Previsão do tempo de resposta de aplicações de big data em ambientes de nuvem

Túlio B. M. Pinto[1], Ana Paula Couto da Silva, Jussara M. Almeida

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Caixa Postal 6.627 – 31.270-901 – Belo Horizonte – MG – Brazil

{tuliobraga,ana.coutosilva,jussara}@dcc.ufmg.br

Abstract. *Data access heterogeneity and irregularity are typical properties of Big Data applications, and, therefore, turn hardware and software resource scheduling much more challenging. However, the flexibility and elasticity provided by cloud environments decrease the difficulty by allowing on-demand resource provisioning. Nonetheless, the performance prediction (e.g.: response time) of such applications increase in complexity as all these characteristics are combined. This work explores an analytical model for Spark applications' response time prediction, a popular platform for large-scale data processing, parametrized by earlier execution logs. This model is evaluated in several scenarios and applications. The results show relative errors lower than 8% for response time prediction, in average.*

Resumo. *Aplicações de big data têm tipicamente propriedades bem específicas, tais como heterogeneidade e irregularidade nos padrões de acesso aos dados, que tornam a alocação de recursos de hardware e software muito desafiadora. Por outro lado, a flexibilidade e a elasticidade provida por plataformas de computação na nuvem facilitam esta alocação à medida em que recursos são alocados sob demanda. Entretanto, estas características também tornam a previsão de desempenho (p.ex: tempo de resposta das aplicações) mais complexa. Este trabalho explora um modelo analítico para a previsão de tempo de resposta de aplicações executando na plataforma Spark, muito popular para processamento de dados em larga escala, que é parametrizado a partir de logs de execuções prévias. O modelo é avaliado em diversos cenários e aplicações, obtendo um erro relativo no tempo de resposta previsto inferior a 8%, em média.*

1. Introdução

Nos últimos anos, o surgimento e consolidação de diferentes aplicações online possibilitaram a geração massiva de dados, em pequenos intervalos de tempo, a partir de diversas fontes (redes sociais, comércio eletrônico, aplicações de Internet das Coisas, etc). Além da quantidade expressiva de dados *per se*, a geração de informação relevante a partir dos mesmos requer grande poder computacional e, em vários casos, respostas em tempo real. Desta forma, a quantidade e heterogeinidade de dados a serem processados por uma nova classe de aplicações (referenciadas como aplicações direcionadas a *Big Data*), apresentam diversos desafios para diferentes áreas do conhecimento [Chen and Zhang 2014].

Uma das áreas que está lidando com estes importantes desafios é a de processamento distribuído, principalmente os sistemas computacionais organizados em nuvem.

A flexibilidade e elasticidade proporcionadas pelos ambientes de nuvem têm sido importantes na minimização dos desafios impostos pelo *Big Data*. A possibilidade de disponibilização de recursos de forma dinâmica, bem como o modelo de pagamento por consumo dos seus recursos, têm estimulado o uso de nuvem para processamento de grandes volumes de dados. O surgimento do MapReduce, um modelo de programação simples e escalável para processamento de dados em paralelo em milhares de máquinas e sua implementação de código aberto mais popular, o Hadoop, impulsinaram este cenário. Diversos outros modelos e plataformas surgiram ao longo dos anos (p. ex. Storm, Pregel), sejam de propósito específico ou geral, como o Spark, o mais popular da atualidade. De acordo com a Databricks [Databricks 2016], as instalações do Spark em nuvem pública cresceram de 51% em 2015 para 61% em 2016 entre os respondentes de sua enquete.

Embora o uso de nuvem pública traga vantagens, a característica compartilhada dos recursos e a heterogeinidade dos dados impactam na regularidade da execução das aplicações. Deste modo, prever o desempenho de aplicações paralelas torna-se uma tarefa desafiadora e de extrema importância, à medida que o *Big Data* se populariza. A proposição de métodos de previsão de tempo de resposta para estas aplicações possibilita o planejamento e gerenciamento eficientes dos recursos computacionais. Por exemplo, durante a execução de uma aplicação que possui certos requisitos de qualidade de serviço, como limite máximo de tempo de resposta, há a necessidade de escalar o *cluster* e prover mais recursos de nuvem para que o sistema alcance o limiar imposto. Esta tarefa combina a complexidade de prever o desempenho de sistemas paralelos, a característica heterogênea dos dados e a base compartilhada dos recursos em nuvem.

Neste trabalho exploramos o uso de um modelo analítico de desempenho baseado em teoria de filas para a previsão do tempo de resposta de aplicações de processamento de grandes volumes de dados executadas utilizando a plataforma Spark. Este modelo, proposto originalmente em [Mak and Lundstrom 1990], considera a precedência das tarefas a serem executadas para o cálculo de métricas de desempenho. As precedências entre as tarefas de uma aplicação são representadas por um grafo direcionado acíclico (do inglês Directed Acyclic Graph ou DAG). O tempo de resposta médio da aplicação é estimado levando em consideração os recursos disponíveis no sistema, como por exemplo núcleos de processamento, modelados como servidores em uma rede de filas. Mais precisamente, o modelo estima os atrasos de sincronização ocorridos em uma aplicação e que são resultantes das relações de precedência e sobreposição entre as tarefas.

O modelo considerado foi originalmente proposto para aplicações paralelas de cunho real. O objetivo é investigar a sua eficácia para cenários específicos de aplicações orientadas a Big Data executando em plataforma Spark na nuvem. Para tal, o modelo é parametrizado a partir de logs de execuções anteriores da aplicação em análise. As previsões obtidas com o modelo são avaliadas considerando sete diferentes cenários, consistindo de diferentes ambientes de virtualização, configurações e recursos, bem como de perfis de aplicação (cargas de trabalho baseadas em SQL e algoritmos de aprendizado de máquina, por exemplo, o algoritmo de clusterização k-means).

