UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Túlio Braga Moreira Pinto

Previsão do tempo de resposta de aplicações paralelas de processamento de dados massivos em ambientes de nuvem

Belo Horizonte
2019

Túlio Braga Moreira Pinto

Previsão do tempo de resposta de aplicações paralelas de processamento de dados massivos em ambientes de nuvem

Versão Final

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientadora: Jussara Marques de Almeida
Coorientadora: Ana Paula Couto da Silva

Belo Horizonte
2019

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa CRB 6ª Região nº 1510

Pinto, Túlio Braga Moreira.

P659p Previsão do tempo de resposta de aplicações paralelas de processamento de dados massivos em ambientes de nuvem / Túlio Braga Moreira Pinto — Belo Horizonte, 2019.
xxiii, 78 f.: il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.
Orientadora: Jussara Marques de Almeida.
Coorientadora: Ana Paula Couto da Silva

1. Computação – Teses. 2. Computação em nuvem – Teses. 3. Sistemas distribuídos – Teses. 3. Processamento massivo de dados – Teses. I. Orientadora. II. Coorientadora. III. Título.

CDU 519.6*22(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Previsão do tempo de resposta de aplicações paralelas de processamento de dados massivos em ambientes de nuvem

TÚLIO BRAGA MOREIRA PINTO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Jussara Marques de Almeida Gonçalves

PROFA. JUSSARA MARQUES DE ALMEIDA GONÇALVES - Orientadora
Departamento de Ciência da Computação - UFMG

PROFA. ANA PAULA COUTO DA SILVA - Coorientadora
Departamento de Ciência da Computação - UFMG

PROF. FABRÍCIO MURAI FERREIRA
Departamento de Ciência da Computação - UFMG

PROF. DORGIVAL OLAVO GUEDES NETO
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 15 de Julho de 2019.

Dedico esta dissertação de mestrado ao meu pai, a minha mãe e ao meu irmão que sempre estiveram presentes me incentivando nos estudos e na carreira.

Agradecimentos

Agradeço à Prof.ª Dr.ª Jussara Marques de Almeida (UFMG) por toda a ajuda e pelos conselhos durante a minha trajetória no mestrado. Agradeço também por ser compreensiva e sempre apoiar minhas decisões durante todo este período em que fui orientado por ela.

Agradeço à Prof.ª Dr.ª Ana Paula Couto da Silva (UFMG) pelo apoio, conselhos e compartilhamento de conhecimento durante todo o meu mestrado e durante o projeto BigSea.

Agradeço ao Prof.º Dr.º Danilo Ardagna (Politecnico di Milano) e demais colegas pelo auxílio e compartilhamento de conhecimento durante todo o projeto do BigSea.

Agradeço à Prof.ª Dr.ª Cristina Duarte Murta (CEFET-MG) e Prof.ª Dr.ª Anolan Yamilé Milanés Barrientos (CEFET-MG) pelos conselhos e indicação ao Programa de Pós-Graduação em Ciência da Computação do DCC/UFMG.

“*Libertem-se da escravidão mental, ninguém além de nós mesmos pode libertar nossas mentes [...] nenhum deles pode parar o tempo.*”

(Bob Marley)

Resumo

A popularização das aplicações online e intensas em dados nos últimos anos trouxe consigo novos desafios à computação. Apesar de a flexibilidade e a elasticidade da computação em nuvem terem facilitado a alocação de recursos de hardware e software sob demanda, a heterogeneidade e a irregularidade nos padrões de acesso das aplicações massivas em dados, por outro lado, tornaram esta tarefa mais desafiadora. Em consequência, a combinação destas características tornam a previsão de desempenho (p. ex: previsão do tempo de resposta das aplicações) mais complexa. Sendo assim, este trabalho explora dois modelos analíticos para a previsão do tempo de resposta de aplicações paralelas na plataforma Spark, muito popular para processamento de grandes volumes de dados. O primeiro modelo é baseado em um *fork/join*, no qual uma aplicação é dividida em N tarefas que são processadas em paralelo em múltiplos servidores. Este modelo captura o tempo do servidor mais lento para computar os atrasos de sincronização. O segundo modelo é baseado em teoria de filas e considera a precedência entre as tarefas para estimar os atrasos de sincronização. Múltiplos cenários experimentais são considerados, incluindo atividades recorrentes como o *wordcount*, algoritmos frequentemente utilizados em aprendizado de máquina, como o SVM, o Logistic Regression e o K-Means, e consultas *ad-hoc* comuns em análise de dados. Para o modelo baseado em precedência de tarefas, os resultados das previsões apresentaram erro médio inferior a 20% para a maioria dos cenários, considerado tipicamente baixo para modelos analíticos. Ainda, com um tempo de execução na casa dos milissegundos, este modelo se mostrou eficaz para a reconfiguração dinâmica de sistemas paralelos, tarefa importante na garantia de qualidade de serviço das aplicações massivas em dados. Ambos os modelos *fork/join* e de precedência de tarefas são comparados com o modelo de simulação *DagSim*, considerado estado da arte para previsão de desempenho de aplicações Hadoop e Spark.

Palavras-chave: Previsão de desempenho, Computação em nuvem, Spark, Processamento massivo de dados, Aplicações paralelas, Sistemas Distribuídos

Abstract

The popularity of online and data-intensive applications presented new challenges to computing. Although cloud computing technology has enabled on-demand resource scheduling, the data access heterogeneity and irregularity of data-intensive applications have increased the difficulty of both hardware and software resource scheduling. Nonetheless, the performance prediction (e.g.: response time) of such applications increase in complexity as all these characteristics are combined. Thus, this research explores two analytical models for the response time prediction of parallel applications running on Apache Spark, one of the most popular frameworks for massive data-processing. The first model is based on a fork/join queues, in which an application is split into N tasks and processed in parallel in multiple servers. This model captures the synchronization delays perceived in the slowest server. The second model is based on queuing networks. It considers the precedence relationship between the application tasks to compute the synchronization delays. Multiple experimental scenarios were considered, including the parallel wordcount algorithm, machine learning common algorithms, such as SVM, Logistic Regression, and K-Means, and ad-hoc data analytics queries. The precedence relationship model presented a mean error less than 20% for most of the experimental scenarios, which is typically considered reasonable for analytical models. Yet, both models presented execution times in the range of milliseconds. Such a low execution time enables the usage of the models for the dynamic provisioning of parallel systems, an important task to guarantee the quality of service of massive data-processing applications. Both the analytical models were compared to the DagSim simulation model, the state-of-art model for performance prediction of Hadoop and Spark applications.

Keywords: Performance prediction, Cloud computing, Spark, Massive data-processing, Parallel applications, Distributed Systems

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Os avanços recentes de hardware para computacão, especialmente no que tange a conectividade e o armazenamento, possibilitaram o surgimento de uma nova gama de aplicações orientadas a dados. Neste cenário, todo tipo de dado gerado pelos dispositivos, pelas aplicações ou pelos próprios usuários é, portanto, armazenado. Para aproveitar a riqueza de informação presente em cada fração de tempo, em toda parte do mundo, aplicações em dispositivos cada vez menores, geralmente conectados a redes sem fio, são frequentemente integradas a sistemas distribuídos [Coulouris et al., 2011]. Muitas vezes, os recursos computacionais utilizados são provenientes do que chamamos de nuvem pública: hardware e software em *datacenters* disponibilizados através de pagamento por demanda, compartilhados com o público geral [Fox et al., 2009].

Assim, considera-se haver uma transição de intensidade em processamento para intensidade em dados, alterando o fator limitante das aplicações. A necessidade de poder computacional soma-se aos desafios de lidar com um grande volume de dados, a complexidade destes dados e a velocidade com que estes dados se multiplicam e precisam ser processados [Kleppmann, 2017]. Estes desafios vão de encontro a uma das mais populares definições para o que são dados massivos (*Big Data*). Para Beyer & Laney [2012] o conceito se trata de "grande volume, velocidade e variedade de informações e requer novas formas de processamento que possibilitem uma melhor tomada de decisões, descoberta de *insights* e otimização de processos". De forma similar, Jacobs [2009] define como sendo dados cujo tamanho nos força a procurar além das alternativas testadas e predominantes à época.

Com a consolidação e popularização de diferentes aplicações online e intensas em dados nos últimos anos (comércio eletrônico, redes sociais, internet das coisas, etc), os desafios da computação em larga escala se tornaram mais evidentes em diferentes áreas do conhecimento [Chen & Zhang, 2014]. Neste quesito, uma das principais vertentes de estudo passou a ser a de sistemas distribuídos para processamento massivo de dados. O surgimento do MapReduce [Dean & Ghemawat, 2004], um modelo de programação simples e escalável para processamento de dados em paralelo em milhares de máquinas e sua implementação tolerante a falhas e de código aberto mais popular, o Hadoop[1],

[1]https://hadoop.apache.org/

impulsinaram este cenário. Posteriormente, diversos outros modelos e plataformas para processamento distribuído tolerante a falhas surgiram ao longo dos anos, sejam de propósito específico ou geral. Como exemplos, pode-se citar o Storm[2] para processamento massivo contínuo (*streaming*) e o Giraph[3] para processamento de grafos de larga escala. Atualmente, o Spark[4] é o sucessor mais popular do Hadoop. Esta plataforma usa a memória para otimizar o acesso a dados. Com uma comunidade de cerca de 225 mil membros e mais de 1200 contribuidores no Github [Databricks, 2016], o Spark é uma plataforma para processamento massivo de dados em paralelo, tolerante a falhas, de propósito geral, com bibliotecas que cobrem os principais algoritmos de aprendizado de máquina e mineração de dados existentes atualmente.

Adicionalmente, a consolidação da computação em nuvem permitiu o acesso a recursos computacionais em larga escala com baixo investimento inicial. Tanto a flexibilidade quanto a elasticidade presentes na natureza dos recursos computacionais e na política de preços dos ambientes em nuvem facilitaram o acesso das aplicações menores a uma gama de recursos antes alcançáveis apenas para as grandes corporações. Tais características têm contribuído para uma maior adoção da nuvem como base para aplicações massivas em dados. Por exemplo, as instalações de Spark em nuvem apresentaram um crescimento de 10% de 2015 para 2016, saltando de 51% das instalações totais para 61% [Databricks, 2016].

Com uma forte tendência de uso de nuvem pública para aplicações intensas em dados, a gestão eficiente dos recursos computacionais disponíveis passa a ser uma tarefa de suma importância, tanto para a redução de custos pagos por demanda, quanto para o alcance de requisitos de qualidade de serviço das aplicações. Portanto, mecanismos que permitam a previsão do tempo de resposta deste tipo de aplicação, de acordo com os recursos computacionais disponíveis, se tornam aliados importantes. Tais previsões podem ser usadas, por exemplo, para determinar a quantidade mínima de recursos necessários para que uma aplicação intensa em dados seja executada, dado um limiar de qualidade de serviço.

Entretanto, devido à característica compartilhada dos recursos, ao volume e à irregularidade dos dados, prever o tempo de resposta nestes cenários torna-se uma tarefa bastante desafiadora. Adicionalmente, pensando em reconfiguração dinâmica de sistemas paralelos, ou seja, o ajuste autônomo dos recursos de um sistema paralelo, como o principal caso de uso dos mecanismos de previsão, esta tarefa se torna ainda mais crítica. Neste cenário, é necessário balancear a rapidez com que as previsões são concluídas e a precisão da previsão em si. Caso esta combinação de fatores não esteja adequadamente calibrada, pode-se chegar a duas situações indesejadas: (i) uma previsão bastante rápida,

[2]https://storm.apache.org/
[3]https://giraph.apache.org/
[4]https://spark.apache.org/

mas de precisão ruim; (ii) uma previsão bastante precisa, mas demorada. Estas situações poderiam levar a: (i) superestimar os recursos necessários para o sistema paralelo executar a aplicação, dado um determinado requisito de qualidade de serviço, neste caso, causando subutilização de recursos; ou (ii) subestimar os recursos necessários, neste caso, prevenindo o sistema paralelo de executar a aplicação atingindo os requisitos de qualidade de serviço predefinidos. Sendo assim, a principal motivação deste trabalho é permitir previsões rápidas e suficientemente precisas para o tempo de resposta de aplicações paralelas massivas em dados.

No campo da previsão de desempenho, diversos modelos foram propostos ao longo dos anos, cada qual com seu cenário alvo. Tratam-se de abstrações com a finalidade de representar, de maneira genérica, o comportamento de um sistema durante a sua execução. Por se tratar de uma abstração, um modelo de desempenho captura apenas o mínimo necessário para alcançar o nível de representatividade adequado para o objetivo da modelagem, sendo que o único modelo completamente confiável de um sistema é o próprio sistema Menasce et al. [2004].

Dito isto, existem dois tipos principais de modelos de desempenho de sistemas: os modelos analíticos e os modelos de simulação. Os modelos analíticos são baseados em um conjunto de equações matemáticas e algoritmos que produzem métricas de acordo com parâmetros de entrada. As representações matemáticas e algorítmicas costumam não conter muitos detalhes sobre o sistema e são geralmente projetadas para execução rápida e eficiente. Já, os modelos de simulação se baseiam em emular a estrutura estática e os aspectos dinâmicos de um sistema de acordo com uma carga de trabalho, sendo ela observada ou sintética. Para que uma simulação ocorra, geralmente necessita-se de uma grande variedade de características sobre o sistema, resultando em uma avaliação detalhada, porém de maior complexidade. Por este motivo, modelos de simulação tendem a ser mais precisos, porém costumam demorar mais que os modelos analíticos para produzir uma previsão.

Dentre os modelos analíticos, Nelson & Tantawi [1988] propuseram um modelo baseado em filas *fork/join*, onde uma tarefa é dividida em subtarefas e processadas em servidores de dados distribuídos. Badue et al. [2010] apresentou um simplificação do modelo de Nelson & Tantawi [1988]. Clement & Quinn [1993] propuseram um modelo para estimar o *speedup* para variações na quantidade de processadores com foco em aplicações de tamanho invariável no *Dataparallel-C*. Mak & Lundstrom [1990] desenvolveram um modelo baseado em teoria de filas e grafo de precedências, representando, respectivamente, o sistema paralelo e as tarefas da aplicação. Liang & Tripathi [2000] estenderam o modelo proposto por Mak & Lundstrom [1990] para contemplar a previsão de múltiplas aplicações concorrentes. Thomasian & Bay [1986] apresentaram um modelo baseado em cadeias de Markov. Adve & Vernon [2004] focaram em aplicações paralelas de memória compartilhada. Ipek et al. [2005] utilizaram modelos de aprendizado de máquina treinados a partir de de execuções prévias das aplicações. Yang & Sun [2011] abordaram a previsão de desempenho de *jobs map-reduce* no Hadoop. Vianna et al. [2013] aplicaram dois modelos previamente propostos por outros autores também na previsão de *jobs map-reduce*.

Já, entre os modelos de simulação, Bagrodia et al. [1999] propuseram o COMPASS, um simulador para previsão de desempenho de aplicações MPI e MPI-IO. Zheng et al. [2004] apresentaram o BigSim, um simulador paralelizável para representar computadores *petaflops*. Uysal et al. [1998] desenvolveram um simulador genérico para aplicações intensas em dados, mas pouco eficiente para simular muitos processadores. Song et al. [2013] apresentaram um analisador para *jobs map-reduce* do Hadoop 1.x. Ardagna et al. [2016], no âmbito de *jobs map-reduce* do Hadoop 2.x, utilizaram as ferramentas JMT [Bertoli et al., 2009], baseada em teoria de filas, e a GreatSPN [Chiola, 1985], de redes estocásticas baseadas em Petri Nets [Reisig et al., 2013]. Mais recentemente, Ardagna et al. [2018] propuseram o DagSim, modelo baseado em simulação de eventos discretos que superam a precisão do JMT e o GreatSPN.

Considerando as características particulares do modelo de execução do Apache Spark, levanta-se neste trabalho que a característica distribuída dos dados e as relações de dependência entre as tarefas das aplicações paralelas executadas nesta plataforma são os fatores de maior representatividade para a execução. Nesta linha, modelos analíticos baseados na divisão das tarefas entre servidores de dados (*Fork/Join* [Nelson & Tantawi, 1988] [Badue et al., 2010]) e modelos que capturam as precedências entre as tarefas [Mak & Lundstrom, 1990][Liang & Tripathi, 2000] costumam capturar bem estes fatores. Portanto, este trabalho tem como hipótese que tais modelos podem ser suficientemente rápidos e precisos para compor o processo de reconfiguração automática de sistemas paralelos Spark. Para suportar esta hipótese, este trabalho compara o tempo de resposta de sistemas paralelos reais com previsões baseadas em execuções prévias para diferentes quantidades de processadores. Ainda, levanta-se que tais modelos podem apresentar precisão próxima ao DagSim [Ardagna et al., 2018], atual estado da arte para previsões por simulação, que são naturalmente mais representativas e complexas.

Em resumo, esta dissertação explora ambos modelos analíticos e modelos de simulação. Os dois modelos analíticos para aplicações paralelas de cunho real escolhidos são: um modelo baseado em uma representação Fork/Join [Nelson & Tantawi, 1988] [Badue et al., 2010] e um modelo baseado na análise do valor médio (MVA), teoria de filas e na precedência de estágios [Mak & Lundstrom, 1990]. A escolha dos modelos não se deu somente pela rapidez em obter uma previsão ou pela precisão demonstrada nos cenários aos quais foram propostos, mas também pela compatibilidade de ambos com a arquitetura e modelo de execução paralela do Apache Spark, plataforma para aplicações massivas em dados, alvo deste trabalho. Para contraste, o *baseline* escolhido é o modelo por simulação DagSim, proposto por [Ardagna et al., 2018], atual estado da arte. Diversos experimentos foram executados no Spark em três diferentes nuvens. Os logs destas execuções prévias foram utilizados para parametrizar os modelos de previsão de desempenho e comparar a precisão das previsões para sistemas paralelos de diferentes quantidades de processadores. Por se tratarem de recursos de máquinas virtuais em nuvem, por coerência e simplificação, a partir deste ponto esta dissertação passará a chamar os processadores (CPUs) de processadores virtuais (vCPUs).

1.1 Objetivos

Este trabalho tem como principal objetivo avaliar a rapidez e a precisão de dois modelos analíticos de previsão de desempenho para sistemas paralelos, na previsão do tempo de resposta de aplicações paralelas e massivas em dados no Spark em nuvem pública. O modelo de Nelson & Tantawi [1988] foi inicialmente proposto como uma aproximação de escalabilidade para a análise de sistemas paralelos *fork/join*, baseados em filas, homogêneos, que consistem de dois ou mais servidores. Já o modelo de Mak & Lundstrom [1990] foi originalmente proposto para sistemas concorrentes onde o paralelismo é definido por uma coleção de tarefas concorrentes com precedência predefinida. Este trabalho visa, portanto, avaliar a precisão dos modelos em um cenário mais instável e heterogêneo, como o das aplicações paralelas massivas em dados em nuvem pública, mantendo a rapidez e com eficácia demonstrada através da comparação com o modelo por simulação DagSim, atual estado da arte para este cenário.

1.2 Contribuições

As principais contribuições desta dissertação são discutidas a seguir. Uma parte delas foi publicada nos trabalhos de Ardagna et al. [2018] e B. M. Pinto et al. [2018]. Os principais resultados mostram que o modelo de precedência de estágios é satisfatório tanto para prever o tempo de resposta para a mesma quantidade de processadores dos logs execuções prévias coletados, quanto para extrapolar de uma quantidade de processadores para outra diferente, com erro, em média, inferior a 20% para a maioria dos cenários.

- **Aplicação de modelos eficientes e precisos em um cenário novo:** este trabalho explora a aplicação de dois modelos analíticos de previsão de desempenho previamente propostos para estimar o tempo de resposta de aplicações Spark em nuvem pública, particularmente considerando cenários de infraestrutura (número de vCPUs disponíveis) diferentes dos utilizados em execuções anteriores. Os resultados da avaliação experimental mostram que o modelo de precedência de estágios produz previsões bastante razoáveis em cenários práticos. Ainda, esta dissertação mostra como os modelos podem ser aplicações em um cenário diferente do proposto, para aplicações paralelas no Apache Spark. As informações dos logs Spark são usadas na modelagem do sistema paralelo através de um sistema de filas, e na modelagem do grafo de precedência de estágios, onde cada par de tarefas possui sua probabilidade de execução concorrente. Além disso, é utilizada uma nova estratégia de aplicação do modelo para previsão em infraestruturas diferentes (quantidade de vCPUs) dos logs de execuções prévias. Vale destacar que uma premissa desta adaptação é a manutenção da complexidade de tempo polinomial do modelo, visando o menor impacto possível no tempo de execução das previsões, e, portanto, possibilitando o uso do modelo para ajustes dinâmicos de infraestrutura em cenários reais. Este trabalho também demonstra a aplicação do modelo *Fork/Join* para a previsão. Apesar de eficiente, os resultados mostram que este modelo não captura detalhes o suficiente do paralelismo no Spark, como as dependências de execução entre os estágios, que causam atrasos ignorados pelo modelo. Como resultado, o modelo *fork/join* apresentou erros que variam de 169,4% a 256,3%.

- **Avaliação diversificada da precisão do modelo:** Na avaliação apresentada neste trabalho, foram propostos cenários diversificados sobre máquinas virtuais com características diferentes. No Microsoft Azure foram utilizados três tipos diferentes de máquina virtuais, tanto de propósito geral quanto intensas em CPU e Memória. Também foram realizados experimentos em uma infraestrutura virtualizada via OpenStack, ampliando a diversidade de ambientes. Além disso, os cenários experimentais cobrem uma vasta gama de aplicações, muitas delas bastante populares. Foram exploradas consultas interativas, através das consultas 26 e 52 do TPC-DS, algoritmos de aprendizado de máquina supervisionados, como o K-Means, Support Vector Machine (SVM), Logistic Regression, e um algoritmo de aprendizado de máquina não-supervisionado, o K-Means. Ainda, foi experimentado um cenário com execuções concorrentes dos mesmos três algoritmos de aprendizado de máquina citados e um caso de *wordcount* na nuvem da *Google Cloud*.

1.3 Organização

O restante desta dissertação está organizada como segue. Trabalhos relacionados são discutidos no próximo capítulo, incluindo detalhes sobre o DagSim, modelo de desempenho de simulação utilizado para fins de comparação com os modelos analíticos experimentados nesta dissertação. No Capítulo 3, é detalhada a fundamentação teórica, com informações sobre ambos os modelos analíticos de previsão de desempenho, o modelo *fork/join* [Nelson & Tantawi, 1988] e o modelo por precedência de estágios [Mak & Lundstrom, 1990], seu funcionamento, suas premissas e as características do Apache Spark. No Capítulo 4, são apresentados os cenários experimentais, a metodologia de avaliação do modelo e a estratégia de aplicação dos modelos para previsão de tempo de resposta de aplicações Spark. O Capítulo 5 apresenta os resultados dos modelos analíticos e do modelo de simulação e faz um comparativo do tempo de resposta e da precisão dos modelos. O Capítulo 6, lista as conclusões desta dissertação considerando a aplicabilidade do modelo de previsão em ambientes reais e propõe trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Modelos de previsão de desempenho podem ser classificados em três categorias distintas: (i) modelos analíticos; (ii) modelos de simulação; e (iii) modelos baseados em aprendizado de máquina. Neste capítulo, são discutidos trabalhos relacionados à previsão de desempenho de sistemas paralelos das três categorias, com foco em traçar um paralelo entre a modelagem analítica proposta nesta dissertação, as diferenças para os modelos de simulação e de aprendizado de máquina, e as características da execução paralela das aplicações Spark. Sendo assim, este capítulo inicia por detalhar os modelos analíticos e suas características. Em seguida, são detalhados os modelos de previsão por simulação, incluindo o DagSim [Ardagna et al., 2018], utilizado para fins de comparação na avaliação dos modelos analíticos aplicados na presente dissertação. Em seguida, trabalhos sobre modelos de aprendizado de máquina são sucintamente descritos. Embora não seja o foco, a reconfiguração dinâmica de sistemas paralelos é também um tópico relacionado a esta dissertação. Portanto, ao final deste capítulo são abordados alguns trabalhos desta linha de pesquisa.

