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

Bruno Vinícius Ávila Machado

BatchKeeper: Processamento de lotes com controle de fluxos de execução

Belo Horizonte
2020

Bruno Vinícius Ávila Machado

BatchKeeper: Processamento de lotes com controle de fluxos de execução

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.

Orientador: Ítalo Fernando Scotá Cunha
Coorientador: Dorgival Olavo Guedes Neto

Belo Horizonte
2020

Machado, Bruno Vinícius Ávila.

M149b Batchkeeper: processamento de lotes com controle de fluxos de execução [manuscrito] / Bruno Vinícius Ávila Machado. — Belo Horizonte, 2020.
84 f. il.; 29 cm.

Orientador: Ítalo Fernando Scotá Cunha.
Coorientador: Dorgival Olavo Guedes Neto
Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação
Referências: f.73-79.

1. Computação – Teses. 2. Processamento em lote. informacional – Teses. 3. Programação de fluxos de execução de dados – Teses. I. Cunha, Ítalo Fernando Scotá.
II. Guedes Neto, Dorgival Olavo III .
Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Computação. IV.Título.

CDU 519.6*34 (043)

Ficha Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa CRB 6/1510 Universidade Federal de Minas Gerais - ICEx

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

FOLHA DE APROVAÇÃO

BatchKeeper: Processamento de Lotes com Controle de Fluxos de Execução

BRUNO VINÍCIUS ÁVILA MACHADO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. ÍTALO FERNANDO SCOTÁ CUNHA - Orientador
Departamento de Ciência da Computação - UFMG

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

PROF. DANIEL FERNANDES MACEDO
Departamento de Ciência da Computação - UFMG

PROF. MARCOS AUGUSTO MENEZES VIEIRA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 26 de Março de 2020.

Dedico este trabalho à minha amada esposa, pai, mãe, irmãs e cunhados

Agradecimentos

Agradeço à minha família por todo o suporte que me deram durante toda esta caminhada. Agradeço a minha esposa por entender minhas ausências para que fosse possível completar esse objetivo. Aos meus pais que me fizeram entender a importância do estudo e as minhas irmãs que sempre me incentivaram a seguir em frente.

Agradeço ao professor Dorgival pela ajuda na busca da contribuição científica do meu trabalho de mestrado. Agradeço também ao professor Ítalo pela grande oportunidade que me deu na iniciação científica e em meu mestrado. Sua grande disponibilidade em ajudar-me em todas as etapas deste trabalho fez com que eu conseguisse fazer progresso durante todo o período da pesquisa. O professor Ítalo foi o principal ator em meu desenvolvimento intelectual desde a época da minha graduação.

"*Pela manhã semeia a tua semente, e à tarde não retires a tua mão, porque tu não sabes qual prosperará: se esta, se aquela, ou se ambas serão boas...*"
(Eclesiastes 11:6)

Resumo

Processamentos em lote tem sido utilizado por muitas décadas na computação. Processamentos em lotes podem ser apenas uma parte de um fluxo de negócio maior. Nesse contexto, visibilidade e controle sobre os processamentos são importantes. Esses objetivos podem ser alcançados através de integrações entre sistemas de processamento em lote, sistemas de monitoramento e programação de fluxos de execução de processamentos; essas funcionalidades podem reduzir custos, intervenções humanas e atrasos em fluxos de negócio. Além disso, o processamento de alguns lotes podem ter diferentes prioridades ou ter prazos máximos de execução.

Neste trabalho, projetamos e desenvolvemos o BatchKeeper: um sistema para processamento em lote que suporta controle e automatização de fluxos de execuções de lotes através de um módulo de workflow, escalonamento com suporte a prazo limite para execução de lotes e integração com sistemas de coleta e monitoramento de informações por múltiplas fontes.

Utilizamos dados históricos de execuções de lotes de uma empresa do mercado financeiro para avaliar o algoritmo previsor de tempo de execução de lotes. O previsor usa regressão para encontrar uma distribuição estatística que aproxime os dados históricos de tempo de execução de tarefas de cada lote. O previsor usa então um percentil configurável das distribuições estatísticas para estimar o término da execução de lotes em execução. O previsor pode prever corretamente se execuções vão ser finalizadas antes de seu prazo máximo de execução em 66% dos lotes mesmo em um contexto com alta variação em tempo de execução de lotes devido ao compartilhamento de recursos computacionais por tarefas concorrentes.

O BatchKeeper é uma solução genérica aplicável em vários cenários. Aplicamos o BatchKeeper em uma empresa que executa centenas de lotes mensalmente com alta taxa de intervenção humana em lotes de longa duração. Nós mostramos que o BatchKeeper foi capaz de automatizar várias tarefas e reduzir intervenções humanas.

Palavras-chave: Processamento em Lote. ECA. Monitoramento. Prazo de execução.

Abstract

Batch processing has been used for several decades for scientific computing by companies and governments. Batch processing can be just one component of an extensive business process. In this context, visibility and control over the computation are important. These objectives can be achieved through the integration of batch processing systems with monitoring frameworks and a programmable execution workflow; these objectives can decrease costs, human interventions, and delays, all of which are key for businesses. Furthermore, the processing of some batches can have different priorities or have deadlines, which requires advanced prediction and resource scheduling.

In this work, we design and evaluate BatchKeeper: a system for batch processing that supports automated control of processing through a workflow module, scheduling with support for deadlines, and collecting monitoring information from multiple sources.

We utilized historical batch execution data from a financial company to evaluate the batch execution time estimator. The estimator uses regression to find a statistical distribution that approximates the historical task runtimes of each batch, then uses a configurable percentile of the runtime distribution to estimate the finish time of executing batches. The estimator can correctly predict whether executions will finish before their deadline for 66% of the batches even in a context where the machines are shared by other concurrent tasks, which leads to long, highly-variable runtimes.

BatchKeeper is a generic solution applicable in several scenarios. We applied BatchKeeper in a company that executes hundreds of batches each month, which had a high human intervention rate on batches with long runtimes. We show that BatchKeeper was capable to automate several tasks and reduce human intervention.

