transferia para a área de *swap* do disco, que nessas situações, provoca um aumento na latência. Além disso, a CPU passa a ser mais requisitada devido ao maior número de operações de transferência de página.

Outro fator que deve ser levado em consideração, é a presença de *Out of Memory (OOM) killers*, que destroem processos que estão utilizando muita memória RAM nos momentos em que a mesma está próxima de ser completamente ocupada.

Na tentativa de variar apenas a quantidade de memória RAM utilizada, o operador efetua o carregamento de uma estrutura de dados de tamanho fixo, que para o cenário executado, ocupava diferentes quantidades de memória. Ao receber um evento, o operador realiza 100 acessos em posições aleatórias dessa estrutura de dados na tentativa de provocar que páginas sejam carregadas da área de *swap*. O *OOM killer* foi desativado para que não interferisse durante as execuções do experimento. Na Tabela 3.2 estão sumarizados os cenários executados.

Cenário	**Tamanho da Estrutura de Dados *(MB)***	**λ *(eventos/s)***	**n_{op}**	***t (s)***
1	256	100	1	300
2	512	100	1	300
3	768	100	1	300

Tabela 3.2: Cenários do experimento de sobrecarga de CPU.

Na Figura 3.5 estão ilustrados os resultados de utilização média de CPU. Como a taxa de chegada de eventos não sofreu variação entre os cenários do experimento, não existiu uma mudança brusca na utilização desse recurso, partindo de 10% no Cenário 1 e não ultrapassando 15% no Cenário 3. Essa variação se deve principalmente ao maior número de operações de *swap*, resultado do crescimento da utilização de memória RAM.

Como descrito acima, o operador fazia o carregamento de uma estrutura de dados que variava de tamanho em cada cenário. O impacto dessa variação na utilização de memória RAM pode ser visto na Figura 3.6.

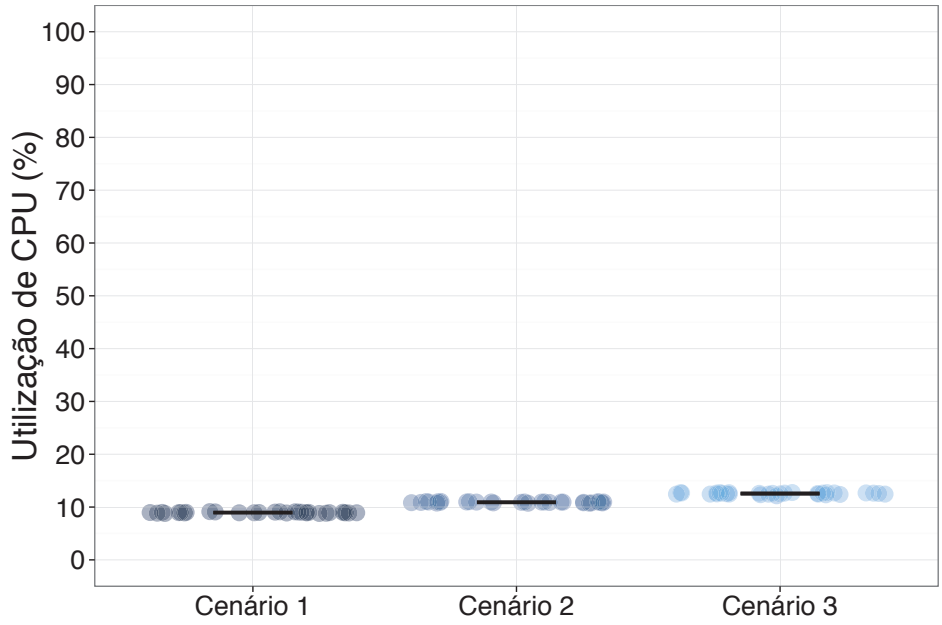


Figura 3.5: Utilização média de CPU para as réplicas dos cenários do experimento de sobrecarga de memória RAM.

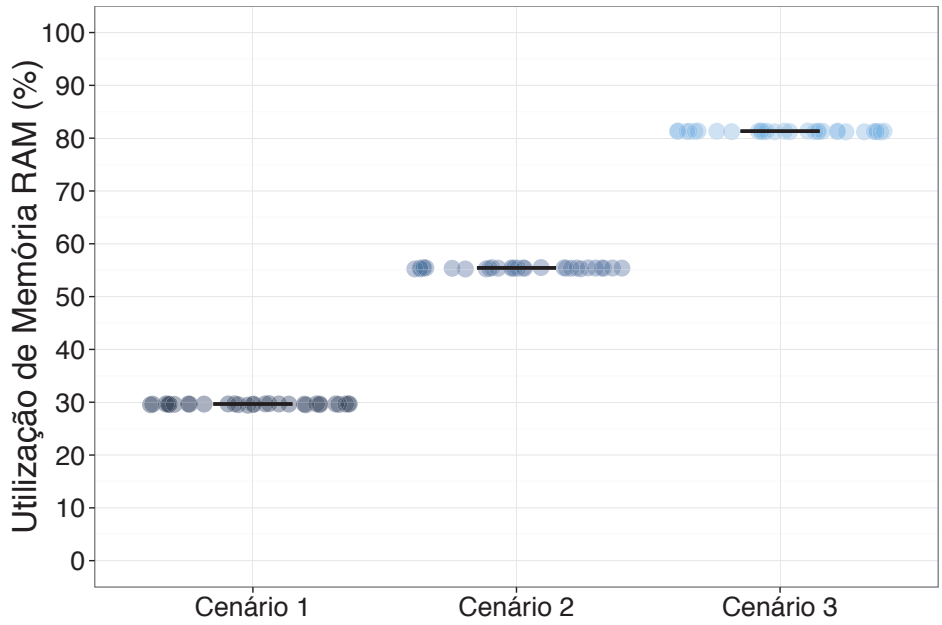


Figura 3.6: Utilização média de Memória RAM para as réplicas dos cenários do experimento de sobrecarga de memória RAM.

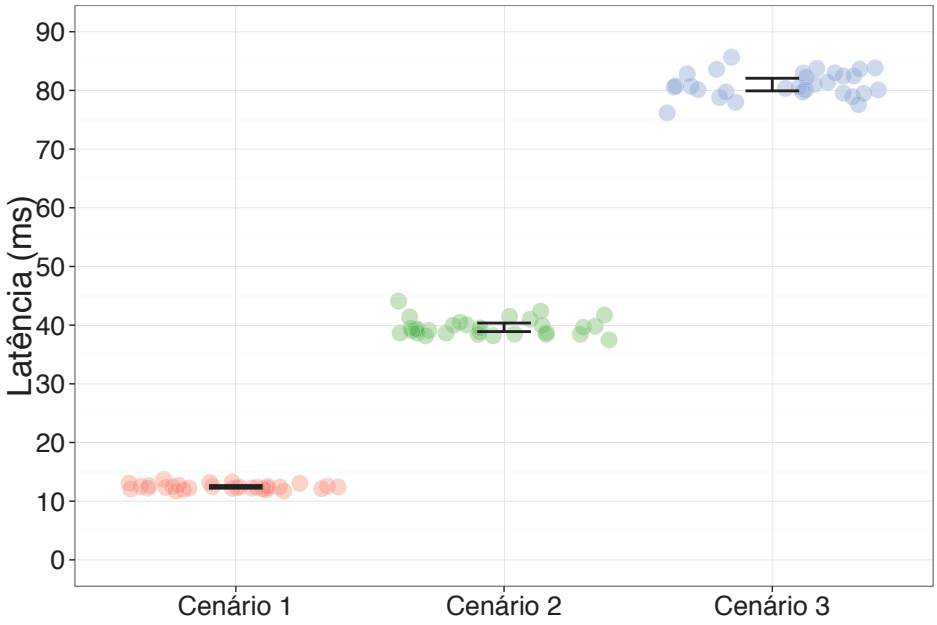


Figura 3.7: Intervalos de confiança da latência nos cenários do experimento de sobrecarga de memória RAM.

As latências de cada cenário podem ser visualizadas na Figura 3.7. Como esperado, a maior utilização de memória RAM provocou que páginas fossem transferidas para área de *swap* e portanto, tiveram um impacto negativo na latência ao serem novamente acessadas pela aplicação.

De forma oposta ao experimento anterior, houve uma piora no desempenho do operador, mas sem que houvesse um aumento considerável na utilização média de CPU. Em um cenário como esse, um escalonador adaptativo que baseia suas ações em limiares fixos de uso de CPU, pode não tomar boas decisões de escalonamento e assim provocar que a aplicação alcance níveis não aceitáveis de latência.

Competição pela banda de memória

A banda de memória representa a taxa com que a memória RAM consegue efetuar a leitura e escrita de dados. Essa capacidade pode ser esgotada quando um processo realiza muitos acessos sequenciais e, consequentemente, outros processos que tentem acessar esse recurso terão que esperar pela sua liberação.

Na tentativa de sobrecarregar esse recurso, quando recebe um evento, o operador realiza a

cópia de uma estrutura de dados de 2 MB entre áreas distintas da memória. Essa abordagem é baseada em *benchmarks* da banda de memória RAM, como o *MBW*[1]. Entre os cenários há a adição de outros operadores realizando o mesmo tipo de operação. A Tabela 3.3 exibe as configurações dos parâmetros utilizados nos diferentes cenários.

Cenário	λ *(eventos/s)*	n_{op}	*t (s)*
1	100	1	120
2	100	2	120
3	100	4	120

Tabela 3.3: Cenários do experimento de sobrecarga da banda de memória RAM

Nas Figuras 3.8 e 3.9 estão ilustrados os resultados de utilização média de CPU e memória RAM, respectivamente. É possível notar um pequeno aumento na utilização dos dois recursos, mesmo sem variação da taxa de chegada de eventos ou tipo de operação realizada pelo operador. Uma possibilidade é que as trocas de contexto causadas pelo maior número de operadores possa ser responsável pelo aumento da utilização média de CPU. Já a variação de utilização de memória RAM pode ser justificado por alguma demora na liberação da memória RAM por parte do *Java Garbage Collector*[2].

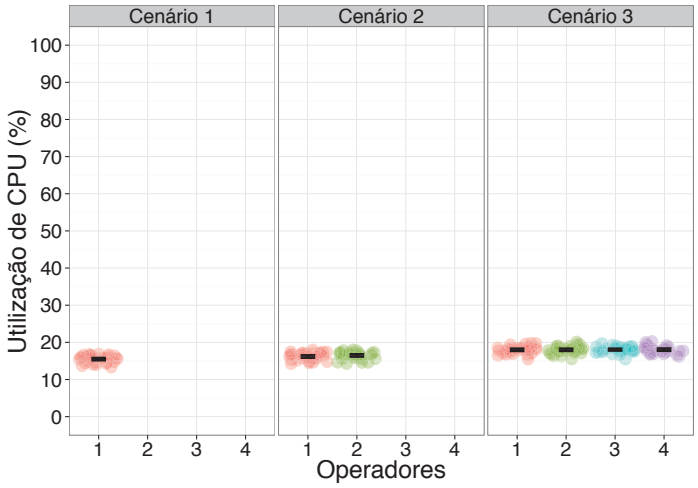


Figura 3.8: Utilização média de CPU para as réplicas dos cenários do experimento de sobrecarga da banda de memória RAM.

[1]https://github.com/raas/mbw

[2]http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

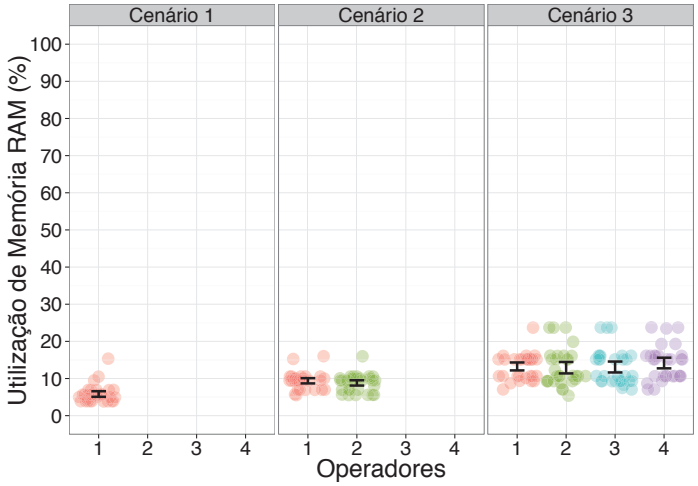


Figura 3.9: Utilização média de memória RAM para as réplicas dos cenários do experimento de sobrecarga da banda de memória RAM.

A Figura 3.10 mostra que a latência sofreu uma pequena variação entre os cenários executados. A variação observada não pode ser atribuída exclusivamente à competição de banda de memória, como também aconteceu um pequeno aumento na utilização de CPU e memória RAM, não está claro qual recurso foi o responsável por essa variação.

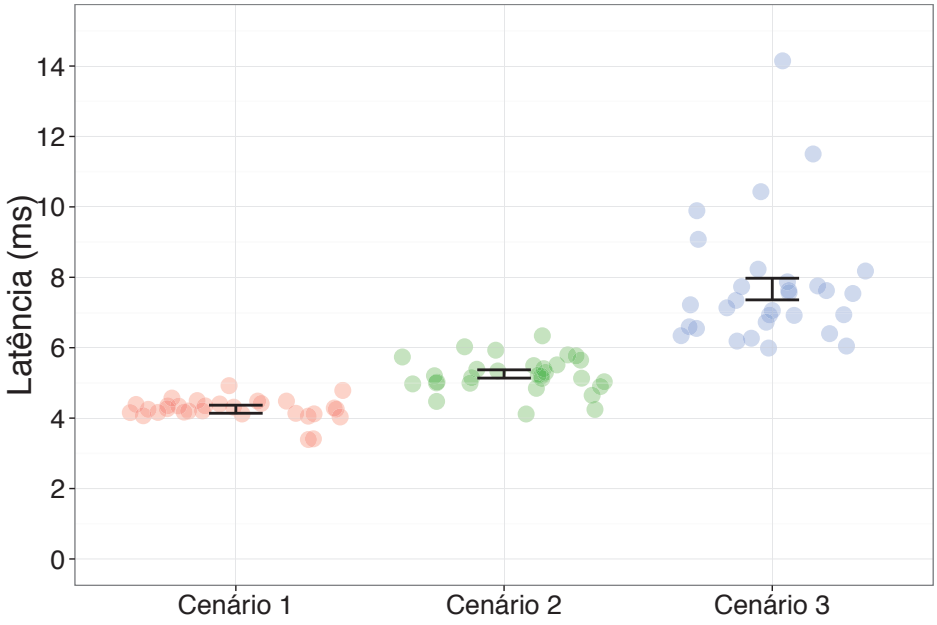


Figura 3.10: Intervalos de confiança da latência nos cenários do experimento de sobrecarga da banda de memória RAM.

3.1.3 Sobrecarga de Rede

Em cenários onde há um aumento de carga substancial em um ou mais operadores localizados em um mesmo nó, a banda de rede pode ser completamente utilizada e pacotes serem perdidos e/ou sofrerem atraso.

Nos experimentos de sobrecarga de rede, a largura de banda, para *download*, da máquina virtual do *worker* foi manualmente limitada em 100 Kbps (através da ferramenta Wonder Shaper[1]). Consequentemente, através do aumento da quantidade de dados recebidos pelo operador (visto na Figura 3.12), a banda foi completamente utilizada. Detalhes das configurações dos cenários podem ser vistos na Tabela 3.4. O operador não realizava qualquer processamento ao receber um evento, que era apenas repassado para o *sorvedouro*.

Cenário	λ ***(eventos/s)***	n_{op}	***t (s)***
1	500	1	120
2	1000	1	120
3	1500	1	120

Tabela 3.4: Cenários do experimento de sobrecarga de rede.

Devido ao aumento da taxa de chegada de eventos, a utilização de CPU (ilustrada na Figura 3.11) também cresceu partindo de um valor próximo a 20% no Cenário 1 e chegando a faixa de 40% no Cenário 3.

Na Figura 3.13, são ilustradas as latências resultantes de cada cenário. É possível notar que a latência no Cenário 3 foi superior a dos outros, como era esperado. Nesse cenário, a capacidade da rede foi completamente utilizada. Entretanto, como também foi observado um aumento da utilização de CPU, se faz necessária uma investigação para verificar se o crescimento na latência pode ser atribuído à sobrecarga de rede, ou se foi causado pela maior utilização de CPU. Uma comparação dos intervalos de confiança da latência do experimento de sobrecarga apenas de CPU em comparação ao de sobrecarga de rede pode ser visto na Figura 3.15. Como pode ser visto na Figura 3.14, foram escolhidos cenários que apresentaram uma média de utilização de CPU semelhante.