2.1 Definição da terminologia

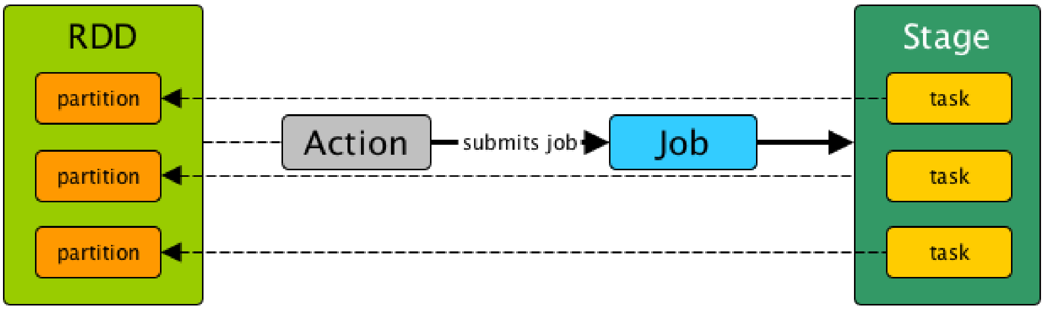
Considerando todos os trabalhos prévios discutidos neste capítulo e também a nomenclatura oficial dos elementos lógicos do modelo de execução paralela do *Apache Spark*, observa-se uma despadronização na terminologia usada pelos diversos autores. A título de clareza, será usada nesta dissertação a mesma terminologia definida por Zaharia et al. [2010] em sua publicação original sobre o Spark. Sendo assim, temos:

- Tarefa: Menor unidade lógica do modelo de execução paralela do Spark. Cada tarefa executa de forma independente em uma partição de dados no sistema paralelo.
- Estágio: Conjunto de tarefas iguais e independentes representando uma transformação uniforme nos dados. Estágios possuem relações de precedência com outros

estágios na execução.

- *Job*: Representação lógica no Spark para uma ação sobre os dados (e.g. salvar no disco, contar, selecionar o máximo, etc). Um *job* é composto por uma ou mais transformações em dados (um ou mais estágios).
- Aplicação paralela ou simplesmente aplicação: Trata-se de um programa escrito pelo usuário representado logicamente no Spark como um conjunto de *jobs*, composto por uma ou mais entradas e uma ou mais saídas. Comumente chamado de *job* por outros autores, inclusive na publicação original do Hadoop.

Figura 2.1: Submissão de um job, estágio e tarefas de uma aplicação Spark



Fonte: Laskowski [2018a]

2.2 Modelos analíticos

Modelos analiticos capturam, por meio de equações e/ou algoritmos, aspectos centrais dos sistemas. O nivel de detalhamento pode variar, mas, no geral, este tipo de modelo tende a ser menos detalhado que modelos de simulação. Consequentemente, modelos analíticos tendem também a ser menos precisos, porém mais rápidos [Menasce et al., 2004]. Um exemplo deste tipo de modelo é a técnica de aproximação para o tempo de resposta médio de sistemas paralelos homogêneos *fork/join* apresentada por Nelson & Tantawi [1988]. Esta técnica aborda uma infraestrutura homogênea formada por k servidores de dados, chamados de *index*, controlados por um computador mestre, chamado de *broker*. Neste modelo a infraestrutura computacional é representada por um sistema de filas *fork/join*, onde um estágio é quebrado em múltiplas tarefas independentes, distribuídas e executadas em cada *index* (*fork*). Os resultados das tarefas são mesclados para obter o resultado final (*join*). A estimativa do tempo de resposta da aplicação usa o tempo médio dos estágios para capturar o tempo da estágio no servidor mais lento. A aproximação proposta pelos autores baseia-se na observação de que existe um limitante superior e um limitante inferior para o tempo de resposta médio da aplicação e que este tempo é composto por uma função $f(k)$ que cresce na mesma taxa que os limitantes. Este modelo desconsidera, por exemplo, informações sobre a dependência entre os estágios presente no modelo de programação do Spark. Porém, ele representa bem a relação mestre-escravo dos nós *worker*, coordenados por um ou mais nós *master*, no Spark. Nesta dissertação, é usada uma modelagem baseada na técnica de aproximação de Nelson & Tantawi [1988], posteriormente apresentada por Badue et al. [2010], que teve como referência as explicações de Menasce et al. [2004]. Aplicações Spark diversas são usadas como carga de trabalho. Mais detalhes sobre o modelo de Nelson & Tantawi [1988] e a simplificação de Menasce et al. [2004] e Badue et al. [2010] serão apresentados no Capítulo 3.

O modelo de precedência de estágios de Mak & Lundstrom [1990] também é um dos modelos experimentados nesta dissertação. Neste, o sistema paralelo é representado através de uma rede de filas que recebem estágios para execução. A aplicação paralela é representada por um grafo de precedências, onde os vértices representam os estágios e as arestas representam as relações de precedência entre eles. Cada par de estágios possui uma probabilidade de execução concorrente específica. Na rede de filas, os recursos computacionais (p. ex. vCPUs) são modelados como centros de serviço e os estágios como clientes, levando em consideração as relações de precedência. Trata-se de um modelo de complexidade polinomial que tem como premissa a execução sequencial dos *jobs* no sistema. Como *jobs* no Spark são por padrão sequenciais, tal premissa se mostrou compatível com o propósito desta dissertação. Além disso, a representação da aplicação como um grafo de precedências muito se assemelha com o modelo de execução baseado em um grafo direcionado acíclico do Spark. Mais detalhes sobre o modelo de precedência de estágios são apresentados no Capítulo 3.

Liang & Tripathi [2000] estendem o modelo proposto por Mak & Lundstrom [1990] de modo a considerar a possibilidade de dois ou mais *jobs* concorrentes para o processo de previsão de desempenho. Neste caso, uma aplicação paralela é modelada como um conjunto de *jobs*, cada qual composto por estágios distintos. De forma similar, o sistema paralelo é também representado como uma rede de filas de centros de serviço (recursos computacionais) enquanto que a aplicação paralela é representada por grafos de precedência entre estágios. Neste caso, podem haver relações de precedência tanto entre estágios de um mesmo job (*intrajobs*) quanto entre estágios de jobs diferentes (*interjobs*). Para obter o tempo de resposta da aplicação, eles consideram dois tipos de atraso: (i) atrasos de sincronização de estágios de acordo com as relações de precedência; (ii) atrasos de enfileiramento de estágios devido à contenção de recursos pela concorrência na utilização. Desta forma, o modelo de Liang & Tripathi [2000] trata de um processo iterativo baseado em uma aproximação da técnica da Análise do Valor Médio (MVA, do inglês *Mean Value*

Analysis) com dois passos principais em cada iteração. No primeiro passo, estima-se de forma aproximada os atrasos de sincronização e as durações da execução concorrente de cada par de estágios através da distribuição do tempo de resposta dos estágios. Com a duração da execução concorrente, o segundo passo é responsável por estimar os atrasos de contenção de recursos. Com ambos os atrasos estimados, ao fim da iteração verifica-se a convergência de ambos os tempos de resposta dos estágios e os tempos de resposta do *jobs* de acordo com um limiar preestabelecido. O modelo possui complexidade polinomial, tanto em tempo quanto em espaço, e assume como premissa que as demandas por recursos podem ser obtidas e fornecidas como entrada para a previsão. Nesta dissertação, usamos as configurações padrão do Spark, ou seja, *jobs* são executados sequencialmente. Sendo assim, o modelo proposto por Mak & Lundstrom [1990] é mais simples e suficiente para prever o tempo de resposta das aplicações aqui abordadas.

Proposto por Nudd et al. [2000], o PACE é um conjunto de ferramentas para avaliação e previsão de desempenho baseado em medição e modelagem analítica de aplicações paralelas. A ferramenta considera diversas informações inerentes ao paralelismo, incluindo o grafo de execução da aplicação, a estratégia de paralelização, os recursos e a arquitetura do sistema paralelo. Para os autores, o modelo tradicional da engenharia de desempenho para previsão deve incluir, além do modelo de execução da aplicação e da rede de filas para centros de serviço, também uma representação da estratégia de paralelismo. Nesta camada adicional são consideradas as características relativas à interação entre os processadores a nível de computação e comunicação. De fato, este modelo captura uma quantidade razoável de informações, porém, isto aumenta a complexidade de uso e a quantidade de informações necessárias, podendo tornar a previsão dependente da implementação. Na presente dissertação, nosso objetivo é fornecer um modelo suficientemente preciso e eficiente em tempo para permitir a elasticidade dos recursos em nuvem de forma independente da aplicação, e, apesar do PACE possibilitar vários níveis de abstração para a modelagem analítica, o modelo de Mak & Lundstrom [1990] é mais simples e contém todas as características aparentemente necessárias para a representação de aplicações paralelas no Spark, que, no geral, segue a estratégia de paralelismo mestre-escravo.

Diversos outros modelos analíticos de previsão de desempenho de sistemas paralelos foram propostos ao longo dos anos. Thomasian & Bay [1986] usam cadeias de Markov para solucionar um modelo baseado em rede de filas e um grafo de precedência de estágios, porém, com complexidade exponencial no número de estágios. Adve & Vernon [2004] propõem um modelo de desempenho para aplicações paralelas de memória compartilhada que leva em consideração o grafo de execução, a política de alocação de estágios e a contenção de recursos. Ipek et al. [2005] usam aprendizado de máquina para treinar modelos com informação proveniente de execuções prévias. Apesar de capturar a complexidade dos sistemas paralelos completamente, a abordagem proposta introduz uma etapa de treinamento que pode levar minutos para completar, depende de uma vasta gama de exemplos de execuções prévias e pode ser influenciada por ruídos presentes no conjunto de dados de treino, comuns em ambientes compartilhados, como é o caso das nuvens públicas.

Mais recentemente, Yang & Sun [2011] propuseram um modelo analítico para avaliar o desempenho de aplicações *map-reduce* no Hadoop, focando em métricas de tempo de resposta e *throughput*. Neste modelo, eles consideram como premissas que (i) os dados são igualmente divididos entre cada máquina do sistema paralelo; (ii) cada tarefa é executada com a mesma capacidade computacional; (iii) a banda de rede entre os nós do sistema é suficientemente grande para suprir a demanda de comunicação sem afetar a velocidade; e (iv) não ocorrerão falhas durante a execução da aplicação. Essas premissas refletem o que geralmente ocorre em configurações Hadoop e Spark. Na modelagem, os autores determinam três tempos diferentes a serem representados, sendo t_1 o tempo de resposta de um estágio *map*, t_2 o tempo de *shuffling* após a finalização do *map* e t_3 o tempo de resposta de um estágio *reduce*. O modelo leva em consideração o volume de dados de entrada e saída dos estágios *map* e *reduce* (incluindo *shuffling*) e a quantidade de tarefas de cada estágio que podem ser executadas simultaneamente. Com estas características, equações matemáticas são parametrizadas e usadas para estimar o tempo de resposta total da aplicação. Este modelo apresenta uma boa precisão, é eficiente em tempo, e captura bem os detalhes de uma aplicação *map-reduce* no Hadoop. Entretanto, as aplicações Spark são muito mais diversificadas e não apresentam apenas estágios *map* e *reduce* em sua composição. Ainda que alguns elementos do trabalho de Yang & Sun [2011] sejam utilizados na presente dissertação, considera-se que um grafo de precedências como o existente no modelo de Mak & Lundstrom [1990] é importante para capturar os atrasos causados pelas dependências entre os estágios, presentes no modelo de execução de uma aplicação Spark.

De forma similar à proposta desta dissertação, Vianna et al. [2013] propõem a utilização de modelos analíticos na avaliação de métricas de desempenho de *jobs map-reduce* no Hadoop, como tempo de resposta médio, *throughput* e utilização de recursos. Os autores baseiam-se em duas estratégias de aproximação. A primeira é baseada no modelo de Liang & Tripathi [2000] que possibilita capturar as relações de precedência entre estágios de um mesmo job e inclui os atrasos de sincronização entre os estágios. A segunda é baseada em um modelo Fork/Join com solução através de um MVA proposto por Varki [1999]. Ambas as alternativas são experimentadas e comparadas com execuções reais e com os resultados de um simulador baseado em uma rede de filas dirigida por eventos, desenvolvido pelos próprios autores, que reproduz as dinâmicas da execução de um *job map-reduce* no Hadoop, levando em conta atrasos por contenção de recursos e sincronização de estágios. Os resultados demonstram, em particular, a precisão do modelo Fork/Join, com cerca de 15% de diferença para os valores das execuções reais no Hadoop. A proposta desta dissertação se assemelha bastante com a proposta de Vianna et al.

[2013]. Como diferenças, esta dissertação usa o modelo de simulação dagSim, estado da arte, especialmente desenvolvido para aplicações Hadoop e Spark, além de usar o modelo fork/join de Nelson & Tantawi [1988].

2.3 Modelos de simulação

Modelos de desempenho de simulação capturam a dinâmica da execução das aplicações e suas estruturas estáticas através de emulação. Geralmente oferecem um nível de detalhes elevado sobre os sistemas representados, o que pode levar a uma boa precisão. Entretanto, costumam ser custosos, tanto em implementação, validação e execução. [Menasce et al., 2004]. Por exemplo, o trabalho de Bagrodia et al. [1999] descreve um simulador para previsão de desempenho de aplicações paralelas, geralmente intensas em comunicação ou em entrada e saída. O COMPASS, como é chamado o simulador, é composto por diversos componentes: (i) simulador do kernel para alocação e execução de *threads*; (ii) simulador de rotinas de comunicação paralelas; (iii) simulador de operações de entrada e saída paralelas; e (iv) simulador de sistema de arquivos paralelo. Estes componentes são acoplados para representar o fluxo de execução. O simulador é aplicável tanto em sistemas paralelos compostos por máquinas de memória distribuída quanto por máquinas de memória compartilhada. Apesar de preciso para os cenários apresentados, o modelo é bastante ineficiente, ambos em tempo e espaço, para simulações detalhadas. Ainda, a característica intensa em computação das aplicações difere das aplicações Spark, que são geralmente intensas em dados.

De forma similar, Zheng et al. [2004] propõem o BigSim, um simulador paralelizável para previsão de desempenho de sistemas paralelos de larga escala. Por se tratar de uma simulação complexa e inviável em máquinas sequenciais, a natureza paralela do simulador permite o *speedup* da execução de simulações de sistemas paralelos compostos por um grande número de processadores. Para simular a classe dos computadores *petaflops*, a abstração consiste de representar cada nó do sistema paralelo através de uma quantidade de *threads* com memória compartilhada. Funções codificadas em mensagens curtas, enviadas de um nó virtual para outro, são extraídas do *buffer* de cada nó virtual destino por *threads* receptoras e, enfim, invocadas para completar a simulação. Trata-se também de paralelismo intenso em computação, que difere do cenário comum de paralelismo de dados presente no Spark.

Uysal et al. [1998] propõem um arcabouço de simulação de baixo acoplamento dividido em (i) emuladores de aplicação, para representar os padrões de comunicação, computação e acesso a dados de aplicações intensas em dados usando grafos; e (ii) uma suite de simuladores, para representar os dispositivos de rede, entrada e saída, disco e processadores. A nível de rede e entrada e saída, os simuladores modelam apenas a concorrência de uso dos *links* e das portas pelos processadores. Com relação ao disco, por sua vez, são capturados o *overhead* do controlador, velocidade de transferência de dados, a velocidade de busca de dados e parâmetros de rotação do disco. Com relação ao processador, são capturados apenas os tempos em que o mesmo está ocupado de acordo com os parâmetros presentes no emulador de aplicação. Embora o modelo permita simulações detalhadas e precisas de sistemas paralelos intensos em dados, as aplicações simuladas no trabalho de Uysal et al. [1998] são de cunho genérico e não capturam um nível alto de detalhes. Apesar disto, é possível observar pelos resultados que a simulação passa a gastar vários segundos à medida que a quantidade de processadores aumenta. Pode-se notar que o aumento no tempo de execução da simulação quando a quantidade de processadores cresce apresenta característica cúbica para todos os cenários avaliados.

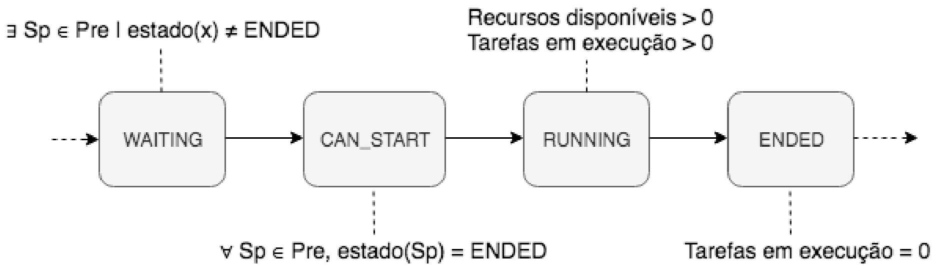
Aproximando-se dos cenários em foco nesta dissertação, previsões por simulação de aplicações Hadoop foram exploradas por outros autores no passado. Song et al. [2013] propõem a avaliação de *jobs map-reduce* no Hadoop através de um analisador. Este analisador é composto por um ambiente de execução virtual baseado no Hadoop que captura os parâmetros de configuração estáticos e dinâmicos da plataforma. Em seguida, um módulo de execução de funções *map-reduce* captura características da aplicação como (i) o tamanho da carga de trabalho de entrada e a quantidade de registros; (ii) a complexidade computacional das funções *map* e *reduce*; e (iii) a taxa entre a entrada e a saída dos estágios de *map* e *reduce*. A previsão, por sua vez, é baseada em um método de regressão local que compara a distância de cada parâmetro da simulação com *logs* de execuções prévias para escolher os mais similares como referência para treinamento do modelo e posterior previsão.

De forma diferente, Ardagna et al. [2016] utiliza duas ferramentas para simulação já existentes para prever o desempenho de *jobs map-reduce* no Hadoop, uma delas baseada em teoria de filas, e a outra baseada em redes estocásticas baseadas em Petri Nets [Reisig et al., 2013]. A primeira abordagem usa o conceito de regiões finitas de capacidade para modelar centros de serviço provenientes de uma fila única de recursos, similar à política *first in first out*, geralmente utilizada no escalonamento de recursos do Hadoop. Desta forma, o tamanho máximo da fila é igual à quantidade de processadores no sistema paralelo. A segunda abordagem captura por completo o comportamento do escalonador de capacidade, possibilitando simular diferentes configurações e cenários, inclusive aqueles de degradação de desempenho causada por falhas. Os autores afirmam que a primeira abordagem pode ser facilmente estendida para o Spark.

Mais recentemente, Ardagna et al. [2018] propõem o dagSim, modelo baseado em simulação de eventos discretos, atual estado da arte para a previsão de desempenho de *jobs map-reduce* e Spark. Este modelo é também utilizado na presente dissertação para fins de comparação com os modelos analíticos. A representação provida pelo modelo é baseada num grafo de precedência de estágios das aplicações paralelas sob avaliação. Nela, considera-se uma lista de estágios (S), a quantidade de nós de processamento (N_{Nodes}), a quantidade de usuários submetendo *jobs* ao sistema (N_{Users}) e o intervalo de tempo entre submissões de *jobs* por um usuário ($\mathcal{Z}$). Cada estágio S_i é composto por um identificador simbólico (id), a quantidade de tarefas do estágio (N_{tasks}), a lista dos estágios antecessores de S_i (Pre), a lista de sucessores diretos de S_i ($Post$) e a distribuição de probabilidade da duração de uma tarefa do estágio S_i, obtida de logs de execuções prévias ($\mathcal{T}$). Com isso, tem-se o modelo do paralelismo com as tuplas $DAG = (S, N_{Nodes}, N_{Users}, \mathcal{Z})$ e $S_i = (id, N_{Tasks}, pre, post, \mathcal{T})$.

O motor de simulação considera quatro estados diferentes para os estágios durante a simulação de um *job*: CAN_START, WAITING, RUNNING, ENDED. Para descrever o processo de simulação, diz-se que estágios que possuem todos seus antecessores já executados ($\forall \ \ S_p \in Pre,\ estado(S_p) = ENDED$) estarão em estado CAN_START e passarão para estado RUNNING a depender da disponibilidade dos recursos. Os estágios com antecessores pendentes de execução, estarão em estado WAITING. Todos os estágios em estado RUNNING controlam a execução de suas tarefas através de um contador de tarefas pendentes. À medida que tarefas são concluídas, o contador é decrementado e o recurso é disponibilizado para uma nova tarefa. Quando um contador chega a zero, o estágio correspondente muda para o estado ENDED e os estágios na lista de sucessores ($Post$) passam para o estado CAN_START desde que todas as suas dependênias tenham sido satisfeitas. Quando todos os estágios do *job* chegam ao estado ENDED, obtém-se o tempo previsto de execução. A Figura 2.2 ilustra a transição de estados.

Figura 2.2: Estados de um estágio durante a simulação



O dagSim usa listas duplamente encadeadas como estrutura de dados para tarefas pertencentes a estágios em estado CAN_START, removendo a necessidade de uma busca completa na lista de estágios para encontrar tarefas aptas a iniciar a execução. Isto torna este modelo mais rápido que os modelos de simulação anteriores. Ainda, o dagSim se mostra preciso, apresentando 6.06% de erro em média para os cenários avaliados pelos autores, com cargas de trabalho diversas e variadas configurações em nuvem. Por se tratar de um modelo eficiente, preciso e estado da arte para previsão de desempenho de aplicações paralelas Hadoop e Spark, este foi adotado para fins de comparação com os modelos analíticos aplicados nesta dissertação.

2.4 Modelos de aprendizado de máquina

Embora modelos de aprendizado de máquina não sejam utilizados nesta dissertação, é importante destáca-los, mesmo que de forma sucinta, como trabalhos relacionados. Como justificativa, destacamos que os modelos analíticos usados na presente dissertação são mais simples de utilizar que os modelos baseados em aprendizado de máquina e, aparentemente, capturam os detalhes principais das aplicações Spark aqui avaliadas.

Motivados pela crescente complexidade dos sistemas paralelos de larga escala, Ipek et al. [2005] e Singh et al. [2007] utilizam redes neurais multicamada para capturar as características dos sistemas paralelos, minimizando os ruídos, treinando modelos a partir de execuções prévias. O modelo apresenta erros bastante baixos, na casa dos 2% a 7% de erro.

Matsunaga & Fortes [2010] experimentam diversas técnicas aprendizado de máquina para a previsão de desempenho de sistemas paralelos. Os autores levam em consideração diversos detalhes de configuração dos sistemas paralelos e das aplicações, como as caracterísitcas CPU, memória, tamanho da entrada, entre outros, na modelagem de atributos. Ainda, os autores abordam cenários distintos em configuração de sistema e em consumo de recursos e propõem o algoritmo *Predicting Query Runtime Regression*, uma generalização da árvore de classificação PQR, que possibilida previsões de granularidade fina. A avaliação dos autores abordam também os algoritmos K-Nearest Neighbor (KNN), a regressão linear, o SVM, entre outros, na previsão de desempenho de aplicações populares de bioinformática.

Considerando os desafios na representação dos sistemas computacionais atuais, Ganapathi [2009] propõe a utilização de análise de correlação canônica através de técnicas de aprendizado de máquina baseadas em estatística (KCCA) para a previsão de desempenho de sistemas paralelos complexos. Detalhes da carga de trabalho, dos parâmetros do sistema e métricas de desempenho prévio são utilizados na modelagem do sistema. Dentre os cenários avaliados pelo autor, está o de *jobs map-reduce*, que apresentou uma precisão tão alta quanto 0,93 (considerando 1,00 como uma previsão perfeita).

2.5 Reconfiguração dinâmica de sistemas

A reconfiguração dinâmica de sistemas foca no ajuste autônomo de um sistema, seja para prover os recursos necessários para a execução ou para promover economia de capacidade. Embora seja campo de estudo relacionado, esta dissertação possui foco nos modelos de previsão de desempenho, sua precisão e rapidez de execução. Ainda assim, é interessante discutir os trabalhos relacionados à reconfiguração dinâmica de sistemas, tendo em vista a interseção com o objetivo, as motivações e os cenários alvo desta dissertação. Nesta seçào, listaremos aqueles ligados à otimização de uso dos recursos em nuvem e à parametrização dinâmica de aplicações massivas em dados.

Para a otimização do uso de recursos em ambientes em nuvem, Truta et al. [2017] baseiam-se em logs prévios de utilização de recursos das aplicações para auxiliar na tomada de decisões para escalonamento eficiente. Com base em um previsor de utilização de capacidade de período curto e um algoritmo de escalonamento de recursos, a proposta dos autores promove decisões sobre o melhor momento de execução dos estágios submetidos ao sistema. Pode-se, portanto, decidir por executar o estágio imediatamente, atrasar a sua execução, ou rejeitar a execução caso não hajam recursos suficientes. Pode-se também acelerar a execução do estágio em andamento caso existam recursos adicionais disponíveis.