Keywords: Batch Processing. ECA. Monitoring. Deadline.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Processamentos em lote são execuções de sequências de tarefas por computadores de forma não interativa. Uma *tarefa* define um programa a ser executado, parâmetros e entradas. O processamento em lote é comumente utilizado em empresas de diversos setores desde a década de 1960 [62]. Além de processamentos em lote serem muito utilizados, estes são críticos para as operações de empresas e governos, como na contagem de votos de eleições, geração de relatórios para business intelligence e processamentos de dados financeiros [35].

Também pode-se observar que o surgimento e aprimoramento da computação em nuvem trouxe maior flexibilidade e menores custos para empresas que utilizam processamentos em lote [39; 40]. Estes benefícios da computação em nuvem são oriundos do escalonamento automático de recursos computacionais, compartilhamento de CPU, memória e disco entre clientes de centros de processamento de dados e da flexibilidade na administração de grandes quantidades de máquinas [20]. Nas últimas décadas algumas empresas passaram a utilizar grandes grupos de máquinas de menor capacidade de processamento para realizar processamento distribuído [12].

Outros modelos de processamento surgiram, como o processamento de fluxo (*streaming*) e o MapReduce. Além disso, ferramentas para efetuar processamentos de grandes conjuntos de dados, como o Spark, Hadoop e outros trouxeram paradigmas flexíveis e facilidade para a implantação e gerenciamento de processamento, permitindo ampla adoção [23; 6]. Também foram desenvolvidos arcabouços para auxiliar no monitoramento e provisionamento necessários de recursos computacionais para realização de processamentos de lotes ou fluxos *(streamings)* [26; 59; 47; 60; 48; 54].

Considere uma empresa que processe e exiba, para os seus usuários, imagens de satélite do planeta Terra. São mostrados nomes de ruas e nomes de locais importantes em cima de suas respectivas ocorrências nas imagens. Estas imagens devem ser atualizadas no website da empresa a cada mês. A figura 1.1 mostra um fluxograma com as etapas de uma possível implementação do sistema de processamento de dados. Como a quantidade de imagens é muito grande, estas são divididas em grupos para processamento. Grupos de imagens referentes a cidades e estados mais populosos devem possuir maior prioridade para processamento. Se houver alguma falha no processamento das imagens, como o não mapeamento de algum nome de rua para alguma imagem, um algoritmo alternativo para

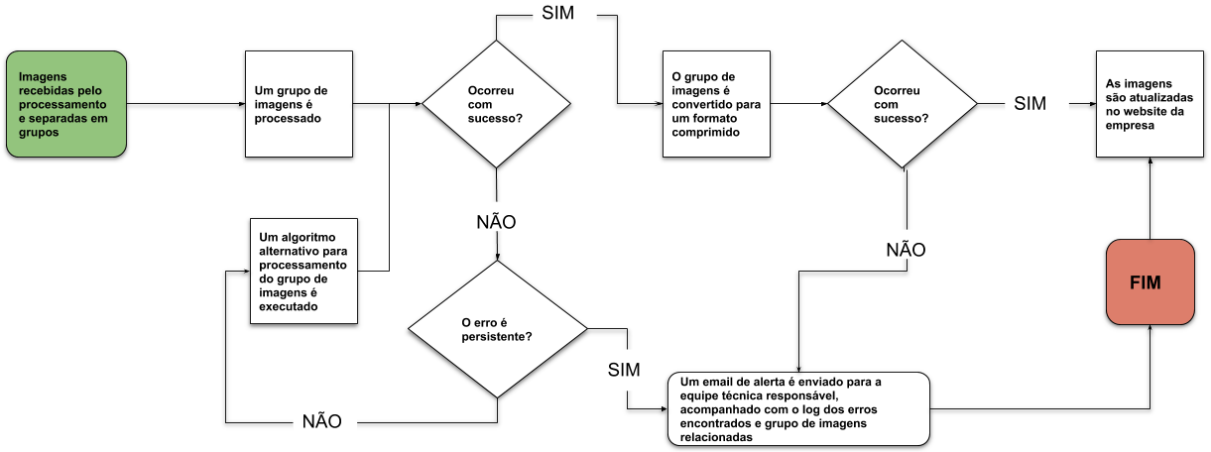


Figura 1.1: Exemplo ilustrativo de cenário de processamento em lote

o processamento da imagem em questão deverá ser executado. Se o processamento de um grupo de imagens ocorrer com sucesso, estas imagens deverão ser convertidas em formato comprimido para exibição na aplicação, substituindo as imagens antigas pelas mais novas. Se alguma falha no reconhecimento de ruas e locais em imagens persistir, o setor responsável pelos processamentos deverá ser avisado, via e-mail, com o log de erros disparados acompanhados das imagens cujo processamento falhou. Cada processamento deverá ter um prazo máximo de execução. Caso esse prazo seja atingido, a equipe responsável pelos processamentos deverá ser avisada, via e-mail com o log do processamento corrente, para efetuar uma análise sobre as causas dessa demora.

O parágrafo anterior descreve um exemplo de processamento de lotes. Assim como no exemplo, é comum a existência de um fluxo de controle em processamentos de lotes. Esses fluxos de controle determinam as ações a serem executadas na ocorrência de falhas ou sucesso dos processamentos. O gerenciamento automatizado de fluxos de controle relacionados aos processamentos em lote é importante para aumentar a produtividade e eficiência de todo o processo que envolve o processamento em lote.

1.1 Controle de fluxo de execuções

Durante processamentos de lotes podem ocorrer erros de software, hardware ou inconsistências de dados, sendo necessário que a continuidade dos processamentos seja devidamente monitorada para que falhas sejam identificadas e tratadas rapidamente. Para que a identificação e resolução de falhas seja ágil, é importante minimizar a quantidade

de intervenções humanas, já que elas adicionam atrasos aos processos e a possibilidade de ocorrências de erros humanos. É desejável, portanto, programar as possíveis tomadas de decisões para casos de interrupções ou falhas em processamentos. Ferramentas de controle e automatização de fluxos são comumente chamadas de ferramentas de workflow [24]. Um exemplo de interface para a configuração de um workflow é a especificação de regras ECA (Evento - Condição - Ação), onde, de acordo com eventos disparados e condições satisfeitas, ações predeterminadas são executadas automaticamente [35].