[1] https://github.com/magnific0/wondershaper

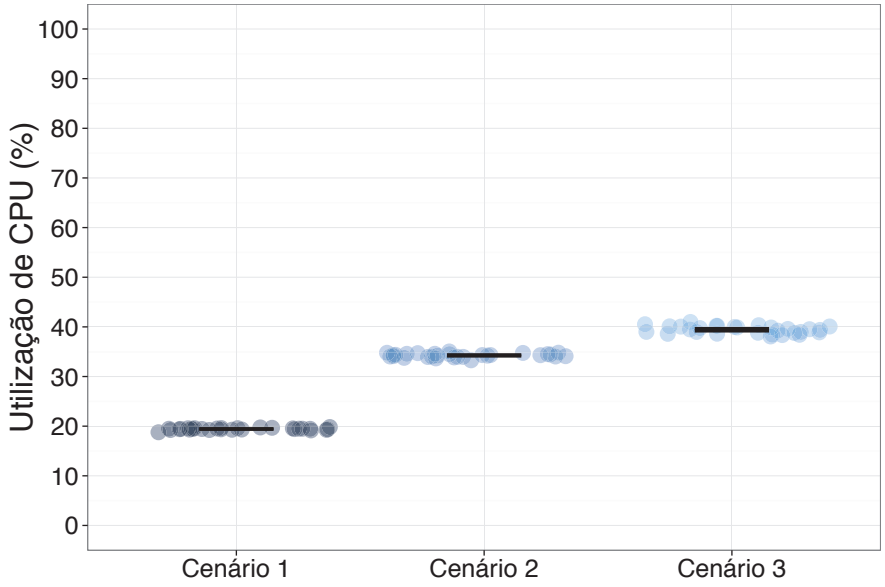


Figura 3.11: Utilização média de CPU para as réplicas dos cenários do experimento de sobrecarga de rede.

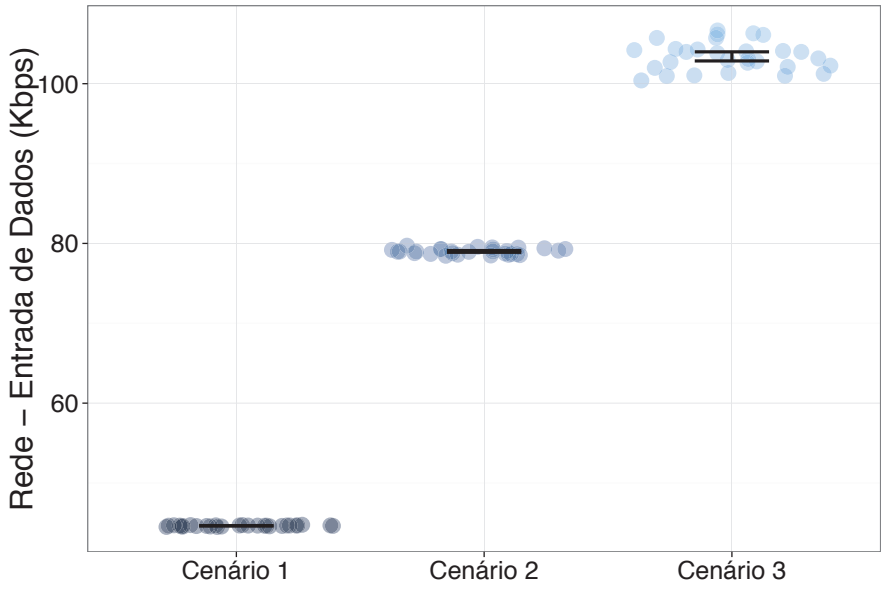


Figura 3.12: Quantidade de dados (em Kbps) chegando ao operador nos cenários do experimento de sobrecarga de rede.

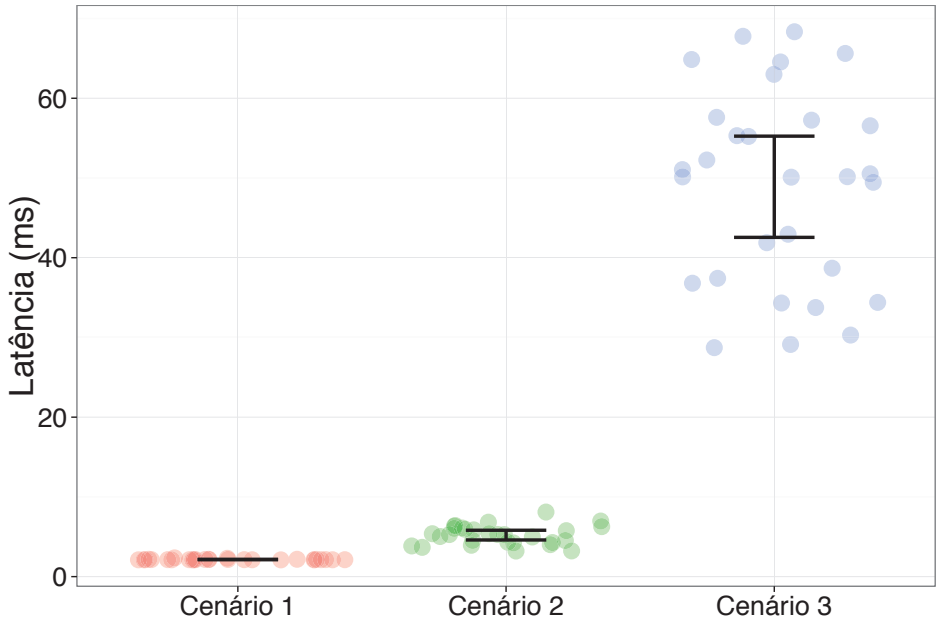


Figura 3.13: Intervalos de confiança da latência nos cenários do experimento de sobrecarga de rede.

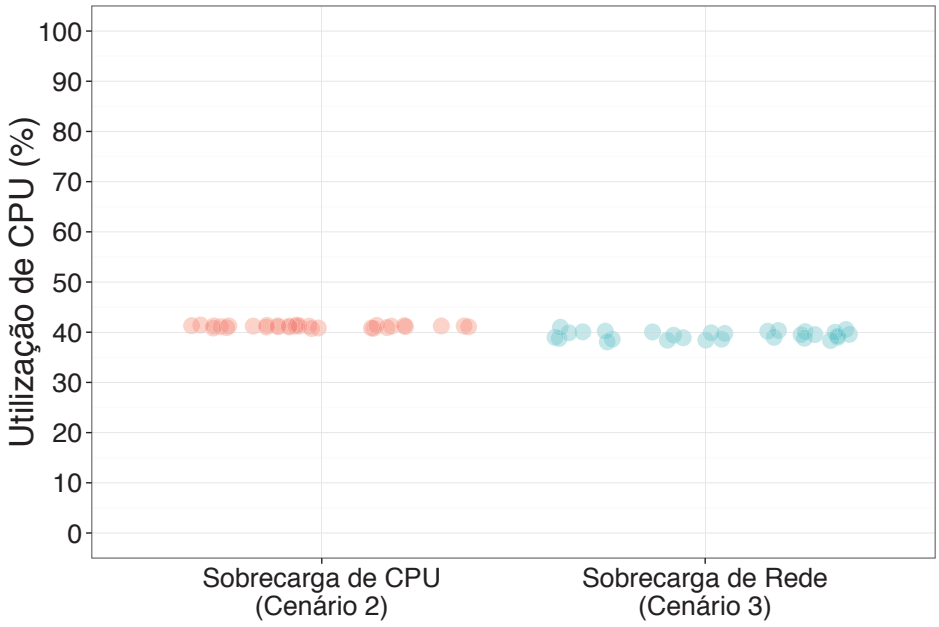


Figura 3.14: Comparação da utilização média de CPU para os cenários comparados dos experimentos de sobrecarga de CPU e de rede.

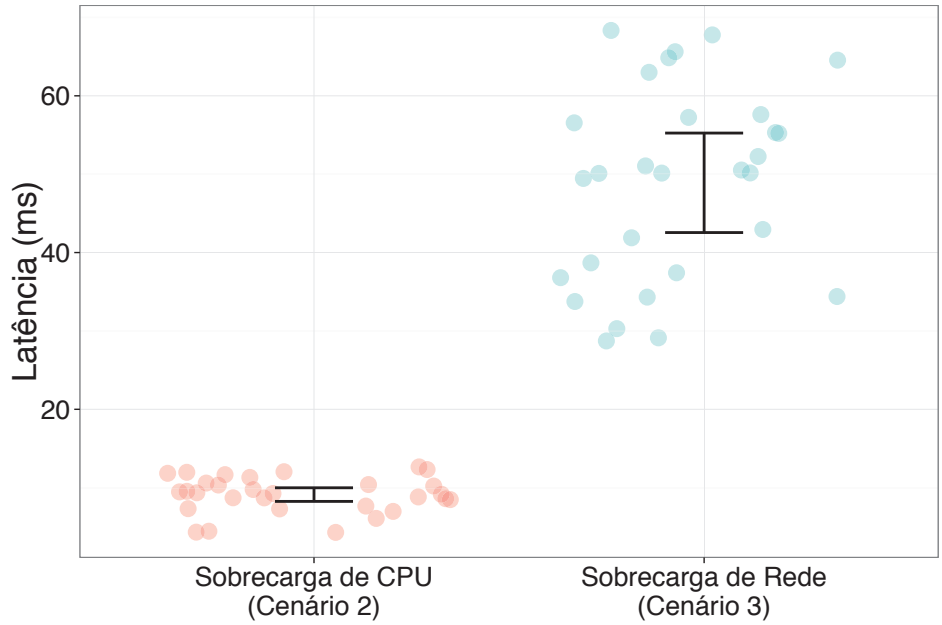


Figura 3.15: Comparação da latência em cenários com aproximadamente a mesma utilização média de CPU para os experimentos de sobrecarga de CPU e de rede.

A diferença maior que 20 *ms* entre os limites dos intervalos de confiança dos cenários comparados reforça que o efeito da sobrecarga de rede foi o maior responsável pela latência resultante do experimento.

3.1.4 Sobrecarga de Disco

Operações de entrada/saída (E/S) do disco podem provocar um atraso significante na latência de uma aplicação. Sob o efeito de uma alta carga, discos se tornam um gargalo e a CPU permanece ociosa enquanto espera pela finalização de operações de E/S. Evitar que o disco se torne um gargalo pode melhorar a performance geral do sistema.

É importante também destacar que a latência de operações de E/S, podem não afetar diretamente a performance da aplicação se existir o uso de operações *assíncronas*. Entretanto, isso nem sempre é possível e a atribuição de dois ou mais operadores que realizem operações de E/S com frequência, no mesmo nó, deve ser evitada sempre que possível.

Para exemplificar situações de sobrecarga de disco, foi executado um experimento onde os operadores realizavam a leitura de 256 KB de dados do disco e realizavam a escrita *síncrona* desses mesmos dados em outro arquivo, também em disco. As configurações dos

cenários podem ser vistas na Tabela 3.5.

Cenário	λ *(eventos/s)*	n_{op}	*t (s)*
1	10	1	120
2	10	2	120

Tabela 3.5: Cenários do experimento de sobrecarga de rede.

Como ilustrado na Figura 3.16, no segundo cenário, o fato de ambos operadores terem de esperar a conclusão de operações de E/S do outro de fato provocou um crescimento na latência.

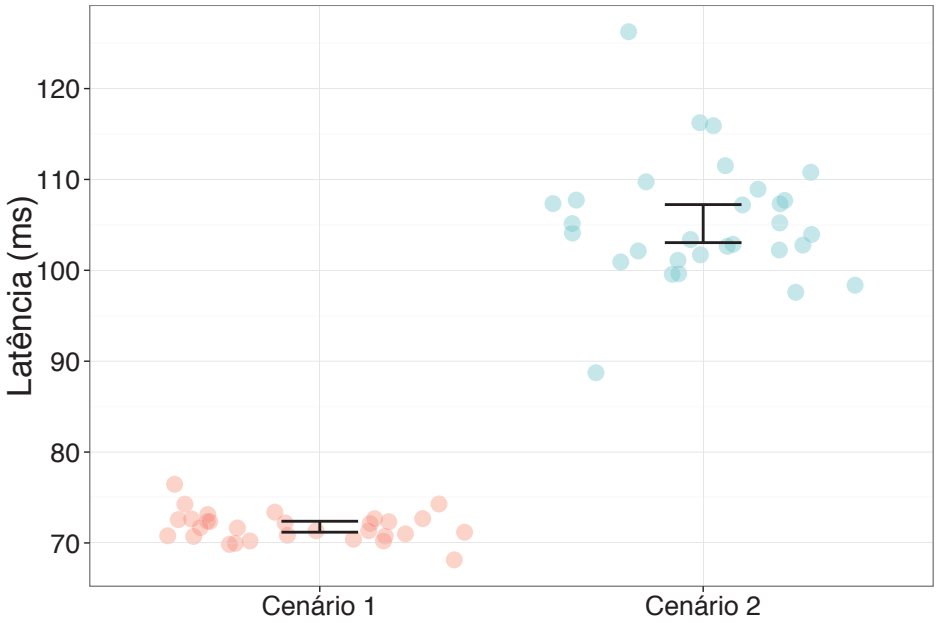


Figura 3.16: Intervalos de confiança da latência nos cenários do experimento de sobrecarga de disco.

3.1.5 Discussão

Os resultados dos experimentos mostrados acima deixam claro que a sobrecarga de vários recursos diferentes podem afetar negativamente o desempenho do sistema, o que torna necessário o monitoramento de diversas métricas de utilização. Em sumário:

- **Utilização de CPU**: como esperado, ao variar apenas a utilização de CPU a latência de processamento do operador foi afetada negativamente.

- **Utilização de Memória RAM**: também teve um efeito negativo no desempenho do operador sem que houvesse variações bruscas de outras métricas, como utilização de CPU. Além disso, outros fatores como a presença de *OOM killers* reforça a necessidade do monitoramento da memória RAM.

- **Entrada e saída de dados da rede**: mesmo que a banda de rede não esteja completamente ocupada, o desempenho dos operadores foi prejudicado sob efeito de uma alta carga.

- **Escritas e leituras em disco e tempo espera por E/S da CPU**: o atraso provocado por operações de E/S no disco podem ser causados por diversos fatores, como: (i) espera em fila; (ii) tipo de escrita/leitura feita (síncrona/assíncrona); (iii) tipo de disco e/ou estado de funcionamento. Ambas métricas podem estar diretamente relacionadas a piora de desempenho de um operador.

Outra situação possível é a existência de um único recurso que sempre esteja com alta utilização quando um operador ou nó está sobrecarregado, podendo assim, ser suficiente na detecção de situações de sobrecarga no sistema. No entanto, nem sempre é claro que o mesmo recurso físico *sempre* será o gargalo de uma aplicação e uma vez que mais desse recurso é adquirido, o gargalo da aplicação pode passar a ser outro recurso. Por exemplo, supondo que a CPU seja o gargalo, mas um *upgrade* de CPU é realizado. É possível que o novo gargalo não seja mais a CPU, e sim outro recurso, que antes não era gargalo porque a CPU estava limitando o uso de tal recurso.

3.2 Algoritmo de Escalonamento Adaptativo

Como discutido anteriormente, em um cenário realista, o comportamento da carga de um sistema *ESP* é difícil de ser previsto - as variações na carga geralmente não seguem um padrão. O algoritmo proposto nessa dissertação lida apenas com a atribuição de operadores a um conjunto de nós fixo e também é considerado que os nós possuem a **mesma** configuração de *hardware*. Adicionalmente, o algoritmo proposto faz uso de métricas de utilização e desempenho para que, de forma reativa, possa minimizar e até anular os efeitos de situações de sobrecarga através da migração de operadores para nós menos sobrecarregados.

Como foi mostrado na seção anterior, são várias as métricas que devem ser monitoradas com o intuito de verificar e/ou prevenir sobrecargas. Entretanto, o uso de métricas de utilização de recursos físicos possui algumas limitações: em geral, tanto em abordagens reativas, quanto proativas, busca-se que determinado recurso físico não ultrapasse um dado valor de utilização. Caso isso ocorra, conclui-se que o recurso está sobrecarregado e alguma ação é disparada. Entretanto, surge o problema da seleção do limiar a ser utilizado para cada recurso monitorado. Muitas vezes, é utilizado um limiar de 75% para utilização de CPU, sob a justificativa de que é necessário algum espaço (25%) para comportar cenários de rajadas na carga. Entretanto, algumas perguntas podem surgir: que valor utilizar para utilização de memória RAM? E quanto ao tempo de espera da CPU por E/S? E se a aplicação for tão sensível a variações de carga, que necessite manter a utilização de CPU abaixo dos 50%? Também existe o cenário inverso, onde a carga é *sempre previsível e constante*. Nesse caso, é esperado que o sistema utilize o máximo de seus recursos para evitar desperdício, como por exemplo, seria possível que os nós mantivessem a utilização de CPU em 90%.