Como principal contribuição este trabalho explora a aplicação de um modelo analítico de previsão de desempenho já existente para estimar o tempo de resposta de aplicações Spark em nuvem, particularmente considerando cenários de infraestrutura (número de núcleos de processamento disponíveis) diferentes dos utilizados em

execuções anteriores. Portanto, é feita uma avaliação diversificada da acurácia do modelo ao utilizar logs de execuções prévias como entrada, bem como um discussão da aplicabilidade da abordagem em um cenário real de ajuste dinâmico de recursos. Nossos principais resultados mostram o modelo é satisfatório tanto para prever o tempo de resposta para a mesma quantidade de núcleos dos logs prévios, quanto para extrapolar de uma quantidade de núcleos para outra. O primeiro processo apresentou um erro médio de 1,87%, enquanto que o segundo apresentou um erro médio de 7,9%. Tais erros podem ser considerados baixos, tendo em vista que erros de até 30% são comuns na previsão de tempo de resposta por modelos analíticos de filas tipicamente [Menasce et al. 2004].

O restante deste artigo está organizado como segue. Trabalhos relacionados são discutidos na próxima seção. A Seção 3 discute características da plataforma Spark e apresenta o modelo analisado. Os resultados experimentais são apresentados na Seção 4, juntamente com uma discussão dos resultados e formas de uso do modelo em ambientes reais. Por fim, conclusões e trabalhos futuros são apresentados na Seção 6.

2. Trabalhos Relacionados

Ao longo dos anos, diversos autores propuseram alternativas para a previsão do desempenho de aplicações paralelas. Elas variam entre modelos analíticos, de simulação ou híbridos. Várias técnicas de modelagem de desempenho com foco em modelos analíticos baseados em teoria de filas são discutidos em [Menasce et al. 2004]. Esta seção descreve os trabalhos relacionados, citando sucintamente modelos de simulação e híbridos, e focando nos modelos analíticos.

Previsões por simulação no Hadoop foram exploradas por outros autores, seja por métodos para análise de *jobs* Hadoop [Song et al. 2013] ou baseando em ferramentas de simulação gerais já existentes [Ardagna et al. 2016]. Neste último, são utilizadas duas ferramentas para simulação, a JMT [Bertoli et al. 2009] baseada em teoria de filas e a GreatSPN [Chiola 1985] para redes estocásticas baseadas em Petri Nets [Reisig et al. 2013]. Já, os modelos híbridos são mais recentes e sofisticados, compostos por uma série de abordagens para previsão de desempenho de sistemas paralelos, por exemplo, combinando técnicas de aprendizado de máquina e métodos analíticos [Popescu 2015].

O trabalho de [Nelson and Tantawi 1988] aborda uma infraestrutura homogênea formada por servidores de dados, chamados de *index*, controlados por computador mestre, chamado de *broker*. Neste modelo a infraestrutura computacional é representada por um sistema de filas *fork/join*, onde uma tarefa é quebrada em múltiplas subtarefas, distribuídas e executadas em cada *index* (fork). Os resultados das subtarefas são mesclados para obter o resultado final *join*. O tempo de execução da aplicação é estimado pelo tempo de execução da subtarefa no servidor mais lento. Este modelo é muito simples para representar o modelo de programação do Spark uma vez que ele não considera as dependências entre os estágios. O modelo utilizado neste trabalho considera a dependência das tarefas (estágios no Spark), sendo esta representada pelo fator de probabilidade de execução em paralelo calculado para cada par de tarefas.

O modelo utilizado neste trabalho foi proposto em [Mak and Lundstrom 1990] e tem como premissa a execução sequencial dos *jobs* no sistema. Este modelo é detalhado na Seção 3.2. os autores em [Tripathi and Liang 2000], estendem o modelo proposto em [Mak and Lundstrom 1990] de modo a considerarem o paralelismo entre dois ou mais

jobs no processo de previsão de desempenho. Para isto, ele modela ambos os paralelismos *intra-jobs* e *inter-jobs*. Como *jobs* no Spark são por padrão sequenciais, o modelo proposto em [Mak and Lundstrom 1990] é mais simples e suficiente para prever o tempo de resposta das aplicações aqui abordadas.

No âmbito da otimização do uso de recursos em ambientes em nuvem, os autores em [Truta et al. 2017] propõem a análise direta dos *jobs* submetidos ao sistema. O processo de otimização se baseia em um algoritmo de escalonamento de recursos combinado a um previsor de utilização de capacidade de curto tempo baseado em logs históricos. Para o escalonamento da tarefa, é considerado o estado de carga atual e o mais recente para decidir entre i) atrasar a execução da tarefa, ii) executar a tarefa imediatamente caso existam recursos disponíveis, iii) acelerar a execução da tarefa caso existam recursos adicionais, iv) rejeitar a terafa se ela requer mais recursos do que disponível.

A escolha por um modelo analítico em detrimento de abordagens por simulação ou híbridas se dá devido à possibilidade de prever rapidamente o tempo de resposta de aplicações. Modelos por simulação, por exemplo, normalmente demoram mais a convergir que modelos analíticos. Combinando uma solução rápida com a elasticidade provida por ambientes em nuvem, é possível projetar sistemas que se ajustam em tempo de execução, garantindo os requisitos de qualidade de serviço da aplicação e otimizando custos.