A integração entre o processamento em lote e ferramentas de workflow vem sendo alvo de trabalhos desde 2007 com o objetivo de administrar e automatizar fluxos de processamentos [35; 55]. O exemplo de processamento de imagens anterior mostra como o processamento em lote pode ser aplicado e diversas ações que poderiam ser executadas de acordo com os resultados de execução de cada tarefa ou lote. O processamento do grupo de imagens, o processamento com o algoritmo alternativo, a conversão de imagens para um formato comprimido e a substituição das imagens antigas pelas imagens processadas recentemente poderiam ser vistos como tarefas. Nesse caso, poderiam haver duas condições: uma para verificar se a execução da tarefa foi realizada com sucesso e outra para verificar se o erro no processamento é persistente. Atreladas a essas condições, poderiam haver duas ações: uma para o envio de e-mails para as equipes responsáveis pelos processamentos e outra para ativar ou desativar a tarefa de processamento do grupo de imagens utilizando o algoritmo alternativo. O fluxo de execução poderia ser descrito como regras ECA em que as condições seriam verificadas após o término de cada passo do fluxo de execução. A utilização de uma ferramenta de workflow pode prover redução de custos de recursos humanos e diminuição das ocorrências de erros humanos, devido à automatização dos fluxos de execução através da aplicação de regras como a ECA.

1.2 Escalonamento com prazos de execução

Lotes de processamento podem ser críticos para a operação de empresas. Em alguns casos, esses lotes podem possuir um prazo máximo de execução. Um exemplo de lote crítico que possa ter um prazo de execução é o de relatório de fechamento de fatura de cartões de crédito, onde pode haver um prazo para entrega da listagem de todas as compras realizadas dentro de um determinado período. Também podem haver grupos de lotes cujas execuções devam possuir prioridades distintas, como no primeiro exemplo da introdução, em que as imagens relacionadas a estados e cidades mais populosos deveriam possuir maior prioridade nos processamentos. Sistemas utilizados para processamento de lotes podem possuir uma fila extensa de lotes para processamento e a definição de um

prazo máximo de execução pode ser importante para identificar possíveis gargalos em processamentos.

Um sistema de processamento em lote pode precisar lidar com um parque heterogêneo de máquinas ou máquinas que não são utilizadas exclusivamente para processamentos de lotes. Nesse caso, é difícil garantir que os recursos necessários para o processamento de lotes críticos e que possuem prazos máximos de execução estejam disponíveis no momento da execução. Portanto, é importante que um escalonador que lide com esse tipo de processamento leve em conta informações como prazo de execução, prioridade de cada lote e recursos computacionais mínimos para execução, para buscar minimizar a violação de prazos de execução de lotes críticos. Uma forma de se minimizar a violação de prazos de execução de lotes críticos é interromper a execução de lotes de pouca prioridade em favor da execução de lotes críticos, em casos de escassez de recursos computacionais.

Para se buscar a minimização da violação de prazos de execuções de lotes, é importante possuir ao menos um histórico de execução de cada lote. Através do histórico, seria possível identificar atrasos anormais nos processamentos e alguma ação poderia ser tomada para diminuir esses atrasos, como interromper o processamento de tarefas de lotes de menor prioridade ou aumentar a prioridade dos processos no sistema operacional da máquina em que esteja ocorrendo o processamento de tarefas atrasadas prioritárias.

1.3 BatchKeeper

Neste projeto desenvolvemos o BatchKeeper, um sistema que integra processamentos em lote com prazos máximos de execução, monitoramento em tempo real, programação para automatização de tomada de decisões através de uma ferramenta de workflow e um escalonador que considera as prioridades de execução de cada lote.

O BatchKeeper gerencia a execução dos processamentos em lote, dependências entre lotes, dependências entre tarefas e as prioridades de cada lote ou tarefa. Além disso, o BatchKeeper suporta um algoritmo que considera dados históricos de processamentos para estimar a data e hora do término da execução do lote, identificar atrasos na execução de lotes e identificar prováveis violações de prazos máximos de execução. O resultado destas previsões de perda de prazos máximos de execução e atrasos podem ser utilizados pelo módulo de workflow.

O módulo de workflow, chamado de ECA (Evento - Condição - Ação), provê suporte ao controle de fluxos de execução de cada lote ou tarefa para, quando necessário, realizar a automatização de ações a serem executadas de acordo com os resultados de execução de cada lote ou tarefa. Algumas ações que podem ser executadas são: interromper lote

ou tarefa em execução, cadastrar novo lote ou tarefa e alterar a alocação de máquina de uma tarefa a ser executada.

Os lotes e seus ambientes de execução podem ser monitorados por sistemas de monitoramento e pelo próprio BatchKeeper. Esse monitoramento leva à identificação de eventos que impactem a execução dos lotes, como falhas de software ou hardware, erros nos dados e outros, permitindo também a obtenção de informações detalhadas das execuções das tarefas e lotes.

O objetivo do BatchKeeper é ampliar a flexibilidade provida por soluções anteriores [55] na automatização de fluxos que envolvem o processamento em lote. Para se ter um sistema de computação em lote que suporte flexibilidade ampliada é necessário que esse proveja flexibilidade ampliada no funcionamento de funcionalidades principais, como escalonamento de lotes e tarefas, e flexibilidade ampliada na automatização dos fluxos que envolvem os processamentos em lote; aliada à segurança.

A ampliação desta flexibilidade no BatchKeeper se dá a partir do suporte a regras ECA poderosas que podem ser alimentadas com informações provenientes dos outros módulos do BatchKeeper e de sistemas externos. Pode-se atuar no enriquecimento e extensão de funcionalidades do BatchKeeper e na automatização de fluxos com informações internas e externas, a partir desta ampliação de flexibilidade. A ampliação da flexibilização na automatização de fluxos provida pelo BatchKeeper envolve a execução de código-fonte externo de condições e ações de regras ECA.