Uma vez que a configuração de limiares adequados para uso dos recursos é uma tarefa difícil, dado que depende do tipo de aplicação e consequentemente, de seus requisitos. Nesse trabalho, é levada em consideração a latência dos operadores como métrica indicativa de sobrecarga. São realizadas rodadas, onde as latências atuais dos operadores são confrontadas com as medidas anteriores. Os operadores com maior variação percentual positiva em sua latência, são aqueles que dão sinais de que algo pode estar afetando seu desempenho e através de métricas de utilização de recursos dos nós, também coletadas, é avaliado se os operadores poderão ser migrados ou não para outro nó da infraestrutura.

A seguir serão apresentados e detalhadas as principais funcionalidades do algoritmo de escalonamento proposto: (i) monitoramento do sistema; (ii) como é feita e quais são as etapas da migração de operadores; (iii) seleção de operadores candidatos à migração e nós destino.

3.2.1 Monitoramento do Sistema

Escalonadores dinâmicos geralmente fazem uso de métricas provenientes do monitoramento do sistema no auxílio das decisões tomadas pelo algoritmo de escalonamento. Na arquitetura utilizada nesta dissertação, os operadores enviam periodicamente *mensagens de controle* que

contêm informações sobre seu estado atual e do nó onde está localizado. Entre as informações contidas nessas mensagens de controle, existem métricas de utilização dos recursos do sistema e desempenho do operador:

1. **Utilização**: utilização de cada CPU, tempo em espera de E/S de cada CPU, total utilizado de memória RAM, quantidade de *bytes* escritos/lidos do disco e quantidade de *bytes* enviados/recebidos pelas interfaces de rede. Naturalmente, informações sobre o *hardware*, como o total de memória RAM disponível, são coletadas.

2. **Desempenho**: latência (em milissegundos).

As métricas de utilização são coletadas através de informações fornecidas pelo próprio sistema operacional [22]. Já no cálculo da latência dos operadores, informações sobre o tempo de chegada e saída dos eventos são inseridas pelo arcabouço nas *mensagens de dados*, que passam a ter os seguintes campos: (i) instante que o evento chegou ao sistema; (ii) número de sequência do evento, utilizado para se descartar eventos fora de ordem ou duplicados, como no uso de replicação ativa; (iii) instante em que o evento saiu do operador anterior. Esse cálculo é ilustrado na Figura 3.17: a latência do operador ***O2*** é dada pela subtração entre os instantes de tempo ***ts2*** e ***ts1***. É necessário destacar que os operadores consideradores podem estar localizados no mesmo nó.

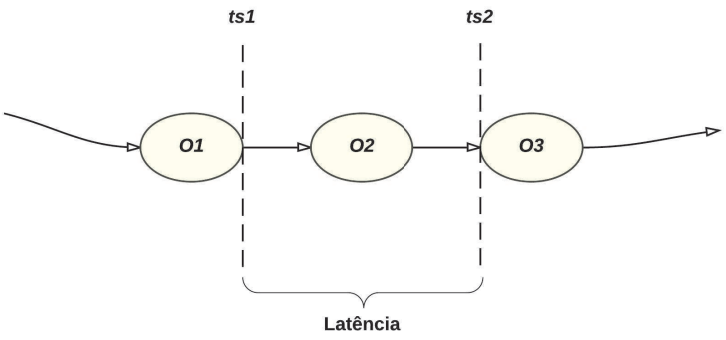


Figura 3.17: Obtenção da latência de um operador.

Onde:

- ***ts1***: instante em que o evento é *enviado* pelo operador localizado anteriormente no fluxo (***O1***).

- ***ts2***: instante em que o evento é *recebido* pelo operador localizado posteriormente no fluxo (***O3***).

Caso o operador realize uma operação de agregação, são considerados os instantes de tempo do último evento pertencente à janela de processamento. Adicionalmente, o conteúdo da informação contida nas *mensagens de controle* é completamente independente entre as réplicas de um mesmo operador. O escalonador realiza o monitoramento de cada réplica de forma individual.

Um fator a ser considerado, é a frequência de envio das mensagens de controle, seria inviável para o sistema se a latência de cada evento processado e a informação de utilização a cada instante fossem reportados. Deste modo, a frequência de envio de mensagens deve ser configurável e as informações de utilização e desempenho são sumarizadas através da **média aritmética** das métricas coletadas durante o período entre envios. Por exemplo, se um operador processou 5 eventos durante o período entre envios das mensagens de controle e as latências de processamento dos eventos foram: 10 *ms*, 15 *ms*, 20 *ms*, 10 *ms*, 20 *ms*. A latência reportada pelo operador será de 15 *ms*, o cálculo pode ser visto a seguir:

$$\frac{10 + 15 + 20 + 10 + 20}{5} = 15 \tag{3.1}$$

A configuração da frequência de envio deve levar em consideração o montante de dados gerados e consequentemente o *overhead* no sistema, como também a granularidade de informação necessária para que o gerente desempenhe suas atividades. Normalmente, o envio de mensagens de controle é feito a **cada segundo**, o que não gera sobrecarga no gerente do sistema por ter que processar muitas mensagens e também permite que o mesmo possua informações detalhadas sobre o estado atual de todos os operadores e nós do sistema.

3.2.2 Migração de Operadores

Migração é uma técnica bastante utilizada na gerência de recursos, principalmente em cenários onde há o problema de atribuição e distribuição dinâmica de um conjunto de componentes aos recursos físicos disponíveis. Nesses cenários, muitas vezes, não existe a possibilidade de crescimento da infraestrutura e por isso, provisionamento não é uma opção. Em sistemas *ESP*, migrações são utilizadas na tentativa de prover uma melhor distribuição de operado-

res nos nós disponíveis durante a sua execução e assim, melhorar o balanceamento de carga entre eles. Para tal efeito, um requisito importante é que a migração provoque interrupções mínimas na execução da aplicação.

Como na pesquisa apresentada nesta dissertação é feito o uso do arcabouço StreamMine3G, na Figura 3.18 (baseada em uma figura presente em [7]) a seguir, é ilustrado o mecanismo de migração de operadores implementado por ele.

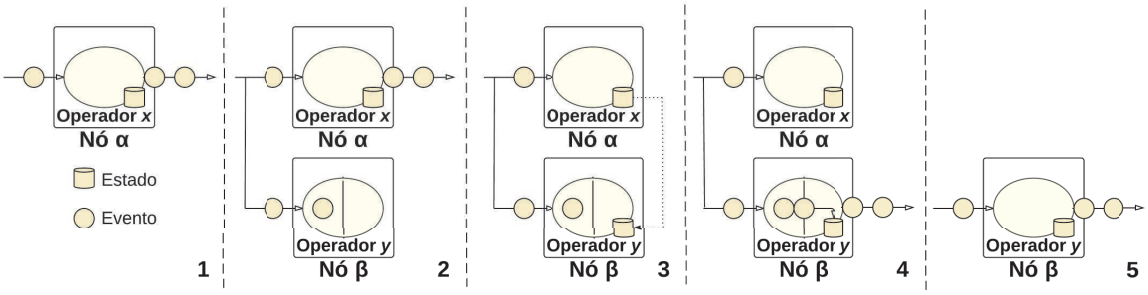


Figura 3.18: Passos da migração de um operador no StreamMine3G.

Seja x um operador localizado no nó α e que será migrado para o nó β:

1. A partir de uma solicitação do escalonador, o gerente do sistema inicia o processo de migração criando um novo operador y em β, inicialmente vazio. Também modifica o fluxo de dados para que as *mensagens de dados* (que contém os eventos) que chegam a x também sejam enviadas para y.

2. Os eventos continuam chegando em x, que os processa normalmente. O novo operador também os recebe, mas são enfileirados para serem processados posteriormente. A cópia do estado de x para y apenas acontece quando os eventos enfileirados em y possuem um número de sequência (identificador), menor ou igual que todos os eventos já processados por x.

3. Antes da cópia do estado, o processamento em x é interrompido.

4. O estado de x é transferido junto de um vetor de instantes de tempo, que indica quais eventos devem ser descartadas por y antes do processamento ser reiniciado. Uma vez que o estado é completamente transferido, y reinicia o processamento.

5. O operador x é então removido de α.

O tempo de duração da migração é diretamente proporcional ao tamanho do estado do operador a ser migrado (passo 4). Em especial, operadores sem estado são facilmente migrados e o impacto que a migração possa ter no sistema é mínimo. Além disso, se existirem *réplicas ativas* do operador a ser migrado, não existe interrupção no processamento.

3.2.3 Etapas do Algoritmo

O algoritmo de escalonamento adaptativo pode ser dividido em duas grandes etapas: seleção de operadores candidatos à migração e seleção de nós destino. Essas duas etapas são detalhadas a seguir.

Seleção de operadores candidatos à migração

Periodicamente, o escalonador realiza um avaliação onde são selecionados operadores candidatos à migração através das latências reportadas a partir das *mensagens de controle*. Durante o tempo entre avaliações, o escalonador guarda todos os valores de latência recebidos. No momento da seleção de quais operadores serão candidatos, o escalonador então realiza um cálculo de uma métrica que irá indicar quais operadores apresentaram uma piora em seu desempenho.

A métrica de piora de desempenho de um operador s (Equação 3.2) é obtida da seguinte forma: seja L o conjunto de todos os valores de latência $\{l_1, l_2, ..., l_n\}$ reportados por um operador na última janela entre avaliações, com l_1 sendo o primeiro valor recebido e l_n, o último. Seja g, a diferença absoluta entre dois valores reportados consecutivamente se a latência estiver aumentando, ou zero, caso contrário, ou seja: $g_i = l_i - l_{i-1}$ se $l_i > l_{i-1} + d \times l_{i-1}$ e $g_i = 0$ se $l_i \leq l_{i-1} + d \times l_{i-1}$. Onde o parâmetro d, chamado de *sensibilidade*, decide a partir de que nível de variação da latência, a diferença deve ser considerada ou não. Por exemplo, se $d = 0,5$, o valor de g será maior que 0 se, e somente se, a variação da latência reportada pelo operador for maior que 50%.

$$s = \frac{\sum_{i=1}^{i=n} g_i}{l_1} \tag{3.2}$$

Como a variável g indica apenas *crescimento*, ela não possui valores negativos. Essa escolha implica que, um operador que apresentou uma variação no valor da latência apenas

em uma medição durante a janela de avaliação e depois voltou a reportar valores baixos de latência, poderia ser migrado. Como na arquitetura utilizada nesse trabalho sempre é considerada a presença de uma ou mais réplicas ativas para o mesmo operador, migrações oferecem a oportunidade de rejuvenescimento do processo, evitando *bugs* em consequência de *envelhecimento de processo*[1] [24].

Para exemplificar a atuação do algoritmo, pode ser considerado um cenário onde o escalonador recebe as latências de três operadores x, y e z, onde: $L_x = \{12, 5, 15\}$, $L_y = \{5, 5, 10\}$ e $L_z = \{20, 30, 20\}$.

Para um $d = 0,25$, temos que:

$$s_x = \frac{(0 + 10)}{12}, s_y = \frac{(0 + 5)}{5}, s_z = \frac{(10 + 0)}{20} \tag{3.3}$$

E como resultado:

$$s_x = 0,8, s_y = 1, s_z = 0,66 \tag{3.4}$$

O operador com maior valor de s foi y. Com uma diferença de 100%, esse operador foi aquele onde a latência teve o maior aumento ao se observar o primeiro e último valor considerados na janela. Caso o valor de d fosse 0,75, os valores de g para o operador z, seriam $\{0, 0\}$. Já que a variação ocorrida entre o primeiro valor reportado, 20, e o segundo, 30, não foi maior que 75%.

Após o valor de s ser calculado para todos operadores, são escolhidos aqueles onde o valor de s é maior que 0. Se nenhum operador é selecionado, o algoritmo não continua, caso contrário, eles são ordenados a partir do que possuir maior valor de s e usados no próximo passo do algoritmo. Adicionalmente, se um operador tiver sido migrado recentemente, é possível que ele ainda esteja um pouco instável por consequência desse processo de migração. Por isso, o **instante de tempo** (*timestamp*) em que um operador foi migrado serve de desempate caso os operadores possuam o mesmo valor de s, ou seja, se dois operadores possuem o valor de s igual a $0,8$ e um deles foi migrado há 10 minutos atrás e o outro há 9 minutos, o primeiro terá maior prioridade.

[1] Quando um processo deve ser executado perpetuamente, reiniciá-lo periodicamente pode evitar que alguns problemas que tendem a piorar com o tempo, como vazamento de memória, prejudiquem a execução do sistema.

Seleção de nós destino

O escalonador possui informações detalhadas da utilização de vários recursos de *hardware* dos nós do sistema. Essa informação é obtida através das *mensagens de controle* enviadas pelos operadores. Para identificar a utilização total de um recurso em um nó, basta que o escalonador reúna a informação reportada pelos operadores localizados atualmente nesse nó.

Para definir qual ou quais operadores serão migrados, o escalonador realiza uma seleção através de uma abordagem gulosa. Seja U_α o conjunto de dados de utilização dos recursos do nó α onde o operador x candidato a migração está atualmente localizado, U_β conjunto de dados de utilização de um nó β a ser considerado para receber o operador x e U_x, conjunto da quantidade de recursos atualmente utilizados por x em α. Temos que x será migrado para β se para algum $u_\alpha \in U_\alpha$, $u_\beta \in U_\beta$ e $u_x \in U_x$:

$$u_\beta + u_x < u_\alpha \tag{3.5}$$

E **não existe** alguma combinação onde:

$$u_\beta + u_x > u_\alpha \tag{3.6}$$

É importante destacar que as Equações 3.5 e 3.6 são válidas apenas para operações envolvendo o mesmo recurso, ou seja, utilização de CPU é somada e comparada apenas com utilização de CPU, e assim por diante.

Resumidamente, a migração do operador x é considerada viável para um nó β se o operador não irá levar a utilização de **algum recurso** de β para um valor **maior** que a utilização do mesmo recurso do nó α onde está o operador x atualmente e exista **pelo menos um** recurso em que a utilização prevista será **menor**. Como não é conhecido para o escalonador, qual ou quais recursos estão afetando o desempenho do operador, a escolha do nó a receber a migração é a mais conservadora possível.

O limite de migrações que podem ser realizadas na mesma rodada, é definida através do parâmetro l_mig. Muitas migrações acionadas ao mesmo tempo, são mais suscetíveis a gerar perturbações indesejáveis. Em contrapartida, se o sistema não puder realizar todas as migrações necessárias, a sobrecarga no sistema em determinados nós ou operadores, pode

ser prolongada. Caso exista limite de migrações, o escalonador deve priorizar sempre os operadores com maior valor de s (métrica de piora de desempenho).

O algoritmo, ao realizar o cálculo acima, também analisa a possibilidade de um operador ser migrado para o nó onde já esteja localizado um outro operador diretamente conectado a ele (a saída de um, é a entrada do outro). Nesse caso, apesar da chance de que esses operadores sejam sobrecarregados ao mesmo tempo, o uso de rede decorrente da transferência de eventos entre eles é anulado. A situação inversa também é provável, operadores diretamente conectados que estejam no mesmo nó, podem ser separados: é possível que exista outro nó com menor utilização de todos recursos, inclusive rede.

Após o final da atribuição dos operadores aos nós destino, o escalonador inicia o processo de migração. As rodadas de avaliação são interrompidas enquanto todas as migrações não são finalizadas.

3.2.4 Considerações Finais

Neste capítulo foi mostrado que a sobrecarga de diferentes recursos de *hardware* podem afetar o desempenho de um operador, e em consequência disso, pode ser necessário que diversas métricas de utilização desses recursos sejam monitoradas. Os experimentos foram importantes no sentido de mostrar que todas as aplicações, independentemente do recurso que está sendo o gargalo, sofrem com o aumento da latência. Então é plausível pensar em um escalonador que não toma decisões com base em uso de recursos mas com base nas variações das latências verificadas para os operadores ao longo do tempo.

Posteriormente, foi detalhado o algoritmo de escalonamento adaptativo proposto nesta dissertação. Esse algoritmo realiza periodicamente rodadas de avaliação onde são selecionados operadores a serem migrados para nós com menor utilização. Na seleção dos candidatos à migração, o algoritmo faz uso de uma métrica baseada na variação percentual ocorrida na latência dos operadores. Em seguida, o escalonador tenta encontrar nós onde os operadores selecionados possam ser migrados. O escalonador realiza a migração se existir algum nó onde as utilizações previstas de seus recursos ao receber o operador não sejam maiores que a do nó ondes o mesmo esteja localizado e possuam pelo menos um recurso com menor utilização.