3. Modelo de Previsão de Desempenho

Nesta seção apresentamos os principais conceitos relacionados ao problema da previsão do tempo de execução de big data em ambientes em nuvem. A Seção 3.2 apresenta o modelo de precedências de tarefas proposto por [Mak and Lundstrom 1990] e suas adaptações para a tarefa de previsão de tempo de execução. Como o foco deste artigo é o estudo de métricas de desempenho no *framework* Spark, a Seção 3.1 apresenta seus principais componentes e como o seu funcionamento impacta nas premissas que devem ser consideradas pelos modelos analíticos a serem aplicados. Por fim, a Seção 3.3 descreve como os logs de execuções prévias de aplicações Spark são utilizados para parametrização do modelo analítico.

3.1. Spark

O Spark é uma plataforma de propósito geral utilizado para o processamento paralelo e distribuído de dados. Projetado para ser tolerante a falhas, é comumente usado em *clusters* de máquinas compostas por processadores multinúcleos, permitindo um ganho de velocidade de processamento devido ao paralelismo. O Spark surgiu como uma alternativa ao modelo *Hadoop MapReduce*, passando a ser amplamente utilizado para o processamento de grandes volumes de dados em nuvem [Zaharia et al. 2010, Databricks 2016].

O Spark permite o processamento de dados em *batch*, ou seja dados previamente armazenados no sistema de arquivos distribuído HDFS (*Hadoop Distributed File System*), ou em tempo real (*streaming*). Neste trabalho focamos no tipo de processamento em *batch*. Uma unidade geral de processamento em *batch* é chamada de aplicação e pode ser composta de um ou múltiplos *jobs*. Os *jobs*, por sua vez, são compostos de operações em dados organizadas em forma de um grafo direcionado acíclico (DAG), podendo ser transformações nos dados ou ações de retorno por agregação. Cada

transformação aplicada aos dados é representada por uma classe de tarefas iguais, chamada de estágio. Os RDDs (*Resilient Distributed Datasets*), abstração de armazenamento em memória do Spark, organizam os dados em blocos com praticamente o mesmo tamanho, conhecidos como partições. Desta forma, os estágios são compostos por uma tarefa igual para cada partição de dados a ser trabalhada. Tais tarefas são completamente independentes entre si e, desta forma, o Spark implementa o paralelismo de dados [Zaharia et al. 2012, Laskowski 2016].

O modelo de execução do Spark baseia-se no DAG de estágios e suas relações de precedência na aplicação, sendo que cada estágio é composto de múltiplas tarefas iguais. Durante a execução dos estágios, cada núcleo de processamento executa apenas uma única tarefa por vez. Como as tarefas de um estágio são independentes e não conhecem o relacionamento entre os estágios, elas são alocadas aos núcleos de acordo com a disponibilidade do recurso e a localização das réplicas de dados. Tarefas restantes são alocadas no momento em que recursos do sistema se tornam disponíveis [Laskowski 2016].

Este trabalho segue as seguintes premissas: 1) núcleos de processamento no *cluster* são representados por um modelo de filas fechado; 2) uma aplicação Spark é composta de um ou mais jobs executados sequencialmente que contém um ou mais estágios cada; 3) estágios podem possuir paralelismo entre si; 4) um estágio é composto por tarefas iguais que executam sobre partições de dados de forma independente, ou seja, não há relação de precedência entre tarefas de um mesmo estágio; 5) o cluster é homogêneo como geralmente ocorre em ambientes de nuvem (p. ex. [Microsoft 2016]), ou seja, cada máquina possui os mesmos recursos, como CPU, memória e disco; 6) modelos de filas fechados são baseados na premissa de que a demanda de tempo das tarefas (neste caso, o tempo dos estágios Spark) em cada recurso (p. ex. núcleo) são exponencialmente distribuídos.

3.2. Modelo de Precedência de Tarefas [Mak and Lundstrom 1990]

O modelo proposto em [Mak and Lundstrom 1990] calcula o tempo esperado da execução de uma aplicação formada por tarefas diferentes que podem ser executadas paralelamente, além de capturar possíveis relações de precedência que possam ocorrer entre elas. Cada aplicação é modelada através de um grafo direcionado acíclico, onde os nós são as tarefas e as arestas representam as relações de precedência. Dois tipos de atrasos podem ocorrer neste tipo de sistema: (i) atrasos decorridos da disputa por recursos computacionais disponíveis, no caso das tarefas que podem ser executadas paralelamente. Este atraso é modelado através de filas e servidores; (ii) atrasos decorridos dos pontos de sincronização das tarefas que possuem relações de precedência. O tempo de execução é estimado por meio de uma extensão da técnica de Análise de Valores Médios (*Mean Value Analysis* ou MVA) [Menasce et al. 2004] para incluir não somente os atrasos por contenção por recursos (tipicamente capturados pelo algoritmo de MVA) mas também os atrasos pela sincronização de tarefas paralelas.

Mais precisamente, atrasos decorridos do nível de paralelismo da aplicação são estimados através da probabilidade de pares de tarefas serem executadas em paralelo (α). São possíveis três níveis de paralelismo: (ii) completo ($\alpha = 1$), onde duas tarefas podem ser executadas ao mesmo tempo, em diferentes recursos; (ii) parcial ($0 < \alpha < 1$), onde tarefas podem ser parcialmente executadas ao mesmo tempo em diferentes recursos; (iii) nulo ($\alpha = 0$), onde pares de tarefas possuem relações de precedência, sendo portanto serializadas (nenhum paralelismo).