Para que esta flexibilização esteja aliada à segurança, o BatchKeeper executa todo código-fonte externo em contêineres Docker, isolando o ambiente de execução de regras ECA do ambiente de execução do BatchKeeper. O BatchKeeper também oferece interfaces seguras para comunicação entre condições ou ações, e o próprio BatchKeeper.

O ECA já foi usado como motor para sistemas de processamento em lote, demonstrando que seu uso permitiria fazer automatizações no fluxo que envolve o processamento em lote [55]. O BatchKeeper aplica o ECA para a automatização de fluxos que envolvem o processamento em lote e amplia o conceito do uso do ECA ao trazer armazenamento de estado local para condições e ações, maior flexibilidade aliada a segurança e integração com sistemas externos. Além disso, o BatchKeeper possui um previsor de término de execução de lotes para identificar possíveis atrasos nas execuções dos processamentos, levando em conta dados históricos de execução.

É preciso possuir ampla quantidade de dados disponíveis para que regras ECA possam implantar automatizações de vários níveis. O BatchKeeper resolve esta questão permitindo o recebimento de fluxos de dados de módulos do BatchKeeper e sistemas externos, como sistemas de monitoramento. A partir desses dados, é possível ter visibilidade para automatizar os fluxos que envolvem os processamentos para-se definir alterações no funcionamento do algoritmo de escalonamento, alocação de máquinas, chamadas a serviços externos e alteração do fluxo de execução de lotes, por exemplo.

As principais contribuições do BatchKeeper são as seguintes:

- ECA flexível com condições e ações expressas em código fonte.
- Previsão do tempo de execução via regressão estatística de dados históricos.
- Integração do ECA, previsão e monitoramento extensivo em um sistema de controle de execução de lotes.

No caso do exemplo do processamento de imagens de satélite anterior, os grupos de processamentos poderiam ser cadastrados como lotes no BatchKeeper. O processamento do vínculo de imagens de satélite com informações de nomes de ruas e locais importantes de uma cidade poderia ser mapeado como um tarefa. Um lote poderia agrupar todas as tarefas relacionadas aos processamentos de imagens de cidades de um mesmo estado. Os fluxos relacionados à operação dos processamentos, como os avisos de falhas para as equipes responsáveis pelos processamentos e criação de um lote para execução do algoritmo alternativo para processamento de imagens, poderiam ser cadastrados como regras ECA no BatchKeeper.

Avaliamos os componentes do BatchKeeper utilizando dados de processamentos de lotes reais de uma empresa do mercado financeiro. Estes dados foram inseridos no BatchKeeper para avaliarmos a qualidade do algoritmo de estimativa de violações de prazos máximos de execução. Além disso, avaliamos qualitativamente a flexibilidade do módulo ECA em se adaptar a diversos fluxos de execução existentes em processamento em lote.

O BatchKeeper está sendo aplicado em produção em uma empresa. Ele tem sido utilizado para a execução, completamente automatizada, de centenas de processamentos em lote a cada mês.

Capítulo 2

Conceitos básicos e trabalhos relacionados

É importante a integração entre processamento em lote e ferramentas de workflow [35]. Tal integração pode prover a automatização dos fluxos de processamentos em lote. Além disso, eventos relacionados ao ambiente de execução dos processamentos podem influenciar nas execuções dos mesmos, fazendo-se desejável a coleta de logs do ambiente de execução e a utilização dessas informações na tomada de decisões. Até onde sabemos, outras plataformas de processamento em lote não suportam a automatização de tomadas de decisões relacionadas aos fluxos de processamentos integrada à coleta de informações detalhadas do ambiente de execução dos processamentos.

2.1 Visão geral da evolução do processamento em lote

A história da computação está intimamente ligada à história do processamento em lote. Em 1890, Herman Hollerith (1860-1929) criou o cartão perfurado para a realização de um processamento em lote para o United States Census Bureau [73]. Na Segunda Guerra Mundial os computadores desempenharam processamentos em lote para desencriptar mensagens encriptadas pela máquina Enigma, utilizada pelos alemães na Segunda Guerra Mundial [31]. Com o passar do tempo as máquinas foram ficando mais rápidas e acessíveis, favorecendo a popularização da computação, e consequentemente, do processamento em lote em empresas e governos menores.

Com o crescimento da Internet, grandes companhias precisaram construir grandes centros de processamento de dados para lidar com a alta carga de trabalho gerada por seus websites. O advento do Internet banking, e-commerce, redes sociais, streaming de vídeos e músicas, a quantidade de dados gerados, e a necessidade do processamento destes dados, contribuíram para um grande crescimento no número de centros de processamento de dados [52]. Porém, em boa parte do ano esses centros de processamento de dados

ficavam ociosos devido ao fato de a carga de trabalho (*workload*) não ser uniformemente distribuída [3]. A computação em nuvem tornou-se uma solução flexível e poderosa o suficiente para a execução de grandes processamentos em lotes, principalmente devido a avanços na redução de custo operacional, compartilhamento de recursos e virtualização, provendo isolamento, segurança e flexibilidade em centros de processamento de dados [78; 67; 20].

A crescente carga de trabalho nos centros de processamento de dados e os altos custos de servidores especializados favoreceram o crescimento da computação distribuída em servidores de uso geral compartilhados. Vários modelos de processamento foram desenvolvidos para ambientes distribuídos como o MapReduce, Pregel e outros [12; 38; 18; 70; 77]. Porém, estes modelos não são genéricos o suficiente a ponto de suportar todos os tipos de computação. No Map Reduce o particionamento entre Map e Reduce nem sempre é trivial, como no problema do caixeiro viajante, e não é eficiente ao lidar com algoritmos iterativos, por exemplo [28].