Schneider et al. [2009] exploram a elasticidade de recursos para garantir o desempenho de aplicações baseadas em análise de *streams* de dados. Mais especificamente, os autores se concentram na linguagem de programação SPADE empregada no System S da IBM para processamento de *streams* em larga escala. Eles propõem um algoritmo adaptativo para detectar tanto variações na carga de trabalho quanto variações na disponibilidade dos recursos computacionais, permitindo a estabilização do desempenho do sistema em tempo de execução. Na visão deles, é importante considerar que não apenas o fluxo de dados possa variar, mas também a disponibilidade de recursos, tendo em vista que nos cenários atuais, aplicações coexistem em um mesmo ambiente paralelo. Ainda, a solução apresentada por eles apresenta sobrecarga mínima no consumo de recursos para os cenários estudados.

Warneke & Kao [2011] apresentam o Nephele, um *framework* para processamento massivo de dados baseado na alocação dinâmica de recursos possibilitada pelas infraestruturas em nuvem. O Nephele representa suas aplicações através de um grafo direcionado acíclico de estágios com suas relações de precedência. Também, segue uma arquitetura mestre-escravo, onde o mestre coordena a alocação de máquinas virtuais em nuvem e a execução dos estágios. Para uma melhor alocação de recursos e escalonamento de estágios, o *framework* implementa um *profiler* baseado no *Java Management Extensions* (JMX) para coleta de dados de execuções com a finalidade de identificar gargalos de processamento e de I/O. De acordo com os autores, gargalos de processamento sugerem a necessidade de um maior grau de paralelismo, enquanto que gargalos de I/O permitem uma melhor decisão sobre canais de transmissão e armazenamento de dados, permitindo reconsiderar os tipos e quantidades de máquinas virtuais em nuvem a serem usadas.

Herodotou et al. [2011] consideram a complexidade envolvida na otimização de desempenho do Hadoop devido aos mais de 190 parâmetros de configuração da plataforma para propor o *Starfish*, um sistema para otimização autônoma (*self-tuning*) do Hadoop para análise de dados em escala. Os autores partem da premissa de que boas configurações dependem da aplicação, dos dados e das características do sistema paralelo. Sendo assim, exploram a otimização em três níveis: aplicação, fluxo de execução, e carga de trabalho. A nível de aplicação, o *Starfish* gera modelos de desempenho a partir de pequenas frações da execução das aplicações. A nível do fluxo de execução, o *Starfish* busca otimizar o escalonamento de tarefas para o mais próximo dos dados, evitando a degradação de desempenho. Já, a nível de carga de trabalho, o *Starfish* explora o uso de aplicações preliminares para computar dados compartilhados, o armazenamento de dados intermediários para reuso e a reorganização dos dados nos nós otimizando os acessos. Trata-se de uma proposta poderosa que opera como um componente da arquitetura do Hadoop.

Em resumo, esta dissertação considera as características da reconfiguração dinâmica de sistemas paralelos e adota dois modelos de previsão de desempenho de sistemas paralelos previamente propostos. Os modelos fork/join e de precedência de estágios buscam capturar: (i) os atrasos resultantes da sincronização entre as tarefas paralelas; (ii) os atrasos devidos à contenção de recursos no sistema paralelo. Tais modelos são parametrizados a partir de *logs* prévios de aplicações paralelas no Spark em cenários diversos.

Capítulo 3

Fundamentação Teórica

Este capítulo inicia com uma breve discussão sobre o que são dados massivos e uma descrição dos principais componentes do *Hadoop Distributed File System* (HDFS) comumente usado como armazenamento persistente para grandes volumes de dados. Em seguida, as principais características do Apache Spark de Zaharia et al. [2010], ambiente alvo deste estudo, são detalhadas. Posteriormente, o modelo analítico de desempenho baseado em *fork/join* de Nelson & Tantawi [1988], um dos modelos considerados no presente estudo, é apresentado. Por fim, o modelo de precedência de estágios [Mak & Lundstrom, 1990] é brevemente descrito, enfatizando as principais diferenças entre ele e o modelo baseado em fork/join.

3.1 Dados massivos

Existem diversas definições para o termo dados massivos (ou *Big Data*) na literatura. Uma das definições mais citadas é a de Beyer & Laney [2012] com os três "Vs", que afirma tratar de "grande volume, velocidade e variedade de informações que requerem novas formas de processamento e possibilitam uma melhor tomada de decisões, descoberta de padrões e otimização de processos". Assunção et al. [2015] destacam a existência de outros dois "Vs", veracidade e valor, e também acescenta o surgimento de um sexto "V", a viabilidade. Manyika et al. [2011] definem dados massivos como sendo "grandes grupos de dados que podem ser capturados, comunicados, agregados, armazenados e analisados". Para Franks [2014], os dados podem ser de pequena quantidade, média quantidade ou grande quantidade, estruturados, semi-estruturados ou não estruturados, estáticos, dinâmicos ou de tempo-real. Estima-se que mais de 95% dos dados usados atualmente em modelos preditivos são encontrados na forma não estruturada [Gandomi & Haider, 2015].

Uma das plataformas de processamento de dados massivos mais conhecidas é o Apache Hadoop[1], que surgiu em 2006 como a primeira implementação de código aberto

[1]https://hadoop.apache.com

do modelo de programação MapReduce [Dean & Ghemawat, 2004] em *cluster*. Como tal, ela oferece uma solução para armazenar e processar dados em paralelo em múltiplos computadores de *hardware* comum. Um dos componentes mais importantes da plataforma é justamente o sistema de arquivos distribuídos *Hadoop Distributed File System*(HDFS) [Borthakur et al., 2008], que permite o armazenamento persistente de grandes volumes de dados de forma particionada e replicada. Para avaliar os modelos de desempenho considerados nesta dissertação, o HDFS foi usado como sistema de armazenamento de dados em conjunto com o Apache Spark, que é considerado por muitos como o sucessor do Hadoop. O HDFS é brevemente descrito na seção a seguir, enquanto que o Spark é detalhado na Seção 3.4.

3.2 O MapReduce e o Hadoop

O MapReduce foi inicialmente proposto por pesquisadores da Google. Consiste de um modelo de programação paralela projetado para processamento de dados em larga escala [Dean & Ghemawat, 2004] composto por duas etapas principais, *map* e o *reduce*. De maneira geral, o *map* é responsável por transformar os dados, que em seguida são agrupados em listas de elementos que contém uma mesma chave em um processo chamado de *shuffle* que são distribuídas para o *reduce*, responsável por processar uma lista de elementos transformados.

O Apache Hadoop é a implementação mais popular, de código aberto, do *MapReduce* [Assunção et al., 2015]. A implementação utiliza o sistema de arquivos *Hadoop Distributed File System* (HDFS) para escrita e leitura de dados, que foi projetado para um funcionamento conjunto eficiente. O Hadoop é uma plataforma horizontalmente escalável e tolerante a falhas [White, 2015], bastante utilizada para processamento massivo de dados.

3.3 Hadoop Distributed File System

O HDFS é um sistema de arquivos distribuído inicialmente construído como infraestrutura para o *Apache Nutch* [Borthakur et al., 2008], posteriormente usado no Apache Hadoop [Shvachko et al., 2010] para particionamento e replicação de dados em múltiplos nós de um *cluster* [Assunção et al., 2015]. Trata-se de um sistema de arquivos distribuídos projetado para rodar em hardware comum [Borthakur et al., 2008]. Na primeira versão do Hadoop, o HDFS esteve acoplado ao MapReduce, sendo posteriormente desacoplado para permitir o uso de outros modelos de programação diretamente na localidade dos dados do sistema de arquivos [Shvachko et al., 2010] [Borthakur et al., 2008].

Como principais características, o HDFS é altamente tolerante a falhas, desenvolvido para ser implantado em *hardware* de baixo custo, fornece alto *throughput* (volume de dados transferidos em uma fração de tempo) para a aplicação e é ideal para grandes volumes de dados [Borthakur et al., 2008]. O sistema de arquivos também fornece uma API eficiente para leitura dos dados, armazenados no sistema de forma distribuída [Shvachko et al., 2010].

O HDFS foi usado nesta dissertação como sistema de arquivos para o armazenamento dos dados de entrada de cada um dos cenários experimentais. O Spark, portanto, cria RDDs a partir dos dados presentes no HDFS. Cada tarefas que opera sobre estes RDDs, portanto, executam sobre uma única partição dos dados armazenados no HDFS. Os cenários experimentais são descritos no Capítulo 4, incluido o volume de dados utilizado para cada aplicação.

3.4 Apache Spark

Proposto por pesquisadores da UC Berkeley [Zaharia et al., 2010], o *Apache Spark* é uma plataforma de código aberto de propósito geral para processamento de dados de forma paralela e distribuída em *clusters* de computadores comuns que consiste no uso de memória distribuída para otimizar o acesso a dados [Laskowski, 2018a]. O Spark apresenta maiores benefícios para aplicações que fazem o reúso dos dados em processos paralelos e iterativos [Kamburugamuve et al., 2013]. Algoritmos iterativos, muitas vezes presentes em processos de aprendizado de máquina e análises interativas, como as consultas exploratórias *ad-hoc* são exemplos de aplicações comumente executadas no Spark. De acordo com Zaharia et al. [2010], o *Spark* pode superar o *Hadoop MapReduce* em tempo de resposta em até 10x para aplicações de aprendizado de máquina iterativas.

3.4.1 Visão geral

A principal característica do *Spark* é a presença de um sistema de armazenamento distribuído de dados em memória, o *Resilient Distributed Dataset* (RDD). Sua natureza distribuída permite o particionamento da coleção de objetos através de computadores comuns, otimizando leituras repetidas durante a execução da aplicação paralela [Zaharia et al., 2010]. Este tipo de armazenamento em memória permite a tolerância a falhas autônoma através da criação de um histórico das transformações utilizadas para construir um RDD, processo chamado de linhagem (ou genealogia).

O Spark é modularizado, permitindo que diversos motores de execução usem a mesma abstração para processamento distribuído em memória: os RDDs. Desta forma, o projeto apresenta soluções para dados estruturados com o Spark SQL, para aprendizado de máquina com a biblioteca MLlib, para grafos com o GraphFrames e para coleções de dados contínuos (*streaming*) com o Spark Streaming. Ainda, o Spark é compatível com diferentes escalonadores de recurso e capacidade, como sua implementação própria (*Spark Standalone*), o Apache Mesos [2], o Hadoop YARN [3] ou Kubernetes [4]. Neste trabalho, são exploradas aplicações baseadas em algoritmos da MLlib e consultas interativas *ad-hoc* utilizando o Spark SQL para aplicações *batch* sobre o HDFS e *Spark Standalone*.

O modelo de programação do *Spark* consiste em dois tipos de operações principais executadas sobre os dados: transformações e ações. O ponto inicial para o *Spark* é a leitura de dados de um armazenamento persistente para uma coleção de objetos que é um RDD. Em seguida, é possível transformar estes dados de diversas formas, podendo, por fim, executar uma ação de agregação, como, por exemplo, a contagem ou sumarização, para posterior armazenamento persistente [Zaharia et al., 2012]. As transformações no Spark não são executadas imediatamente quando as operações são avaliadas. Uma execução acontece comumente quando os dados transformados vão ser de fato agregados através de uma ação, tendo a memória liberada logo após o uso (*lazy transformations*). De fato, *Spark* e os RDDs funcionam de forma similar a um sistema de memória multi-nível, sendo que em alguns momentos um RDD precisa ser armazenado em um sistema de arquivos, como por exemplo o HDFS, para que haja espaço para outros dados. Para esta situação, *Spark* permite que o programador defina quais RDDs devem ir para o disco primeiro, dando uma maior flexibilidade e controle sobre o que tem prioridade para ficar na memória devido à constante reutilização.

Esta abordagem tem inúmeras vantagens em comparação com as memórias compartilhadas distribuídas, especialmente pelo fato de utilizar granularidade grossa e tratar

[2] http://mesos.apache.org/

[3] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

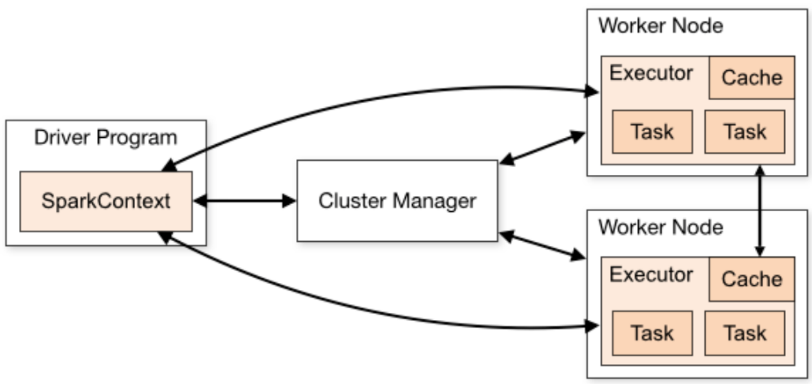
[4] https://kubernetes.io/

de dados imutáveis criados a partir de transformações. Em outras palavras, um RDD não pode ser alterado, mas apenas transformado para a criação de outro RDD. Isto evita escritas diretas na memória e permite tolerar falhas sem a necessidade de replicação de dados, possibilita manter o desempenho caso a memória não seja suficiente (similar às hierarquias de memória e recuperação de RDDs por linhagem), possibilita o processamento na localidade dos dados e garante imutabilidade, este último permitindo tarefas como *backup* com custo consideravelmente baixo [Zaharia et al., 2012].

3.4.2 Funcionamento do *cluster*

Um *cluster* Spark funciona com estratégia de paralelização mestre-escravo, podendo ter um ou mais mestres coordenando a execução dos nós trabalhadores (*workers*). Neste sentido, a nível de software, uma aplicação Spark é um conjunto de processos coordenados por um principal, chamado de *driver program*, acessível a nível de código por um objeto chamado de SparkContext. Através do SparkContext é possível solicitar ao módulo de gestão do *cluster* (Spark Standalone, Mesos, YARN) a alocação de recursos, que instancia executores de acordo com a disponibilidade de cada nó. Cada executor é um processo com múltiplas *threads* que recebe do SparkContext instruções no formato de tarefas para serem executadas em dados de forma paralela. Este fluxo é ilustrado na Figura 3.1.

Figura 3.1: Fluxo de execução de uma aplicação Spark



Fonte: [Spark-Docs, 2018a]

Uma aplicação no Spark é, basicamente, um programa escrito pelo usuário com uma instância de *driver program* (SparkContext) disponível e executores alocados pelo módulo de gestão do *cluster*. Usando Spark Standalone, por padrão, as aplicações são submetidas para execução de acordo com a política FIFO de forma que cada aplicação tentará usar todos os recursos disponíveis no *cluster*.

Como dito anteriormente, o Spark possui dois tipos de operações: transformações e ações. Cada ação (por exemplo: *save, collect, count*) em uma aplicação delimita uma unidade computacional chamada no Spark de *job*. Por padrão, os *jobs* também são submetidos para execução em FIFO [5]. Cada *job*, é dividido em unidades menores, dependentes entre si, chamadas de estágios, que englobam uma sequência de transformações a dados que não requerem sincronização dentro daquele *job*. Cada estágio de um *job* modifica uniformemente uma coleção de dados (RDDs) e, portanto, é dividido em tarefas independentes, uma para cada partição do dado. As tarefas são submetidas de forma paralela para os executores, cada qual com múltiplas threads, de forma que uma única tarefa pode estar ativa por vez em cada núcleo de processamento [Spark-Docs, 2018b] [Laskowski, 2018a]. A lista a seguir sumariza o fluxo de execução de uma aplicação Spark em 5 etapas.

1. O plano de execução é construído de acordo com o código da aplicação. Cada ação (p. ex. *save, collect, count*) e as transformações correspondentes são estruturadas como um *job*;

2. *Jobs* são, por padrão, submetidos sequencialmente ao escalonador da aplicação (*DAG Scheduler*), que organiza cada *job* em um DAG de estágios de acordo com as relações de precedência dos conjuntos de transformações, representando o fluxo de execução;

3. Os estágios são submetidos ao escalonador de tarefas de acordo com a ordem topológica descrita pelo DAG. Cada estágio possui múltiplas tarefas iguais e independentes que executam a mesma porção de código para cada partição de dados correspondente;

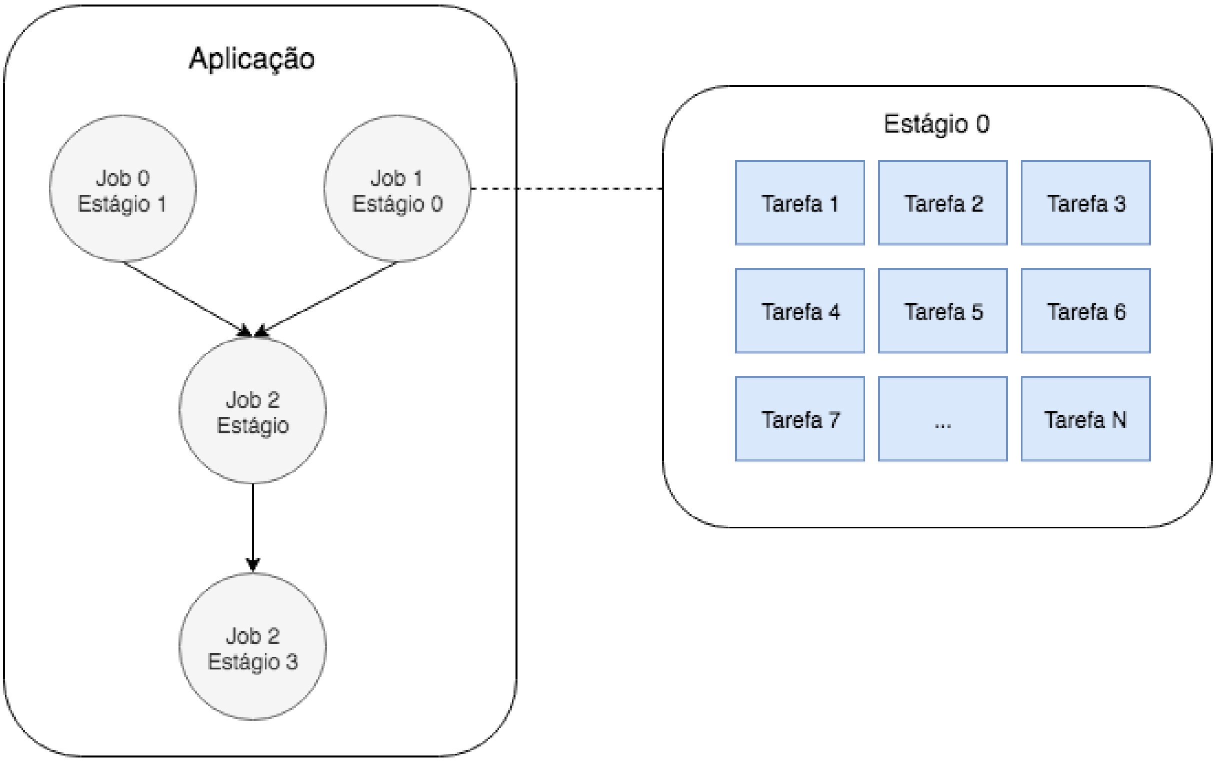
4. O escalonador de tarefas submete as tarefas aos executores de acordo com a disponibilidade. Os executores são responsáveis por servir as tarefas com as suas respectivas partições de dados.

5. Ao final, o resultado de cada tarefa é mesclado e distribuído como entrada para o estágio seguinte do DAG, até que o DAG chegue ao fim. Ao fim de cada estágio, há sempre uma etapa de *shuffling* para construir a saída, o que resulta em um atraso de sincronização. O estágio seguinte só tem acesso à saída do estágio imediatamente anterior após a sincronização.

A Figura 3.2, de própria autoria, ilustra uma aplicação Spark, composta por *jobs*, estágios e tarefas.

[5]A estratégia *Fair Scheduler* do Spark permite uma distribuição de recursos estilo "round robin"para jobs concorrentes. Porém, todos os experimentos apresentados nesta dissertação seguem a estratégia padrão (FIFO).

Figura 3.2: Exemplo de DAG de uma aplicação Spark

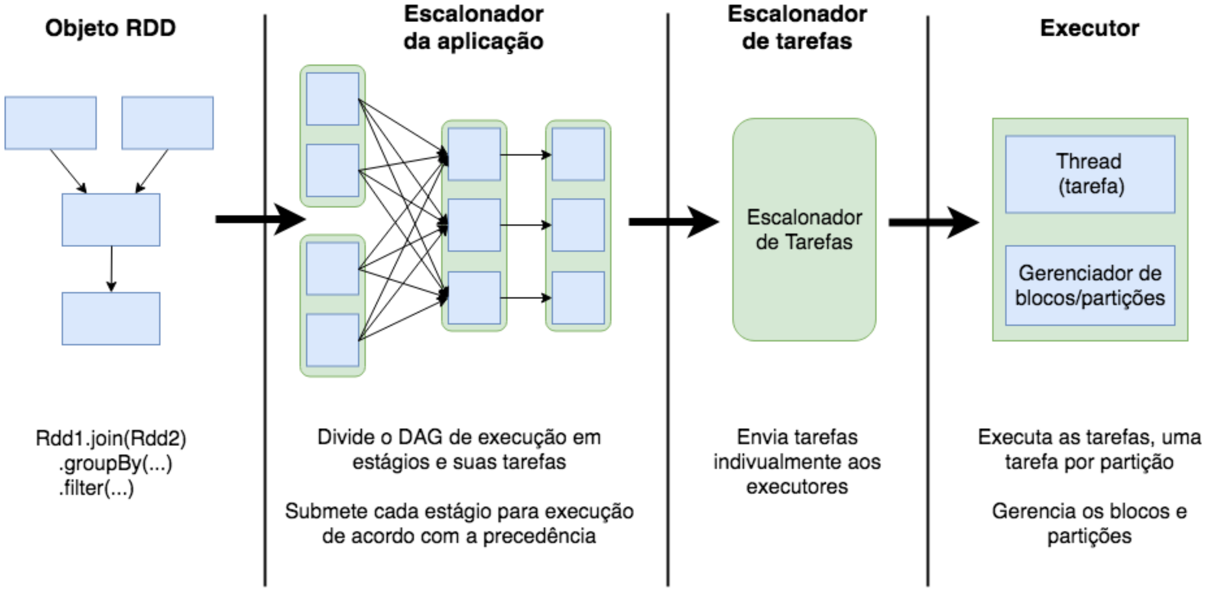


Considerando todas as características do Spark descritas nesta seção, é importante destacar aquelas que servirão de premissa para a modelagem de desempenho apresentada neste trabalho. São elas:

- Apesar do Spark possibilitar a execução de aplicações e de *jobs* em paralelo de acordo com configurações específicas, a política FIFO é padrão e amplamente adotada.
- Dois ou mais estágios podem ser executados em paralelo desde que não haja dependências entre eles.
- Embora um estágio possua múltiplas tarefas, uma para cada partição, todas estas tarefas são iguais e independentes, tendo seus resultados individuais combinados ao final.

A Figura 3.3 ilustra em alto nível o escalonamento de um *job* no Spark, baseando nas explicações de Arggawal [2015].

Figura 3.3: Escalonamento de um *job* no Spark



3.5 Sumarização dos modelos analíticos

O modelo *fork/join* de Nelson & Tantawi [1988] e o modelo de precedência de estágios de Mak & Lundstrom [1990] possuem características específicas para a previsão de desempenho de aplicações paralelas. Antes de apresentar formalmente estes modelos, o modo de aplicação dos modelos em cenários práticos desta dissertação é brevemente listada. A sumarização dos modelos apresentada a seguir inclui: (i) entrada esperada; (ii) saída; e (iii) atrasos capturados. As entradas de ambos os modelos são extraídas de *logs* de execuções prévias de aplicações Spark. No Capítulo 4 é apresentada a estratégia de parametrização dos modelos.

- Modelo Fork/Join:

  Entradas:

  T_1: Tempo de resposta médio de um servidor;

  K: Quantidade de núcleos de processamento no *cluster*

  Saída:

  T_K ou T_{prev}: Tempo de resposta previsto

  Atrasos capturados:

  Atrasos de contenção de recursos.

  Atrasos de sincronização desconsiderando precedência de estágios.

- Modelo de Precedência de Estágios

  Entradas:

  N: Número de estágios

  D: Matriz $D \times K$ de demandas de cada estágio por cada núcleo de processamento.

  R: Matriz de tempo de resposta de cada estágio em cada núcleo.

  P: Matriz $E \times E'$ de fatores de execução concorrente entre pares de estágios.

  K(Opcional): Quantidade de núcleos de processamento (vCPUs) no *cluster* para prever;

  Saídas:

  T_K ou T_{prev}

  Atrasos capturados

  Atrasos de contenção de recursos.

  Atrasos de sincronização de estágios considerando as precedências de estágios.