O modelo segue um processo iterativo, onde ao final de cada iteração, os tempos das tarefas são agregados de acordo com as respectivas probabilidades de execução em paralelo. O grafo com as relações de precedência entre as tarefas é reduzido para refletir estas agregações. No início da primeira iteração, o tempo estimado do sistema (R_s) é definido como zero. Ao final da iteração, uma nova previsão (R_p) é calculada. O critério de parada do algoritmo e dado pela diferença entre duas iterações consecutivas. Caso $R_s - R_p$ seja menor do que uma tolerância L, o algoritmo é finalizado. Caso contrário, R_s recebe o valor da previsão naquela iteração dado por R_p e o algoritmo inicia a próxima iteração. Mais detalhes sobre o modelo estão disponíveis no artigo original [Mak and Lundstrom 1990].

Para aplicar o modelo de previsão de tempo de resposta em aplicações Spark, considera-se o grafo de precedência entre estágios do modelo de programação da plataforma como sendo o grafo de precedência de tarefas do modelo analítico de previsão. Neste caso, as probabilidades de execução em paralelo são calculadas entre pares de estágios e os tempos de resposta dos nós do grafo de precedência do modelo analítico são os tempos de resposta dos estágios da aplicação Spark. Estas informações são extraídas dos logs de execuções prévias e usadas para estimar a etapa de sincronização entre estágios Spark. Caso não existam logs prévios para a configuração de *cluster* desejada, deve-se fazer o processo de extrapolação, descrito na seção 3.3.

3.3. Parametrização do Modelo

Assumimos como entrada do modelo os tempos de início e fim da execução dos estágios (tarefas) que compõem a aplicação. Assim, o foco do trabalho está na estimativa do tempo de resposta da aplicação frente às relações de precedência entre os vários estágios que a compõem. Os tempos de início e fim de cada estágio são extraídos dos logs de execuções anteriores da aplicação em análise. Com base nos tempos calculados, o modelo estima as probabilidades de execução paralela (total/parcial/nula) entre cada par de tarefas assim como o tempo total de resposta da aplicação.

Neste artigo são considerados dois contextos de aplicação do modelo. O primeiro, definido como previsão, ocorre quando existem logs de execuções prévias para a mesma aplicação (ou uma aplicação com perfil de paralelismo semelhante) à que se quer prever, com recursos parecidos e mesma quantidade de núcleos. O segundo, definido como extrapolação, ocorre quando são utilizados logs de execuções prévias da mesma aplicação (ou aplicação semelhante) mas em uma infraestrutura (em termos de recursos disponíveis) diferente, mais precisamente diferentes quantidades de núcleos de processamento. Desta forma, é possível parametrizar o modelo com as informações presentes em um histórico de execuções de diferentes perfis de aplicações Spark. A previsão ou extrapolação do tempo de execução de uma nova tarefa no sistema pode ser calculada tendo como base uma aplicação executada anteriormente no sistema. Para tal, a mesma deve possuir perfil similar ao da aplicação sendo executada no momento, como por exemplo, mesmo grau de paralelismo ou de quantidade de dados sendo manipulada.

Os logs do Spark são estruturados por eventos. Considerando o escopo deste trabalho, os principais eventos a serem analisados são os eventos de submissão do primeiro *Job* e fim da aplicação ($SparkListenerJobStart$ e $SparkListenerApplicationEnd$), os eventos de submissão e finalização dos estágios ($SparkListenerStageSubmitted$

e $SparkListenerStageCompleted$) e os eventos de início e fim das tarefas ($SparkListenerTaskStart$ e $SparkListenerTaskEnd$). O tempo de inicialização do Spark, ou seja, o tempo necessário para a reserva de recursos de cada unidade de execução e o carregamento do contexto do programa (p. ex. informações sobre como acessar o *cluster*) não é considerado na parametrização do modelo. Assim, o evento de submissão do primeiro *Job* é usado como o tempo de início da aplicação. Para aplicações intensivas em dados, o tempo de inicialização do *cluster* pode se tornar desprezível, tendo em vista que seria muito menor que o tempo de execução da aplicação. Além disso, pode-se somar a média do tempo de inicialização ao tempo de previsão final ($SparkListenerApplicationStart - SparkListenerJobStart$, tempo de início da aplicação menos o tempo de início do primeiro job) para obter um tempo que compreende tanto a aplicação quanto o tempo da inicialização do *cluster*.

No caso da previsão do tempo de resposta de uma determinada aplicação, onde a configuração do sistema permanece a mesma de execuções anteriores, a principal informação a ser obtida dos logs é o tempo de início e fim de cada um dos estágios da aplicação, presentes nos eventos *SparkListenerStageSubmitted* e *SparkListenerStageCompleted* da aplicação Spark. Assim, é possível calcular tanto o tempo de execução quanto a probabilidade da execução em paralelo dos pares de estágios. Estes valores são utilizados como entradas do modelo de previsão.

No caso da extrapolação, considera-se a mesma quantidade de partições e, consequentemente, a mesma quantidade de tarefas dos estágios presentes nos logs de referência. Como cada tarefa executa em um único núcleo, os valores de tempo de resposta são mantidos. Como deseja-se estimar o tempo de resposta da aplicação executando em um número de núcleos diferente dos logs de referência, é preciso simular a alocação de tarefas para os núcleos disponíveis. Considera-se aqui um esquema de alocação que segue a política FIFO, sendo que cada tarefa é alocada a um núcleo, criando, portanto, uma fila de tarefas a serem executadas em cada núcleo. A partir dos tempos de início e fim de cada tarefa do estágio, o tempo de execução de cada tarefa é calculado. O somatório dos tempos das tarefas das filas é considerado como a demanda de tempo do estágio pelos núcleos. Desta forma, tempo de execução total do núcleo mais lento é considerado como o tempo de execução do estágio na nova configuração (nova quantidade de núcleos disponível). A partir da extrapolação dos tempos de execução dos estágios, as probabilidades de execução em paralelo são calculadas. Apesar de simples e diferente do esquema padrão de alocação de tarefas do Spark, que tende a priorizar núcleos próximos das réplicas da partição, este método de alocação se mostrou bastante eficiente, conforme será discutido na Seção 4.