A computação distribuída em nuvem vem sendo muito utilizada também para a execução de serviços, como websites, sistemas de arquivos distribuídos, SGBDs e *appliances* oferecidos por provedores de aplicações. Muitos esforços vêm sendo empregados no desenvolvimento e aprimoramento de tecnologias para processamentos de propósito geral. A tendência é que processamentos e recursos sejam gerenciados por sistemas operacionais de centros de processamento de dados (*datacenter*), responsáveis por escalonar a execução de serviços e processamentos de acordo com os recursos computacionais disponíveis no parque de máquinas. O Apache Mesos [23] é um exemplo de escalonador de recursos de centro de processamento de dados, assim como o Borg [6]. O Mesos também possui diversos projetos relacionados que, ao se conectarem com ele, ampliam a sua utilidade, como o Chronos (agendador de tarefas) [7] e o Marathon (orquestrador de contêineres para o Mesos e DC/OS) [41].

A flexibilização e popularização do uso de contêineres teve um grande impacto em sistemas utilizados para processamento em lote. O Apache Mesos, Borg, IBM Menagerie e Kubernetes já nasceram com suporte à contêineres justamente devido à facilidade de se realizar processamentos em qualquer máquina, dada a característica de portabilidade de aplicações provida pelos contêineres. Os contêineres também possuem desempenho maior do que as máquinas virtuais, chegando à um desempenho muito próximo ao alcançado por aplicações instaladas na máquina física [13].

2.2 Monitoramento de logs

Um tópico que vem ganhando muita atenção é o de coleta e monitoramento de logs de servidores e de aplicações [50]. Com o avanço da computação distribuída e da computação em nuvem, a quantidade de máquinas virtuais e contêineres aumentou consideravelmente, e, consequentemente, também aumentaram as dificuldades de se administrar grandes quantidades de logs de máquinas virtuais e contêineres. Devido a esse problema, ferramentas de monitoramento de logs de servidores e aplicações ganharam muita popularidade pois, através da criação de filtros, é possível monitorar diversas métricas de utilização e funcionamento dos sistemas [60; 47; 48]. Estas métricas são utilizadas para guiar atualizações futuras para melhoria de desempenho de sistemas e correção de erros [71].

Um sistema de monitoramento de logs deve ser escalável o bastante para suprir as necessidades da computação em nuvem. O Chukwa [57] é um sistema de coleta de logs em larga escala. Esse sistema foi criado para prover confiabilidade na persistência de logs em ambientes distribuídos. Em um ambiente distribuído, as escritas de logs também ficam distribuídas. O Chukwa coleta estes logs para aumentar o desempenho de leitura dos mesmos. Os logs são coletados por um agente dedicado em cada máquina que esteja configurada para ser monitorada. Cada agente é responsável por produzir metadados, lidar com falhas eventuais e integrar-se com fontes de dados existentes. Os agentes são responsáveis por assegurar que os logs foram escritos com sucesso no disco pelo menos uma vez, reenviando esse log para o Chukwa quando necessário.

Uma forma simples de se encaminhar logs para processamento por outras aplicações é utilizando plataformas de fluxos de dados distribuído do tipo produtor-consumidor (*publish-subscribe*) [44; 1; 19]. Um exemplo deste tipo de plataforma é o Apache Kafka [27]. O Kafka cria tópicos em que é possível publicar e consumir dados. É tolerante a falhas, escalável e rápido [75]. O Kafka armazena os dados de tópicos de forma segura ao mesmo tempo que os encaminha para as aplicações que tenham se inscrito para consumir estes dados. Sistemas de coleta de logs podem ser publicadores de dados em tópicos do Kafka. Pode-se configurar o Kafka para fazer processamentos distribuídos leves em fluxos de dados [14; 10] e, após tais processamentos, disponibilizar os fluxos em outros tópicos que podem ser consumidos por outros sistemas. Assim, o Kafka pode ser utilizado como um integrador de sistemas com a possibilidade de se realizar adaptações em fluxos de dados para serem consumidos de forma flexível por outros sistemas [76; 11]. Além disso, pode-se utilizar ferramentas externas de análise (*analytics*) para visualização de métricas, como o Grafana [32]. Esta integração é importante para justamente ser possível escrever logs em tópicos, efetuar consultas em cima de dados de logs em tempo de execução que podem ser consumidas por outros sistemas, criar painéis para melhor visualização do

status de execução de sistemas, dentre outros.

Sistemas de mensageria também podem ser utilizados para transportar dados de logs entre sistemas produtores e sistemas consumidores em filas ou tópicos [53; 69; 45]. Há sistemas de mensageria que podem ser configurados para prover persistência de dados por um tempo determinado. Sistemas de mensageria geralmente não permitem processamentos de dados armazenados em filas ou tópicos, porém podem ser integrados a outros sistemas que poderiam ser consumidores de filas, fazer processamentos e produzir os resultados destes processamentos em novas filas do sistema de mensageria.

Sistemas de monitoramento são importantes para identificar eventuais problemas em aplicações e em seus ambientes de execução. Sistemas de monitoramento podem monitorar o desempenho de rede em um conjunto de máquinas, logs de erros de aplicações, desempenho de máquinas e até enviar notificações de acordo com métricas que podem ser especificadas. Alguns sistemas de monitoramento muito utilizados são o OpenStack Monasca, Riemann, Nagios e Prometheus [47; 60; 48; 54]. Apesar das diferenças entre estes sistemas, todos permitem criar regras de monitoramento e expor o que foi monitorado para ferramentas externas, como o Apache Kafka. O Kafka poderia ser utilizado para disponibilizar grupos de dados à outras plataformas de forma simples, como escrevendo estes dados em uma tabela de banco de dados.

É importante a existência de um monitoramento constante dos recursos utilizados nos processamentos para que, quando possível, os recursos necessários sejam provisionados e erros sejam resolvidos rapidamente. O Morpheus [26] é um sistema que foi projetado para oferecer recomendações de alteração na alocação de recursos e reprovisionar estes recursos dinamicamente quando necessário. O Morpheus consegue fazer isto através da análise dos logs de execuções anteriores de cada processamento. Esta inferência de recursos necessários para processamentos é ainda mais desejável em ambientes cujos processamentos são críticos e possuem restrições de tempo de execução e recursos.