3.6 Modelo *Fork/Join*

Proposto originalmente por Nelson & Tantawi [1988], este modelo permite a previsão do tempo de resposta de aplicações paralelas com modelo de execução *fork/join*, na qual um *job* é dividido em N tarefas (*fork*) e processadas em paralelo em $K \geq 2$ servidores, para posterior junção dos resultados (*join*). No cenário principal descrito pelos autores, as aplicações têm taxa de chegada igual a λ, determinada por um processo de Poisson. A aplicação é dividida em N tarefas idênticas, independentes, exponencialmente distribuídas e possuem tempo médio de execução igual a $1/\mu$, submetidas a K sistemas de filas homogêneas M/M/1 (uma tarefas para cada servidor) com taxa de utilização igual a $\rho = \lambda/\mu$.

A aproximação sugerida por eles para solucionar este cenário baseia-se na observação de que existe um limitante inferior e um limitante superior para o tempo de resposta médio da aplicação (T_K) que crescem a mesma taxa que uma função $f(K)$, onde K é a quantidade de servidores do sistema paralelo. Os autores sugerem que quando o sistema está com praticamente todos os recursos disponíveis os limitantes seguem o crescimento de uma série harmônica H_K. Considera-se que o tempo de resposta médio da aplicação é dado pelo tempo gasto pelo servidor que mais demorou para executar sua tarefa (apesar de servidores idênticos, processos não determinísticos podem levar os servidores a diferentes estados, influenciando, por exemplo, taxas de *hit* e *miss* da cache).

Sendo assim, o limitante inferior desconsidera os atrasos causados pela contenção de recursos e os atrasos causados pela sincronização entre as tarefas e é dado em função da série harmônica H_K e do tempo médio da tarefa $1/\mu$, este último que sempre será menor que o tempo no servidor mais lento. Para o limitante superior, uma aproximação do método da análise do valor médio (MVA, do inglês Mean Value Analysis), uma técnica recursiva aplicada a redes de filas fechadas, é usada para computar o tamanho das filas e o tempo de espera das tarefas por recursos. Este limitante pode ser definido em função de H_K e do tempo médio de execução da aplicação em um único servidor M/M/1, aqui chamado de T_1, que considera o tempo médio da tarefa e captura ambos os atrasos não considerados pelo limitante inferior. A equação é dada por:

$$H_K \frac{1}{\mu} \leq T_K \leq H_K T_1 \tag{3.1}$$

A partir daí, observa-se que ambos o limitante inferior e superior possuem caráter de crescimento igual a $O(ln\ K)$ à medida em que a quantidade de servidores cresce. Sendo assim, conclui-se que uma boa aproximação para o tempo de resposta do sistema segue o mesmo caráter. A equação do tempo de resposta médio previsto (T_K) desenvolvida pelos autores, é dada por:

$$T_K \simeq \left[\frac{H_K}{H_2} + \alpha(\rho)\left(1 - \frac{H_K}{H_2}\right)\right] T_2 \tag{3.2}$$

Nelson & Tantawi [1988] usam simulação com K = 4, 8, 16, 32 para encontrar uma boa aproximação para o $\alpha(\rho)$ que satisfaça a Equação 3.2. Porém, informações colhidas de execuções prévias podem ser usadas e, geralmente, são mais precisas.

De acordo com Menasce et al. [2004], o número harmônico H_k presente nas equações é um fator de inflação suficiente para capturar os atrasos resultantes da sincronização entre várias tarefas durante a operação de *join*. Os autores afirmam que "a justificativa para esta aproximação surge do fato de que a expectância de uma variável aleatória definida como o máximo de k variáveis aleatórias exponencialmente distribuídas com média T_1 é igual a $H_k T_1$". Esta afirmativa simplifica, portanto, a equação da previsão do tempo de resposta T_k de Nelson & Tantawi [1988] para apenas a equação do limitante superior.

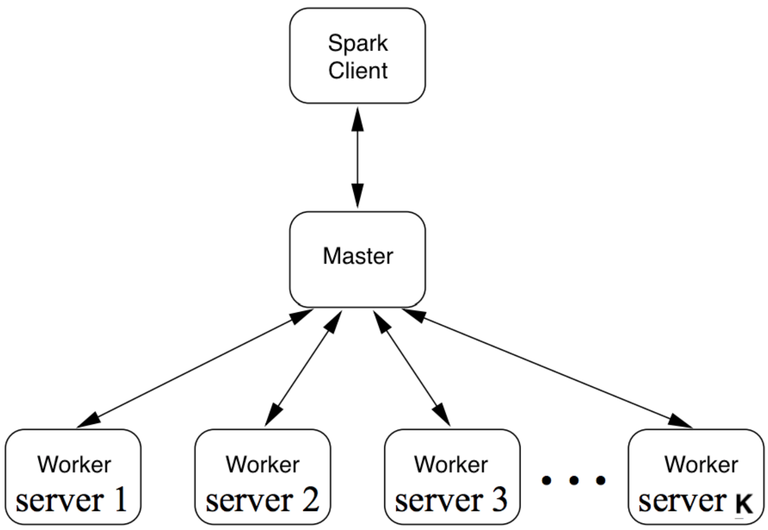
$$T_K \leq H_K T_1, \quad onde \quad H_K = 1 + 1/2 + 1/3 + \cdots + 1/K \tag{3.3}$$

Nesta linha, Badue et al. [2010] utilizaram a simplificação proposta por Menasce et al. [2004] e aplicaram no cenário de motores de buscas verticais, onde o sistema paralelo é composto por um servidor *broker* e servidores *index*. O primeiro é responsável por controlar consultas em cada um dos servidores *index*. Os servidores *index*, por sua vez, armazenam porções de dados e processam as consultas de forma independente e paralela na localidade dos dados. O *broker* é responsável por combinar os resultados individuais de cada *index* para obter a resposta final.

De acordo com os experimentos de Badue et al. [2010], esta adaptação do modelo *fork/join* apresentou resultados precisos para a estimativa do limitante superior no cenário experimental dos autores, demonstrando que trata-se de uma boa estimativa para o tempo de resposta médio da aplicação. Sua simplicidade, quando comparado ao modelo de Nelson & Tantawi [1988], apresenta forte apelo para o uso em cenários onde há a necessidade de rápidas estimativas de tempo de resposta.

A estratégia de execução apresentada no cenário de motores de buscas verticais por Badue et al. [2010] é análoga à estratégia de execução do Spark, baseada em uma execução mestre-escravo. No Spark, os nós *master* coordenam a execução, enquanto que os nós *worker* armazenam e processam as tarefas, de forma independente, também na localidade dos dados. A Figura 3.4 ilustra os elementos do Spark representados através do modelo de desempenho *fork/join*.

Figura 3.4: Modelo fork/join para aplicações Spark



Apesar de os *estágios* no Spark serem submetidos de acordo com as relações de precedência descritas pelo DAG de execução, característica não capturada por este modelo, há o enfileiramento de suas tarefas de acordo com a taxa de utilização dos servidores, podendo ocasionar atrasos por contenção de recursos. Ainda, cada tarefa dos estágios é submetida a um nó *worker* associado a uma partição de dados, priorizando execução na localidade dos dados.

Dadas as semelhanças do Spark com o cenário apresentado por Badue et al. [2010], o presente trabalho baseia-se, portanto, em uma implementação da adaptação proposta por Menasce et al. [2004]. Esta adaptação considera ambos os tempos dos servidores *workers* e os tempos do servidor *master*. A equação geral do limitante superior proposta por eles é dada a seguir, onde R é o tempo de resposta médio da aplicação, R_{broker} representa o tempo de resposta médio de um único servidor *master*, R_{index} representa o tempo de resposta médio do servidor *worker* e H_K representa o k-ésimo número harmônico, onde K é o número de servidores *worker* do sistema paralelo:

$$R \leq H_K R_{index} + R_{broker} \tag{3.4}$$

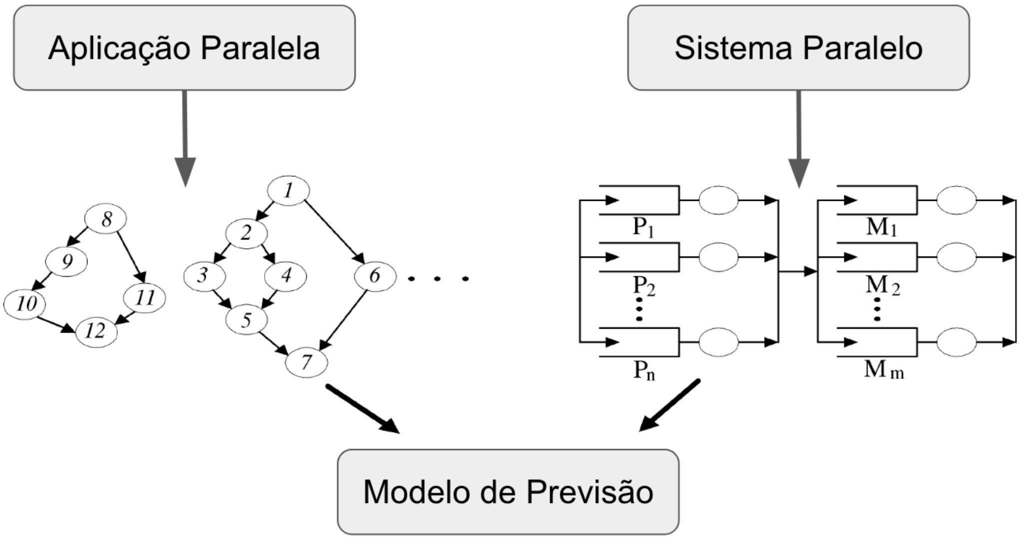
Apesar de não capturar as relações de precedência entre os estágios, um modelo com este grau de simplicidade, é, consequentemente, bastante rápido em tempo de execução. Desde que preciso, o modelo é bastante útil como suporte à elasticidade horizontal de sistemas paralelos, garantindo requisitos de qualidade de serviço das aplicações ao passo em que permite a economia de recursos. Detalhes de como os *logs* de execuções prévias foram usados para parametrizar este modelo são descritos na Seção 4.2.

3.7 Modelo de precedência de estágios

Mak & Lundstrom [1990] propuseram um modelo de desempenho para sistemas paralelos no qual o fluxo de execução é baseado em um conjunto de estágios que executam de forma concorrente e cooperativa. Além do tempo de resposta dos estágios paralelos, este modelo captura os atrasos de enfileiramento por recursos e sincronização de estágios de forma mais detalhada que o modelo *fork/join*. O modelo de previsão tem como entrada duas representações principais: (i) o sistema paralelo homogêneo modelado como uma rede de filas de servidores, obtido a partir das características do sistema paralelo (no caso desta dissertação, cada vCPU é um centro de serviço compondo a fila de servidores); e (ii) a aplicação paralela modelada como um grafo direcionado acíclico (DAG) de estágios (vértices) e suas relações de precedência de execução (arestas). A Figura 3.5 ilustra a aplicação do modelo.

Para solucionar o sistema de filas considerando as precedências, os autores propõem uma modificação no método da análise do valor médio (MVA) que inclui um fator de inflação relativo à etapa de sincronização entre os estágios, levando em consideração as

Figura 3.5: Modelagem do sistema paralelo e aplicação paralela



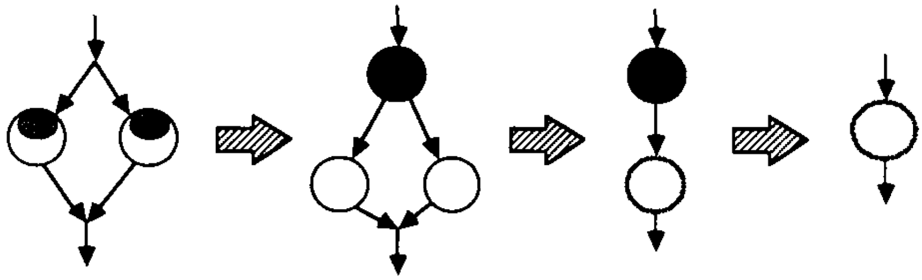
probabilidades de execução concorrente entre eles. A aproximação utilizada pelos autores permite, portanto, a solução de sistemas de filas fechados de múltiplas classes, como é o caso das aplicações com fluxo de execução baseado em um DAG.

As probabilidades α de execução concorrente são necessárias durante a redução do grafo de precedência a fim de estimar os atrasos de sincronização de estágios. Estes valores podem ser obtidos ao comparar o tempo de início e fim de permanência de cada par de estágios nos servidores. No estado inicial, o sistema está livre de contenção de recursos, e portanto, é possível usar a demanda d de cada estágio t em cada servidor k para estimar os tempos de início e fim dos estágios e derivar as probabilidades de execução concorrente. Quando há relação de precedência entre um par de estágios i e j, não há paralelismo ($\alpha_{ij} = 0$). Quando não há relação de precedência entre um par de estágios, o paralelismo pode ser parcial ($0 < \alpha_{ij} < 1$) quando os estágios competem por um mesmo recurso ou total ($\alpha_{ij} = 1$) quando os estágios podem ser executados simultaneamente sem restrições.

Considerando que utilizar informações provenientes de *logs* prévios é geralmente mais preciso [Menasce et al., 2004], obtém-se diversos valores necessários para a execução do modelo a partir de *logs* ao invés de usar as equações originais. São eles: (i) as probabilidades de execução paralela p_{ij}, onde i e j são estágios; (ii) a duração da execução paralela d_{ij}; (iii) a demanda por serviço D_{ik} do estágio i no servidor k. Outros fatores são derivados a partir dos valores obtidos dos logs e das equações propostas por Mak & Lundstrom [1990]. São eles: (iv) o tamanho da fila A_{ik} no momento em que o estágio i entra na fila do servidor k; e (v) o tempo de permanência médio R_{ik} do estágio i no servidor k, que considera os atrasos de enfileiramento; e (vi) o tempo de permanência médio R_i do estágio i no sistema paralelo.

Com os tempos de permanência de cada estágio estimados, é necessário reduzir o grafo de precedências para obter os tempos de resposta previstos para a aplicação paralela. Ao reduzir estágios em série, deve-se somar os tempos. No caso dos estágios em paralelo, considera-se que eles são formados por duas partes, uma dependente e que não pode ser executada em paralelo devido à contenção de recursos mútua, e outra independente, que pode ser executada em paralelo. A Figura 3.6 demonstra as etapas da redução de dois estágios paralelos, representando a parte dependente em preto e a parte não dependente em branco [Mak & Lundstrom, 1990].

Figura 3.6: Redução de tarefas paralelas. Inicialmente, reduz as etapas dependentes, em preto, somando os tempos de resposta de cada uma delas (similar a uma execução sequencial). Em seguida, reduz as etapas não dependentes, em branco, utilizando o maior tempo entre elas como o tempo de resposta da etapa não dependente (execução paralela).



No processo de redução de dois estágios paralelos i e j, considera-se que o tempo de permanência do estágio i é afetado pela contenção gerada pelo estágio concorrente j. Logo, o tempo de permanência de i é composto pela soma de um fator dependente β_{ij} e de um fator independente α_{ij}, ou seja, $R_i = \beta_{ij} + \alpha_{ij}$.

Para aplicar o modelo, os tempos de permanência médios são inicializados para uma estimativa definida pelo usuário. Logo, para estimar o tempo de permanência dos estágios em um sistema paralelo de múltiplos servidores, como é o caso das aplicações tratadas nesta dissertação, Mak & Lundstrom [1990] sugerem o seguinte passo a passo:

1. O fator de concorrência, θ_{ij}, é obtido a partir da probabilidade de concorrência p_{ij}, do intervalo de duração da concorrência d_{ij} e do tempo de permanência R_i do estágio i computado na iteração anterior.
   para todo estágio i e todo estágio j
   $\theta_{ij} = \frac{p_{ij}\ d_{ij}}{R_i}$

2. O tempo médio da fila, Q_{jk}, é obtido a partir dos tempos de permanência do estágio computado na iteração anterior.
   para todo estágio i e todo centro de serviço k
   $Q_{ik} = \frac{R_{ik}}{R_i}$

3. O tempo de permanência do estágio j no centro k quando o estágio i é removido do sistema, R_{jki}, é obtido a partir dos fatores de probabilidade e dos tempos médio da fila computados nos passos (1) e (2), e do tamanho da fila no momento de chegada de i computado na iteração anterior. Considerando m como a quantidade total de servidores, caso a quantidade total de estágios for menor que a quantidade de servidores, nenhuma espera é necessária. Se a quantidade de estágios for maior que m, m se torna um fator de multiplicação para a taxa de execução de estágios considerando um servidor.
   para todo estágio i, todo estágio j e todo centro de serviço k
   $R_{jki} = \frac{D_{jk}(1+\hat{A}_{jk}-\theta_{ji}Q_{ik})}{min(1+\hat{A}_{ik}-\theta_{ji}Q_{ik},\ m)}$

4. Novas estimativas para os tamanhos da fila no momento da chegada de cada estágio, $\hat{A}_{ik}$, são computados a partir R_{ijk}.
   para todo estágio i e todo estágio j
   $R_{ji} = \sum_{k=1}^{K} R_{jki}$
   para todo estágio i e todo centro de serviço k
   $\hat{A}_{ik} = \sum_{j=1, j\neq i}^{N} \theta_{ij} \frac{R_{jki}}{R_{ji}}$

5. O tempo de permanência R_{ik} de cada estágio i em cada centro de serviço k é obtido a partir das demandas e do tamanho da fila no momento da chegada de i.
   para todo estágio i e todo centro de serviço k
   $R_{ik} = \frac{D_{ik}(1+\hat{A}_{ik})}{min(1+\hat{A}_{ik},\ m)}$

6. O tempo de permanência de cada estágio, R_i, é obtido a partir do tempo de permanência de cada estágio nos centros de serviço.
   para todo estágio i
   $R_i = \sum_{k=1}^{K} R_{ik}$

Nesta dissertação, utiliza-se informações mais precisas presentes nos *logs* de execuções prévias de aplicações Spark para parametrizar o modelo de desempenho. Neste caso, considera-se o tempo médio de execução dos estágios como sendo as demandas D_{jk} dos estágios pelos servidores. A partir dos tempos de início e fim dos estágios, é possível também computar as probabilidades médias de execução paralela entre os pares de estágio. Desta forma, o processo iterativo é iniciado com valores medidos de execuções anteriores. Isto simplifica bastante a aplicação do modelo, bastando seguir o passo a passo descrito anteriormente para obter as previsões. Detalhes de como os *logs* do Spark foram utilizados para parametrizar este modelo são descritos em detalhes na Seção 4.2.

3.7.1 Premissas de uso do modelo

Os autores definem sete premissas para uso do modelo de precedência de estágios. São elas:

1. Os recursos do sistema paralelo podem ser modelados de forma independente, com cada recurso (p. ex. memória, cpu, disco) representando um centro de serviço, ou unificada, com um servidor completo representando um centro de serviço. No caso desta dissertação, consideramos um servidor completo como um centro de serviço.

2. O modelo possibilita o uso de três políticas de escalonamento diferentes para estágios concorrentes: (i) primeiro a chegar, primeiro a servir (FCFS); (ii) último a chegar, primeiro a servir - interromper e retomar (FCFS); e (iii) compartilhamento dos processadores (PS). Este trabalho considera aplicações com compartilhamento dos processadores para estágios concorrentes, enquanto que as tarefas executadas em cada partição de dados seguem FCFS.

3. A distribuição do tempo de serviço de um centro (recurso) que segue política FCFS é exponencial e é o mesmo para todas os estágios.

4. As relações de precedência entre os estágios da aplicação paralela podem ser representadas por um DAG. Este é o caso das aplicações Spark, como explicado na Seção 3.4.

5. Estágios concorrentes são independentes entre si. Como dito anteriormente, a execução da aplicação no Spark representada por um DAG segue uma ordem topológica. Estágios só poderão executar caso todos os seus precedentes tenham concluído. Isto é fácil de visualizar, tendo em vista que as saídas dos estágios anteriores (RDD transformado) servem de entrada para o estágio em execução.

6. O tempo de permanência de um estágio em um centro de serviço (servidor) é modelado como uma variável aleatória com distribuição exponencial.

7. O estágio irá circular na rede de filas inúmeras vezes antes de sair, comportamento geralmente associado a estágios de granularidade grossa. Um estágio no Spark geralmente engloba múltiplas instruções delimitadas por operações de *shuffling*, e, portanto, geralmente não trata-se apenas de um estágio de granularidade fina (p. ex. adição, subtracão). Casos de estágios de granularidade fina podem violar esta premissa afetando a precisão do modelo.

3.8 Aplicação dos modelos no contexto do Spark

Tanto o modelo *fork/join* de Nelson & Tantawi [1988] adaptado por Menasce et al. [2004], quanto o modelo de precedência de estágios de Mak & Lundstrom [1990] possuem representações compatíveis com a estratégia de execução paralela do Spark. A adaptação do modelo *fork/join* é aparentemente adequada para capturar a característica mestre-escravo do Spark, além de ser bastante simples e rápidos. Já o modelo de precedência de estágios vai além e captura também as relações de precedência entre os estágios da aplicação.

No contexto do Spark, usa-se *logs* de execuções prévias para parametrizar ambos os modelos. No caso do *fork/join*, o modelo é parametrizado através do enfileiramento das tarefas dos estágios, sem considerar as relações de precedência. No caso do modelo de precedência de estágios, as probabilidades de execução concorrente dos estágios, a duração da execução concorrente e as demandas dos estágios pelos servidores são extraídas diretamente dos logs. A metodologia de parametrização de ambos os modelos é explicada em detalhes no Capítulo 4 a seguir.

Capítulo 4

Metodologia de Avaliação

Neste capítulo, inicialmente serão apresentadas as métricas utilizadas na análise dos resultados. Em seguida, são definidas duas tarefas de previsão para a avaliação dos modelos de previsão de desempenho, chamadas de validação e extrapolação. Também são apresentados detalhes da modelagem de desempenho utilizando logs de execuções prévias do Spark e os modelos descritos no Capítulo 3. Em seguida, os cenários experimentais são apresentados destacando as principais características dos sistemas paralelos e das aplicações paralelas avaliadas.

4.1 Métricas de avaliação

A título de clareza, define-se duas principais métricas consideradas para avaliar as previsões de desempenho obtidas com os modelos *Fork/Join* e Precedência de Estágios: (i) precisão da previsão, representada nos resultados através do erro relativo; e (ii) o tempo médio gasto na solução do modelo. Estas duas métricas são apresentadas juntamente com os resultados de todos os cenários experimentais. Os cenários experimentais considerados neste trabalho são apresentados na Seção 4.4.

A precisão da previsão é estimada a partir do erro relativo calculado considerando os tempos de resposta previstos ($T_{PREVISTO}$) e os respectivos tempos de resposta reais (T_{REAL}) obtidos de *logs* de execuções prévias. Para isto, foi feita uma extensa coleta de dados através da execução de aplicações Spark em clusters de três diferentes ambientes de nuvem. O tempo de resposta real é dado pela média do tempo de resposta das repetições de execução para cada cenário, os quais são comparados com o tempo de resposta previsto pelos modelos, que são determinísticos. A previsão através dos modelos analíticos apresentados neste trabalho é um processo determinístico, e, portanto, para entradas iguais apresenta as mesmas saídas sempre. A análise da precisão permite determinar até que ponto os modelos capturam bem as características de desempenho das aplicações Spark, e, portanto, podem substituir, sem grandes perdas, um modelo de maior complexidade como os de simulação. A título de clareza, a Equação 4.1 detalha como é calculado o erro relativo.

$$\epsilon_R = \frac{T_{REAL} - T_{PREVISTO}}{T_{REAL}} \tag{4.1}$$

O tempo gasto na solução do modelo refere-se simplesmente ao tempo que as implementações dos modelos levaram para, a partir das entradas pré-computadas, resolver as respectivas equações e obter o tempo de resposta previsto para as entrada. Os modelos são executados 10 vezes para cada experimento de cada cenário. As discussões deste capítulo baseiam-se, portanto, no tempo médio gasto para solucionar os modelos. Esta métrica é importante para avaliar a viabilidade dos modelos para a reconfiguração dinâmica de *clusters*.

4.2 Parametrização dos modelos

Para parametrizar ambos os modelos de previsão, baseados no *Fork/Join* e precedência de estágios, são usados *logs* de execuções prévias colhidos a partir de cenários variados de aplicações intensas em dados que serão descritos mais adiante. O parâmetro de configuração *spark.eventLog.enabled* permite o monitoramento da execução de aplicações Spark através da coleta e gravação dos eventos ocorridos em um arquivo de *log*. Com o *History Server* configurado, é possível visualizar detalhes dos *logs*, como por exemplo o DAG de cada *job*, através de uma interface visual acessível pelo navegador. Cada linha de um arquivo de *log* representa um evento (p. ex. estágio submetido ao escalonador, estágio finalizado, tarefa iniciada), em ordem cronológica, estruturado em atributos, em notação JSON[1]. Neste trabalho, eventos específicos são utilizados para parametrizar os modelos de previsão. Os principais eventos utilizados estão dispostos na Tabela 4.1 a seguir.