A previsão tende a usar uma informação mais acurada sobre o tempo do estágio do que a extrapolação. Enquanto a previsão usa o tempo de início e término de um estágio diretamente, a extrapolação usa o tempo de início e término das tarefas iguais dentro de um estágio para obter o tempo total do estágio, ignorando possíveis atrasos ocorrido entre o término de uma tarefa e início da tarefa seguinte. No caso da extrapolação, o tempo de execução de cada estagio é aproximado através da simulação da execução de cada tarefa do estágio distribuindo-as de acordo com a quantidade de núcleos que se deseja extrapolar. A eficácia do método é discutida na Seção 4.

4. Resultados Experimentais

Esta seção apresenta os cenários experimentais e discute os resultados. Nesta avaliação, exploramos duas classes de aplicações: consultas interativas, através da execução das consultas Q26 e Q52 do TPC-DS[1][Nambiar and Poess 2006, Poess et al. 2007] e as cargas de trabalho iterativas, através das execuções dos algoritmos K-Means, SVM e Logistic Regression [Zaki et al. 2014]. Os experimentos foram executados na Microsoft Azure.

4.1. Descrição dos Cenários

Com o objetivo de cobrir os principais casos de uso do Spark em nuvem, foram selecionados algoritmos iterativos comuns em processos de aprendizado de máquina, bem como consultas interativas populares atualmente no Spark. Especificamente, foram escolhidos os algoritmos K-Means, SVM e Logistic Regression, muito usados em aplicações de *big data* (i.e., *big data analytics*) e disponíveis na biblioteca de aprendizado de máquina do Spark (MLlib) e as consultas Q26 e Q52 do TPC-DS, disponíveis para SparkSQL. De acordo com uma pesquisa da Databricks [Databricks 2016], ambos MLlib e SparkSQL estão entre os componentes mais usados do Spark, com 67% e 38% de crescimento, respectivamente, de 2015 para 2016. Portanto, foram planejados sete cenários experimentais.

O Spark foi instalado em três tipos diferentes de máquinas virtuais (VMs), considerando diferentes quantidades de nós e núcleos de processamento. Para os experimentos, foram utilizadas as versões 1.6.2 e a 2.1.0 do Spark. As VMs são classificadas em instâncias de propósito geral e instâncias otimizadas em CPU e memória. O principal objetivo é avaliar a acurácia do modelo de previsão e extrapolação em diferentes recursos computacionais. A Tabela 1 lista as características dos três tipos de máquina virtual usados para os nós *workers* (máquinas que armazenam e processam dados em paralelo).

Tabela 1. Configuração das máquinas virtuais

Tipo da VM	Num. Núcleos	Núcleos Exec.	RAM Exec.	RAM Driver	RAM VM	Disco local persistente	Tipo da VM
D12v2	4	2	2GB	4GB	28GB	200GB SSD	CPU/Memória
A3	4	2	2GB	4GB	7GB	250GB disco	Propósito Geral
D4v2	8	4	10GB	8GB	28GB	400GB SDD	Propósito Geral

O *cluster* foi configurado com dois nós *master* (nós que coordenam a execução e mantém a localização das réplicas de dados) sobre VMs D12v2 em todos os cenários. Para os cenários com nós *workers* de VMs A3 ou D12v2, foram instalados Ubuntu 14.04 e Spark 1.6.2. Os cenários com *workers* em VMs D4v2 consistem de Ubuntu 16.04 e Spark 2.1.0. A Tabela 2 detalha os cenários, explicitando o algoritmo, tipo da VM, número de nós e núcleos de processamento e carga de trabalho em termos de volume. Nos cenários 1 a 4, carga sintética com o ferramental oficial (tpc.org). De 5 a 7, a partir de duplicações do *dataset* HIGGS (https://archive.ics.uci.edu/ml/datasets/HIGGS).

Tabela 2. Descrição dos Cenários

#	Aplicação	VM	Nós	Núcleos 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 13	até 4	500
4	TPCDS Q52	A3	3 a 13	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

[1]TPC-DS é um benchmark proposto pelo TPC para cobrir aspectos de sistemas de suporte a decisões.

Cada experimento foi executado 10 vezes. Foram feitas previsões para as mesmas configurações usando logs de execução prévias de cada cenário. Estes resultados são chamados simplesmente de previsão. Foram também realizadas extrapolações para *clusters* com quantidades de núcleos diferentes da usada em logs de execuções prévias (logs de referência). Para as extrapolações, três diferentes casos para logs de referência (indicados nas tabelas pelo rótulo "REF.") foram validados em todos os cenários: i) logs com a menor quantidade de núcleos disponíveis para o cenário (extrapolação A), ii) logs com uma quantidade intermediária de núcleos disponíveis para o cenário (extrapolação B), iii) logs com a maior quantidade de núcleos disponíveis para o cenário (extrapolação C). Por exemplo, para o cenário 1, a menor quantidade de núcleos com logs disponíveis é 12 núcleos (3 nós, 4 núcleos por nó), enquanto que no caso intermediário o total de núcleos é 32 (8 nós, 4 núcleos por nó) e a maior é 52 núcleos (13 nós, 4 núcleos por nó). Enfatiza-se que a razão para experimentar com três logs de referência diferentes não é para encontrar qual extrapolação tem melhor acurácia, mas sim demonstrar que o modelo produz bons resultados mesmo quando os logs disponíveis são para *clusters* bem diferentes (menores ou maiores) que o em análise. Para medir a acurácia do modelo de previsão de tempo de resposta, foi usada a métrica do erro relativo (ϵ), com a média do tempo medido no *cluster* real e o valor da previsão (ou extrapolação) para cada cenário, ou seja: $\varepsilon = \frac{T_{\text{real}} - T_{\text{modelo}}}{T_{\text{real}}}$.