É importante dizer que, apesar do constante uso de sistemas de monitoramento, é difícil, em muitos casos, identificar causas de erros de forma automática. Esta dificuldade se dá pelo fato de existirem muitos ruídos em logs, isto é, falhas que não necessariamente levam a erros importantes. Porém existem trabalhos que focam em identificar a causa raiz de problemas de forma automática em aplicações específicas [64; 65]. Para isso, existem sistemas que analisam anomalias no uso de recursos computacionais e até documentações de arquiteturas de softwares para possibilitar a identificação de tais falhas [8; 9].

2.3 Escalonamento

No ambiente de processamento em lote, muitas vezes a ordem de execução dos lotes e tarefas é importante. É desejável que um algoritmo de escalonamento nesse ambiente, além de distribuir tarefas de acordo com os recursos disponíveis, leve em conta prioridades, prazos máximos de execução de lotes e possíveis dependências entre tarefas e entre lotes.

Existem diversos algoritmos de escalonamento de tarefas. O objetivo de alguns algoritmos de escalonamento varia em função da aplicação. Em sistemas iterativos, um requisito importante é reduzir o tempo de resposta. Em sistemas de computação científica, ou em lote, é diminuir o (*turnaround*) ou aumentar a taxa de processamento (*throughput*). Além disso, existem algoritmos de escalonamento que tratam apenas de uma máquina local, como os algoritmos utilizados no escalonamento de tarefas por sistemas operacionais, e outros que tratam do escalonamento de tarefas em ambiente distribuído.

2.3.1 Arquitetura

O escalonador pode ter uma arquitetura centralizada, hierárquica ou distribuída. Em uma arquitetura centralizada, um escalonador possui todas as informações de todos as tarefas a serem executadas e todos os recursos disponíveis de todas as máquinas do grid. Na arquitetura hierárquica, um escalonador central distribui as tarefas para outros escalonadores locais de acordo com algum algoritmo de previsão de início da execução de cada tarefa. Já na arquitetura distribuída, cada máquina possui um escalonador e estes trocam mensagens entre si a respeito de suas cargas de trabalho, enviando, quando necessário, tarefas para outros escalonadores a fim de balancear a carga de trabalho do conjunto de máquinas utilizadas para os processamentos [66]. [66] desenvolveu o *K-Distributed Model*, onde a arquitetura distribuída foi utilizada. Nesta solução, são enviadas requisições de execução de cada tarefa para k escalonadores locais; quando uma tarefa é escolhida para ser executada, é enviada uma requisição de cancelamento para todos os outros k-1 escalonadores que haviam recebido a requisição de execução desta tarefa. O principal problema dessa abordagem é o desperdício de trabalho causado quando é necessário cancelar uma tarefa já em execução.

O Apache Mesos [23] utiliza um mecanismo de escalonamento em dois níveis. O Apache Mesos mantém agentes em todas as máquinas que serão usadas para processamentos, enviando frequentemente as informações de recursos disponíveis a um nó mestre. O nó mestre envia ofertas de recursos computacionais para clientes. Clientes cuidam do

escalonamento de tarefas, enviando informações para execução de tarefas e recursos computacionais ao nó mestre ao aceitar uma oferta de recursos computacionais. Um cliente pode recusar ofertas de recursos computacionais caso uma tarefa deva ser executada em outra máquina específica ou caso a quantidade de recursos computacionais seja insuficiente para a execução de suas tarefas, por exemplo. Os clientes são softwares que podem ser plugados no Mesos, retirando do Mesos a responsabilidade de escalonar tarefas para recursos computacionais. Assim, sistemas como Hadoop e Spark podem manter suas próprias políticas de escalonamento e utilizarem o Apache Mesos de forma integrada para a execução de tarefas [58].

2.3.2 Algoritmos de máquina única clássicos

Quanto aos algoritmos de máquina única, o *First come, first served* (FCFS)[68] é o algoritmo mais básico. O FCFS distribui as tarefas de acordo com suas ordens de chegada, levando em conta se os requisitos mínimos de cada tarefa podem ser satisfeitos de acordo com os recursos disponíveis. Há também algoritmos que executam tarefas de acordo com a prioridade de cada tarefa na fila de tarefas prontas para execução. Esse tipo de algoritmo pode ser útil em casos em que existam tarefas mais críticas do que outras; porém, pode ser difícil otimizar o fluxo de execução caso haja muitas tarefas com prioridades distintas. O *Fastest Processor to Largest Task First* (FPLTF) [43] distribui as tarefas de tempo maior de processamento para os processadores mais rápidos. Trazendo para o contexto de processamento em lote em múltiplas máquinas, as tarefas de maiores requisitos por recursos computacionais poderiam ser distribuídas para as máquinas de maior poder de processamento disponível. Pode fazer sentido aplicar esse algoritmo caso o principal objetivo seja minimizar as chances de falta de recursos computacionais.

Algoritmos de escalonamento de processos, como o "O(1)" [25], CFS [46; 21; 29] e o BFS [30; 21] buscam equilibrar o tempo de resposta com o desempenho do sistema. Esses algoritmos interrompem tarefas e as continua posteriormente e são chamados de preemptivos. Esses algoritmos são utilizados em vários sistemas operacionais, dentre eles o Linux. O algoritmo "O(1)" aloca um tempo fixo para cada tarefa. O "O(1)" utiliza dois vetores principais. Um vetor armazena as tarefas que possuem tempo restante de execução alocado (ativas) e o outro vetor armazena as tarefas que já não possuem tempo restante de execução alocado (expiradas). O escalonador escolhe a tarefa de maior prioridade para execução e, após o término do tempo alocado, a move para o vetor de tarefas expiradas. Quando o vetor de tarefas ativas fica vazio, o vetor de tarefas expiradas passa a ser o vetor de tarefas ativas e o vetor de tarefas ativas passa a ser o vetor de tarefas expiradas. Todas

as operações desse algoritmo possuem complexidade "O(1)". Os principais problemas do "O(1)" é alto tempo de resposta, que o torna inadequado para aplicações interativas, e baixo desempenho (*throughput*) em execuções de tarefas em segundo plano [25].