Capítulo 4

Avaliação e Resultados

Neste capítulo serão discutidos os métodos utilizados na avaliação do algoritmo de escalonamento proposto no Capítulo 3. Também serão apresentados e analisados os resultados dessa avaliação.

Os métodos de avaliação foram definidos a fim de atingir os seguintes objetivos:

1. Analisar diferentes configurações dos parâmetros do algoritmo proposto (*sensibilidade* e *limite de migrações por rodada*), com o objetivo de avaliar qual impacto cada um possui sobre o desempenho do escalonador.

2. Verificar se o uso do algoritmo de escalonamento adaptativo pode melhorar o desempenho de aplicações que executam em sistemas *ESP*.

Nas próximas seções, as métricas utilizadas na avaliação serão detalhadas e os cenários avaliados serão descritos. Por fim, serão apresentados os resultados dos cenários executados.

4.1 Métricas de Avaliação

A fim de responder as questões de pesquisa estabelecidas no início do capítulo, foram consideradas duas métricas de avaliação: *latência* e *número total de eventos processados*.

4.1.1 Latência

A forma como é obtida a latência foi detalhada no Capítulo 3 (Seção 3.2.1). Para fins de avaliação, foram considerados a média e o percentil 99[1] da latência do processamento de todos os operadores envolvidos na execução dos experimentos.

Formalmente, seja L o conjunto de todos os valores de latência $\{l_1, l_2, ..., l_n\}$ medidos durante uma réplica dos experimentos e n o total de eventos processados. A latência média $\hat{L}$, é calculada a partir da Equação 4.1.

$$\hat{L} = \frac{\sum_{i=1}^{i=n} l_i}{n} \tag{4.1}$$

E para o cálculo do 99 percentil L_{99}, primeiramente, os valores de latência são ordenados e, em seguida, o cálculo é realizado a partir das Equações 4.2 e 4.3:

$$L_{99} = l_k \tag{4.2}$$

Onde:

$$k = \left\lfloor \frac{99(n+1)}{100} \right\rfloor \tag{4.3}$$

4.1.2 Total de eventos processados

Além da *latência*, o número total de eventos processados n se torna uma métrica importante na avaliação do desempenho dos operadores. Principalmente, quando há a presença de migrações. Durante a migração de um operador, há a interrupção de seu processamento, prejudicando a eficiência do sistema. Se existir uma réplica ativa desse operador, caso a réplica com melhor desempenho seja migrada, o número total de eventos processados diminui em consequência na queda da vazão desse operador. Consequentemente, mesmo que não exista diferença na *latência* entre dois cenários distintos, o desempenho do sistema influencia diretamente na quantidade de eventos processados. Além disso, caso exista uma réplica ativa do

[1] Ao analisar a latência de uma aplicação, o percentil 99 é mais representativo que a média, e geralmente utilizada quando se quer afirmar que uma aplicação possui um tempo de resposta menor que um determinado valor [30].

operador, a escolha da réplica mais sobrecarregada para ser migrada, não diminui a vazão de processamento.

4.2 Cenários de Avaliação

Partindo dos objetivos acima citados, foram elaborados diversos cenários para execução de experimentos e avaliação dos parâmetros de configuração e do desempenho do algoritmo de escalonamento adaptativo. Em todos os cenários, o tempo de execução e a quantidade de réplicas, foram mantidos fixos. Para que a análise pudesse ser feita através da comparação das médias dos intervalos de confiança dos cenários, foram executadas 30 réplicas. Cada réplica tinha um tempo de execução de 600 segundos (10 minutos).

No contexto da avaliação do **impacto dos parâmetros de configuração**, as variáveis independentes (fatores) são a sensibilidade (d) e o limite de migrações por rodada (l_mig). Na Tabela 4.1, estão resumidos os níveis avaliados para cada variável. Para a variável de sensibilidade, os níveis escolhidos podem ser classificados em *sensibilidade alta* (0,25), *sensibilidade média* (0,50) e *sensibilidade baixa* (0,75). É esperado que, quanto menor for o valor de d, mais operadores sejam selecionadas para a fase de seleção de nós destino. Já para variável de limite de migrações por rodada, foram escolhidos dois níveis: 1 e sem limite. No primeiro, apenas um operador poderá ser migrado por rodada e assim, vários operadores podem ter seu período de sobrecarga prolongado. No segundo, o número de migrações é ilimitado, mas, dependendo da variável de sensibilidade, pode ser que poucos sejam os operadores candidatos à migração.

Variáveis Independentes (Fatores)	**Níveis**
d	0,25 (25%), 0,50 (50%), 0,75 (75%)
l_mig	1, sem limite

Tabela 4.1: Resumo dos níveis das variáveis independentes utilizados na avaliação do impacto dos parâmetros de configuração do algoritmo de escalonamento.

Na avaliação de **desempenho do algoritmo proposto**, foram definidas duas variáveis independentes: carga de trabalho e estratégia de escalonamento. Os níveis das variáveis

estão resumidos na Tabela 4.2.

Variáveis Independentes (Fatores)	**Níveis**
Carga de trabalho	estável, variada
Estratégia de escalonamento	adaptativa, aleatória, sem escalonamento

Tabela 4.2: Resumo dos níveis das variáveis independentes utilizados na avaliação de desempenho do algoritmo de escalonamento.

Para o cenário com carga *estável*, a taxa de chegada de eventos era mantida fixa em 50 ou 100 *eventos/s* dependendo do operador. No nível com carga *variada*, a taxa de chegada de eventos muda de forma aleatória, partindo de 50 ou 100 *eventos/s* para 550 ou 600 *eventos/s*. A variação na carga de trabalho acontece em ciclos de 60 segundos, 10 vezes em cada execução do experimento. O envio das mensagens de controle é feito a cada 1 segundo e as rodadas de avaliação do escalonador, a cada 10 segundos (é realizado o **mesmo número** de rodadas de avaliação em todos os cenários). A escolha de quais momentos terão carga alta ou baixa foi feita previamente (para que pudesse ser reproduzida da mesma forma em todas as réplicas do experimento) e, para cada operador, era definida uma sequência da seguinte forma:

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Alta	Baixa	Baixa	Alta	Baixa	Alta	Baixa	Alta	Alta	Baixa

Tabela 4.3: Exemplo da definição da variação da carga de trabalho de um operador.

Na Figura 4.1 está ilustrado o comportamento da variação de taxa de chegada de eventos provocada pela configuração da Tabela 4.3. A transição entre os níveis não é feita imediatamente, tanto o crescimento da taxa, quanto a sua diminuição, são feitos de forma gradativa. No Apêndice A desta dissertação estão detalhadas todas as cargas de trabalho utilizadas nos cenários de avaliação.

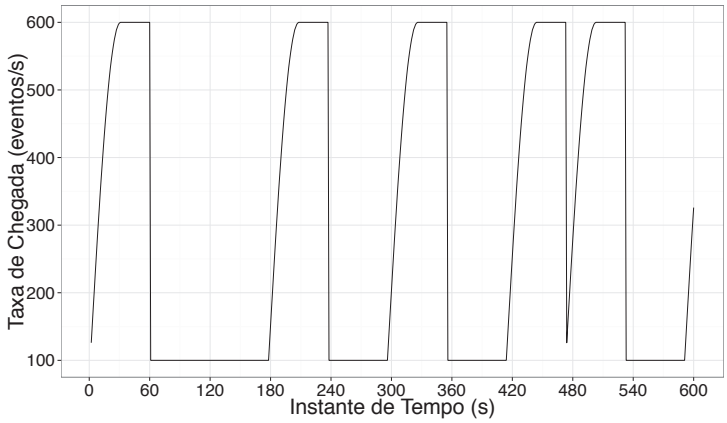


Figura 4.1: Exemplo da variação da taxa de chegada de eventos para um operador, no cenário com taxa de envio variada.

Na Figura 4.2, está ilustrada a taxa de chegada de eventos agregada para todos os operadores localizados em um mesmo nó.

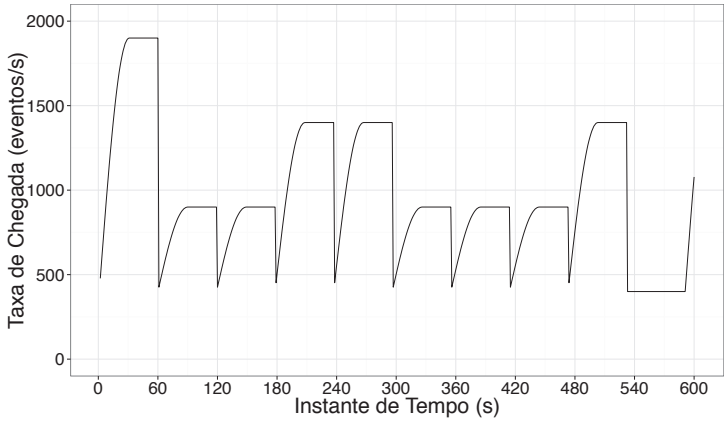


Figura 4.2: Exemplo da variação da taxa de chegada de eventos agregada para todos operadores localizados em um mesmo nó, no cenário com taxa de envio variada.

Já os níveis escolhidos para a variável de estratégia de escalonamento, estão detalhados a seguir:

- **Adaptativa**: algoritmo de escalonamento proposto nesta dissertação. Utiliza a métrica de piora de desempenho na seleção de operadores candidatos à migração.
- **Aleatória**: a cada rodada, para cada réplica dos operadores em execução no sistema é feita uma escolha aleatória se essa réplica será candidata à migração ou não. A seleção

de nós destino é realizada da mesma forma que no algoritmo adaptativo, naturalmente, sem a ordenação dos operadores a partir da métrica de piora de desempenho.

- **Sem escalonamento**: não existe a presença de um escalonador, ou seja, nenhuma ação de mudança na topologia do sistema, como migrações, é realizada.

4.3 Ambiente e Topologia dos Experimentos

Na reprodução dos cenários de avaliação descritos na Seção 4.3, foram utilizadas onze máquinas virtuais em um ambiente OpenStack[1] de *computação na nuvem* privado. Nessas máquinas, foram distribuídos 39 operadores, 13 deles possuindo uma réplica ativa, o que totaliza 52 operadores. Adicionalmente, as ma?quinas virtuais foram sincronizadas via Network Time Protocol (NTP). Maiores detalhes sobre a topologia estão na Tabela 4.4.

Máquina Virtual	**Operadores**
1	Gerente
2	Fonte 1, Fonte 2, ..., Fonte 6
3	Fonte 7, Fonte 8, ..., Fonte 13
4	Sorvedouro 1, Sorvedouro 2, ..., Sorvedouro 6
5	Sorvedouro 7, Sorvedouro 8, ..., Sorvedouro 13
6	Worker 1, Worker 7, Worker 13, Worker 6, Worker 12
7	Worker 2, Worker 8, Worker 1, Worker 7, Worker 13
8	Worker 3, Worker 9, Worker 2, Worker 8
9	Worker 4, Worker 10, Worker 3, Worker 9
10	Worker 5, Worker 11, Worker 4, Worker 10
11	Worker 6, Worker 12, Worker 5, Worker 11

Tabela 4.4: Localização dos operadores envolvidos nos experimentos.

Existem 13 *Fontes* que geram eventos a taxas diversas. Os *Workers*, e suas réplicas, recebem esses eventos, os processam, e os enviam para os *Sorvedouros*. Nessa configuração,

[1] http://www.openstack.org/

os únicos operadores a serem monitorados pelo escalonador do sistema são os *Workers*, e consequentemente, migrações ocorrem **apenas** entre as máquinas virtuais 6 a 11. Na Figura 4.3 está ilustrado a comunicação entre uma *Fonte*, um *Worker* (com sua réplica ativa) e um *Sorvedouro*.

As onze máquinas virtuais executavam Ubuntu Server 12.04 LTS (Precise Pangolin), com o arcabouço StreamMine3G instalado. As configurações de *hardware* eram:

- Máquinas Virtuais 1, 2, 3, 4 e 5: 8 VCPUs, 4 GB de memória RAM e 160 GB de disco (magnético).
- Máquinas Virtuais 6, 7, 8, 9, 10 e 11: 4 VCPUs, 2 GB de memória RAM e 80 GB de disco (magnético).

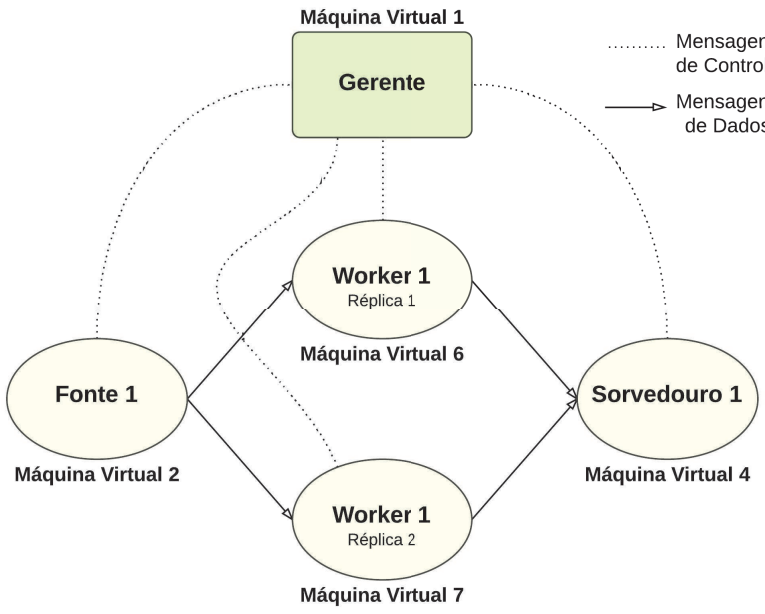


Figura 4.3: Topologia dos experimentos de avaliação do algoritmo de escalonamento para um *Worker*.

Além disso, não foi efetuada nenhuma alteração nas configurações padrão do arcabouço StreamMine3G.

4.4 Apresentação e Análise dos Resultados

Nesta seção serão apresentados e analisados os resultados dos experimentos descritos acima. Para fins de comparação, serão exibidas figuras que irão agrupar os resultados dos diferentes cenários. A análise será feita a partir de intervalos de confiança da média, que ao não se sobreporem, sugerem que os grupos possuem amostras provenientes de populações diferentes.

4.4.1 Parâmetros de Configuração

A seguir está detalhada a avaliação realizada dos parâmetros de configuração do algoritmo proposto: *sensibilidade* e *limite de migrações por rodada*.

Sensibilidade

Na avaliação da sensibilidade (d), foram executados seis cenários, que estão detalhados na Tabela 4.5.

Cenário	d	l_mig	Carga de Trabalho	Estratégia de Escalonamento
1	25%	1	Variada	Adaptativo
2	50%	1	Variada	Adaptativo
3	75%	1	Variada	Adaptativo
4	25%	1	Estável	Adaptativo
5	50%	1	Estável	Adaptativo
6	75%	1	Estável	Adaptativo

Tabela 4.5: Resumo dos níveis das variáveis independentes utilizados na avaliação da sensibilidade.

Os cenários foram separados em dois blocos de acordo com a sua carga de trabalho. O limite de migrações (l_mig) por rodada foi mantido fixo em 1. As Figuras 4.4 e 4.5 exibem a quantidade de migrações realizadas nos cenários. Como esperado, ao variar o valor da sensibilidade (d), a quantidade de migrações a serem realizadas é controlada indiretamente: quanto menor o valor de d, mais variações na latência são detectadas e consequentemente, mais operadores são selecionados para fase de migração.

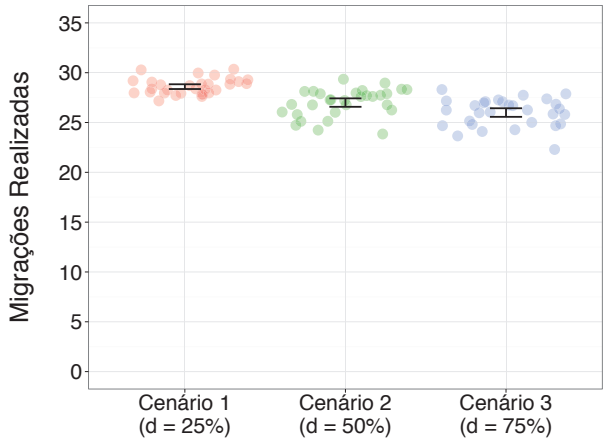


Figura 4.4: Quantidade de migrações realizadas nos cenários com carga variada.