4.2. Cenários 1 e 2 (VMs D12v2)

Esta seção apresenta os resultados das previsões e extrapolações das consultas Q26 e Q52 nas VMs D12v2 para diversos tamanhos de *cluster* (cenários 1 e 2), com nós de 4 núcleos em um *cluster* homogêneo. Nos experimentos de extrapolação, as três configurações de logs de referência consideradas foram: 3 nós e 12 núcleos (extrapolação A), 8 nós e 32 núcleos (extrapolação B) e 13 nós e 52 núcleos (extrapolação C). A Tabela 3 apresenta os erros para cada experimento: o maior erro observado está em negrito, e o menor com preenchimento cinza. Os erros nos cenários 1 e 2 são sumarizados na Tabela 4, que apresenta erros médios, mínimos, máximos e desvio padrão para as previsões e extrapolações.

Tabela 3. Previsão e Extrapolação nos cenários 1 e 2 (cluster homogêneo)

núcleos	real (s)	prev. (s)	erro	ext. A (s)	erro A	ext. B (s)	erro B	ext. C (s)	erro C
Consulta 26 do TPC-DS									
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	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.
Consulta 52 do TPC-DS									
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 4. Sumarização dos erros dos cenários 1 e 2 (cluster homogêneo)

	Erro para consulta Q26				Erro para consulta Q52			
	previsão	ext. A	ext. B	ext. C	previsão	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%

No cenário 1, observa-se um erro médio na previsão de apenas 4.3%. No geral este erro se manteve próximo para todos os experimentos, dado o baixo desvio padrão (1.3%) e a pequena diferença entre valores mínimo (1.7%) e máximo (6.3%). Para extrapolações, os erros médios e máximos são um pouco maiores, o que era esperado já que o modelo tem como entradas parâmetros de execuções prévias em configurações diferentes da corrente. Estas diferenças refletem também em uma maior variabilidade dos erros (maior desvio padrão). Nas três extrapolações, a que usou o menor número de núcleos como referência (extrapolação A) foi a que levou aos maiores erros. Entretanto, ressalta-se que mesmo o maior erro observado – 18,2% – ainda é bastante razoável do ponto de vista prático, principalmente considerando a simplicidade do modelo analítico [Menasce et al. 2004].

Para o cenário 2, os erros no geral tendem a ser bem inferiores (exceto para a extrapolação C). Para a previsão, os erros variaram de 0.3% a 1.1%, refletindo uma excelente acurácia. Os desvios padrões dos erros também são bem menores. Estes resultados mais consistentes, se comparados ao do cenário 1, são devido a uma menor variabilidade nos tempos medidos para execuções prévias da consulta Q52. Isto afeta consideravelmente a precisão da extrapolação e da previsão. Note que mesmo para o pior caso da extrapolação, que ocorre quando os logs de referência têm configurações maiores que a em análise, os erros são ainda razoavelmente baixos (máximo de 12.7%).

4.3. Cenários 3 e 4 (VMs A3)

Diferentemente dos cenários 1 e 2, nos cenários 3 e 4 não há uma homogeneidade na quantidade de núcleos de processamento por nó no *cluster*. Para estes cenários, foi simulada uma situação de contenção de recursos, como se alguns núcleos e memória RAM já estivessem em utilização para outros processos, não podendo ser alocados pelo Spark para as aplicações a serem executadas. As mesmas alternativas (extrapolações A, B e C) foram mantidas, com i) a extrapolação A usando logs de referência para uma configuração de *cluster* com 4 nós e 10 núcleos no total, ii) a extrapolação B considerando 9 nós e 30 núcleos no total, e iii) a extrapolação C considerando 13 nós e 48 núcleos no total. A Tabela 5 apresenta os resultados para cada configuração testada para os cenários 3 e 4, e a Tabela 6 sumariza estes resultados, apresentado erros médios, mínimo e máximo (assim como desvio padrão) para previsão e extrapolação, para cada cenário.

Assim como observado para os cenários 1 e 2, os erros médios e máximos no cenário 3 tendem a ser maiores que no cenário 4. O mesmo vale para o desvio padrão dos erros. A razão é a mesma discutida na seção anterior já que cenários 1 e 3 (2 e 4) correspondem a execuções da mesma consulta Q26 (Q52). Isto é, a menor variabilidade dos tempos de execuções da Q52 levam a resultados mais consistentes do modelo. Nota-se também que erros médios e principalmente máximo tendem a ser maiores nos cenários 3 e 4, quando comparados aos correspondentes 1 e 2. Estes resultados refletem a característica mais imprevisível de um *cluster* com contenção de recursos, situação simulada