O algoritmo CFS (*completely fair scheduler*) tenta diminuir injustiças na alocação de tempo de CPU para as tarefas. Cada tarefa recebe um tempo virtual. O tempo virtual mede a alocação de tempo de CPU para cada tarefa, sendo inicialmente um valor baixo. Após o término de execução de uma tarefa por uma quantia de tempo, o tempo virtual desta tarefa é aumentado, sinalizando que esta foi alocada para um processador. O CFS escalona tarefas a partir dos menores tempos virtuais existentes em sua fila de execução. Assim, tarefas que não foram alocadas a um processador, ou foram alocadas poucas vezes, possuem maior prioridade para serem escalonadas pelo CFS do que tarefas que tenham sido alocadas por mais tempo. O tempo virtual de cada tarefa é armazenado em uma árvore red-black. Uma árvore do tipo red-black é auto-balanceada e operações de inserção e remoção de elementos possuem um bom desempenho, habilitando boa escalabilidade ao CFS. Caso existam tarefas com prioridades pré-definidas, sem ser pelo escalonador, o CFS ajusta dinamicamente o tamanho da janela de alocação de tempo para que tarefas com maior prioridade possam conseguir maior alocação de tempo para sua execução.

O algoritmo BFS (*Brain Fuck Scheduler*) [30; 21] coloca todas as tarefas que não estejam sendo executadas em uma fila. Cada tarefa recebe um tempo virtual e um tamanho de janela de tempo em que poderá ser executada pelo processador ao ser adicionada na fila. O tempo virtual indica quando uma tarefa seria executada, porém sem garantias de execução. O BFS escalona as tarefas de acordo com o tempo virtual de cada tarefa, em que tarefas de tempo virtual menores e mesma prioridade (*niceness*) são escalonadas primeiro. Tarefas de maior prioridade recebem um tempo virtual menor. Quando uma tarefa é bloqueada a espera de outros recursos, como disco, ela é colocada de volta na fila para execução, mantendo seu tempo virtual e seu tamanho restante da janela de tempo, alcançando alta prioridade para sua execução.

2.3.3 Algoritmos orientados a prazos máximos de execução

Em alguns sistemas, os algoritmos de escalonamento levam em conta a informação do prazo de execução de cada processo. Um algoritmo preemptivo simples utilizado em sistemas de tempo real é o EDF (Earliest Deadline First) [34]. O EDF consiste em distribuir as prioridades entre processos de forma inversamente proporcional à proximidade dos prazos de execução de cada um. Um problema do EDF é que esse algoritmo, uma vez iniciado, não se ajusta levando em conta feedbacks contínuos a respeito do andamento

das execuções de tarefas. Além disso, o EDF não funciona bem em ambientes onde recursos computacionais são insuficientes para a execução de toda a carga de trabalho. O FC-EDF [37] aborda estas limitações do EDF, adicionando um controlador de admissão e recebendo feedback, através de sensores, sobre o andamento de cada processo. O FC-EDF consegue melhorar a vazão (em taxa de processamento) a partir da configuração de um parâmetro que define a frequência máxima de violações, flexibilizando o escalonamento e melhorando a vazão de execução de tarefas ao custo de eventuais violações de prazos de execução. A partir do feedback recebido, o FC-EDF consegue ajustar a distribuição de recursos computacionais de forma a minimizar a violação de prazos de execução. Sem o parâmetro de frequência máxima de violação de prazo de execução, o escalonador sempre teria que ser muito conservador, diminuindo a vazão dos processamentos de tarefas. Há diversos sistemas que são muito conservadores na hora de escalonar tarefas para não correr o risco de perder prazos de execução, reservando todos os recursos necessários no momento do cadastro de tarefas [74].

2.3.4 Algoritmos que lidam com um parque heterogêneo de máquinas

Os algoritmos acima lidam bem com o problema de escolher o momento em que cada tarefa será executada em uma máquina, porém, pensando em um parque heterogêneo de máquinas para processamento de lotes de prioridades distintas, estes algoritmos, isoladamente, podem ser insuficientes para processamento em lote, por não levar em conta as variáveis envolvidas no problema de processamento em lote. O balanceamento de carga no conjunto de máquinas é importante pois faz com que tarefas possam ser executadas mais rapidamente em máquinas com maior quantidade de recursos computacionais disponíveis e que atendam aos requisitos computacionais mínimos de cada tarefa [49; 59]. Além disso, é importante que o algoritmo de escalonamento seja dinâmico, já que a disposição e recursos computacionais livres pode mudar em tempo de execução, principalmente se estas máquinas não forem utilizadas exclusivamente para processamentos.

O *Dominant Resource Fairness*(DRF) [15] é um exemplo de algoritmo de escalonamento de tarefas que leva em conta diferentes tipos de recursos computacionais. O DRF escalona tarefas de acordo com a relação entre os recursos computacionais existentes e os requisitos de cada tarefa. O recurso computacional proporcionalmente mais utilizado de uma tarefa em relação aos recursos computacionais totais é chamado de recurso dominante. Cada tarefa possui um recurso dominante. O DRF escalona as tarefas priorizando a tarefa que possua a menor taxa de uso de seu recurso dominante.

As principais propriedades que fazem o DRF se destacar são: Um usuário não se beneficia mentindo sobre os requisitos computacionais mínimos ao cadastrar uma tarefa para execução; a utilização de recursos computacionais é compartilhada entre as tarefas dos diversos usuários; Como a alocação de recursos de cada usuário é a melhor para cada usuário, nenhum usuário preferiria trocar sua alocação pela alocação de outro usuário; não há como aumentar a alocação de recursos de um usuário sem diminuir a alocação de outro.

2.3.5 Utilização de aprendizado de máquina

Há escalonadores que utilizam aprendizado de máquina para prever os recursos computacionais necessários para que novas tarefas atinjam determinado prazo de execução [59]. O uso de aprendizado de máquina nesse caso é interessante pois permite prever os recursos necessários para executar tarefas que nunca foram executadas anteriormente. Outros algoritmos utilizam aprendizado de máquina para escolher as melhores máquinas para a execução das tarefas, de acordo com o uso de recursos de cada tipo de tarefa, reduzindo a fila de tarefas a serem executadas no conjunto de máquinas utilizada para os processamentos [49].