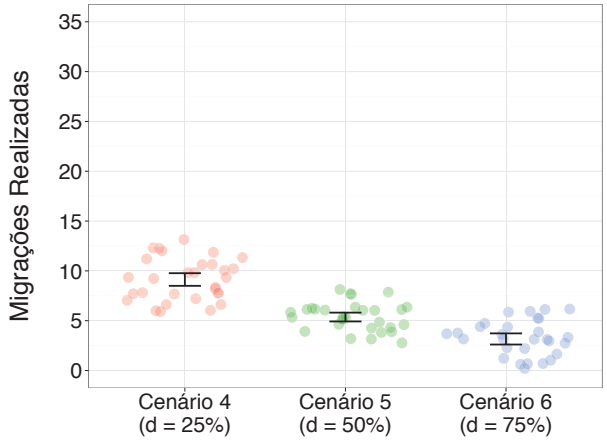
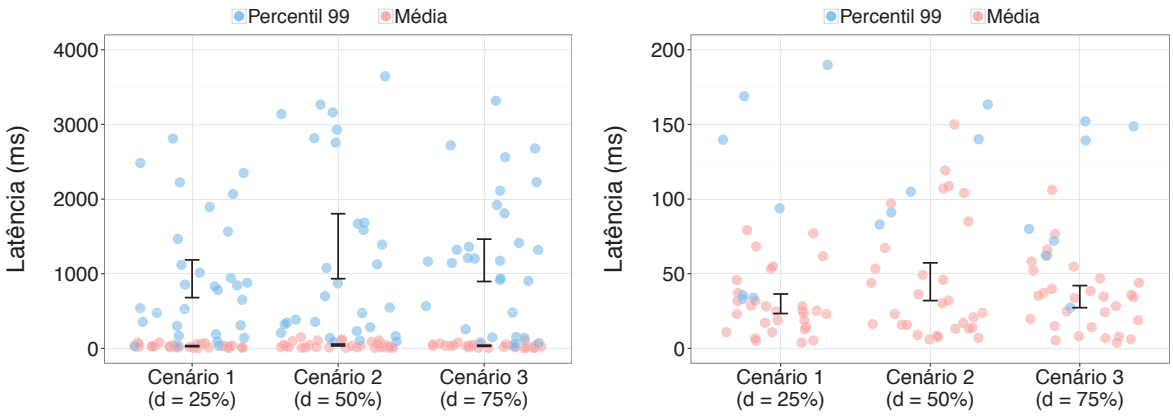


Figura 4.5: Quantidade de migrações realizadas nos cenários com carga estável.

A presença de uma ou mais réplicas ativas torna interessante a comparação da latência não só das réplicas de melhor desempenho, como também entre as réplicas de pior desempenho. Caso a falha de um nó aconteça pode ser que a réplica com melhor desempenho (**réplica rápida**) de um operador seja perdida, o que torna importante o gerenciamento da réplica de pior desempenho (**réplica lenta**).

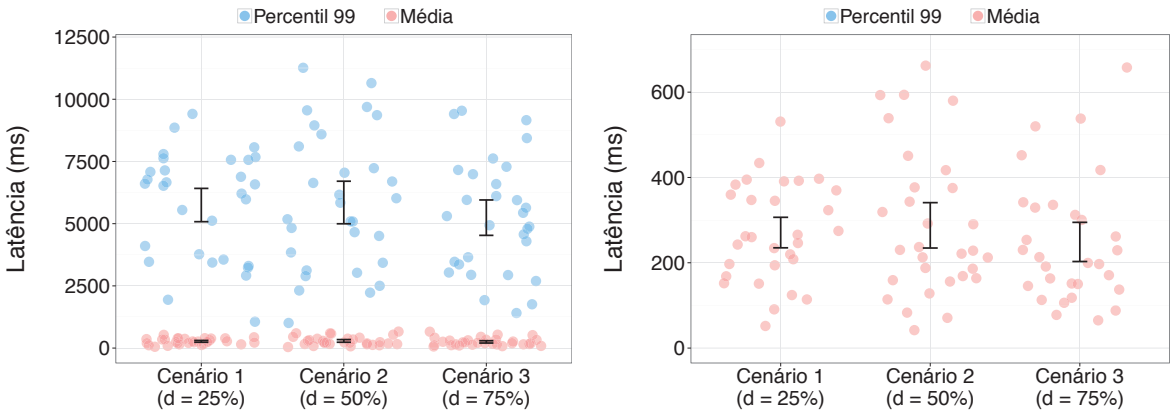
A latência resultante pode ser verificada nas Figuras 4.6 e 4.7. Na Figura 4.6 estão os resultados para a réplica rápida, enquanto que na Figura 4.7 estão os resultados para a réplica lenta. Nesses gráficos, estão representados duas medidas diferentes de latência: o percentil 99 e a média. Quando necessário, um gráfico que faz uma aproximação nos valores que ficaram difíceis de visualizar devido à escala, é exibido à direita do gráfico original. Pela

semelhança entre os intervalos de confiança exibidos nos gráficos foram realizados testes estatísticos para confirmar que não existe diferença entre as médias dos valores de latência para os três cenários, que comprovaram essa igualdade.



(a) Gráfico original. (b) Gráfico com aproximação.

Figura 4.6: Latências da réplica rápida, para os cenários com carga variada.

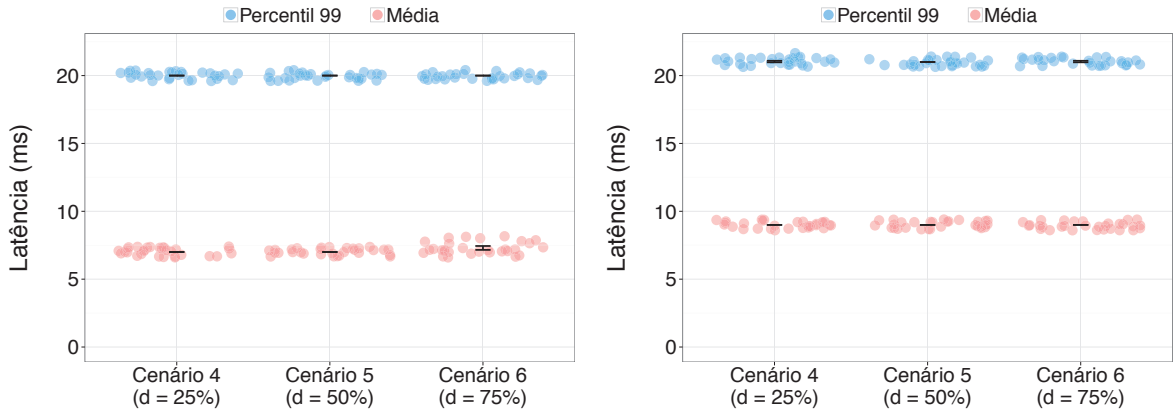


(a) Gráfico original. (b) Gráfico com aproximação.

Figura 4.7: Latências da réplica lenta, para os cenários com carga variada.

Em decorrência da manutenção da carga de forma estável nos Cenários 4, 5 e 6. Não existiram picos de sobrecarga, o que provocou que a latência, ilustrada na Figura 4.8, permanecesse praticamente uniforme, mesmo entre as diferentes réplicas. Nos seis cenários

executados, a quantidade de eventos processados foi maior para valores maiores de d. A diferença entre as médias do Cenário 3 e 1 foi de cerca 9500 eventos (0,55%), e entre os Cenários 6 e 4 (0,17%), de 600 eventos. Esses resultados estão ilustrados nas Figuras 4.9 e 4.10.



(a) Réplica rápida. (b) Réplica lenta.

Figura 4.8: Latências para os cenários com carga estável.

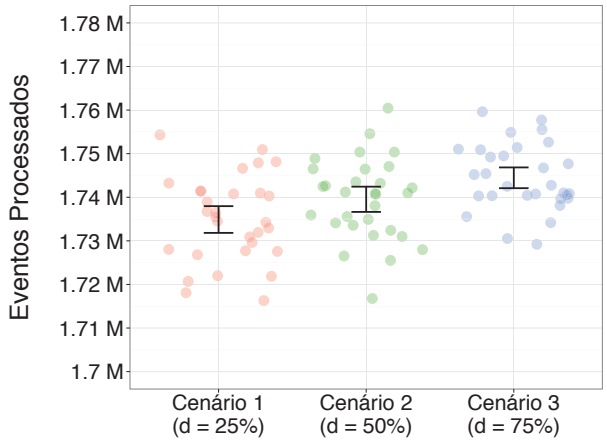


Figura 4.9: Quantidade de eventos processados nos cenários com carga variada. Os valores estão exibidos de forma abreviada (1.75 M é igual a 1,75 milhões).

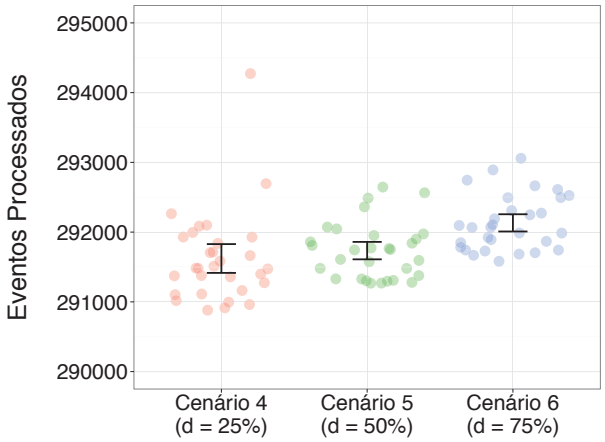


Figura 4.10: Quantidade de eventos processados nos cenários com carga estável.

Limite de migrações por rodada

Em adição aos experimentos executados na avaliação da sensibilidade, foram definidos três cenários para a avaliação do limite de migrações por rodada (l_mig). Esses cenários estão detalhados na Tabela 4.6. A carga de trabalho foi mantida fixa como *variada*. Dessa forma, a análise dos cenários adicionais pode ser feita através da comparação com os Cenários 1, 2 e 3. A comparação então é feita entre cenários com o mesmo nível dos fatores *sensibilidade* e *carga de trabalho*.

Cenário	***d***	***l_mig***	**Carga de Trabalho**	**Estratégia de Escalonamento**
1	25%	1	Variada	Adaptativo
2	50%	1	Variada	Adaptativo
3	75%	1	Variada	Adaptativo
7	25%	Sem limite	Variada	Adaptativo
8	50%	Sem limite	Variada	Adaptativo
9	75%	Sem limite	Variada	Adaptativo

Tabela 4.6: Resumo dos níveis das variáveis independentes utilizados na avaliação do limite de migrações por rodada.

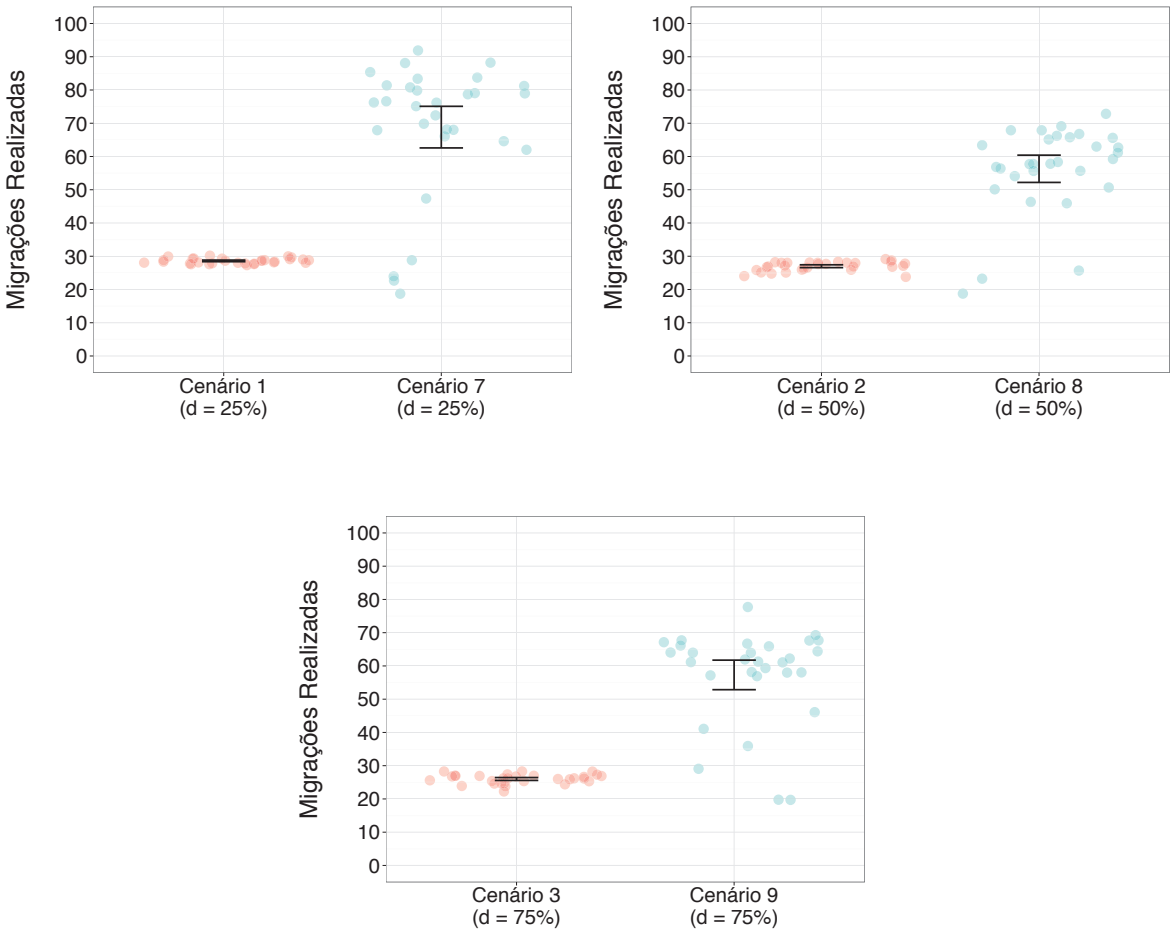


Figura 4.11: Quantidade de migrações realizadas.

Como mostra a Figura 4.11, o número de migrações realizadas foi maior nos cenários sem limite de migrações por rodada. De forma semelhante à avaliação da sensibilidade, as interrupções causadas pelas migrações tiveram um impacto negativo na quantidade de eventos processados (Figura 4.12). Nas três comparações, não houve sobreposição nos intervalos de confiança.

A diferença entre a média do número de migrações realizadas chegou a 28 e 68 para os Cenários 1 e 7, respectivamente. Em relação a média da quantidade de eventos processados, ilustrados na Figura 4.12, foram 1,74 milhões para o Cenário 3 e 1,68 milhões para o Cenário 9, uma diferença de 3,4%.