Tabela 5. Previsão e extrapolação nos cenários 3 e 4 (contenção de recursos)

nós	núcleos	real (s)	prev. (s)	error	ext. A (s)	error A	ext. B (s)	error B	ext. C (s)	error C
					Consulta 26 do TPC-DS					
4	10	1718.5	1763.6	-2.6%	REF.	REF.	1788.5	-4.1%	1753.7	-2.0%
4	12	1632.3	1674.5	-2.6%	1517.3	7.0%	1518.9	6.9%	1463	10.4%
5	14	1381.1	1414	-2.4%	1321.8	4.3%	1305.7	5.5%	1262.9	8.6%
5	16	1214	1243.3	-2.4%	1171.3	3.5%	1163.5	4.2%	1119.1	7.8%
6	18	1069.5	1099.8	-2.8%	1039.7	2.8%	1036.4	3.1%	995.6	6.9%
6	20	1036.2	1064.2	-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	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	-7.6%
9	32	565.6	589	-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	-4.7%	562.6	-0.2%
10	36	511.2	532.4	1.6%	565.2	-10.6%	565.3	-10.6%	536	-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	-21.1%	486.1	-19.6%	461.4	**-13.6%**
13	46	383.2	405.6	-1.1%	472	-23.2%	455.1	-18.8%	435.5	**-13.6%**
13	48	367.2	387.5	**-5.5%**	459	**-25.0%**	452.1	**-23.1%**	REF.	REF.
					Consulta 52 do TPC-DS					
4	10	1270.6	1276.4	-0.5%	REF.	REF.	1354.5	-6.6%	1370.8	-7.9%
4	12	1067.4	1072.5	-0.5%	1075.8	-0.8%	1141.7	-7.0%	1152	-7.9%
5	14	918.9	924	-0.6%	929.7	-1.2%	986.8	-7.4%	993	-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	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	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	-4.6%
9	32	428	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	-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	**-1.5%**	303.7	-1.1%	321.2	-6.9%	REF.	REF.

nos cenários 3 e 4. Entretanto, mesmo nestes casos, os erros ainda são bem razoáveis demonstrando que mesmo em situações mais complexas o modelo tem acurácia satisfatória.

Tabela 6. Sumarização dos erros dos cenários 3 e 4 (contenção de recursos)

	Erro para consulta Q26				Erro para consulta Q52			
	previsão	ext. A	ext. B	ext. C	previsão	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%

4.4. Cenários 5, 6 e 7 (VMs D4v2)

Estes cenários apresentam os resultados de alguns dos algoritmos mais comuns em processos de aprendizado de máquina, o K-means, o Logistic Regression e o SVM. São consideradas duas configurações de *cluster*, uma com 3 nós e 24 núcleos e outra com 6 nós 48 núcleos. Os demais recursos são homogêneos. Adicionalmente, consideramos quantidades diversas de tamanhos de dados processados: 8GB, 48GB e 96GB. Note que nestes cenários, dados os logs disponíveis, é possível realizar apenas duas extrapolações: de 24 para 48 núcleos e vice-versa. Por esta razão, diferentemente das tabelas anteriores, para estes cenários as duas quantidades de núcleos dos quais existem logs prévios são dispostas lado a lado e organizadas em blocos por algoritmo. Mais ainda, a coluna extrap.

refere-se aos resultados para a extrapolação correspondente, mantendo o mesmo conjunto de dados. Por exemplo, para 24 núcleos, a coluna extras corresponde aos resultados de extrapolação usando como referência execuções prévias em 48 núcleos. Para 48 núcleos, a coluna mostra resultados usando como referência os logs de 24 núcleos.

Tabela 7. Previsão e extrapolação nos cenários 5, 6 e 7 (cluster homogêneo)

	24 núcleos					48 núcleos				
data	real (s)	prev. (s)	erro	extrap. (s)	erro A	real	prev. (s)	erro	extrap. (s)	erro A
K-means										
8GB	83.4	81.9	1.8%	73.8	11.5%	75.3	74	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%	1148.3	-35.6%	575.1	572.9	0.4%	523.0	9.1%
Logistic Regression										
8GB	164.7	159.5	3.2%	160.7	2.4%	166.6	161	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	1418.9	1414.1	0.3%	2367.6	-66.9%	1200.8	1192.6	0.7%	766.3	36.2%
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	1353.5	1349.6	0.3%	1216.4	10.1%	636.0	631.1	0.8%	758.7	-19.3%

Note que para estes cenários, embora no geral os erros estejam razoavelmente baixos, há casos de erros mais elevados (entre 41.3% e 89.8%). Estes erros maiores ocorrem para extrapolações para Logistic Regression com 96 GB de dados e para o SVM com 48 GB de dados. Porém, note que neste casos os tempos de execução do algoritmo para 24 e 48 núcleos são bem próximos. O tempo de execução do SVM com 48GB no cluster real com 24 núcleos foi 341,5 segundos, em média. Já para o cluster com 48 núcleos, este tempo foi 358 segundos. Dada a variação de recursos disponíveis, esperaríamos uma variação maior dos tempos de execução, mais consistente com o previsto pelo modelo. Possíveis razões que explicam o observado são: (i) congestionamento de rede nos dispositivos físicos das instâncias; ii) possível gargalo pela quantidade de memória disponível para as operações executadas; iii) falha nas réplicas de dados; iv) configuração incorreta dos *data nodes*, mantendo dados apenas em 3 dos 6 nós disponíveis. Nenhuma destas situações é capturada pelo modelo. A despeito dos poucos casos de erros elevados, os vários cenários explorados demonstram, em sua maioria, erros muito satisfatórios, principalmente considerando a aplicabilidade prática do modelo. Por exemplo, no K-Means, a extrapolação de 24 núcleos para 48 núcleos levou a um erro absoluto máximo de apenas 9,1% com a carga de 96GB. Para o mesmo caso, o erro absoluto mínimo foi de 0,1%.