Para simplificar, nomeia-se cada um dos intervalos dados pelos eventos. Neste trabalho, o intervalo delimitado por *SparkListernerApplicationStart* e *SparkListenerApplicationEnd* é chamado de tempo da aplicação (T_A). O intervalo entre *SparkListernerJobStart* e *SparkListenerJobEnd* é chamado de tempo do *job* (T_J). O intervalo entre *SparkListernerStageSub* e *SparkListenerStageCompleted*, tempo do estágio (T_S). O intervalo entre *SparkListernerTaskSta* e *SparkListenerTaskEnd*, tempo da tarefa (T_T). Além destes intervalos facilmente identificados pela tabela acima, o tempo real da execução T_{real} (ou T_{RE}, tempo da execução) é computado como o intervalo entre os eventos *SparkListenerStageSubmitted* do primeiro

[1] https://www.json.org/

Tabela 4.1: Principais eventos dos log spark

Evento	Descrição	Atributo principal
ApplicationStart	Início da execução de uma aplicação.	*Timestamp*
ApplicationEnd	Fim da execução de uma aplicação.	*Timestamp*
JobStart	Início da execução de um job.	*Submission Time*
JobEnd	Fim da execução de um job.	*Completion Time*
StageSubmitted	Submissão do estágio ao escalonador.	*Submission Time*
StageCompleted	Fim da execução de um estágio.	*Completion Time*
TaskStart	Início da execução de uma tarefa.	*Launch Time*
TaskEnd	Fim da execução de uma tarefa.	*Finish Time*

estágio e *SparkListenerStageCompleted* do último estágio, seguindo a ordem cronológica refletida nos *logs*. Estes tempos são computados a partir das médias dos tempos presentes em repetições de execução de um mesmo experimento (*logs* de repetições da execução de uma aplicação para um configuração específica do sistema paralelo). A estimativa de tempo de resposta obtida através da execução dos modelos é chamada de $T_{previsto}$ (ou T_{prev}).

De modo geral, ambos os modelos representam especificamente o trabalho realizado durante a execução da aplicação e não capturam detalhes sobre alocação de recursos de hardware, inicialização de ambiente de software e orquestração que precede a execução. Neste caso, é necessário evitar a absorção destes atrasos no cálculo do erro da previsão. Portanto, ao invés de usar o tempo da aplicação (T_A) que engloba não apenas a execução paralela, mas também todas as atividades que precedem e sucedem a execução, usa-se o tempo real de execução (T_{RE}). Sendo assim, a previsão leva em conta apenas o tempo gasto de fato na execução dos estágios, suas tarefas e os atrasos de sincronização entre elas. Apesar disto, é possível obter os atrasos médios na submissão e finalização de um *job* a partir da diferença do tempo do *job* e o tempo de resposta do *job* (tempo de submissão do primeiro estágio do *Job* menos tempo de conclusão do último estágio do *Job*). Também, é possível acrescer ao tempo final da previsão a diferença média entre T_A e T_{RE} para capturar os demais atrasos.

Para os experimentos, são incluídos na modelagem do sistema paralelo a quantidade de vCPUs presentes nos nós *worker* como servidores (ou centros de serviço). Os nós de dados do HDFS são configurados para execução em conjunto com os *workers* do Spark com a parametrização de quantidade de réplicas padrão, ou seja, com fator de replicação igual a 3. Sendo assim, os resultados refletem o impacto causado ao variar a quantidade de vCPUs em *clusters* homogêneos. É importante ressaltar que, para configuração de cada cenário, o *cluster* é instalado uma única vez. Sendo assim, a variabilidade na localidade em que as máquinas vituais podem ser alocadas em nuvem não é avaliada nesta dissertação.

É totalmente possível expandir a modelagem para considerar, além dos vCPUs, outros tipos de recursos como memória e disco, por exemplo, embora a abordagem geral para melhorar o desempenho em sistemas paralelos consista na inclusão de novos nós ao sistema e, consequentemente, mais vCPUs. As especificidades na parametrização de cada modelo são descritas nas subseções a seguir. As variadas configurações de *cluster* são apresentadas na Seção 4.4.

4.2.1 Fork/Join

Para aplicar o modelo Fork/Join baseado na abordagem previamente apresentada por Badue et al. [2010], utiliza-se uma estimativa para o tempo de resposta de cada estágio no vCPU mais lento como entrada para o modelo. Para obter estas estimativas, utiliza-se o tempo médio das tarefas para cada estágio e a quantidade de vCPUs disponíveis para construir sistemas de filas independentes de tarefas uniformemente distribuídas, de forma que o tamanho da fila mais longa multiplicado pelo tempo médio da tarefa determina o tempo de resposta desejado. Em outras palavras, as tarefas de são distribuídas igualmente entre os vCPUs disponíveis. O tamanho da maior fila multiplicado pelo tempo médio das tarefas representa o tempo de resposta do vCPU mais lento. Como este modelo não representa as relações de precedência entre os estágios, soma-se os tempos obtidos para cada estágio, um a um, para obter o tempo de resposta previsto.

Apesar de não capturar o paralelismo entre estágios possível nas aplicações Spark, é provável que este modelo apresente boa precisão de previsões. O paralelismo entre estágios causa competição por recursos entre tarefas de estágios distintos, o que pode causar o tempo do estágio a ser consideravelmente maior que o somatório dos tempos das tarefas do estágio, conforme descrição dos eventos interceptados no escalonador de Laskowski [2018b]. Sendo assim, apesar de não capturar as relações de precedência, utilizar apenas os tempos das tarefas evita que os atrasos causados pela concorrência por recursos absorvidos nos tempos dos estágios sejam considerados.

Neste modelo os tempos do servidor *broker* são considerados como nulos, ou seja, considera-se que os atrasos causados pelos nós *mestre* do Spark já são capturados pelos eventos presentes nos *logs*. Sendo assim, o parâmetro referente ao tempo do *broker* é definido como zero, enquanto que o tempo do *index* é calculado a partir dos *logs* de execuções prévias, conforme descrito acima. Em um cenário real de uso do modelo, é esperado que as entradas para o modelo sejam preconstruídas a partir de *logs* de cenários diversos, de forma que o previsor apenas precise buscar a entrada que mais se assemelhe ao cenário a prever em um banco de dados auxiliar.

4.2.2 Precedência de estágios

Para aplicar o modelo de precedência de estágios, primeiramente considera-se o sistema paralelo como um sistema de filas de estágios para cada um dos k vCPUs. A aplicação paralela é dada através do grafo de precedência entre os estágios da aplicação Spark. Desta forma, são contruídas três matrizes de entrada para o modelo, sendo: (i) uma matriz $i \times k$ de demanda de estágios por centros de serviço; (ii) uma matriz $i \times k$ de tempo de residência dos estágios nos centros de serviço; e (iii) uma matriz $i \times j$ de probabilidades de execução concorrente entre pares de estágios.

As demandas dos estágios são dadas pelo tempo em milissegundos de cada estágio obtido a partir dos *logs* de execução prévia. Como as tarefas dos estágios são executadas em múltiplos servidores paralelamente, define-se que a demanda do estágio é a mesma para cada vCPU (centro de serviço no sistema de filas). Esta dissertação não aborda as variações que os dados podem causar na execução das aplicações. Os tempos de residência dos estágios é cumulativa de acordo com a posição de cada estágio no sistema de filas. Já, as probabilidades de execução concorrente p_{ij} entre os estágios são valores entre 0 e 1 dados pela fração do intervalo concorrente pelo intervalo total da execução do par de estágios. Em outras palavras, estes valores indicam a porcentagem do tempo total do par de tarefas em que ambas executam em paralelo.

Os valores de demanda por serviço, tempo de residência dos estágios no sistema de filas e as probabilidades de execução concorrente entre pares de estágios são computados a partir de uma quantidade de 5 a 10 repetições para cada experimento. Sendo assim, os valores usados tratam-se de valores médios considerando múltiplas repetições de um mesmo experimento.

4.3 Tarefas de previsão: validação e extrapolação

Após entender como os modelos são parametrizados, é importante distinguir duas tarefas de previsão de desempenho consideradas neste trabalho. A primeira delas é chamada de validação e trata-se da modelagem e previsão de tempo de resposta de determinada aplicação e sistemas paralelos idênticos (ou muito semelhantes em recursos físicos e perfil de paralelismo) aos *logs* de execuções prévias utilizados na parametrização dos modelos. A segunda tarefa é chamada de extrapolação. No caso da extrapolação, a previsão é feita para determinada aplicação em um sistema paralelo diferente em termos de quantidade de recursos (centros de serviço) daquele aos quais os *logs* de execução prévia se referem.

Considerando uma aplicação paralela de perfil α, um carga de trabalho de perfil δ e um sistema paralelo de perfil σ, a validação tratará da previsão do tempo de resposta de α sobre δ em um sistema σ de k vCPUs, com parametrização dos modelos a partir de *logs* de execuções prévias de uma aplicação de perfil α sobre δ em um sistema de perfil σ também de k vCPUs. Esta tarefa é importante para validar se os modelos representam bem a execução paralela no Spark e capturam os atrasos de contenção de recursos e sincronização de estágios satisfatoriamente. Na extrapolação, entretanto, a parametrização dos modelos é feita com base em *logs* de uma aplicação de perfil α sobre δ em um sistema de perfil σ com $n \neq k$ vCPUs. Ou seja, a extrapolação corresponde à previsão do tempo de resposta de sistemas paralelos com maior ou menor quantidade de vCPUs que do sistema referente aos *logs* préviamente colhidos.

No caso da extrapolação, há uma particularidade no cálculo do tempo dos estágios que determinam a demanda D_{ik} por um servidor. Nesta abordagem, a demanda não é dada pela diferença entre o tempo de submissão e o tempo de finalização do estágio, mas sim de acordo com os tempos das tarefas. Sendo assim, deve-se redistribuir as tarefas dos estágios de acordo com os $n \neq k$ vCPUs existentes no sistema paralelo alvo e computar novos tempos para cada estágio. Para isto, considera-se nesta nova configuração de *cluster* que a quantidade de partições de dados dos estágio em RDDs no Spark se mantém a mesma, tendo em vista que a aplicação α e carga de trabalho δ são as mesmas. Consequentemente, a quantidade de tarefas dos estágios não muda, tendo em vista que uma única tarefa é lançada para cada partição de dados referente àquele estágio. Como cada tarefa executa em um único vCPU, os valores de tempo de resposta também são mantidos visto que os recursos de hardware utilizado por cada tarefa não variam. Para redistribuir as tarefas de acordo com a quantidade de vCPUs alvo da previsão, considera-se a política FIFO, sendo que cada tarefa é alocada em fila aos vCPUs de forma alternada. A partir dos tempos de início e fim de cada tarefa do estágio, calcula-se o tempo das tarefas. Dado o somatório de cada fila independente de tarefas, o maior tempo encontrado é considerado como o novo tempo do estágio. Apesar de possuir uma sutil diferença com relação ao esquema padrão de alocação de tarefas do Spark, que tende a priorizar vCPUs próximos das réplicas da partição, este método de alocação se mostrou bastante verossímil, conforme será discutido no Capítulo 5.

Por fim, para o modelo *Fork/Join*, os novos tempos dos estágios são suficientes para a parametrização na extrapolação. No entanto, para o modelo de precedência de estágios é necessário usar os novos tempos dos estágios para ajustar em cascata novos tempos de submissão e finalização para os estágios. Com isto, é possível computar também novas probabilidades de execução concorrente para os pares de estágios. Diferentemente da validação, que usa informações mais precisas com relação ao tempo dos estágios colhidas diretamente dos *logs* prévios, a extrapolação usa os *logs* para estimar novos tempos para os estágios através dos tempos das tarefas, podendo inclusive ignorar alguns atrasos não capturados pelo tempo de início e fim das tarefas.

4.4 Cenários experimentais

Para uma avaliação ampla dos modelos de previsão para aplicações Spark, os experimentos se basearam em classes de aplicações bastante comuns na plataforma. Os cenários experimentais descritos a seguir incluem algoritmos iterativos com implementações presentes na *MLlib*, consultas *ad-hoc* usando o SparkSQL e mesmo um *wordcount* com múltiplos arquivos como fonte de dados. De acordo com uma enquete da Databricks [2016], a MLlib apresentou um crescimento de popularidade de 67% de 2015 para 2016, enquanto que o SparkSQL cresceu 38%, ambos figurando entre as abstrações mais usadas do Spark.

Dentre os algoritmos avaliados estão três algoritmos comuns em aprendizado de máquina: o algoritmo K-Means de agrupamento de dados não supervisionado, o algoritmo Support Vector Machines (SVM) de classificação e o método de regressão logística. Os três algoritmos foram avaliados individualmente. Além disso, foi avaliado um cenário com o SVM, Logistic Regression e Naive Bayes em conjunto em uma mesma aplicação. Todos estes algoritmos são muito comuns em aplicações de dados massivos (p. ex. *big data analytics*). As consultas *ad-hoc* avaliadas são implementações em SparkSQL das consultas Q26 e Q52 descritas no *benchmark* TPC-DS[2]. Já a aplicação baseada em *wordcount* consiste da contagem de palavras de três arquivos distintos armazenados no HDFS em paralelo. Este cenário em específico apresenta um maior grau de paralelismo entre os estágios. Além disso, o *wordcount* costuma ser uma tarefa base para diversos outros algoritmos, inclusive, no processamento de linguagem natura para computar a frequência de palavras.

Para avaliar a precisão dos modelos de previsão em diferentes recursos de hardware e software, foram usadas quatro diferentes versões do Spark, três diferentes ambientes de nuvem e máquinas virtuais distintas para cada configuração de *cluster*. As versões do Spark usadas nos experimentos são: (i) 1.6.2; (2) 2.1.0; (iii) 2.3.1 e (iv) 2.3.2, cada qual instalada em cenários específicos. Os ambientes de nuvem utilizados consistem de máquinas virtuais na Microsoft Azure HDInsight[3], na Google Cloud (GCE)[4] e em ambiente

[2]https://www.tpc.org/tpcds/
[3]https://docs.microsoft.com/en-us/azure/hdinsight/spark/apache-spark-overview
[4]https://cloud.google.com/compute/

OpenStack[5] instalado em servidores no laboratório SPEED do DCC-UFMG. Cada *cluster* Spark é configurado de forma homogênea, ou seja, composto por máquinas virtuais com as mesmas configurações. Foram usadas tanto máquinas virtuais de propósito geral quanto máquinas virtuais otimizadas em CPU e memória.

A Tabela 4.2 lista as principais características dos cinco tipos de máquina virtual usados para os nós *workers* do Spark (para armazenamento distribuído de dados e processamento na localidade dos dados). São elas a máquina virtual otimizada em memória e CPU Azure D12v2, e as de propósito geral Azure A3, Azure D4v2, Google Compute Engine n1-standard-8 e OpenStack m1.large. Para o Microsoft Azure, foi selecionada a configuração recomendada com dois nós mestre por *cluster*, enquanto que nos demais apenas um. Como os modelos são parametrizados levando em consideração apenas a quantidade de vCPUs dos nós *worker*, a diferença na configuração dos nós mestre não afetam o resultado das previsões.

Tabela 4.2: Configuração das máquinas virtuais (VMs)

VM dos nós worker					VM dos nós master		Software	
VM	vCPU	Disco	RAM	Spark Executors	VM	Driver	Ubuntu	Spark
D12v2	4	200 SSD	28GB	2x 2 vCPU e 2GB RAM	D12v2	2GB	14.04	1.6.2
A3	4	250 HDD	7GB	2x 2 vCPU e 2GB RAM	D12v2	2GB	14.04	1.6.2
D4v2	8	400 SSD	28GB	2x 4 vCPU e 10GB RAM	D12v2	8GB	16.04	2.1.0
m1.large	4	80 HDD	8GB	1x 4 vCPU e 6912MB RAM	m1.large	6GB	16.04	2.3.1
n1-std-8	8	250 HDD	30GB	1x 8 vCPU e 28GB RAM	n1-std-4	13GB	16.04	2.3.2

Com base nas máquinas virtuais e nas aplicações previamente selecionadas, foram planejados nove cenários experimentais conforme descrito pela Tabela 4.3. Os cenários 1, 2, 5, 6, 7, 8 e 9 foram executados em condições normais do *cluster*, enquanto que os cenários 3 e 4 foram executados com o *cluster* em contenção de recursos induzida. Nesta situação, uma aplicação externa foi responsável por alocar, de forma padronizada, núcleos e memórias do *cluster* impedindo que o Spark iniciasse a execução com todos os recursos possíveis para cada configuração do sistema paralelo. Todos os quatro primeiros cenários foram planejados por pesquisadores do Politecnico de Milano. Nestes cenários, foi utilizada uma carga de trabalho sintética gerada pelo ferramental oficial do *benchmark* TPC-DS (tpc.org) para testar as consultas Q26 e Q52. Para os cenários 5 a 7, foram feitas duplicações do *dataset* HIGGS (https://archive.ics.uci.edu/ml/datasets/HIGGS) para atingir os tamanhos de 8GB, 48GB e 96GB e experimentar os algoritmos K-Means, *Logistic Regression* e SVM, individualmente. Para o cenário 8, o dataset KDD CUP 2010 em formato compatível com a LibSVM (https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html) foi duplicado para atingir o tamanho desejado para executar uma aplicação de multiclassificador que combina o SVM, o *Logistic Regression* e o *Naive Bayes*. Para o cenário 9, foram usadas duplicações de três datasets de diálogos transcritos (https://github.com/Phylliida/Dialogue-

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

Datasets, BNCCorpus, BNCSplitWordsCorpus e MovieCorpus) para executar uma aplicação de contagem de palavras com três fontes de dados distintas (MultiWordcount).

Tabela 4.3: Descrição dos Cenários

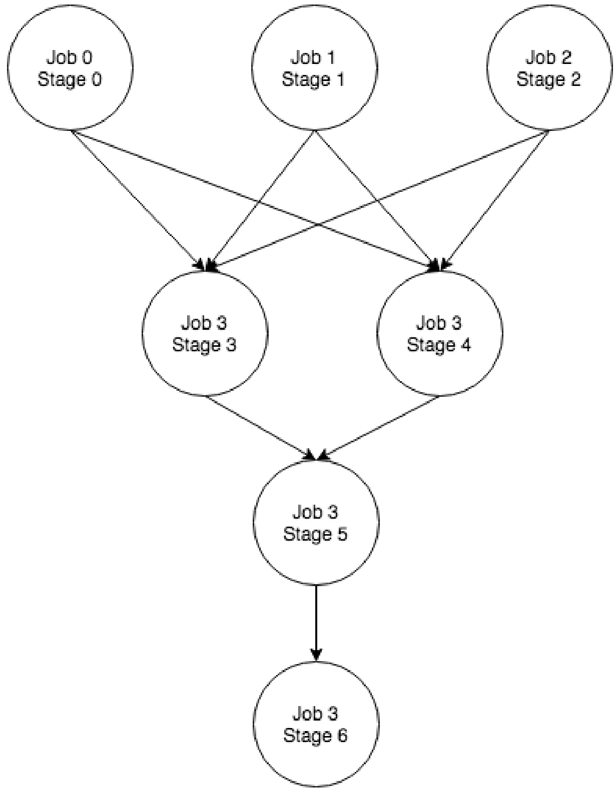
Cenários	Aplicação	VM	Nós	vCPU por nó	Carga de trabalho (GB)
1	TPCDS Q26	D12v2	3 a 13	4	500
2	TPCDS Q52	D12v2	3 a 13	4	500
3	TPCDS Q26	A3	3 a 12	até 4	500
4	TPCDS Q52	A3	3 a 12	até 4	500
5	K-Means	D4v2	3 e 6	8	8,48,96
6	Log. Regression	D4v2	3 e 6	8	8,48,96
7	SVM	D4v2	3 e 6	8	8,48,96
8	MultiWordcount	n1-std-8	6, 9 e 12	8	88
9	Multiclassificador	m1.large	3 a 5	4	4

Cada um dos experimentos foram executados entre 5 e 20 vezes (mais detalhes na Seção 4.6). A validação foi executada para cada experimento e comparada com o tempo de resposta real. Para as extrapolações, foram usados três diferentes contextos de *logs* de referência para a modelagem de desempenho (indicados nas tabelas de resultados pelo rótulo "REF."). As extrapolações foram feitas de *logs* de execuções prévias de referência para as demais quantidades de servidores. Os contextos de referência escolhidos para cada cenário foram: i) *logs* com a menor quantidade de vCPU disponíveis para o cenário (extrapolação A), ii) logs com uma quantidade intermediária de vCPUs disponíveis para o cenário (extrapolação B), iii) logs com a maior quantidade de vCPUs disponíveis para o cenário (extrapolação C). Por exemplo, para o cenário 1, a menor quantidade de vCPUs com *logs* disponíveis é 12 vCPUs (3 nós, 4 vCPUs por nó), enquanto que no caso intermediário o total de vCPUs é 32 (8 nós, 4 vCPUs por nó) e a maior é 52 vCPUs (13 nós, 4 vCPUs por nó). Enfatiza-se que a razão para experimentar com três logs de referência diferentes não é para encontrar qual extrapolação tem melhor precisão, mas sim demonstrar que o modelo produz bons resultados mesmo quando os logs disponíveis são para *clusters* bem diferentes (menores ou maiores em quantidade de vCPUs) que a referência.

4.5 Descrição das aplicações

A consulta 26 do TPC-DS está presente em dois cenários desta dissertação. O primeiro deles é o cenário 1 com *cluster* homogêneos. O segundo é o cenário 3, com contenção de recursos. Esta consulta computa a média de valores de quatro campos relativos a um catálogo de produtos: (i) quantidade; (2) preço de venda; (3) desconto; (4) preço promocional de venda e permite filtrar por ano, gênero, estado civil e formação educacional. Trata-se portanto de uma filtragem e agregação de valores em uma base de dados de grande volume. No Spark, esta consulta é executada como um DAG que possui três estágios paralelos no primeiro nível e dois estágios paralelos no segundo nível do grafo de precedência de estágios, como pode ser visto na Figura 4.1.

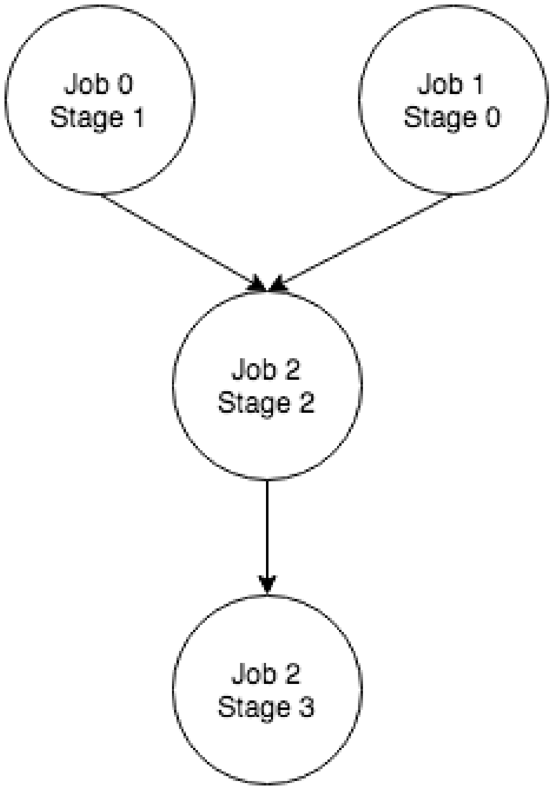
Figura 4.1: DAG da consulta 26



A consulta 52 do TPC-DS está também presente em dois cenários experimentais desta dissertação, o cenário 2 e o cenário 4. Esta consulta lista o preço promocional estendido para todos os itens de uma determinada marca, ano e mês. Trata-se de uma consulta que explora filtro, agrupamento, agregação e ordenação em dados massivos. Diferentemente da consulta 26, esta consulta em específico possui um grafo de precedência de estágios mais simples e menos paralelo, como pode ser visto na Figura 4.2.