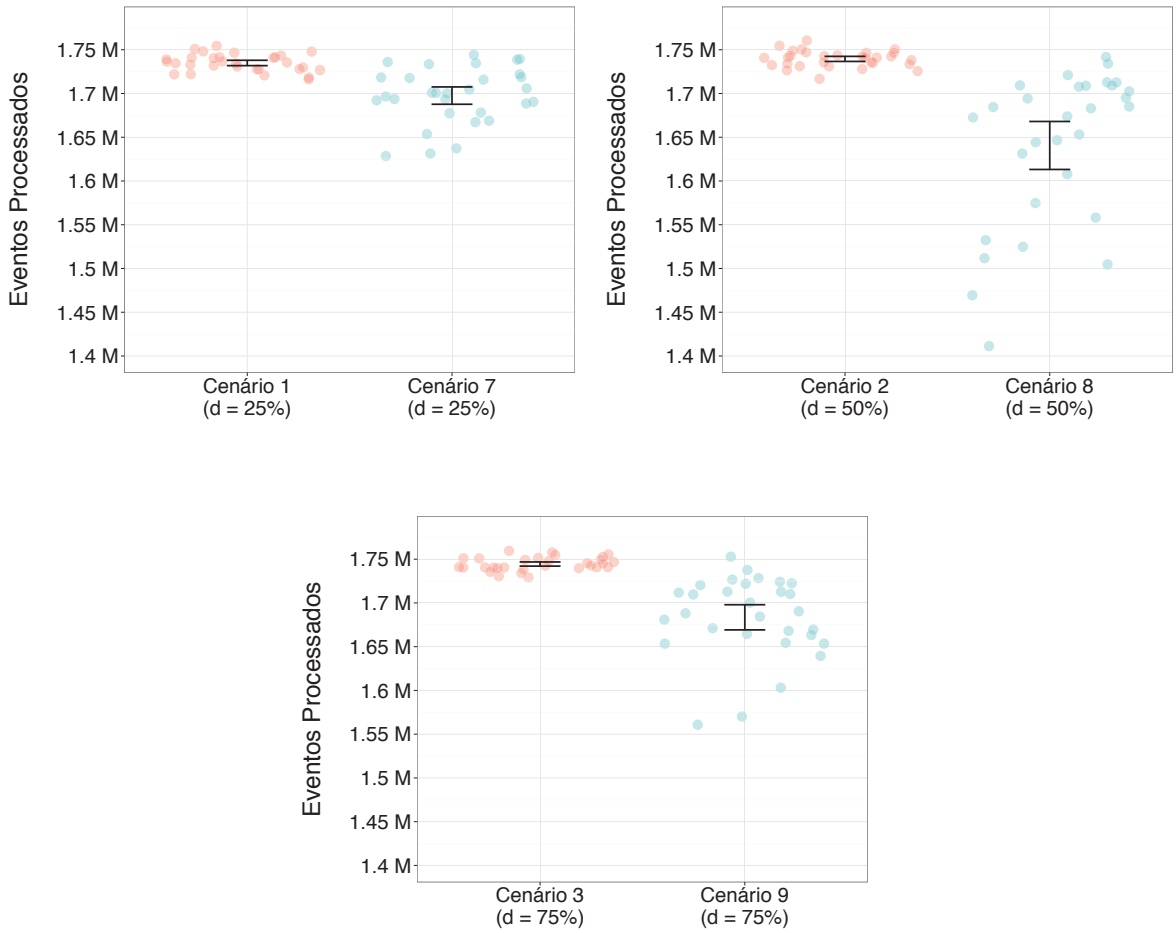
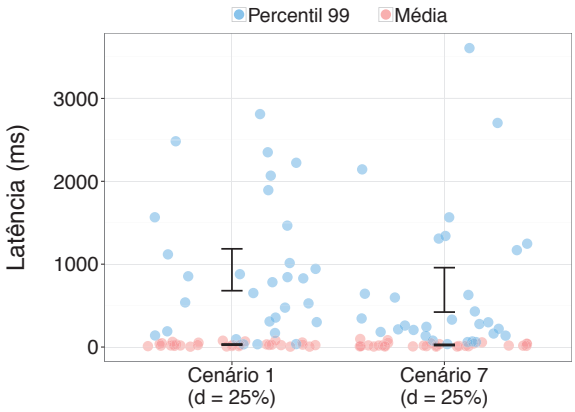
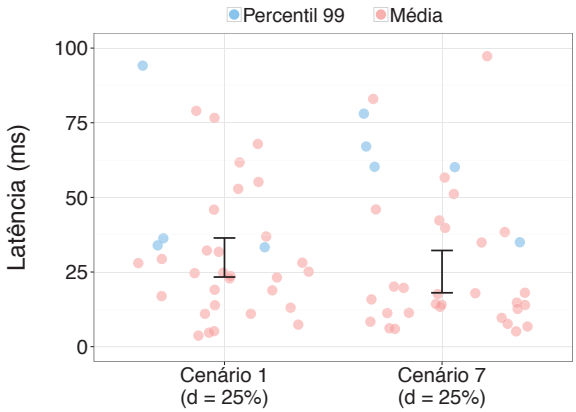


Figura 4.12: Quantidade eventos processados.

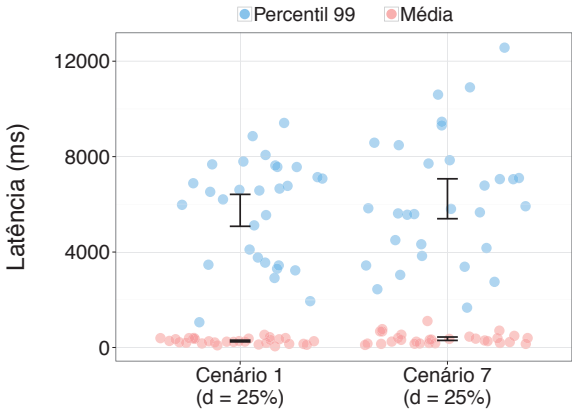
Para ilustrar o impacto na latência, a comparação entre os Cenários 1 e 7 pode ser vista na Figura 4.13. A possibilidade de migração de vários operadores ao mesmo tempo e assim evitar que alguns deles permanecessem sobrecarregados por mais tempo, não foi refletida nos resultados. Na comparação direta entre os níveis do limite de migrações por rodada, os cenários com $l\_mig = 1$ possuíram um desempenho semelhante na latência, mas superior na quantidade de eventos processados.



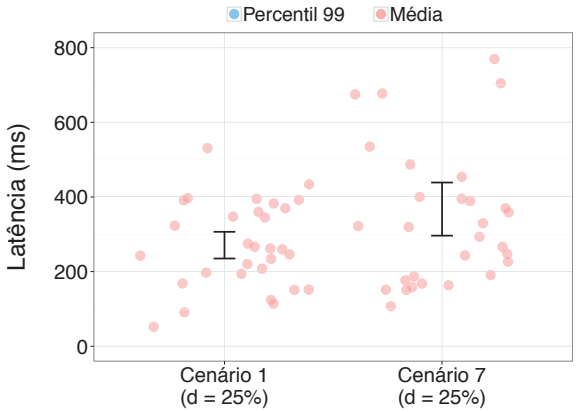
(a) Réplica rápida. Gráfico original.



(b) Réplica rápida. Gráfico com aproximação.



(c) Réplica lenta. Gráfico original.



(d) Réplica lenta. Gráfico com aproximação.

Figura 4.13: Latências para o cenário com sensibilidade igual a 25%.

Discussão

A avaliação dos parâmetros de configuração mostrou que a quantidade de migrações realizadas, pode interferir diretamente no desempenho do sistema. Ao variar o parâmetro de *sensibilidade*, o número de migrações era maior para valores menores de d, dada a forma com que os operadores a serem migrados são selecionados. O maior número de migrações nos cenários com menor valor de d interferiu na quantidade de eventos processados. Esse resultado sugere que valores baixos de d, provocam uma maior quantidade de erros na seleção dos operadores a serem migrados, o escalonador pode ter decidido migrar uma réplica

que poderia estar com um melhor desempenho, se comparada a outra réplica do mesmo operador. Entretanto, ao utilizar valores menores de d, há uma maior garantia de que o sistema poderá lidar com pequenas variações de carga e dependendo da aplicação, pode ser a melhor escolha.

Da mesma forma, ao variar o parâmetro de *limites de migrações por rodada*, a quantidade de migrações é afetada. Nesse caso, as migrações simultâneas geraram uma perturbação ainda maior no sistema, provocando uma diminuição mais acentuada na quantidade de eventos processados. O que poderia ser diferente se existisse um maior número de nós disponíveis: com uma maior quantidade de nós, a perturbação gerada pelas migrações poderia não existir, dado que as migrações poderiam acontecer entre nós que não compartilhassem a rede, pode exemplo.

Ao verificar que migrações podem ter um efeito negativo no desempenho do sistema, a seleção de operadores a serem migrados se torna um passo ainda mais importante. Migrar a réplica com melhor desempenho (mais rápida) pode ser prejudicial. Por exemplo, se um operador x possui duas réplicas x_1 e x_2 com uma vazão constante de 100 e 105 *eventos/s*, respectivamente, ou seja, o operador possui uma vazão de 105 *eventos/s*, que é a vazão de x_2, a réplica de melhor desempenho. Mas ambas tenham apresentado uma piora latência durante a rodada de avaliação. Caso o escalonador não encontre um nó para x_1 e x_2 seja migrada, a vazão do operador será reduzida para 100 *eventos/s* pelo menos até que a migração de x_2 seja concluída, o que não é desejável.

4.4.2 Desempenho do Algoritmo de Escalonamento Adaptativo

Aqui serão apresentados os resultados de desempenho das estratégias de escalonamento. Além dos resultados já analisados anteriormente, foram executados outros quatro cenários que estão detalhados na Tabela 4.7. O limite de migrações por rodada foi fixado em 1.

Cenário	*l_mig*	Carga de Trabalho	Estratégia de Escalonamento
10	1	Variada	Aleatória
11	1	Estável	Aleatória
12	1	Variada	Sem escalonamento (Base)
13	1	Estável	Sem escalonamento (Base)

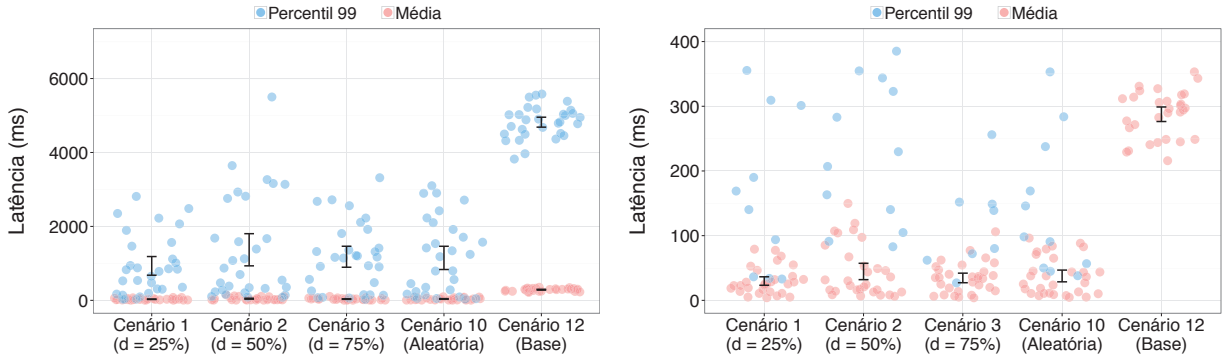
Tabela 4.7: Resumo dos níveis das variáveis independentes utilizados na avaliação das estratégias de escalonamento. O valor de sensibilidade não está exibido já que esse parâmetro não é utilizado pelas estratégias de escalonamento.

Para efeito de comparação, nesse experimento foram considerados os Cenários 1, 2 e 3 já utilizados na Seção 4.4.1. Nesses cenários, o *limite de migrações por rodada* (l_mig) é mantido fixo em 1, enquanto que o valor da *sensibilidade* (d) é variado: 25% (Cenário 1), 50% (Cenário 2) e 75% (Cenário 3).

A Figura 4.14 mostra um gráfico com a latência (percentil 99 e média) das estratégias de escalonamento. O cenário sem escalonamento é representado como *Base*. Nessa figura, são comparadas as latências da réplica mais rápida, nos cenários com carga de trabalho variada.

Ao se considerar a réplica mais rápida, não existe uma diferença significativa entre a latência dos Cenários 1, 2 e 3, com escalonador adaptativo, e o Cenário 10, com escalonador aleatório. Mesmo que o escalonador aleatório selecione uma réplica com bom desempenho para ser migrada, essa migração só acontece se existir outro nó com menor utilização de seus recursos físicos. Ao ser migrado para outro nó menos sobrecarregado, o operador não tem sua latência afetada negativamente.

O ganho de desempenho obtido da utilização da estratégia adaptativa, pode ser visto na comparação entre os Cenários 1, 2 e 3 e o Cenário 12 (*Base*). A média do percentil 99 da latência, foi de 932 *ms*, 1368 *ms* e 1179 *ms* para os Cenários 1, 2 e 3. Já para o *Base*, foi de 4819 *ms*, um ganho de 80%, 71% e 75%, respectivamente. Com relação à latência média, os valores foram 29 *ms*, 44 *ms* e 34 *ms* para os cenários com escalonamento adaptativo e 287 *ms* para o cenário *Base*. Um ganho de 89%, 84% e 87%.

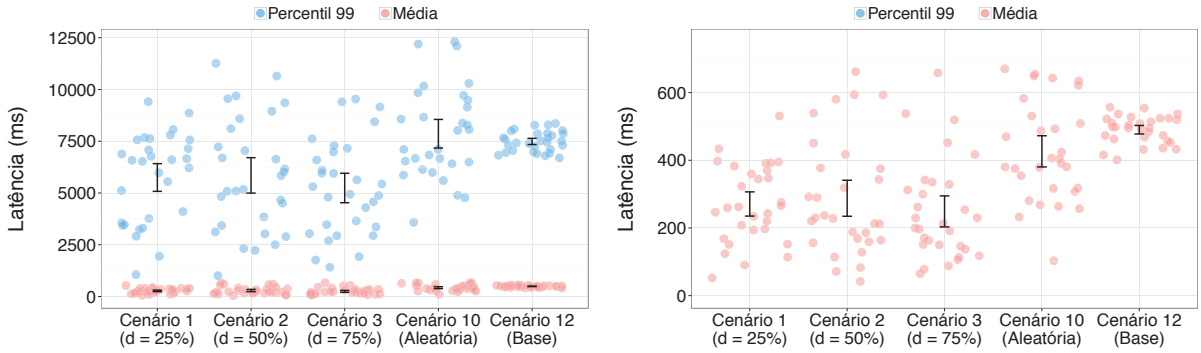


(a) Gráfico original.

(b) Gráfico com aproximação.

Figura 4.14: Latências das réplicas mais rápidas nos cenários com carga variada.

Os resultados da latência para a réplica mais lenta estão na Figura 4.15. Nesses gráficos, ficam evidentes as diferenças entre as estratégias de escalonamento: os operadores que são migrados na estratégia aleatória, não necessariamente são aqueles que realmente estão apresentando sinais de sobrecarga. O Cenário 10 (estratégia aleatória) apresentou um desempenho pior que o Cenário 12 (*Base*) ao se considerar a média do percentil 99 da latência: 7862 *ms*, contra 7490 *ms*. Já ao se considerar a latência média, o ganho foi de 13%.



(a) Gráfico original.

(b) Gráfico com aproximação.

Figura 4.15: Latências das réplicas lentas nos cenários com carga variada.

Na comparação da latência entre os cenários que utilizam o escalonador adaptativo (Cenários 1, 2 e 3) e o cenário que utiliza o escalonador aleatório (Cenário 10), temos que o

ganho, ao considerar o percentil 99, foi de 26%, 25% e 33%. Para a latência média, o ganho foi de 36%, 32% e 41%.

A Figura 4.16 mostra os resultados para os cenários com carga estável. Mais uma vez, os resultados são semelhantes. A quantidade de migrações e/ou a escolha de qual operador deve ser migrado não tem impacto na latência quando não existe a presença de sobrecarga, ainda que exista um nível elevado (estável) na utilização dos recursos.

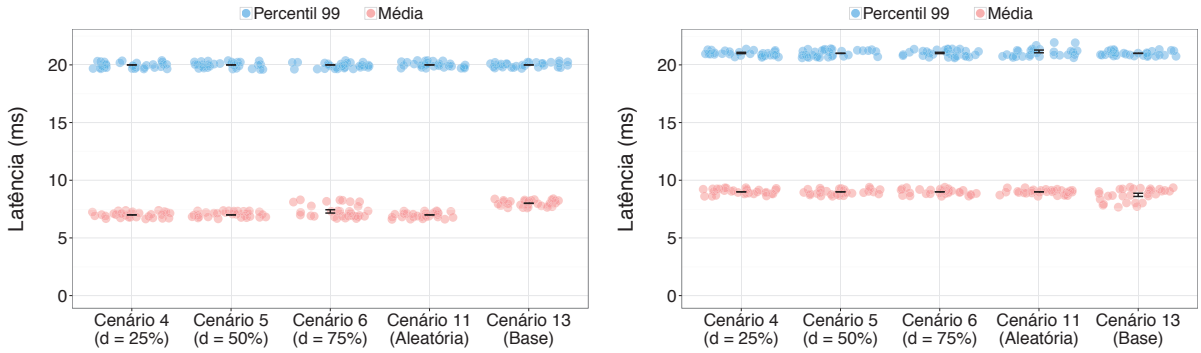


Figura 4.16: Latências das réplicas lentas nos cenários com carga estável.

Com relação à quantidade de eventos processados, os resultados são exibidos pelas Figuras 4.17 e 4.18. Nos experimentos com carga variada, a diferença entre a quantidade de eventos processados foi de 4213, 8853 e 13783 eventos dos Cenários 1, 2 e 3 para o Cenário 10. O mesmo aconteceu nos experimentos com carga estável, os Cenários 4, 5 e 6 processaram respectivamente 722, 835 e 1233 mais eventos que o Cenário 11.

O efeito da seleção aleatória de operadores a serem migrados, pode ser notado pela menor quantidade de eventos processados. Além da possível interrupção de processamento, ao migrar a réplica com melhor desempenho, a vazão do operador é temporariamente afetada, dado que a réplica mais lenta se torna responsável pelo processamento durante a migração.

Apesar da maior quantidade de eventos processados quando comparado aos cenários com escalonamento aleatório, nos experimentos com carga variada (Figura 4.17), os cenários com escalonamento adaptativo processaram menos eventos que o cenário sem escalonamento (*Base*). Entretanto, essa diferença não passou de 1,18%. Sendo de 1,18%, 0,91% e 0,63% a diferença entre o Cenário 12 e os Cenários 1, 2 e 3, respectivamente.