4.5. Discussão

As seções anteriores apresentaram resultados quanto à precisão do modelo para previsão e extrapolação do tempo de resposta. Um outro ponto importante é o tempo gasto para resolver o modelo, que segue um processo iterativo. Em todos os experimentos executados, o tempo de execucao para o modelo foi pequeno. Especificamente, executando em um computador com processador dual-core Intel Core i5 processor de 2.4GHz com 3MB cache L3 compartilhada, 8GB de RAM 1600MHz DDR3L e um sistema operacional Mac OS X El Captain 10.11.6, o tempo de execução do modelo, no cenário mais custoso, ficou abaixo de 60 milissegundos. Portanto, é possível utilizar o modelo em ambientes reais e prover previsões rápidas que possibilitem o ajuste dinâmico de recursos através da elasticidade e flexibilidade dos ambientes em nuvem.

Durante a fase experimental deste trabalho, a extrapolação foi proposta com a finalidade de validar o modelo com logs prévios bastante diferentes (distantes) do caso ao qual deseja-se estimar. Esta estratégia foi adotada para demonstrar que nos casos onde os logs de referencia possuem quantidade muito menor (ou maior) de núcleos que a configuração a qual deseja-se estimar o tempo de resposta, o erro da estimativa não é muito diferente daqueles obtidos com logs prévios com quantidade de núcleos mais próxima da configuração desejada. Portanto, a alternativa de validar logs prévios com uma quantidade média de núcleos foi importante na comparação com os casos extremos.

Num cenário de aplicação real, o serviço de previsão de tempo de resposta deve buscar sempre dados históricos o mais próximo possível do qual se deseja extrapolar. Desta forma, a chance de se obter estimativas de tempo de resposta mais próximas do valor real é maior. Um outro passo importante ao explorar cenários reais é o processamento prévio dos logs do Spark, facilitando a execução do modelo. Apesar de não ilustrado nas tabelas, o tempo gasto na leitura de processamento de logs históricos é consideravelmente maior que o tempo gasto executando o modelo. Portanto, uma boa alternativa seria processar previamente os logs Spark, deixando apenas a execução em tempo real da simulação de tarefas, criação das entradas e execução do modelo.

5. Conclusões e Trabalhos Futuros

Neste artigo, avaliamos a acurácia do modelo analítico de precedência de tarefas apresentando em [Mak and Lundstrom 1990] usando logs de execuções prévias de aplicações Spark em nuvem. Uma extensão baseada em simulação simples para extrapolar a previsão para quantidades de núcleos diferentes dos logs históricos foi proposta e avaliada. Por fim, discutimos a aplicabilidade do modelo em ambientes reais. Foram avaliados sete cenários, compostos por diferentes máquinas virtuais, variações de recursos e perfis de aplicações que abragem as consultas interativas e algoritmos iterativos.

Os resultados se mostraram promissores, com erros consideravelmente baixos para a previsão do tempo de execução para cargas de trabalho nos mesmos recursos dos logs das execuções prévias. Adicionalmente, os erros também se mostraram relativamente baixos na extrapolação para uma quantidade de núcleos diferente daquela explorada nos logs prévios. Para as previsões nos sete cenários, obteve-se um erro médio de 1,87% considerando todos os 79 previsões executados. Já, para a extrapolação, apesar de alguns resultados inconsistentes, obteve-se um erro médio de 7,90% considerando todas as três alternativas com um total de 189 extrapolações. Apesar da premissa de demanda exponencial, observa-se que esta característica (ou a falta) nas tarefas influenciou pouco na precisão da previsão. Uma maior incidência de erros para mais são justificáveis visto que os logs não diferenciam execução, disparo e finalização de tarefas na marcação de tempos.

Como trabalhos futuros, seria interessante explorar outros algoritmos paralelos (p.ex outros da MLlib), novas consultas interativas e fluxos completos de processamento de dados, compostos por vários algoritmos ou consultas, assim como cenários com *jobs* paralelos e aplicações concorrentes no mesmo *cluster*. Também, cabe avaliar o modelo para adicionar ou remover recursos de forma proativa ao cluster respeitando a qualidade de serviço e provendo maior eficiência. É também interessante avaliar o comportamento do modelo para os algoritmos selecionados com diferentes entradas, bem como comparar os resultados com outros modelos como, por exemplo, o Starfish [Herodotou et al. 2011].

Referências

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

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

Chen, C. P. and 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. In *International Workshop on Timed Petri Nets, Italy, 1985*. p. 136–143.

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

Herodotou, H. et al. (2011). Starfish: A self-tuning system for big data analytics. In *Proceedings of the 5th CIDR*, pages 261–272.

Laskowski, J. (2016). Mastering Apache Spark. Disponível em: https://gitbook.com/book /jaceklaskowski/mastering-apache-spark. Acessado em: 28/06/2017.

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

Menasce, D. A., Almeida, V. A., Dowdy, L. W., and Dowdy, L. (2004). *Performance by design: computer capacity planning by example*. Prentice Hall Professional.

Microsoft (2016). What is PaaS? Disponível em: https://azure.microsoft.com/en-us/overview/what-is-paas/. Acessado em: 15/06/2017.

Nambiar, R. O. and Poess, M. (2006). The making of tpc-ds. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*. p. 1049–1058.

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

Poess, M., Nambiar, R. O., and Walrath, D. (2007). Why you should run tpc-ds: A workload analysis. In *Proceedings of the 33rd VLDB*. p. 1138–1149.

Popescu, A. D. (2015). *Runtime Prediction for Scale-Out Data Analytics*. PhD thesis, École Polytechnique Fédérale de Lausanne.

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

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

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

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

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX HotCloud*.

Zaharia, M. et al. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX NSDI*.

Zaki, M. J., Meira Jr, W., and Meira, W. (2014). *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press.