O K-Means, presente no cenário 5, é um algoritmo iterativo de clusterização. Como primeiro passo do algoritmo, os centróides dos K clusteres são definidos (geralmente de forma aleatória). Em seguida, cada ponto na coleção de dados é alocada a um cluster levando em consideração a distância entre o ponto e os centróides. Por fim, um novo centróide é definido para o cluster que recebeu o novo item e o algoritmo continua a iteração. Sua implementação em Spark utilizada nesta dissertação apresentou apenas paralelismo de dados (ou seja, paralelismo na execução das tarefas nas partições de dados), não apresentando paralelismo entre estágios. Isto é justificável tendo em vista que as etapas do algoritmo são bastante isoladas e sequencialmente dependentes. De forma similar, os algoritmos de regressão logística presente no cenário 6, o SVM presente no cenário 7 e o classificador múltiplo composto pelos algoritmos SVM, logistic regression e

Figura 4.2: DAG da consulta 52



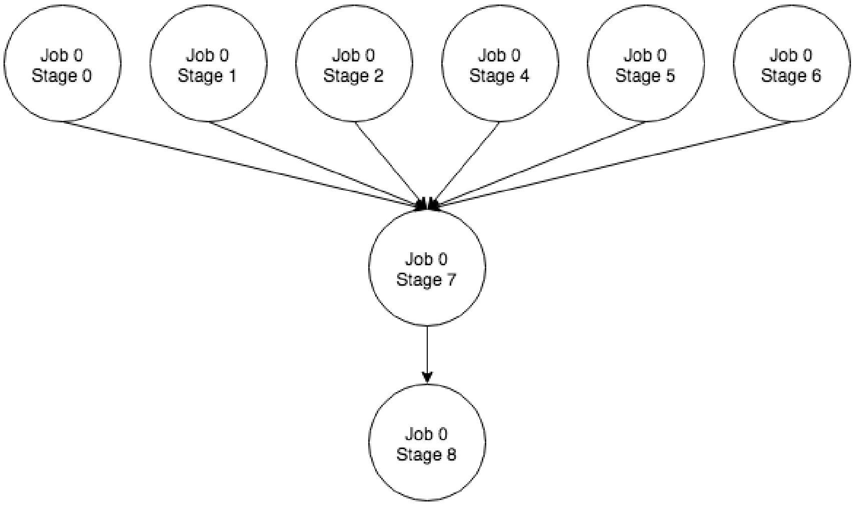
Naive-Bayes também apresentam DAGs sem paralelismo entre estágios. Cabe ressaltar que apesar disto, o classificador múltiplo é o algoritmo que apresentou maior demanda por recursos.

O algoritmo do cenário 9 é um wordcount múltiplo. Esta implementação usa 6 fontes de dados diferentes presentes no HDFS e executa um wordcount em paralelo em cada uma das fontes de dados. O wordcount em si trata-se basicamente de um estágio de map e um estágio de reduce. O grafo de precedência de estágios é apresentado na Figura 4.3. Além do paralelismo, o wordcount múltiplo é o cenário que apresenta maior demanda por disco dentre os nove cenários. Apesar de o volume de dados ser menor que outros cenários, isto é justificável devido à necessidade constante de escritas intermediárias ao longo da execução do algoritmo.

4.6 Cargas de trabalho

Esta seção apresenta uma descrição das cargas de trabalho de cada cenário, com uma análise estatística das amostras de logs de execuções prévias colhidos ao longo dos experimentos. Serão listadas as médias e os intervalos de confiança 95% de confiança.

Figura 4.3: DAG do wordcount múltiplo



4.6.1 Cenários 1 e 2

Os cenários 1 e 2 contam com um total de 20 execuções repetidas para cada experimento realizado. Considerando um total de 11 diferentes *clusters* variando de 3 a 13 nós com 4 núcleos cada (12 a 52 núcleos), tanto para o cenário 1 quanto o cenário 2 foram coletados um total de 220 logs de execuções prévias cada. Assim como nos demais cenários, parte dos dados presentes nos logs são usados como base para a modelagem de desempenho e entrada para os algoritmos experimentados. As médias, os coeficientes de variação (CV) e os intervalos de confiança para 95% de confiança são listados na Tabela 4.4.

Tabela 4.4: Caracterização do tempo de resposta da aplicação de acordo com os logs prévios dos cenários 1 e 2. Tempos em milissegundos.

#	Cenário 1			Cenário 2		
	Média (ms)	CV	IC 95%	Média (ms)	CV	IC 95%
12	660.424,50	0,052	644.006,86; 676.842,14	653.272,15	0,014	648.819,73; 657.724,57
16	553.475,05	0,082	531.791,78; 575.158,32	510.063,50	0,013	506.789,37; 513.337,63
20	461.454,05	0,100	438.351,33; 484.556,77	411.218,10	0,011	408.993,14; 413.443,06
24	380.070,50	0,130	355.589,36; 404.551,64	352.168,30	0,017	349.240,22; 355.096,38
28	350.711,60	0,150	325.173,19; 376.250,01	298.712,65	0,014	296.758,52; 300.666,78
32	305.550,00	0,180	279.639,50; 331.460,50	264.233,80	0,023	261.378,32; 267.089,28
36	243.701,65	0,150	225.651,96; 261.751,34	241.392,55	0,022	238.826,76; 243.958,34
40	224.130,75	0,180	204.973,24; 243.288,26	209.789,60	0,006	209.141,76; 210.437,44
44	193.543,30	0,130	181.517,56; 205.569,04	194.991,25	0,019	193.253,52; 196.728,98
48	185.828,10	0,010	184.961,71; 186.694,49	185.784,25	0,011	184.821,21; 186.747,29
52	169.462,85	0,008	168.842,84; 170.082,86	174.642,55	0,011	173.687,72; 175.597,38

Ambos os cenários 1 e 2 apresentam um coeficiente de variação baixo. O cenário 1 apresenta coeficiente de variação inferior a 0,2 para todas as configurações. Já, o cenário 2 apresenta coeficiente de variação inferior a 0,03 para todas as configurações. A maior estabilidade no tempo de resposta das execuções repetidas para o cenário 2 reflete em uma previsão mais precisa, como será visto no Capítulo 5. Isto é justificável pela consulta

26 apresentar uma maior quantidade de estágios paralelos que a consulta 52, o que traz também um maior atraso por sincronização de tarefas e estágios.

4.6.2 Cenários 3 e 4

Os cenários 3 e 4 contam com uma quantidade variável de execuções repetidas por experimento realizado, de 2 a 38 repetições. A intenção neste cenário foi de avaliar a precisão do modelo tanto para um ambiente de contenção de recursos quanto para casos em que a quantidade de logs prévios para as configurações do sistema paralelo pode ser tanto consideravelmente grandes quanto muito pequenas. É importante destacar que não foram coletados *logs* para a configuração de 28 nós com contenção de recursos para o cenário 3. As médias e os coeficientes de variação (CV) são listados na Tabela 4.5. A coluna com o número de repetições de cada configuração é identificada com o símbolo #.

Tabela 4.5: Caracterização do tempo de resposta da aplicação de acordo com os logs prévios dos cenários 3 e 4. Tempos em milissegundos.

vCPU	Cenário 3				Cenário 4			
	#	Média (ms)	CV	IC 95%	#	Média (ms)	CV	IC 95%
13	15	1.715.509,07	0,094	1.622.703,03; 1.808.315,10	10	1.257.027,93	0,015	1.246.094,21; 1.267.961,64
12	5	1.628.389,40	0,028	1.566.094,32; 1.690.684,48	6	1.054.972,33	0,009	1.044.339,81; 1.065.604,86
14	23	1.377.622,96	0,027	1.361.303,84; 1.393.942,07	18	907.059,44	0,026	894.977,78; 919.141,11
16	2	1.211.137,50	0,004	1.144.881,00; 1.277.394,00	7	816.277,71	0,019	800.484,60; 832.070,83
18	19	1.065.195,37	0,079	1.023.636,45; 1.106.754,29	18	748.714,78	0,037	734.725,78; 762.703,77
20	8	1.032.754,63	0,028	1.007.070,89; 1.058.438,36	9	671,521,56	0,035	652.378,20; 690.664,91
22	22	931.828,18	0,037	916.077,48; 947.578,88	19	598.753,00	0,016	594.019,97; 603.486,03
24	5	846.356,80	0,012	832.631,98; 860.081,62	7	551.734,71	0,009	546.941,56; 556.527,86
26	30	654.022,37	0,122	623.624,09; 684.420,64	9	499.386,33	0,006	497.114,03; 501.658,63
28	0	-	-	-; -	21	465,621.95	0,009	463.581,81; 467.662,10
30	22	590.449,36	0,086	567.533,98; 613.364,75	9	453.085,89	0,037	439.329,58; 466.842,20
32	2	563.997,00	0,003	541.024,18; 586.969,82	15	419.804,27	0,037	410.807,61; 428.800,92
34	38	558.299,66	0,029	552.964,17; 563.635,15	17	390.204,35	0,007	388.824,31; 391.584,40
36	2	507.582,50	0,020	379.675,49; 635.489,51	23	370.389,74	0,009	368.907,37; 371.872,11
38	29	479.687,57	0,022	475.478,48; 483.896,66	13	367,859.38	0,034	359.924,42; 375.794,35
40	6	461.385,00	0,027	447.249,96; 475.520,04	22	346,827.59	0,029	342.234,85; 351.420,33
42	32	422.407,75	0,039	416.434,72; 428.380,78	17	322,515.71	0,009	320.933,36; 324.098,06
44	8	403.478,00	0,041	388.688,05; 418.267,95	23	314,367.30	0,012	312.672,38; 316.062,23
46	20	378.806,20	0,059	368.146,45; 389.465,95	18	307.137,67	0,030	302.411,72; 311.863,62
48	15	364.521,73	0,068	350.342,62; 378.700,85	23	293.739,95	0,017	291.472,84; 296.007,07

O cenário 3 apresentou coeficientes de variação abaixo de 0,13, enquanto que o cenário 4 apresentou coeficientes de variação inferiores a 0,04. Como ilustrado na Seção 4.5 pelos grafos de precedência de estágios, pode-se observar que a consulta 26, presente no cenário 3, explora mais o paralelismo que a consulta 52, presente no cenário 4. Esta característica em particular indica a existência de mais pontos de sincronização de estágios e tarefas para a consulta 26 e, consequentemente, mais pontos causadores de atrasos na execução da aplicação. Isto justifica uma maior estabilidade para a consulta 52 quando comparada à consulta 26.

4.6.3 Cenários 5, 6 e 7

Os cenários 5, 6 e 7 contam com um total de 5 repetições para cada experimento realizado. Ao todo, são 18 configurações de experimentos diferentes, resultado da combinação de 3 fatores: (1) algoritmo; (2) quantidade de núcleos; e (3) tamanho do dataset. Os algoritmos experimentados são K-Means, SVM e Logistic Regression, em configurações de 24 e 48 núcleos, com datasets de 8GB, 48GB e 96GB.

Para o dataset de 8GB, podemos observar que a diferença entre os tempos de execução com 3 workers e com 6 workers é mínima. Isto sugere que a quantidade de dados não foi suficiente para esgotar os recursos dos sistemas paralelos em praticamente nenhum momento da execução. Pode-se dizer que a demanda por recursos foi menor que a quantidade recursos disponíveis em ambas as configurações.

Tabela 4.6: Caracterização dos tempos de resposta da aplicação de acordo com logs prévios para uma carga de trabalho de 8GB. Tempos em milissegundos.

Exec #	Tempo em 24 vCPUs (ms)			Tempo em 48 vCPUs (ms)		
	K-Means	SVM	Logistic Regression	K-Means	SVM	Logistic Regression
Média	98.461	190.752	178.900	90.394	189.741	181.978
CV	3,69%	1,18%	0,74%	1,72%	1,32%	1,20%
IC95%	88.375; 108.547	184.511; 196.992	175.252; 182.548	86.070; 94.718	182.785; 196.698	175.941; 188.014

Para o dataset de 48GB já é possível notar a diferença no tempo de execução de algoritmos iguais em configurações com 3 workers e 6 workers. O tempo de execução é consideravelmente menor para o caso com mais nós. Para o K-Means, o tempo com 6 workers é 42,88% menor, enquanto que para o SVM é 41,72% menor e para o Logistic Regression é 43,94% menor.

Tabela 4.7: Caracterização dos tempos de resposta da aplicação de acordo com logs prévios para uma carga de trabalho de 48GB. Tempos em milissegundos.

Exec #	Tempo em 24 vCPUs (ms)			Tempo em 48 vCPUs (ms)		
	K-Means	SVM	Logistic Regression	K-Means	SVM	Logistic Regression
Média	340.983	639.213	683.157	194.768	372.529	382.952
CV	3,79%	9,09%	1,00%	2,20%	1,57%	1,12%
IC95%	330.623; 351.344	591.075; 687.352	678.201; 688.113	189.241; 200.295	364.264; 380.795	377.282; 388.622

Pode-se destacar a partir da Tabela 4.7 que os intervalos de confiança são estreitos, mesmo para uma confiança de 95%. Isto indica uma certa estabilidade do sistema mesmo utilizando apenas 5 repetições de cada configuração.

Para o dataset de 96GB, destacamos um coeficiente de variação mais alto que o normal para o K-Means com 3 workers, podendo representar uma interferência nas VMs em nuvem do Azure causada por alguma anormalidade no hardware físico. Apesar disto, podemos destacar tempos com redução de 31,06%, 52,41% e 24,85% para o K-Means, SVM e Logistic Regression, respectivamente.

Tabela 4.8: Caracterização dos tempos de resposta da aplicação de acordo com logs prévios para uma carga de trabalho de 96GB. Tempos em milissegundos.

Exec #	Tempo em 24 vCPUs (ms)			Tempo em 48 vCPUs (ms)		
	K-Means	SVM	Logistic Regression	K-Means	SVM	Logistic Regression
Média	862.059	1.366.995	1.431.853	594.310	650.509	1.219.151
CV	69,23%	1,89%	3,72%	8,42%	1,21%	5,66%
IC95%	672.824; 1.051.294	1.356.989; 1.377.000	1.413.395; 1.450.312	573.177; 615.443	644.133; 656.886	1.190.573; 1.247.729

4.6.4 Cenários 8 e 9

Os cenários 8 e 9 contam com um total de 10 repetições para cada experimento realizado. Considerando um total de 3 diferentes clusters cada, tanto o cenário 8 quanto o cenário 9 contam com um total de 30 logs de execuções prévias cada. Vale lembrar também que o ambiente de nuvem para estes dois cenários são diferentes, sendo o cenário 8 configurado em um ambiente OpenStack de nuvem privada do laboratório Speed da UFMG, e o cenário 9 configurado na Google Cloud Platform.

A Tabela 4.9 lista as médias, os coeficientes de variação e os intervalos de confiança de 95% para os cenários 8 e 9. Considerando as amostras de logs prévios para 16 núcleos, destacamos um desvio padrão mais de 3 vezes superior que aqueles encontrados para as amostras de 12 e 20 núcleos. Apesar disto, nenhuma das três configurações apresentaram valores altos para o coeficiente de variação, não havendo sobreposição nos intervalos de confiança.

Tabela 4.9: Caracterização dos logs prévios do cenário 8 e 9

Cenário 8				Cenário 9			
#	Média	CV	IC 95%	#	Média	CV	IC 95%
48	472.678	0,009	469.373; 475.983	12	1.812.070	0.020	1.781.699; 1.842.441
72	315.894	0,009	313.801; 317.988	16	1.466.617	0,095	1.361.975; 1.571.260
96	238.258	0,011	236.201; 240.314	20	1.289.298	0,030	1.260.431; 1.318.164

No geral, foram realizados experimentos diversificados ao longo desta dissertação. Ao todo, foram experimentados 9 cenários com diferentes aplicações e dados para variadas configurações de *cluster*, conforme detalhado neste capítulo. Ainda, o processo de validação e de extrapolação, duas tarefas distintas de previsão aplicadas nesta dissertação foram detalhadas com a finalidade de facilitar o entendimento dos resultados que serão apresentados no Capítulo 5.

A estratégia de parametrização dos modelos usando *logs* de execuções prévias de cada cenário foi explicada para ambos os modelos *fork/join* e precedência de estágios. No capítulo seguinte, serão apresentados os resultados das previsões, seguido de uma discussão, inclusive, comparando ambos os modelos com o DagSim, modelo de simulação atual estado-da-arte para previsão de desempenho para aplicações Hadoop e Spark. Os resultados para o DagSim foram cedidos por pesquisadores parceiros do Politecnico di Milano e publicados por Ardagna et al. [2018].

Capítulo 5

Resultados Experimentais

Este capítulo apresenta e discute os principais resultados obtidos nos experimentos realizados conforme metodologia descrita no Capítulo 4. Os experimentos cobrem cenários bastante diversificados e representativos de aplicações práticas: (i) consultas interativas Q26 e Q52 do TPC-DS, (ii) algoritmos iterativos como o K-Means, SVM e Logistic Regression, (iii) o multiclassificador, e (iv) uma operação mais simples porém recorrente em paralelismo, o *wordcount*. Ainda, ao considerar três diferentes ambientes de nuvem, Microsoft Azure, Google Cloud e uma nuvem privada em OpenStack, este capítulo mostra de forma variada que o modelo de precedência de estágios é uma ferramenta razoável para suportar a reconfiguração dinâmica de sistemas paralelos devido à sua boa precisão e rapidez de execução. Os resultados para os modelos analíticos são detalhados e comparados aos resultados do modelo de simulação DagSim, estado-da-arte para Hadoop e Spark. Os resultados do DagSim foram cedidos por pesquisadores parceiros do Politecnico di Milano e públicados por Ardagna et al. [2018].

5.1 Disposições gerais

Nas próximas seções, serão listados os resultados para os cenários de 1 a 9. Num primeiro momento, serão apresentados os tempos médios de execução dos modelos para cada cenário. Esta informação será utilizada durante a discussão dos resultados e da precisão dos modelos que serão apresentados em seguida. Nas tabelas de resultados, o maior erro observado é destacado em negrito e o menor erro observado com preenchimento cinza.

Os resultados dos modelos são apresentados por cenário na seguinte ordem: (1) modelo de simulação DagSim; (2) modelo *fork/join*; (3) modelo de precedência de estágios. O modelo de simulação DagSim é apresentado primeiro com a finalidade de estabelecer a base comparativa para os modelos analíticos *fork/join* e de precedência de estágios. Ainda, cabe ressaltar que a abordagem do modelo DagSim e do modelo *fork/join* acrescentam ao final das previsões os atrasos de inicialização da aplicação Spark. Desta forma, o tempo de resposta real (T_{real}) compreende o intervalo de tempo entre os eventos *ApplicationStart* e *ApplicationEnd*. Diferentemente, o modelo de precedência de estágios considera os tempos relativos à execução do grafo de estágios da aplicação e compreende o intervalo de tempo entre os eventos *SparkListenerStageSubmitted* do primeiro estágio e *SparkListenerStageCompleted* do último estágio.

É importante notar que o modelo *fork/join* foi executado apenas para os cenários 1 e 2 conforme será explicado na próxima seção. Já, o modelo de simulação DagSim foi executado para os cenários 1 a 7, tendo em vista que estes foram os cenários avaliados e publicados pelos autores do modelo. Os cenários 8 e 9 foram exclusivamente preparados para esta dissertação e para a avaliação do modelo de precedência de estágios com aplicações de alta demanda por recursos e com maior grau de paralelismo.

5.2 Tempo médio de execução dos modelos

A Tabela 5.1 apresenta os tempos de resposta médios (T_e) e os coeficientes de variação para 10 execuções do modelo para cada configuração às quais foram aplicados. Os tempos são apresentados em milissegundos.

Tabela 5.1: Tempo médio de execução dos modelos e coeficientes de variação

Scenario	Fork/Join		Task Precedence		DagSim	
	T_e (ms)	CV	T_e (ms)	CV	T_e (ms)	CV
1	3.13	2.59	5.35	11.96	3,085.29	5.54
2	2.67	2.98	4.59	4.81	763.45	5.97
3	-	-	6.26	3.95	3,264.96	1.43
4	-	-	5.08	2.72	815.47	2.70
5	-	-	9.42	31.97	1,238.11	91.56
6	-	-	28.38	29.20	2,417.94	84.10
7	-	-	59.82	36.64	4,923.00	87.35
8	-	-	139.37	52.12	-	-
9	-	-	123.82	12.53	-	-

Observa-se a partir dos tempos médios de execução que os modelos analíticos são, em alguns casos, até 3 ordens de magnitude mais rápidos que o modelo de simulação DagSim. Com tempos na casa dos milissegundos, é interessante a utilização de ambos modelos na reconfiguração dinâmica desde que os mesmos apresentem precisão satisfatória para tal. A precisão dos modelos em cada cenário aos quais foram aplicados é feita nas seções a seguir.

5.3 Cenários 1 e 2 (VMs D12v2)

5.3.1 Modelo de simulação DagSim

Esta subseção apresenta os resultados cedidos pelos pesquisadores do Politecnico di Milano sobre o modelo de simulação estado-da-arte para previsão de desempenho de aplicações paralelas Hadoop e Spark para os cenários 1 e 2. Estes resultados foram publicados por Ardagna et al. [2018] e servem de comparativo na avaliação dos resultados dos modelos *fork-join* e precedência de estágios para esta seção. Os resultados para este modelo são dispostos na Tabela 5.2.

Tabela 5.2: Validação com o modelo *DagSim* (cluster homogêneo)

#	Cenário 1			Cenário 2		
Núcleos	T_{Real} (s)	$T_{Previsto}$ (s)	Erro	T_{Real} (s)	$T_{Previsto}$ (s)	Erro
12	722,2	682,3	5,5%	719,9	716,0	0,6%
16	582.9	526,5	9,7%	562,7	559,6	0,6%
20	515.9	455,3	11,8%	471,8	468,3	0,8%
24	447.6	394.3	11,9%	417,7	415,3	0,6%
28	415.7	348,4	**16,2%**	364,1	360,7	**0,9**%
32	366.1	312,4	14,7%	324,7	322,3	0,7%
36	306.1	290,3	5,2%	306,8	304.2	**0,9**%
40	287.5	270,3	6,0%	275,2	273,1	0,8%
44	259.7	250,6	3,5%	258,8	257,0	0.7%
48	248.6	249,0	-0,1%	250,0	248,3	0,7%
52	220.2	221,0	-0,4%	226,1	224,2	0,8%

Para o cenário 1, o DagSim apresentou um erro médio de 7,27% com erro absoluto mínimo de 0,10% e erro absoluto máximo de 16,20%. Pra o cenário 2, os erros foram bem inferiores, todos abaixo de 1%, o que destaca a precisão do modelo de simulação para este cenário. O DagSim apresentou erro absoluto médio de 0,74%, com erro absoluto máximo igual a 0,9% e erro absoluto mínimo igual a 0,6%.

5.3.2 Modelo *fork/join*

Esta subseção traz os resultados das previsões para o modelo *fork/join* para os cenários 1 e 2, com execução em *clusters* homogêneos de máquinas virtuais Microsoft Azure D12v2, cada uma com 4 núcleos. Os *clusters* variam de 3 nós (12 núcleos) até 13 nós (52 núcleos). Para o *fork/join*, foram realizados apenas experimentos de validação. Os resultados podem ser vistos na tabela Tabela 5.3.

Tabela 5.3: Validação com o modelo *fork/join* (cluster homogêneo)

# Núcleos	Cenário 1 T_{Real} (s)	Cenário 1 $T_{Previsto}$ (s)	Cenário 1 Erro	Cenário 2 T_{Real} (s)	Cenário 2 $T_{Previsto}$ (s)	Cenário 2 Erro
12	722,2	1.945,3	-169,4%	719,9	2.041,5	-183,6%
16	582.9	1573.8	-170,0%	562,7	1763,2	-213,3%
20	515.9	1408.9	-173,1%	471,8	1.454,2	-208,2%
24	447.6	1266.4	-182,9%	417,7	1.335,4	-219,7%
28	415.7	1138.7	-173,9%	364,1	1.172,8	-222,1%
32	366.1	1037.4	-183,4%	324,7	1.068,4	-229,0%
36	306.1	953.8	-211,6%	306,8	1.004,2	-227,3%
40	287.5	903.9	-214,4%	275,2	920,0	-234,3%
44	259.7	855.1	-229,3%	258,8	884,2	-241,7%
48	248.6	865.7	-248,2%	250,0	857,7	-243,1%
52	220.2	784.5	**-256,3%**	226,1	805,6	**-256,3%**

Apesar de rápido, o modelo *fork/join* se mostrou impreciso para prever o tempo de resposta para aplicações Spark em todas as configurações de *cluster* experimentadas. Deve-se ressaltar aqui que, de acordo com os resultados, o modelo não captura detalhes suficientes da estratégia de execução paralela do Spark. Trata-se de um modelo que usa como parâmetros apenas o tempo médio de resposta em um único núcleo (vCPU) e a quantidade total de núcleos do sistema paralelo para produzir uma previsão.