Já nos experimentos com carga estável (Figura 4.18), os cenários que utilizavam o esca-

lonador adaptativo, conseguiram processar mais eventos que o cenário *Base*. Visualmente, os intervalos de confiança dos Cenários 4, 5 e 6 não se sobrepõem aos do Cenário 13 e a diferença entre as médias é de 0,20%, 0,23% e 0,37%. Isso demonstra que a distribuição inicial dos operadores nos nós não era a melhor possível. Após o início da execução, o escalonador adaptativo realizou algumas migrações, que fizeram com que a distribuição dos operadores nos nós fosse otimizada e assim melhorasse o desempenho.

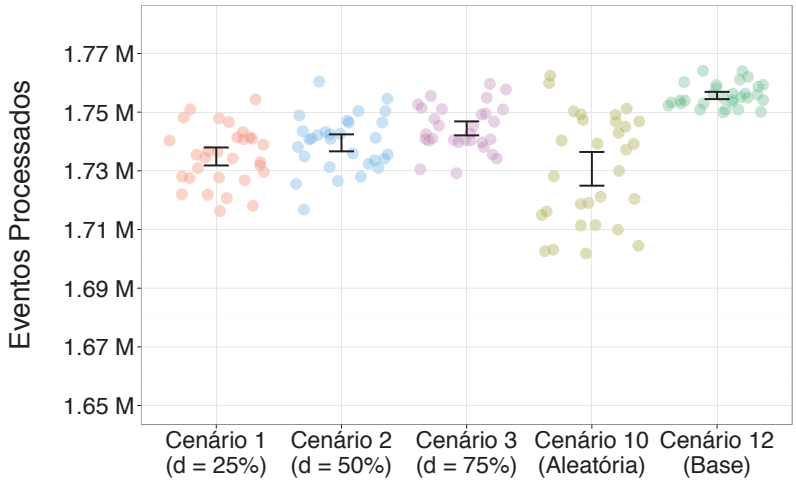


Figura 4.17: Quantidade de eventos processados nos cenários com carga variada. Os valores estão exibidos de forma abreviada.

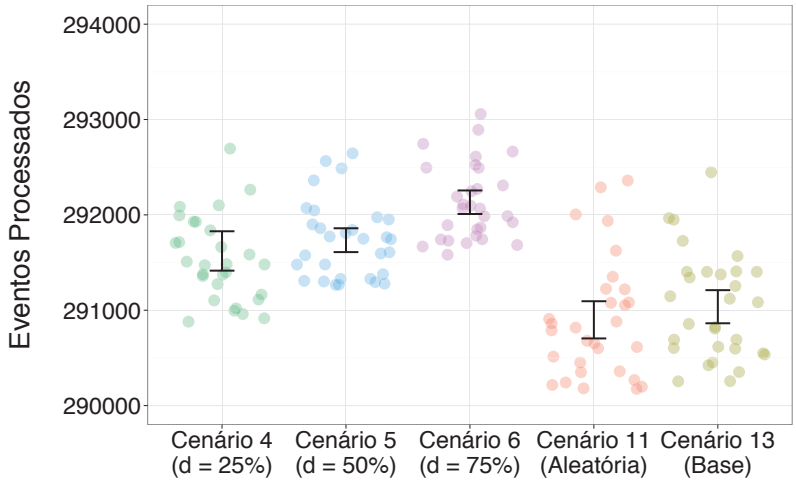


Figura 4.18: Quantidade de eventos processados nos cenários com carga estável.

4.4.3 Considerações Finais

Neste capítulo foram detalhados os cenários de avaliação e a topologia dos experimentos executados e posteriormente, foram apresentados os resultados desses experimentos. Foram analisados os diferentes parâmetros de configuração do algoritmo de escalonamento e o desempenho do mesmo, quando comparado a uma estratégia de escalonamento aleatória e a um cenário sem escalonamento.

Foi observado que, ao variar a sensibilidade e o limite de migrações por rodada, existiu uma influência na quantidade de migrações realizadas. Uma réplica de um operador, ao ser migrada, tem seu processamento interrompido e, caso essa seja a réplica de melhor desempenho, a vazão do operador é reduzida, já que durante esse período, a réplica mais lenta é a única responsável pelo processamento. Esse comportamento reforça que a escolha de qual operador deve ser migrado é de extrema importância para a efetividade do escalonamento. Os resultados demonstraram que, escolhas aleatórias podem não ser a melhor opção, mesmo que essa escolha ajude no balanceamento da utilização dos recursos disponíveis.

O ganho no desempenho demonstrado pelo algoritmo proposto em cenários com carga variada e estável, mostra que a solução se adapta a diferentes situações, ou seja, os mecanismos de monitoramento e migração utilizados, não dependem do tipo de aplicação que está sendo executada.

Capítulo 5

Conclusão

5.1 Sumário

Neste trabalho de dissertação foi proposto um algoritmo de escalonamento reativo para sistemas de processamento contínuo de eventos, que se adapta a diferentes situações de carga e tipo de aplicação. Em especial, o algoritmo proposto faz uso de uma métrica baseada na variação da latência dos operadores e utiliza migrações como estratégia de balanceamento da carga do sistema. Periodicamente, é feita uma avaliação onde os operadores com maior valor dessa métrica são selecionados para serem migrados. Em seguida, o algoritmo avalia se existe algum nó disponível onde a utilização de recursos prevista, caso receba a migração, seja menor que a do nó onde o operador está atualmente localizado. Finalmente, as migrações necessárias são realizadas. Através dessa abordagem, é realizado um balanceamento dinâmico do sistema com o principal objetivo de evitar que possíveis sobrecargas provoquem o aumento da latência para valores não aceitáveis.

A abordagem utilizada no algoritmo proposto partiu da avaliação do impacto da sobrecarga que diferentes recursos de *hardware* (CPU, memória RAM, rede e disco) podem ter no desempenho de um sistema *ESP*. Foram executados experimentos onde a utilização dos recursos era alterada de forma isolada e era observado o impacto decorrente na latência dos operadores do sistema. Todos os recursos avaliados provocaram algum efeito negativo na latência e, consequentemente, seu monitoramento se torna importante no contexto de escalonamento de sistemas *ESP*. Entretanto, não ficou claro que valores poderiam ser utilizados pelo escalonador do sistema na detecção de recursos sobrecarregados, sobretudo, de forma

genérica (independente de aplicação). Em alguns casos, uma utilização abaixo dos 50% em um recurso já pode representar uma ameaça ao desempenho do sistema, em outros, uma utilização de 90% pode ser perfeitamente aceitável. Ao considerar esses resultados e com o objetivo de prover uma solução adaptativa e independente de aplicação, é que foi introduzida a ideia de utilizar a variação do desempenho dos operadores como indício de sobrecarga.

O algoritmo proposto utiliza dois parâmetros de configuração: sensibilidade e limite de migrações por rodada. A sensibilidade indica a variação mínima percentual de latência entre duas medições que deve ser considerada para que a réplica em questão tenha a chance de ser migrada para outro nó. O limite de migrações por rodada define o número máximo de migrações que podem ser realizadas em cada rodada de avaliação. Para melhor entender o efeito que cada parâmetro possui no sistema, foram realizados experimentos onde foram atribuídos diferentes níveis a cada um desses parâmetros e também foi variado o tipo de carga recebida pelo sistema: estável ou variada. Os resultados demonstraram que o maior número de migrações provocados por um baixo valor de sensibilidade, ou um alto valor do limite de migrações por rodada, pode causar uma queda na quantidade de eventos processados pelo sistema. Essa queda acontece em decorrência das interrupções que acontecem durante o processo de migração dos operadores.

Também foi avaliado o desempenho do algoritmo proposto quando comparado a uma estratégia chamada de "aleatória". Nessa estratégia, ao invés de se basear em alguma métrica monitorada pelo sistema, operadores são selecionados aleatoriamente e migrados caso exista algum nó com menor utilização de recursos. Nos cenários executados, o desempenho do algoritmo de escalonamento adaptativo foi superior, o ganho na latência média foi maior que 32%.

A partir dos diferentes experimentos executados foi possível constatar que a estratégia de seleção de operadores a serem migrados, utilizada no algoritmo proposto nessa dissertação, proporcionou um ganho de desempenho. O ganho no desempenho foi demonstrado em todos os cenários avaliados, não importando o valor dos parâmetros utilizados. A latência média de processamento chegou a ter uma diminuição maior que 84%, enquanto que a quantidade de eventos processados caiu apenas 1,18% em um cenário com variações bruscas de carga. Além disso, o algoritmo também proporciona uma solução adaptativa e flexível, dado que provê um mecanismo de monitoramento independente do tipo de aplicação a ser executada.

5.2 Limitações e Trabalhos Futuros

O algoritmo de escalonamento adaptativo proposto neste trabalho de dissertação, juntamente com sua avaliação, possuem algumas fraquezas decorrentes das limitações dos cenários avaliados e premissas consideradas, que de certa forma, reduzem o escopo dos resultados. A seguir essas limitações estão descritas, bem como possíveis trabalhos futuros que poderiam saná-las:

- A escala de avaliação do algoritmo foi limitada pela disponibilidade de máquinas virtuais do ambiente de *computação na nuvem* utilizado, ambiente esse, localizado no Laboratório de Sistemas Distribuídos. Em uma infraestrutura maior, com dezenas de máquinas virtuais, seria possível que o algoritmo de escalonamento adaptativo demonstrasse uma melhora mais evidente no desempenho do sistema. Existiria a possibilidade de que em cada rodada, houvesse mais nós disponíveis para receber a migração dos operadores, o que também poderia permitir que a quantidade migrações por rodada de avaliação fosse maior. Além de poder verificar a escalabilidade do algoritmo proposto e da ferramenta utilizada, StreamMine3G.

- Outra limitação da avaliação se deve à aplicação utilizada nos experimentos. A aplicação possuía apenas três etapas de operadores que realizavam uma computação específica para a sobrecarga dos recursos de *hardware*. Em avaliações futuras, poderiam ser utilizadas aplicações reais, como a de análise de anomalias em redes elétricas e transações no mercado financeiro.

- Uma simplificação realizada nos experimentos foi a utilização de operadores sem estado, o que interfere diretamente nas interrupções geradas por migrações. A transferência do estado do operador aumenta significantemente o tempo total que uma migração demora para ser realizada. Sendo assim, o algoritmo poderia ser modificado para levar em consideração o custo da transferência dos operadores com estado durante a fase seleção dos candidatos à migração.

- Os experimentos foram executados considerando-se uma infraestrutura com número fixo de nós. O algoritmo pode ser estendido para englobar situações de provisionamento, onde seriam incluídos ou removidos nós da infraestrutura automaticamente,

dependendo da situação de sobrecarga dos operadores do sistema.

- O algoritmo está limitado a melhora e/ou manutenção do desempenho do sistema. Há a possibilidade de adequação do algoritmo para atingir outros objetivos, como economia de energia.

Um possível desdobramento seria a avaliação da abordagem utilizada em outros contextos, como por exemplo, no problema do escalonamento de máquinas virtuais em ambientes de *computação na nuvem*. Outra possibilidade seria um estudo dirigido para a avaliação e evolução da abordagem aleatória de seleção de operadores a serem migrados, que foi utilizada nos experimentos descritos no Capítulo 4.

Bibliografia

[1] Apache Software Foundation. Samza Project Incubation Status. `https://samza.incubator.apache.org/`, 2014. [Online; accessed 28-August-2014].

[2] Apache Software Foundation. Storm Project Incubation Status. `https://storm.incubator.apache.org/`, 2014. [Online; accessed 28-August-2014].

[3] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, et al. The design of the borealis stream processing engine. In *CIDR*, volume 5, pages 277–289, 2005.

[4] FJ Almeida Morais, F Vilar Brasileiro, R Vigolvino Lopes, R Araujo Santos, Wade Satterfield, and Leandro Rosa. Autoflex: Service agnostic auto-scaling framework for iaas deployment models. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 42–49. IEEE, 2013.

[5] Mauro Andreolini, Sara Casolari, Michele Colajanni, and Michele Messori. Dynamic load management of virtual machines in cloud architectures. *Cloud Computing*, pages 201–214, 2010.

[6] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[7] Raphaël Barazzutti, Thomas Heinze, André Martin, Emanuel Onica, Pascal Felber, Christof Fetzer, Zbigniew Jerzak, Marcelo Pasin, and Etienne Riviere. Elastic scaling of

a high-throughput content-based publish/subscribe engine. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, pages 567–576. IEEE, 2014.

[8] R.S. Barga, J. Goldstein, M. Ali, and M. Hong. Consistent streaming through time: A vision for event stream processing. *arXiv preprint cs/0612115*, 2006.

[9] Dominique Bellenger, Jens Bertram, Andy Budina, Arne Koschel, Benjamin Pfänder, Carsten Serowy, Irina Astrova, Stella Gatziu Grivas, and Marc Schaaf. Scaling in cloud environments. *Recent Researches in Computer Science*, 2011.

[10] Kenneth P Birman, Lakshmi Ganesh, and Robbert Van Renesse. White paper running smart grid control software on cloud computing architectures. *Computational Needs for the Next Generation Electric Grid*, 2011.

[11] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Mobile Data Management*, pages 3–14. Springer, 2001.

[12] Andrey Brito, Andre Martin, Thomas Knauth, Stephan Creutz, Diogo Becker, Stefan Weigert, and Christof Fetzer. Scalable and low-latency data processing with stream mapreduce. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 48–58. IEEE, 2011.

[13] Javier Cervino, Evangelia Kalyvianaki, Joaquin Salvachua, and Peter Pietzuch. Adaptive provisioning of stream processing systems in the cloud. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 295–301. IEEE, 2012.

[14] Sharma Chakravarthy and Raman Adaikkalavan. Events and streams: harnessing and unleashing their synergy! In *Proceedings of the second international conference on Distributed event-based systems*, pages 1–12. ACM, 2008.

[15] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, F. Reiss, and M.A. Shah. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668. ACM, 2003.

[16] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing, and Stanley B Zdonik. Scalable distributed stream processing. In *CIDR*, volume 3, pages 257–268, 2003.

[17] T. Condie, N. Conway, P. Alvaro, J.M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 21–21, 2010.

[18] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[19] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *Software Engineering, IEEE Transactions on*, (5):662–675, 1986.

[20] Andrew Fox, Andrew Turner, and Hyong S Kim. Resource contention-aware virtual machine management for enterprise applications. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 1641–1646. IEEE, 2012.

[21] Lukasz Golab and M Tamer Özsu. Issues in data stream management. *ACM Sigmod Record*, 32(2):5–14, 2003.

[22] Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. Pearson Education, 2013.

[23] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. A catalog of stream processing optimizations. *ACM Computing Surveys (CSUR)*, 46(4):46, 2014.

[24] Yennun Huang, Chandra Kintala, Nick Kolettis, and N Dudley Fulton. Software rejuvenation: Analysis, module and applications. In *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on*, pages 381–390. IEEE, 1995.

[25] Jeong-Hyon Hwang, Sanghoon Cha, Ugur Cetintemel, and Stan Zdonik. Borealis-r: a replication-transparent stream processing system for wide-area monitoring applicati-

ons. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1303–1306. ACM, 2008.

[26] W. Jiang, V.T. Ravi, and G. Agrawal. Comparing map-reduce and freeride for data-intensive applications. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–10. IEEE, 2009.

[27] Wilhelm Kleiminger, Evangelia Kalyvianaki, and Peter Pietzuch. Balancing load in stream processing with the cloud. In *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on*, pages 16–21. IEEE, 2011.

[28] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, 2011.

[29] Andre Martin, Christof Fetzer, and Andrey Brito. Active replication at (almost) no cost. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*, pages 21–30. IEEE, 2011.

[30] Cary Millsap. Thinking clearly about performance. *Queue*, 8(9):10, 2010.

[31] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 170–177. IEEE, 2010.

[32] Joel Wolf, Nikhil Bansal, Kirsten Hildrum, Sujay Parekh, Deepak Rajan, Rohit Wagle, Kun-Lung Wu, and Lisa Fleischer. Soda: An optimizing scheduler for large-scale stream-based distributed computer systems. *Middleware 2008*, pages 306–325, 2008.

[33] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 358–369. VLDB Endowment, 2002.

Apêndice A

Cargas de Trabalho Utilizadas

Neste capítulo é apresentada, através de tabelas e gráficos, a carga de trabalho utilizada nos cenários de avaliação dos experimentos executados nesta dissertação. O tipo de carga possui dois níveis: estável e variada. No caso da carga variada, a taxa de chegada de eventos muda durante a execução, sempre em ciclos de 60 segundos. As tabelas ilustram a carga para os 10 períodos do experimento, dado que cada execução tem duração de 600 segundos. Para os experimentos com carga estável, estão detalhados apenas as taxas de chegada iniciais, que permanecem fixas durante toda a execução.

A.0.1 Tipo de Carga de Trabalho Estável

Workers	**Taxa de Chegada de Eventos (eventos/s)**
1, 3, 5, 7, 9, 11, 13	100
2, 4, 6, 8, 10, 12	50

Tabela A.1: Taxas de chegada de eventos nos cenários com carga de trabalho estável.

A.0.2 Tipo de Carga de Trabalho Variada

Carga de trabalho do Worker 1

1	2	3	4	5	6	7	8	9	10
Alta	Alta	Baixa	Baixa	Alta	Alta	Alta	Baixa	Baixa	Baixa

Tabela A.2: Variação da carga de trabalho do Worker 1.

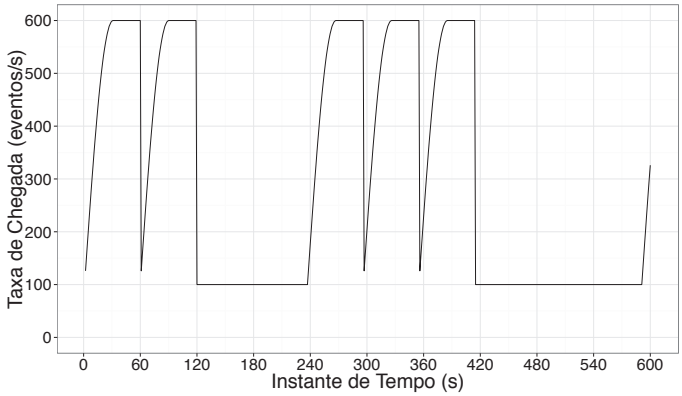


Figura A.1: Taxa de chegada de eventos para o Worker 1.

Carga de trabalho do Worker 2

1	2	3	4	5	6	7	8	9	10
Alta	Baixa	Baixa	Baixa	Alta	Alta	Alta	Alta	Baixa	Baixa

Tabela A.3: Variação da carga de trabalho do Worker 2.

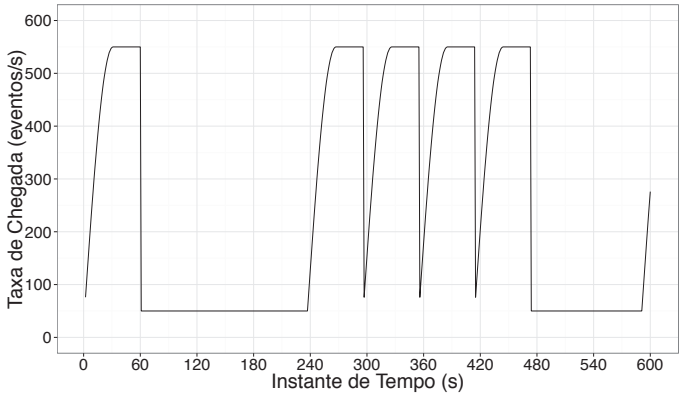


Figura A.2: Taxa de chegada de eventos para o Worker 2.

Carga de trabalho do Worker 3

1	2	3	4	5	6	7	8	9	10
Baixa	Alta	Baixa	Alta	Baixa	Baixa	Baixa	Alta	Alta	Alta

Tabela A.4: Variação da carga de trabalho do Worker 3.

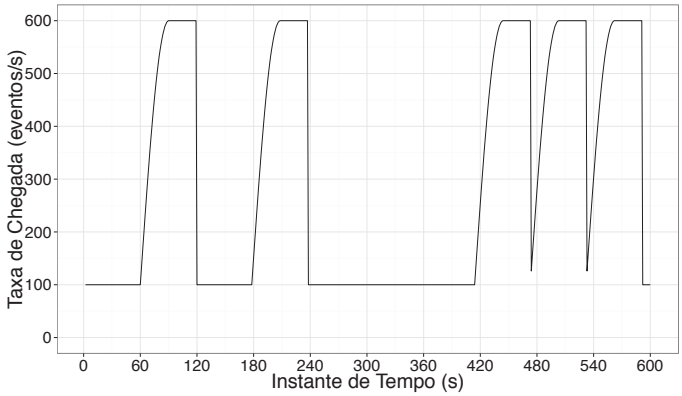


Figura A.3: Taxa de chegada de eventos para o Worker 3.

Carga de trabalho do Worker 4

1	2	3	4	5	6	7	8	9	10
Alta	Baixa	Baixa	Alta	Baixa	Alta	Baixa	Alta	Alta	Baixa

Tabela A.5: Variação da carga de trabalho do Worker 4.

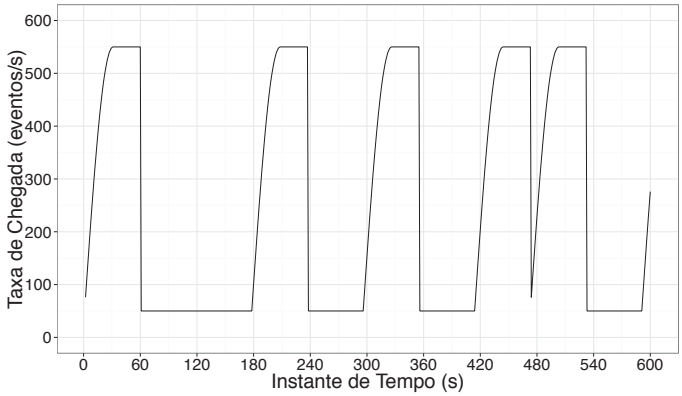


Figura A.4: Taxa de chegada de eventos para o Worker 4.

Carga de trabalho do Worker 5

1	2	3	4	5	6	7	8	9	10
Alta	Baixa	Baixa	Baixa	Alta	Baixa	Baixa	Baixa	Alta	Baixa

Tabela A.6: Variação da carga de trabalho do Worker 5.

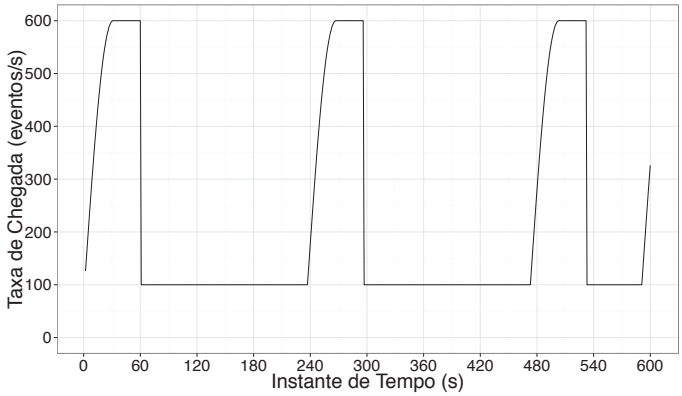


Figura A.5: Taxa de chegada de eventos para o Worker 5.

Carga de trabalho do Worker 6

1	2	3	4	5	6	7	8	9	10
Alta	Alta	Alta	Baixa	Alta	Baixa	Alta	Baixa	Baixa	Baixa

Tabela A.7: Variação da carga de trabalho do Worker 6.

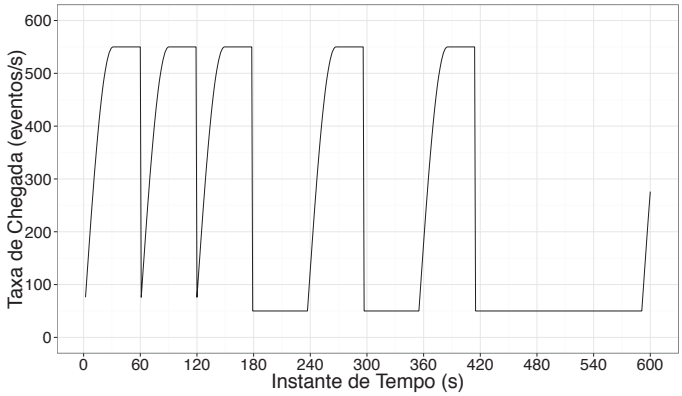


Figura A.6: Taxa de chegada de eventos para o Worker 6.

Carga de trabalho do Worker 7

1	2	3	4	5	6	7	8	9	10
Baixa	Alta	Alta	Baixa	Baixa	Alta	Alta	Alta	Alta	Baixa

Tabela A.8: Variação da carga de trabalho do Worker 7.

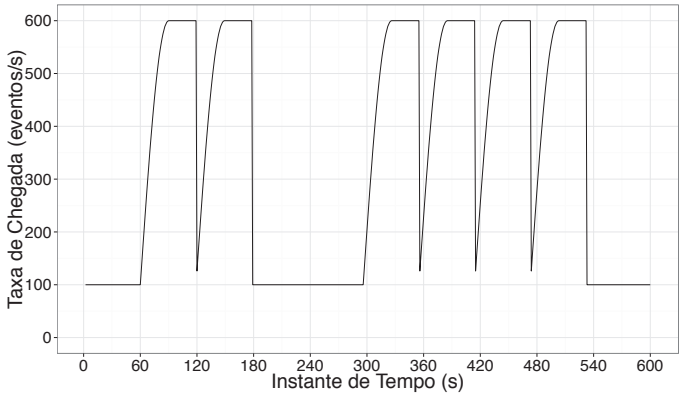


Figura A.7: Taxa de chegada de eventos para o Worker 7.

Carga de trabalho do Worker 8

1	2	3	4	5	6	7	8	9	10
Alta	Alta	Baixa	Alta	Baixa	Baixa	Baixa	Baixa	Baixa	Baixa

Tabela A.9: Variação da carga de trabalho do Worker 8.

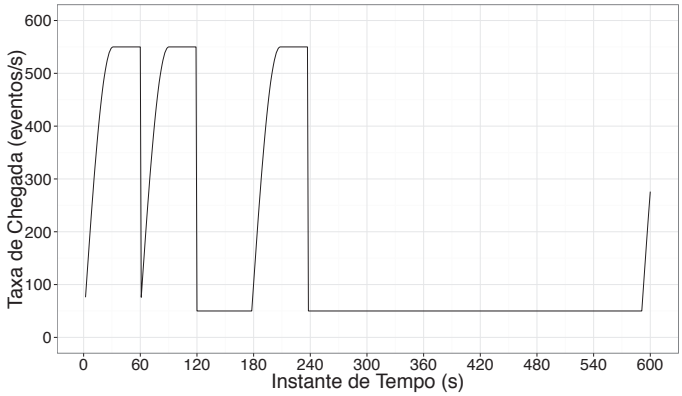


Figura A.8: Taxa de chegada de eventos para o Worker 8.

Carga de trabalho do Worker 9

1	2	3	4	5	6	7	8	9	10
Baixa	Alta	Alta	Alta	Baixa	Baixa	Baixa	Baixa	Alta	Alta

Tabela A.10: Variação da carga de trabalho do Worker 9.

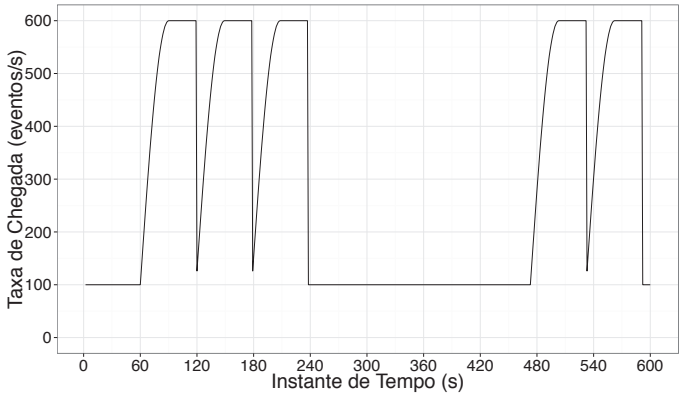


Figura A.9: Taxa de chegada de eventos para o Worker 9.

Carga de trabalho do Worker 10

1	2	3	4	5	6	7	8	9	10
Baixa	Alta	Baixa	Baixa	Baixa	Baixa	Alta	Baixa	Baixa	Alta

Tabela A.11: Variação da carga de trabalho do Worker 10.

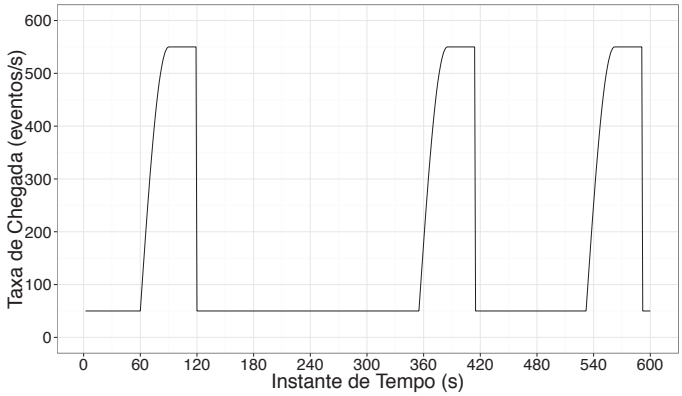


Figura A.10: Taxa de chegada de eventos para o Worker 10.

Carga de trabalho do Worker 11

1	2	3	4	5	6	7	8	9	10
Baixa	Baixa	Alta	Alta	Baixa	Baixa	Baixa	Baixa	Alta	Baixa

Tabela A.12: Variação da carga de trabalho do Worker 11.

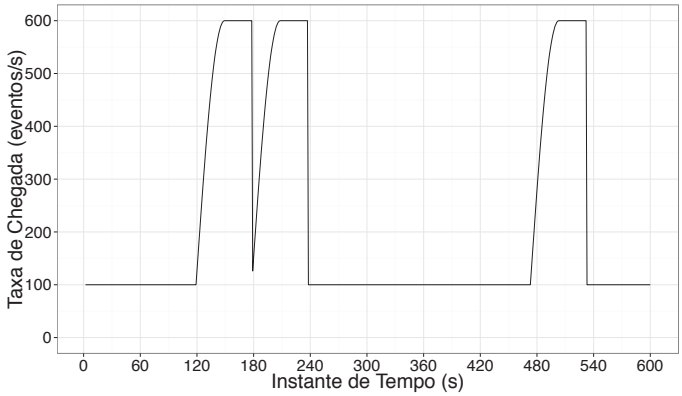


Figura A.11: Taxa de chegada de eventos para o Worker 11.

Carga de trabalho do Worker 12

1	2	3	4	5	6	7	8	9	10
Baixa	Alta	Alta	Baixa	Alta	Baixa	Alta	Alta	Alta	Alta

Tabela A.13: Variação da carga de trabalho do Worker 12.

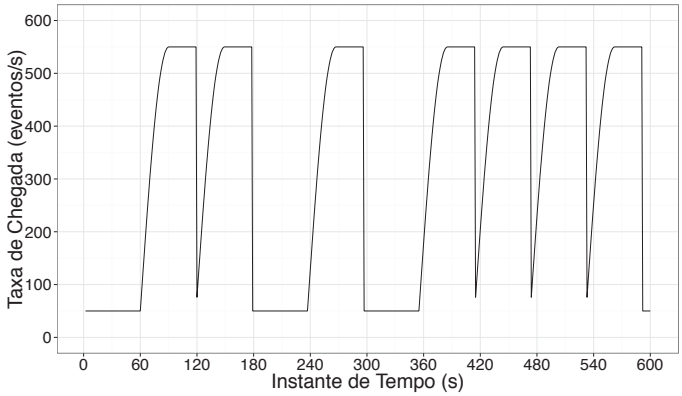


Figura A.12: Taxa de chegada de eventos para o Worker 12.

Carga de trabalho do Worker 13

1	2	3	4	5	6	7	8	9	10
Alta	Alta	Baixa	Baixa	Baixa	Baixa	Alta	Baixa	Baixa	Baixa

Tabela A.14: Variação da carga de trabalho do Worker 13.

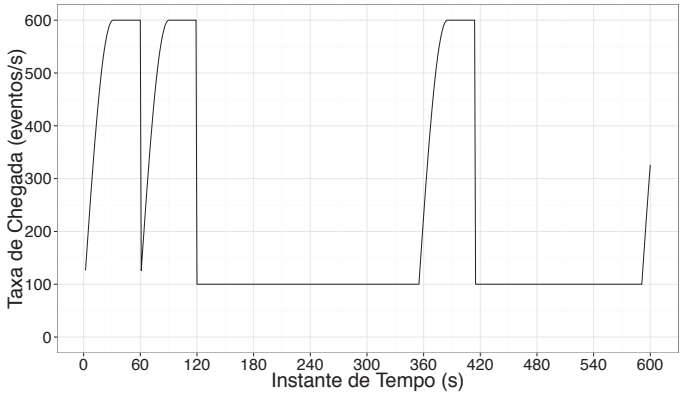


Figura A.13: Taxa de chegada de eventos para o Worker 13.