A justificativa aqui é que o modelo superestima o tempo de resposta por computar atrasos para os passos de sincronização de forma repetida quando a demanda por recursos é elevada. Como geralmente os estágios Spark são separados em inúmeras tarefas e cada núcleo de processamento só pode executar uma tarefa por vez, o modelo *fork/join* pode acabar por considerar cada rodada de tarefas como um passo passível de sincronização. Isto causa uma inflação nos valores dos atrasos de sincronização que não reflete o que de fato acontece internamente no Spark.

A execução do modelo *fork/join* apresentou tempos de resposta abaixo dos 5 milissegundos. Entretanto, devido à baixa precisão em cenários simples, como os cenários 1 e 2 de *clusters* homogêneos, a análise da rapidez do modelo torna-se dispensável. Também, opta-se por não executar o modelo *fork/join* para os demais cenários tendo em vista os resultados ruins para as previsões usando o modelo para um cenário relativamente mais estável que os demais.

5.3.3 Modelo de precedência de estágios

De forma similar à subseção anterior, esta subseção apresenta os resultados das tarefas de validação e extrapolação do modelo de precedência de estágios para os cenários 1 e 2. Note que para as extrapolações foram considerados o menor número de núcleos, i.e., 12, como referência para a extrapolação do tipo A, um número intermediário, i.e., 32, para a extrapolação do tipo B e o maior número de núcleos considerados nestes cenários, i.e., 52, para a extrapolação do tipo C. Os três valores de referência aparecem na tabela marcados com REF nas colunas correspondentes a resultados de extrapolação.

Os erros para os experimentos de cada configuração são apresentados na Tabela 5.4. Para cada tarefa de previsão, o maior erro observado está em negrito, e o menor com preenchimento cinza. Em seguida, é apresentada uma sumarização dos erros para este cenário tanto para as validações quanto para as extrapolações. A sumarização contém erros médios, máximos, mínimos e desvio padrão.

Tabela 5.4: Validação e Extrapolação com o modelo de precedência de estágios (cluster homogêneo)

Núcleos	T_{real} (s)	Validação		Extrapolação A		Extrapolação B		Extrapolação C	
		T_{prev}(s)	Erro	T_{prev}(s)	Erro	T_{prev}(s)	Erro	T_{prev}(s)	Erro
Cenário 1 - Q26									
12	660,8	690,2	-4,4%	REF.	REF.	670,9	-1,5%	722,3	-9,3%
16	534,8	543,9	-1,7%	495,7	7,3%	511,4	4,4%	555,4	-3,9%
20	454,1	469,0	-3,3%	401,5	11,6%	417,0	8,2%	454,3	-0,04%
24	385,7	398,3	-3,3%	334,6	13,2%	347,8	9,8%	375,1	2,7%
28	354,2	367,2	-3,7%	289,6	**18,2%**	303,1	**14,4%**	324,9	8,3%
32	304,1	316,5	-4,1%	259,2	14,8%	REF.	REF.	290,4	4,5%
36	244,3	256,1	-4,8%	231,4	5,3%	239,6	1,9%	259,0	-6,0%
40	225,5	236,8	-5,0%	217,8	3,4%	223,8	0,8%	244,7	-8,5%
44	199,0	209,6	-5,3%	199,7	-0,4%	206,4	-3,7%	222,6	**-11,9%**
48	186,7	197,2	-5,6%	185,7	0,5%	193,7	-3,7%	207,1	-10,9%
52	170,6	181,4	**-6,3%**	169,7	0,5%	173,2	-1,5%	REF.	REF.
Cenário 2 - Q52									
12	658,5	660,8	-0,3%	REF.	REF.	693,5	**-5,3%**	741,8	-12,6%
16	515,3	517,3	-0,4%	512,5	0,5%	523,7	-1,6%	566,2	-9,9%
20	410,6	412,7	-0,5%	408,4	0,5%	420,5	-2,4%	448,4	-9,2%
24	356,3	358,3	-0,6%	347,0	2,6%	355,5	0,2%	379,9	-6,6%
28	302,8	304,7	-0,6%	300,0	0,9%	304,9	-0,7%	328,1	-8,4%
32	263,1	265,0	-0,7%	267,2	-1,6%	REF.	REF.	291,3	-10,7%
36	245,1	247,0	-0,8%	237,1	**3,3%**	241,2	1,6%	258,3	-5,4%
40	213,4	215,2	-0,8%	220,3	-3,2%	223,9	-4,9%	240,4	**-12,7%**
44	198,1	200,2	**-1,1%**	203,7	-2,8%	207,1	-4,5%	222,9	-12,5%
48	188,9	190,7	-1,0%	188,8	0,1%	192,3	-1,8%	205,6	-8,8%
52	177,4	179,3	**-1,1%**	173,9	2,0%	175,5	1,1%	REF.	REF.

Tabela 5.5: Sumarização dos erros dos cenários 1 e 2 (cluster homogêneo)

	Erro para a consulta Q26				Erro para a consulta Q52			
	Validação	Ext. A	Ext. B	Ext. C	Validação	Ext. A	Ext. B	Ext. C
Média	4,3%	7,5%	5,3%	6,6%	0,70%	1,80%	2,40%	9,70%
MIN	1,7%	0,4%	0,8%	0,0%	0,30%	0,10%	0,20%	5,40%
MAX	6,3%	18,2%	14,4%	11,9%	1,10%	3,30%	5,30%	12,70%
Desv. Padrão	1,3%	6,6%	4,3%	3,8%	0,30%	1,20%	1,80%	2,50%

Considerando todas as configurações de *cluster* para o cenário 1, tanto a validação quanto a extapolação apresentaram erros médios abaixo de 10%. No caso da validação o erro médio do cenário foi de apenas 4,3% com um desvio padrão de 1,3%. Estes resultados refletem a pequena diferença entre o erros absolutos mínimo e máximo da validação para a consulta 26, com valores de 1,3% e 6,3%, respectivamente. Para as extrapolações, o maior erro médio encontrado foi de 9,70% para a extrapolação do tipo C. Já, o erro absoluto máximo aconteceu para a extrapolação tipo A, com 18,2% de erro para a configuração de 28 núcleos. Ainda, estes erros são condizentes com o esperado para modelos analíticos de previsão de desempenho para sistemas paralelos em cenários práticos [Menasce et al., 2004].

Mais especificamente, já era esperado que as validações apresentassem uma maior precisão que as extrapolações. Isto se justifica pelo fato de as extrapolações terem como referência para a construção das entradas *logs* de execuções prévias em configurações diferentes da qual se deseja prever. Estas diferenças também são notadas a partir do desvio padrão dos erros, que possuem uma maior variabilidade para a extrapolação.

Nas três extrapolações, a que usou o menor número de núcleos como referência (extrapolação A) foi a que levou aos maiores erros. Uma justificativa para este fato é a incidência de mais contenção de recursos, o que pode causar uma maior variabilidade nos tempo de resposta das tarefas e, consequentemente, uma maior variabilidade nos tempos de resposta dos estágios.

Considerando que o cenário 2 possui uma menor quantidade de paralelismo entre estágios, conforme apresentado na Seção 4.5, é possível observar nos resultados uma tendência a menores erros para as previsões (exceto para a extrapolação C). A validação apresentou erro médio de 0,70%, com erro absoluto mínimo igual a 0,3% e erro absoluto máximo igual a 1,1%, o que representa uma precisão melhor que o esperado para o modelo. Além computar menos atrasos devido a um grafo de precedência mais simples, este cenário também possui uma menor variabilidade nos tempos de resposta reais medidos e apresentados na Seção 4.6 relativos às execuções prévias da consulta 52 para o cenário 2. Ambas as características afetam consideravelmente a precisão da validação e da extrapolação. É importante pontuar que mesmo para a extrapolação do tipo C, que apresentou erros maiores que as demais previsões, os erros são ainda razoavelmente baixos, com erro absoluto mínimo igual a 5,4% e erro absoluto máximo igual 12,7%.

O modelo de precedência de estágios se mostrou bem melhor que o modelo *fork/join* e apresentou resultados satisfatórios tanto para a validação quanto para a extrapolação para os cenários 1 e 2. Isto se deve ao fato de o modelo de precedência de estágios capturar, além das demais características, as relações de precedência entre os estágios no Spark. Desta forma, o modelo é capaz de contabilizar ambos os atrasos por contenção de recursos durante a solução da aproximação do MVA quanto os atrasos relativos à sincronização refletidos pelas relações de precedências entre os estágios, dadas as probabilidades de execução concorrente dos estágios.

Os resultados do modelo de simulação DagSim suportam ainda mais nossa afirmativa de que o modelo de precedência de estágios é satisfatório em precisão para previsão do tempo de resposta de aplicações Spark. Para o cenário 1, o modelo DagSim apresentou erro médio de aproximadamente 7,73% enquanto que o modelo de precedência de estágios apresentou erro médio de 4,31% para a tarefa de validação. Para o cenário 2, o erro médio foi de 0,74% para o DagSim e de 0,72% para o modelo de precedência de estágios também para a tarefa de validação.

Para as tarefas de extrapolação, o modelo de precedência de estágios apresentou erros mais elevados, com erro absoluto máximo de 18,2% (extrapolação tipo A) e 12,7% (extrapolação tipo C) para os cenários 1 e 2, respectivamente. Apesar de valores mais elevados, os resultados são bastante satisfatórios. Conforme afirmado por Menasce et al. [2004], modelos analíticos comumente apresentam erros na previsão de até 30%.

5.4 Cenários 3 e 4 (VMs A3)

5.4.1 Modelo DagSim

Esta subseção apresenta os resultados cedidos pelos pesquisadores do Politecnico di Milano sobre o DagSim para os cenários 3 e 4 de contenção de recursos. Estes resultados servem de comparativo na avaliação dos resultados do modelo de precedência de estágios para esta seção. Cabe ressaltar que no cenário de contenção de recursos o tempo dispendido na inicialização da aplicação e na alocação dos recursos foi, no geral, maior que nos demais cenários. Isto é refletido na diferença entre os tempos reais medidos para o DagSim e para o modelo de precedência de estágios. O modelo de simulação acresce tais atrasos ao final das previsões enquanto que o modelo de precedência de estágios apenas considera o intervalo de tempo relativo à real execução da aplicação. Os resultados para o DagSim são dispostos na Tabela 5.6.

Para o cenário 3, o DagSim apresentou erro médio de 2,95%, com erro absoluto máximo igual a 7,10% e erro absoluto mínimo igual a 0,01%. Já, para o cenário 4, este modelo apresentou erros abaixo de 1,2% (exceto para 30 núcleos, que apresentou 11,6% de erro devido à instabilidade desta configuração observável pelos *logs*). Considerando os resultados, pode-se dizer que o DagSim é um modelo bastante estável, que captura, inclusive, características comuns em cenários de contenção de recursos como os representados pelo cenário 3 e 4.

Tabela 5.6: Validação com o modelo *DagSim* (contenção de recursos)

# Nós	# Núcleos	Cenário 3 T_{Real} (s)	$T_{Previsto}$ (s)	Erro	Cenário 4 T_{Real} (s)	$T_{Previsto}$ (s)	Erro
3	6	2532,3	2538,5	-0,3%	2.158,9	2.153,0	0,3%
3	8	2071,2	2086,1	-0,7%	1.709,2	1.702,5	0,4%
4	10	1778,8	1778,6	0.01%	1.327,4	1.316,3	0,8%
4	12	1690,6	1704,5	-0,8%	1.124,5	1.117,4	0,6%
5	14	1.439,3	1.452,9	-0,9%	976,4	970,5	0,6%
5	16	1.271,6	1.281,0	-0,7%	884,9	880,4	0,5%
6	18	1.127,2	1.152,9	-5,4%	816,5	809,8	0,8%
6	20	1.093,6	1.095,2	-0,1%	738,8	733,3	0,7%
7	22	996,7	1.050,4	5,4%	667,9	663,3	0,7%
7	24	911,2	933,9	-2,5%	620,1	615,8	0,7%
8	26	720,8	735,6	-2,1%	572,1	568,4	0,6%
8	28						
9	30	658,8	691,6	-5,0%	525,9	465,0	**11,6%**
9	32	629,8	643,7	-2,2%	492,3	487,9	0,9%
10	34	625.8	647.6	-3.5%	462,2	457,4	1,0%
10	36	577.4	602.9	-4.4%	442,1	439,3	0,6%
11	38	546.6	572.1	-4.7%	438,7	436,8	0,4%
11	40	530.6	555.2	-4.6%	418,4	415,6	0,7%
12	42	488.4	510.6	-4.6%	392,1	390,2	0,5%
12	44	470.7	493.8	-4.9%	383,7	379,5	1,1%
13	46	446.4	455.2	-2.0%	378,4	380,1	-0,4%
13	48	430.5	460.9	**-7.1**	362,8	359,3	1,0%

5.4.2 Modelo de precedência de estágios

O cenário 4 possui características diferentes do sistema paralelo para as mesmas consultas Q26 e Q52 apresentadas para o cenário 1 e 2. Nestes cenários, os *clusters* explorados não são homogêneos. Mais especificamente, os nós não possuem a mesma quantidade de núcleos de processamento (vCPU), além de representar um cenário de contenção de recursos simulada. Desta forma, as aplicações destes cenários enfrentaram situações em que os núcleos e a memória RAM já estavam alocados para outros processos, o que causou uma maior heterogeineidade no sistema paralelo.

Note que para as extrapolações foram considerados o menor número de núcleos, i.e., 4 nós e 10 núcleos, como referência para a extrapolação do tipo A, um número intermediário, i.e., 9 nós e 30 núcleos, para a extrapolação do tipo B e o maior número de núcleos considerados nestes cenários, i.e., 13 nós e 48 núcleos, para a extrapolação do tipo C. A Tabela 5.7 apresenta os resultados para cada configuração testada para os cenários 3 e 4, e a Tabela 5.8 sumariza estes resultados, apresentado erros médios, mínimo e máximo (assim como desvio padrão) para previsão e extrapolação, para cada cenário.

Considerando a validação, o cenário 3 apresentou erro médio baixo com valor igual a 2,5%, com erros absolutos mínimo e máximo de 0,2% e 5,5%, respectivamente. O cenário 4, apresentou erro médio, máximo e mínimo ainda mais baixos, seguindo a tendência apresentada e justificada para os cenários 1 e 2. Tanto os cenários 1 e 2, quanto os cenários 3 e 4 possuem como aplicação paralela as mesmas consultas Q26 e Q52 do TPC-DS.

Tabela 5.7: Validação e extrapolação nos cenários 3 e 4 (contenção de recursos)

Nós	Núcleos	T_{real} (s)	Validação T_{prev}(s)	Validação Erro	Extrapolação A T_{prev}(s)	Extrapolação A Erro	Extrapolação B T_{prev}(s)	Extrapolação B Erro	Extrapolação C T_{prev}(s)	Extrapolação C Erro
Cenário 3 - Q26										
4	10	1.718,5	1.763,6	-2,6%	REF.	REF.	1.788,5	-4,1%	1.753,7	-2,0%
4	12	1.632,3	1.674,5	-2,6%	1.517,3	7,0%	1.518,9	6,9%	1.463,0	10,4%
5	14	1.381,1	1.414,0	-2,4%	1.321,8	4,3%	1.305,7	5,5%	1.262,9	8,6%
5	16	1.214,0	1.243,3	-2,4%	1.171,3	3,5%	1.163,5	4,2%	1.119,1	7,8%
6	18	1.069,5	1.099,8	-2,8%	1.039,7	2,8%	1.036,4	3,1%	995,6	6,9%
6	20	1.036,2	1.064,2	-2,7%	936,8	9,6%	936,6	9,6%	908,6	12,3%
7	22	936,4	963,2	-2,9%	866,7	7,4%	868,9	7,2%	829,1	11,5%
7	24	850,6	874,8	-2,8%	805,4	5,3%	803,4	5,5%	766,0	9,9%
8	26	657,4	682,2	-3,8%	757,6	-15,2%	757,6	-15,2%	715,6	-8,9%
9	30	593,6	617,6	0,8%	658,4	-10,9%	REF.	REF.	639,0	-7,6%
9	32	565,6	589,0	-3,3%	627,6	-11,0%	629,9	-11,4%	605,6	-7,1%
10	34	561,4	584,3	5,2%	599,6	-6,8%	588,0	-4,7%	562,6	-0,2%
10	36	511,2	532,4	1,6%	565,2	-10,6%	565,3	-10.6%	536,0	-4,9%
11	38	482,3	503,1	-1,0%	557,1	-15,5%	544,5	-12.9%	527,9	-9,5%
11	40	466,3	487,3	4,1%	520,5	-11,6%	517,1	-10.9%	496,5	-6,5%
12	42	425,2	447,3	-0,6%	491,8	-15,7%	494,6	-16.3%	468,9	-10,3%
12	44	406,3	427,6	0,2%	492,0	-21,1%	486,1	-19,6%	461,4	**-13,6%**
13	46	383,2	405,6	-1,1%	472,0	-23,2%	455,1	-18,8%	435,5	**-13,6%**
13	48	367,2	387,5	**-5,5%**	459,0	**-25,0%**	452,1	**-23,1%**	REF.	REF.
Cenário 4 - Q52										
4	10	1.270,6	1.276,4	-0,5%	REF.	REF.	1.354,5	-6,6%	1.370,8	-7,9%
4	12	1.067,4	1.072,5	-0.5%	1.075,8	-0,8%	1.141,7	-7,0%	1.152,0	-7,9%
5	14	918,9	924,0	-0,6%	929,7	-1,2%	986,8	-7,4%	993,0	-8,1%
5	16	827,7	832,6	-0,6%	822,7	0,6%	869,7	-5,1%	873,8	-5,6%
6	18	759,6	764,8	-0,7%	728,3	**4,1%**	775,1	-2,0%	780,7	-2,8%
6	20	682,0	687,3	-0,8%	653,7	**4,1%**	693,9	-1,7%	700,3	-2,7%
7	22	608,7	613,6	-0,8%	605,6	0,5%	638,9	-5,0%	647,5	-6,4%
7	24	561,2	566,2	-0,9%	553,6	1,4%	588,6	-4,9%	589,9	-5,1%
8	26	508,0	512,7	-0,9%	527,6	-3,9%	561,7	**-10,6%**	559,8	**-10,2%**
8	28	474,2	478,8	-1,0%	479,8	-1,2%	509,8	-7,5%	510,8	-7,7%
9	30	461,8	466,6	-1,0%	457,7	0,9%	REF.	REF.	483,0	-4,6%
9	32	428,0	432,7	-1,1%	429,2	-0,3%	453,7	-6,0%	459,3	-7,3%
10	34	397,8	402,5	-1,2%	402,8	-1,3%	424,1	-6,6%	431,1	-8,4%
10	36	377,9	382,6	-1,2%	379,3	-0,4%	399,5	-5,7%	405,9	-7,4%
11	38	375,6	380,2	-1,2%	377,9	-0,6%	397,5	-5,8%	403,0	-7,3%
11	40	354,3	359,1	-1,4%	354,5	-0,1%	370,2	-4,5%	375,4	-6,0%
12	42	329,5	334,2	-1,4%	328,7	0,2%	346,2	-5,1%	352,4	-6,9%
12	44	321,3	325,9	-1,4%	328,7	-2,3%	344,1	-7,1%	351,2	-9,3%
13	46	314,2	318,8	**-1,5%**	303,6	3,4%	321,8	-2,4%	324,7	-3,3%
13	48	300,4	305,0	**-1,5%**	303,7	-1,1%	321,2	-6,9%	REF.	REF.

Tabela 5.8: Sumarização dos erros dos cenários 3 e 4 (contenção de recursos)

	Erro para consulta Q26 Validação	Ext. A	Ext. B	Ext. C	Erro para consulta Q52 Validação	Ext. A	Ext. B	Ext. C
Média	2.5%	11.5%	10.5%	8.4%	1.0%	1.5%	5.7%	6.6%
MIN	0.2%	2.8%	3.1%	0.2%	0.5%	0.1%	1.7%	2.7%
MAX	5.5%	25.0%	23.1%	13.6%	1.5%	4.1%	10.6%	10.2%
Desv. Padrão	1.5%	6.6%	6.0%	3.6%	0.3%	1.4%	2.1%	2.1%

Com relação às extrapolações, pode-se notar que os erros médios e os erros máximos são maiores para os cenários 3 e 4 que para os cenários 1 e 2. Isto se justifica pelo incidência de uma maior contenção de recursos para os cenários 3 e 4, que, consequentemente, causa uma maior variabilidade nos tempos de resposta medidos para as tarefas e os estágios, dificultando as previsões para configurações diferentes dos *logs* prévios usados como referência. Apesar disso, os resultados das extrapolações se mantiveram dentro do esperado para modelos analíticos de previsão de desempenho de sistemas paralelos.

5.5 Cenários 5, 6 e 7 (VMs D4v2)

5.5.1 Modelo DagSim

Esta subseção apresenta os resultados do DagSim cedidos pelos pesquisadores do Politecnico di Milano para os cenários 5, 6 e 7 baseados em algoritmos comuns em aprendizado de máquina (K-Means, regressão logística e SVM) para diferentes volumes de dados. Parte destes resultados foram publicados por Ardagna et al. [2018] e servem de comparativo na avaliação dos resultados do modelo de precedência de estágios para esta seção. Os resultados para este modelo são dispostos na Tabela 5.9.

Tabela 5.9: Validação nos cenários 5, 6 e 7

	24 núcleos			48 núcleos		
Volume	T_{real} (s)	T_{prev} (s)	erro	T_{real}	T_{prev} (s)	erro
Cenário 5 - K-Means						
8GB	99,0	75,6	23,6%	90,3	70,3	22,1%
48GB	342,2	364,6	-6.5%	195,0	219,2	-12.4%
96GB	862,1	788,4	8,5%	594,3	746,2	**-25,6%**
Cenário 6 - Regressão Logística						
8GB	164,6	156,1	5,1%	166,5	156,5	**6,0%**
48GB	669,4	671,7	-0,3%	368,2	362,9	1,4%
96GB	1.418,8	1.404,9	0,9%	1.200,7	1.193,9	0,5%
Cenário 7 - SVM						
8GB	190,9	167,7	12,2%	189,7	164,0	**13,5%**
48GB	356,7	358,1	0,4%	3.725,0	352,2	5,4%
96GB	1.367,0	1.323,9	3,2%	650,5	635,4	2,3%

O erro médio para o cenário 5 (K-Means) foi de 16,45%, enquanto que o cenário 6 (regressão logística) apresentou erro médio de 2,37% e o cenário 7 (SVM) de 6,17%.

5.5.2 Modelo de precedência de estágios

Com configurações homogêneas de 24 núcleos (3 nós) e 48 núcleos (6 nós) e tamanhos de dados variando entre 8GB, 48GB e 96GB, estes cenários apresentaram resultados razoáveis para as previsões, salvo algumas exceções. Para estes cenários, as extrapolações são feitas de 24 núcleos para 48 núcleos e de 48 núcleos para 24 núcleos, tendo em vista que estas são as duas configurações de *cluster* experimentadas. Sendo assim, esta tabela é organizada de forma diferente dos cenários anteriores. Os resultados são exibidos na

Tabela 5.10.

Tabela 5.10: Validação e extrapolação nos cenários 5, 6 e 7

	24 núcleos					48 núcleos				
		Validação		Extrapolação			Validação		Extrapolação	
Volume	T_{real} (s)	T_{prev} (s)	Erro	T_{prev} (s)	Erro	T_{real}	T_{prev} (s)	Erro	T_{prev} (s)	Erro
Cenário 5 - K-Means										
8GB	83,4	81,9	1,8%	73,8	11,5%	75,3	74,0	1,7%	81,7	-8,5%
48GB	326,3	325,1	0,4%	351,9	-7,8%	180,1	178,8	0,7%	180,0	0,1%
96GB	847,0	845.9	0,1%	1.148,3	-35,6%	575,1	572,9	0,4%	523,0	9,1%
Cenário 6 - Regressão Logística										
8GB	164,7	159,5	3,2%	160,7	2,4%	166,6	161,0	3,4%	159,2	4,4%
48GB	669,5	664,4	0,8%	705,2	-5,3%	368,3	362,5	1,6%	368,4	-0,03%
96GB	1.418,9	1.414,1	0,3%	2.367,6	-66,9%	1.200,8	1.192,6	0,7%	766,3	36,2%
Cenário 7 - SVM										
8GB	176,3	171,7	2,6%	169,8	3,7%	174,9	170,2	2,7%	171,3	2,1%
48GB	341,5	339,3	0,7%	648,0	-89,8%	358,0	353,3	1,3%	210,3	41,3%
96GB	1.353,5	1.349,6	0,3%	1.216,4	10,1%	636,0	631,1	0,8%	758,7	-19,3%

Os cenários 5, 6 e 7 apresentaram erros baixos para as validações, com valores de erros médios iguais a 0,85%, 1,67% e 1,40% para os cenários 5 (K-Means), 6 (regressão logística) e 7 (SVM), respectivamente. Estes valores para erro na validação demonstram uma precisão mais elevada que o DagSim. Para as extrapolações, os erros também foram razoavelmente baixos. Porém, pode-se observar erros absolutos elevados variando entre 36,2% e 89,9%, para a regressão logística com 96 GB de dados e para o SVM com 48 GB de dados. A expectativa para esta situação é que o DagSim seja mais estável e preciso, o que poderia ser validado em trabalhos futuros.

É importante notar que nas extrapolações em que o modelo teve precisão baixa, os tempos reais de resposta dos algoritmos (T_{real}) foram bastante parecidos tanto para 24 quanto para 48 núcleos. Considerando a maior disponibilidade de recursos para o cenário com 48 núcleos, a expectativa era de que os tempos de resposta diminuissem consideravelmente para estas configurações, assim como ocorreu para as demais.

Possíveis justificativas para a instabilidade notada para as configurações mencionadas acima são: (i) congestionamento de rede nos dispositivos físicos onde as máquinas virtuais estavam alocadas; ii) gargalo causado pela quantidade de memória disponível para as operações executadas; iii) gargalo causado por pouco espaço disponível em disco para essas configurações em específico, o que pode causar atrasos na leitura e escrita de dados, ou mesmo durante a alocação de memória virtual; iv) falha nas réplicas de dados; v) configuração incorreta dos *data nodes*, mantendo dados apenas em 3 dos 6 nós disponíveis. Nenhuma destas situações é capturada pelo modelo.

Apesar de, em menor quantidade, estes cenários apresentarem configurações em que o modelo não teve boa precisão, as demais configurações, em sua maioria com resultados precisos, demonstram que o modelo é satisfatório para a previsão de aplicações com as características apresentadas pelo K-Means, regressão logística e SVM. Como exemplo, pode-se destacar que a a extrapolação de 24 núcleos para 48 núcleos do K-Means apresentou erro absoluto máximo de apenas 9,1%, como destacado para a carga de trabalho de 96GB. Para a extrapolação do mesmo algoritmo, o erro absoluto mínimo foi de 0,1%.

5.6 Cenários 8 e 9 (VMs D12v2)

Estes cenários apresentam os resultados de um algoritmo de contagem de palavras com múltiplas fontes de dados e de um multiclassificador composto pelos algoritmos SVM, regressão logística e Naive-Bayes executando de forma concorrente. Ambos os cenários apresentam demanda elevada por recursos, o que pode causar maiores atrasos por contenção de recursos. Por se tratarem de cenários planejados exclusivamente para avaliar os modelos analíticos, experimentos do DagSim não foram executados até então, podendo ser realizados em trabalhos futuros. Ainda, de todos os cenários avaliados, o *wordcount* múltiplo é o algoritmo com o maior nível de paralelismo de estágios, conforme descrito na Seção 4.5. A Tabela 5.11 lista os resultados para as validações e extrapolações.

Tabela 5.11: Validação e extrapolação nos cenários 8 e 9

Cores	$T_{real}(s)$	Validação $T_{previsto}(s)$	Erro	Extrapolação A $T_{previsto}(s)$	Erro	Extrapolação B $T_{previsto}(s)$	Erro	Extrapolação C $T_{previsto}(s)$	Erro
Cenário 8									
48	474,70	593,5	-25,03%	REF.	REF.	640,0	-34,83%	685,4	-44,40%
72	317,5	416,3	-31,12%	386,1	-21,62%	REF.	REF.	445,9	-40,46%
96	239,7	330,3	-37,82%	286,0	-19,35%	308,3	-28,64%	REF.	REF.
Cenário 9									
12	1.814,0	1.701,3	6,21%	REF.	REF.	1.811,6	0.13%	1.965,5	-8,35%
16	1.468,6	1.358,7	7,48%	1.276,0	13,12%	REF.	REF.	1.474,1	-0.38%
20	1.291,2	1.179,3	8,67%	1.020,8	20,94%	1.087,0	15,82%	REF.	REF.

O cenário 8 apresentou erros razoáveis para a extrapolação A, na qual *logs* prévios do menor tamanho de *cluster* são usados como referência para extrapolar para tamanhos maiores de *cluster*. A validação e extrapolação B excederam um pouco o valor considerado razoável para erro em modelos analíticos (>30%), enquanto que a extrapolação C apresentou erros superiores a 40%. Uma justificativa para estes erros mais elevados é similar à apresentada nos cenários anteriores. Neste caso, o disco se manteve com aproximadamente 90% de taxa de ocupação, o que prejudica o desempenho individual das máquinas virtuais, seja na leitura e escrita de dados no disco ou na alocação de memória virtual. Além disso, a leitura simultânea de 6 arquivos diferentes do HDFS pode ter causado contenção de recursos de disco e rede.

Por outro lado, o cenário 9 se mostrou estável, com erros inferiores a 21% no pior dos casos para a extrapolação. A validação apresentou erro médio de 7,45%, considerado bastante baixo para modelos analíticos. Cabe ressaltar que este cenário foi executado em um *cluster OpenStack* na nuvem privada do laboratório Speed/UFMG, que é formada por *clusters* compartilhados entre os alunos do laboratório. O modelo de precedência de estágios se mostrou satisfatório para este cenário.

5.7 Considerações finais sobre os resultados

No geral, um total de 85 experimentos de validação diferentes foram executados para o modelo de precedência de estágios. O erro médio encontrado considerando todos os cenários foi de 3,10%. Para a extrapolação, o número de experimentos foi ainda maior, totalizando 207 experimentos diferentes. Para a extrapolação, o cenário 8 apresentou o maior erro médio, com 31,55% de erro, seguido do cenário 7 (SVM), com 27,72% de erro e do cenário 6 com 19,21% de erro. Todos os demais cenários apresentaram erro médio da extrapolação inferior a 10%.

No próximo capítulo, será feita uma recapitulação de forma a elucidar a linha seguida por esta dissertação, da proposta até a discussão de resultados. Serão apresentadas as conclusões desta dissertação, tendo como base a proposta e os resultados apresentados no Capítulo 5. Por fim, serão sugeridos trabalhos futuros, tanto para preencher as lacunas deixadas por esta dissertação como para expandir a aplicabilidade dos modelos estudados.

Capítulo 6

Conclusão

Dadas as características da computação em nuvem e do paralelismo para o processamento de dados massivos, conforme apresentado no Capítulo 1, a previsão de desempenho de sistemas paralelos se tornou tarefa importante na garantia da qualidade dos serviços online e na redução dos custos de infraestrutura. Ambos os modelos considerados nesta dissertação, o *fork/join* e o precedência de estágios, capturam (i) atrasos por contenção de recursos; e (ii) atrasos por sincronização de tarefas, e foram apresentados em detalhes no Capítulo 3 juntamente com as características do Apache Spark, plataforma tolerante a falhas mais popular para processamento de dados massivos. No caso do modelo de precedência de estágios, os atrasos por sincronização de tarefas são computados levando em consideração aspectos da execução concorrente dos estágios. Para isto, este modelo utiliza um grafo direcionado acíclico para representar as relações de precedência entre os estágios da aplicação e as probabilidades de execução concorrente entre os pares de estágios. Já, o modelo baseado em *fork/join* é mais simples e não captura os detalhes da precedência entre os estágios, tornando-o menos preciso.

A aplicação de ambos os modelos para a previsão do tempo de resposta de aplicações Spark seguiu uma abordagem diferenciada. Nesta dissertação, algumas das equações dos modelos foram substituídas pelo uso de informações extraídas dos *logs* de execuções prévias, o que é considerado geralmente mais preciso [Menasce et al., 2004]. Desta forma, as entradas para os modelos apresentadas no Capítulo 3 são construídas através de *logs* de execuções prévias planejadas e executadas em três ambientes de nuvem diferentes, a nuvem da Google (Google Cloud), a nuvem da Microsoft (Microsoft Azure) e uma nuvem privada configurada em OpenStack no laboratório SPEED da UFMG.

Levando em consideração o erro da previsão e o tempo de execução do modelo como métricas de avaliação dos modelos, os cenários experimentais foram apresentados no Capítulo 4, incluindo detalhes sobre a configuração dos *clusters* nos ambientes de nuvem e as características das aplicações executadas. Ao todo, 9 cenários diversos foram experimentados, incluindo ambos *clusters* homogêneos e heterogêneos, com e sem contenção de recursos e diferentes volumes de dados. Os algoritmos utilizados contemplam consultas *ad-hoc* com paralelismo moderado entre estágios, algoritmos comuns em aprendizado de máquina, como o K-Means, a regressão logística, e o SVM, um classificador múltiplo com alta demanda por recursos e um contador de palavras múltiplo com um maior paralelismo entre estágios.

As previsões foram executadas diversas vezes para cada cenário e os tempos médios de execução dos modelos foram apresentados no Capítulo 5. Os tempos médios de execução do modelo de simulação DagSim e os erros de previsão deste modelo foram cedidos por pesquisadores parceiros do Politecnico di Milano e publicados por Ardagna et al. [2018]. Os dados deste modelo são usados para fins de comparação com os modelos analíticos avaliados nesta dissertação.

Também no Capítulo 5 os erros insatisfatórios do modelo *fork/join* são apresentados para os cenários 1 e 2 e comparados aos erros do modelo de precedência de estágios. Devido à baixa precisão do modelo *fork/join*, nos cenários de 3 a 7 o modelo de precedência de estágios foi comparado apenas com os resultados do modelo DagSim. Ao final, os resultados de dois cenários, os cenários 8 e 9, ambos com alta demanda por recursos, são apresentados apenas para o modelo de precedência de estágios. Como trabalhos futuros, seria interessante executar o DagSim para estes cenários também para fins de comparação.

No geral, o modelo de precedência de estágios se mostrou satisfatório, com erro médio de interpolação, considerando todos os cenários, de 3,10%. Para a interpolação, um total de 85 experimentos diferentes foram executados para 9 cenários variados. Já, para a extrapolação foram feitos um total de 207 experimentos diferentes. Dentre os cenários, o que apresentou os piores resultados para a extrapolação foi o cenário 8, com o contador de palavras múltiplo, apresentando erro médio de 31,55%, seguido do cenário 7 (SVM), com 27,72% de erro médio e do cenário 6 com 19,21% de erro médio. Todos os demais cenários apresentaram erro médio da extrapolação inferior a 10%. Para modelos analítico de desempenho de sistemas paralelos, os erros encontrados são satisfatórios, dado que erros de até 30% são geralmente razoáveis para modelos desta categoria [Menasce et al., 2004].

A comparação de ambos o tempo de execução e a precisão do modelo com o estado-da-arte, o DagSim, mostra também que o modelo de precedência de estágios é um ótimo candidato para o uso de previsão de desempenho como suporte à reconfiguração dinâmica de sistemas paralelos em ambientes de nuvem. Com tempos de execução do modelo na casa dos milissegundos, o modelo de precedência de estágios é até 3 ordens de magnitude mais rápido que o DagSim. A aplicação na reconfiguração dinâmica poderia permitir tanto a garantia da qualidade de serviço de aplicações online massivas em dados quanto a redução do custo de infraestrutura em nuvem, melhorando a experiência dos usuários e reduzindo os custos de manutenção dos serviços.

6.1 Trabalhos futuros

Previsão de desempenho de sistemas paralelos é uma tarefa bastante específica, mas também bastante útil nos dias atuais. Da perspectiva desta dissertação, são diversas possibilidades de trabahos futuros que serão enumeradas nesta seção. Elas variam de novas estratégias de aplicação dos modelos, a novos cenários experimentais, como também a utilização prática dos modelos para a reconfiguração dinâmica de sistemas paralelos.

Uma possiblidade de trabalhos futuros seria a proposta de novos cenários experimentais, explorando aplicações com grafos de precedência de estágios diferentes dos já explorados. Neste caso, seria interessante tanto realizar experimentos para outras consultas *ad-hoc* do TPC-DS quanto explorar novos algoritmos da MLlib do Spark. De forma similar, seria interessante também explorar os mesmos algoritmos para outras cargas de trabalho e outros volumes de dados.

Considerando os cenários experimentais já existentes, seria interessante também executar o DagSim para os cenários 8 e 9 para fins de comparação. Estes resultados seriam bastante importantes para a análise e discussão dos resultados do cenário 8, que se mostrou menos preciso que os demais para o modelo analítico de precedência de estágios. Ainda, dadas as inconsistências de resultados apontadas para os cenários 6 e 7, seria interessante executar os mesmos experimentos em um ambiente de nuvem mais robusto, evitando impactos causados por contenções de recursos e com espaço o suficiente livre no disco.

Uma outra possibilidade é a de explorar outras estratégias de aplicação de modelos simples como o do *fork/join* para o Spark. Uma dessas estratégias seria similar à extrapolação feita para o modelo de precedência de estágios, distribuindo as tarefas entre a quantidade de núcleos disponíveis no *cluster* e usar o tempo médio da tarefa para encontrar uma estimativa para o tempo de cada estágio da aplicação.

Ainda, seria possível expandir os experimentos dessa dissertação com a finalidade de segmentar os recursos, ao invés de tratar um servidor completo como um centro de serviço. Assim, a análise do tempo de resposta levaria em consideração, individualmente, as características da CPU, da memória RAM, do disco e de outros elementos específicos do sistema paralelo com os estágios trafegando por redes de filas distintas para cada tipo de recurso representado. Este tipo de análise é perfeitamente possível e suportada pelo modelo de precedência de estágios.

Também como trabalhos futuros, seria interessante fazer um comparativo da abordagem apresentada com outras ferramentas para melhoria de desempenho de aplicações Hadoop/Spark através de *self-tuning*, como o Starfish [Herodotou et al., 2011].

Por fim, seria bastante pertinente dar sequência a esta dissertação através da proposta e avaliação de uma arquitetura de reconfiguração dinâmica de sistemas paralelos com a aplicação do modelo de precedência de estágios. Existem diversas abordagens para seguir a partir daqui. Uma delas consiste no preprocessamento e armazenamento de *logs* de aplicações prévias para consulta e utilização em tempo de execução. Basicamente, um módulo gerenciador do sistema paralelo alocaria recursos para execução de uma determinada aplicação levando em consideração o tempo de resposta necessário para garantia da qualidade de serviço. Durante a execução da aplicação, em intervalos de tempo preestabelecidos, o módulo gerenciador do sistema paralelo poderia executar novas previsões de tempo de resposta, podendo solicitar o provisionamento ou liberação de recursos do sistema paralelo. A primeira situação se trataria de uma ação proativa para garantia da qualidade de serviço, de forma que a aplicação consiga finalizar sua execução dentro do tempo de resposta limite. A segunda situação permitira a economia de recursos em nuvem garantindo o tempo de reposta limite. Para isto, seria necessário projetar e implementar no Spark estratégias de reconfiguração em tempo de execução. Possíveis momentos para ajuste poderiam ser, por exemplo, os intervalos de sincronização entre estágios.

Referências Bibliográficas

Adve, V. S. & Vernon, M. K. (2004). Parallel program performance prediction using deterministic task graph analysis. *ACM Transactions on Computer Systems (TOCS)*, 22(1):94--136.

Ardagna, D.; Barbierato, E.; Evangelinou, A.; Gianniti, E.; Gribaudo, M.; Pinto, T. B. M.; Guimarães, A.; Couto da Silva, A. P. & Almeida, J. M. (2018). Performance prediction of cloud-based big data applications. Em *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, pp. 192--199, New York, NY, USA. ACM.

Ardagna, D. et al. (2016). Modeling performance of hadoop applications: A journey from queueing networks to stochastic well formed nets. 16th ICA3PP. p. 599–613.

Arggawal, S. (2015). Apache spark introduction and resilient distributed dataset basics and deep dive. Disponível em: https://www.slideshare.net/differentsachin/apache-spark-introduction-and-resilient-distributed-dataset-basics-and-deep-dive. Acessado em: 01 de setembro de 2018.

Assunção, M. D.; Calheiros, R. N.; Bianchi, S.; Netto, M. A. & Buyya, R. (2015). Big Data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79:3--15.

B. M. Pinto, T.; Couto da Silva, A. P. & Almeida, J. M. (2018). Previs ão do tempo de resposta de aplicações de big data em ambientes de nuvem. Em *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, volume 36. SBC.

Badue, C.; Almeida, J.; Almeida, V.; Baeza-Yates, R.; Ribeiro-Neto, B.; Ziviani, A. & Ziviani, N. (2010). Capacity planning for vertical search engines. *arXiv preprint arXiv:1006.5059*.

Bagrodia, R.; Deeljman, E.; Docy, S. & Phan, T. (1999). Performance prediction of large parallel applications using parallel simulations. Em *ACM SIGPLAN Notices*, volume 34, pp. 151--162. ACM.

Bertoli, M.; Casale, G. & Serazzi, G. (2009). JMT: performance engineering tools for system modeling. *SIGMETRICS Performance Evaluation Review*, 36(4). p. 10–15.

Beyer, M. A. & Laney, D. (2012). The importance of 'big data': a definition. *Stamford, CT: Gartner*.

Borthakur, D. et al. (2008). Hdfs architecture guide. *Hadoop Apache Project*, 53(1-13):2.

Chen, C. P. & Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275. p. 314–347.

Chiola, G. (1985). A software package for the analysis of generalized stochastic petri net models. Em *International Workshop on Timed Petri Nets, Italy, 1985*. p. 136–143.

Clement, M. J. & Quinn, M. J. (1993). Analytical performance prediction on multicomputers. Em *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pp. 886--894. ACM. Portland, Oregon, USA.

Coulouris, G.; Dollimore, J.; Kindberg, T. & Blair, G. (2011). *Distributed Systems: Concepts and Design*. International computer science series. Addison-Wesley, USA, 5th edição. ISBN 0132143011, 9780132143011.

Databricks (2016). Apache Spark Survey 2016. Disponível em: https://databricks.com/2016-spark-survey. Acessado em: 30/11/2017.

Dean, J. & Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. *Proceedings of 6th Symposium on Operating Systems Design and Implementation*, pp. 137--149. USENIX Association.

Fox, A.; Griffith, R.; Joseph, A.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A. & Stoica, I. (2009). Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009.

Franks, B. (2014). *The Analytics Revolution: How to improve your business by making analytics operational in the Big Data era*. Wiley, primeira edição.

Ganapathi, A. (2009). *Predicting and optimizing system utilization and performance via statistical machine learning*. Tese de doutorado, UC Berkeley.

Gandomi, A. & Haider, M. (2015). Beyond the hype: Big Data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137–144. ISSN 0268-4012.

Herodotou, H.; Lim, H.; Luo, G.; Borisov, N.; Dong, L.; Cetin, F. B. & Babu, S. (2011). Starfish: A self-tuning system for big data analytics. Em *Cidr*, volume 11, pp. 261--272.

Ipek, E.; De Supinski, B. R.; Schulz, M. & McKee, S. A. (2005). An approach to performance prediction for parallel applications. Em *European Conference on Parallel Processing*, pp. 196--205. Springer.

Jacobs, A. (2009). The pathologies of big data. *Communications of the ACM*, 52(8):36--44. ISSN 0001-0782.

Kamburugamuve, S.; Fox, G.; Leake, D. & Qiu, J. (2013). *Survey of Apache Big Data Stack*. Tese de doutorado, Ph. D. Qualifying Exam, Dept. Inf. Comput., Indiana Univ., Bloomington, IN.

Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. "O'Reilly Media, Inc.".

Laskowski, J. (2018a). Mastering apache spark (2.3.1). Disponível em: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/. Acessado em: 01 de Setembro de 2018.

Laskowski, J. (2018b). Mastering apache spark (2.3.1) - sparklistenerinterface — internal contract for spark listeners. Disponível em: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/. Acessado em: 01 de Setembro de 2018.

Liang, D.-R. & Tripathi, S. K. (2000). On performance prediction of parallel computations with precedent constraints. *IEEE TPDS*, 11. ISSN 1045-9219. p. 491-508.

Mak, V. & Lundstrom, S. (1990). Predicting performance of parallel computations. *IEEE Transactions on Parallel & Distributed Systems*, 1. ISSN 1045-9219. p. 257-270.

Manyika, J.; Chui, M.; Brown, B.; Bughin, J.; Dobbs, R.; Roxburgh, C. & Byers, A. H. (2011). Big data: The next frontier for innovation, competition, and productivity. *Information Systems*.

Matsunaga, A. & Fortes, J. A. (2010). On the use of machine learning to predict the time and resources consumed by applications. Em *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 495--504. IEEE Computer Society.

Menasce, D. A.; Almeida, V. A.; Dowdy, L. W. & Dowdy, L. (2004). *Performance by design: computer capacity planning by example*. Prentice Hall Professional. Chapters 3, 11, 12, and 15.

Nelson, R. D. & Tantawi, A. N. (1988). Approximate analysis of fork/join synchronization in parallel queues. *IEEE Trans. Computers*, 37(6). p. 739-743.

Nudd, G. R.; Kerbyson, D. J.; Papaefstathiou, E.; Perry, S. C.; Harper, J. S. & Wilcox, D. V. (2000). Pace—a toolset for the performance prediction of parallel and distributed systems. *The International Journal of High Performance Computing Applications*, 14(3):228--251.

Reisig, W.; Rozenberg, G. & Thiagarajan, P. S. (2013). *In Memoriam: Carl Adam Petri*. Springer Berlin Heidelberg, Berlin, Heidelberg. p. 1–5.

Schneider, S.; Andrade, H.; Gedik, B.; Biem, A. & Wu, K. (2009). Elastic scaling of data parallel operators in stream processing. Em *2009 IEEE International Symposium on Parallel Distributed Processing*, pp. 1–12. ISSN 1530-2075.

Shvachko, K.; Kuang, H.; Radia, S. & Chansler, R. (2010). The hadoop distributed file system. Em *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pp. 1– 0. IEEE.

Singh, K.; İpek, E.; McKee, S. A.; de Supinski, B. R.; Schulz, M. & Caruana, R. (2007). Predicting parallel application performance via machine learning approaches. *Concurrency and Computation: Practice and Experience*, 19(17):2219--2235.

Song, G.; Meng, Z.; Huet, F.; Magoules, F.; Yu, L. & et al. (2013). A hadoop mapreduce performance prediction method. Em *Proceedings of the HPCC 2013*. p. 820-825.

Spark-Docs (2018a). Apache spark documentation: Cluster mode overview. Disponível em: https://spark.apache.org/docs/latest/cluster-overview.html. Acessado em: 01 de setembro de 2018.

Spark-Docs (2018b). Apache spark documentation: Job scheduling. Disponível em: https://spark.apache.org/docs/latest/job-scheduling.html. Acessado em: 01 de setembro de 2018.

Thomasian, A. & Bay, P. F. (1986). Analytic queueing network models for parallel processing of task systems. *IEEE Transactions on Computers*, (12):1045--1054.

Truta, H.; Vivas, J. L.; Brito, A. & Nobrega, T. (2017). A predictive approach for enhancing resource utilization in paas clouds. Em *Proceedings of the SAC 2017*.

Uysal, M.; Kurc, T. M.; Sussman, A. & Saltz, J. (1998). A performance prediction framework for data intensive applications on large scale parallel machines. Em *International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, pp. 243--258. Springer.

Varki, E. (1999). Mean value technique for closed fork-join networks. Em *ACM SIGMETRICS Performance Evaluation Review*, volume 27, pp. 103--112. ACM.

Vianna, E.; Comarela, G.; Pontes, T.; Almeida, J.; Almeida, V.; Wilkinson, K.; Kuno, H. & Dayal, U. (2013). Analytical performance models for mapreduce workloads. *International Journal of Parallel Programming*, 41(4):495--525.

Warneke, D. & Kao, O. (2011). Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):985–997. ISSN 1045-9219.

White, T. (2015). *Hadoop: The definitive guide*. O'Reilly Media, Inc., 4th edição.

Yang, X. & Sun, J. (2011). An analytical performance model of mapreduce. Em *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, pp. 306--310. IEEE.

Zaharia, M.; Chowdhury, M.; Das, T.; Dave, A.; Ma, J.; McCauley, M.; Franklin, M. J.; Shenker, S. & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Em *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 2--2. USENIX Association.

Zaharia, M.; Chowdhury, M.; Franklin, M. J.; Shenker, S. & Stoica, I. (2010). Spark: cluster computing with working sets. Em *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, p. 10. USENIX Association.

Zheng, G.; Kakulapati, G. & Kalé, L. V. (2004). Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. Em *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 78. IEEE.