2.4 Workflow

Paralelamente às inovações no campo de processamento em lote, foram criadas ferramentas para controle de fluxos de trabalho, chamadas de ferramentas de workflow [5]. As ferramentas de workflow servem para modelar e automatizar processos de negócios. Essas ferramentas permitem a instanciação de processos e aplicações de acordo com fluxos modelados e eventos disparados no ambiente de execução [17].

Lotes são compostos por uma ou mais tarefas. Estas tarefas podem ser executadas paralelamente ou sequencialmente, podendo haver também uma ordem específica de execução entre algumas tarefas. Uma ferramenta de workflow pode ser útil para executar lotes ou tarefas que fazem parte de um fluxo de processos de negócio [16; 35]. Dessa forma, o fluxo de processos envolvidos em processamento em lote pode ser automatizado, bem como a própria execução de processamentos em lote.

Uma outra forma de se utilizar ferramentas de workflow é através de uma integração

interna com sistemas para processamento em lote, transferindo funcionalidades de uma ferramenta de workflow para dentro de um sistema para processamento em lote. Nesse caso, um sistema para processamento em lote com funcionalidades de uma ferramenta de workflow poderia suportar o gerenciamento das dependências entre lotes e dependências entre tarefas, instanciações de lotes e automatização de processos, em alto nível [35]. Regras do tipo Evento-Condição-Ação (ECA) são apropriadas para a configuração do workflow de processamentos em lote, pois descrevem as ações a serem tomadas de acordo com eventos disparados e suas circunstâncias [16; 72; 4]. Regras do tipo ECA já são muito populares em ferramentas de workflow e em bancos de dados [56; 2; 51].

Uma ferramenta de workflow integrada a processamentos em lote poderia ser utilizada para alterar o comportamento do próprio sistema de processamento em lote, como o escalonamento de tarefas, para que esse possa se adequar à mudanças no ambiente de execução dinamicamente [55]. Em um cenário em que exista um cadastro de manutenções preventivas nos servidores utilizados para processamentos em lote, uma ferramenta de workflow poderia reagendar automaticamente tarefas, previamente agendadas para serem executadas em horários de manutenções, para serem executadas em horários sem manutenções, por exemplo.

Alguns exemplos de ferramentas para processamento em lote que utilizam conceitos de workflow estão na seção 2.5.1;

2.5 Integração de Trabalhos Relacionados com o BatchKeeper

Um dos objetivos do BatchKeeper é ampliar a flexibilidade na automatização de fluxos de execução e fluxos que envolvem o processamento em lote. O BatchKeeper provê um algoritmo de escalonamento básico de arquitetura centralizada orientado a prioridades e delega a distribuição de prioridades a algoritmos externos que podem ser implementados através da utilização do ECA. Esta abordagem permite a implantação de diferentes políticas que podem transformar este algoritmo básico em qualquer algoritmo de escalonamento. O BatchKeeper também permite delegar a alocação de tarefas para cada máquina a algoritmos externos através da utilização do ECA. Assim, algoritmos mais complexos, como os que utilizam aprendizado de máquina [59], podem ser utilizados tanto para o escalonamento quanto na alocação de máquinas [49].

Ao contrário de soluções anteriores [55], o BatchKeeper não utiliza o ECA como seu motor principal para o controle de execução de lotes e tarefas. Porém, é possível

adicionar, remover e alterar tarefas e lotes através de regras ECA. O BatchKeeper provê automatização dos fluxos que envolvem o processamento em lote de forma mais flexível do que em outras soluções, através da funcionalidade de descrever condições e ações de regras ECA como código-fonte. Além disso, o BatchKeeper provê armazenamento de estado durante a execução de condições e ações, algo não encontrado nas outras soluções de mesmo fim.

Conforme a seção 2.2, o monitoramento de execuções de processos em geral e seus respectivos ambientes de execução vem sendo cada vez mais utilizado. Dados de monitoramento podem ser úteis na automatização de fluxos que envolvem o processamento em lote, na melhoria de algoritmos de escalonamento e alocação de tarefas às máquinas do conjunto de máquinas utilizadas para processamento. Tarefas que estejam alocadas para execução em uma máquina que esteja em manutenção poderiam ser adiadas ou serem alocadas para execução em outras máquinas disponíveis, por exemplo. O BatchKeeper provê fácil integração com sistemas externos em que estes podem enviar mensagens que podem ser capturadas pelo módulo ECA. Regras ECA podem ser criadas para utilizar os dados recebidos de sistemas externos e aplicar para ações programadas, como a alteração da alocação de máquinas ou alteração da priorização da execução de lotes, por exemplo.

O BatchKeeper possui suporte nativo a execuções de lotes e tarefas em contêineres. Modelos de computação como o MapReduce e o Pragel podem ser executados por tarefas cadastradas no BatchKeeper. O BatchKeeper não apresenta um modelo de computação novo, sendo possível a execução de alguns modelos existentes de computação no BatchKeeper.

2.5.1 Diferenças entre o BatchKeeper e sistemas similares

Foram analisados os seguintes sistemas similares ao BatchKeeper: Mistral; combinação de Mesos e Chronos; Menagerie; Cook; GoDocker e JobServer. Estes sistemas processam lotes de tarefas ou apenas tarefas e são, em sua maioria, código aberto. As principais diferenças em relação ao funcionamento destes sistemas estão apresentadas na tabela 2.1.

Alguns sistemas possuem a noção de lotes, outros possuem apenas a noção de tarefas, porém, todos os sistemas executam os comandos cadastrados em um conjunto de máquinas pertencentes a um cluster.

Uma diferença significativa entre o BatchKeeper e os outros sistemas é a presença do módulo ECA como uma ferramenta para automatizar todo o fluxo envolvido nos processamentos dos lotes.

Funcionalidades	**BatchKeeper**	**Mistral**	**Mesos + Chronos**	**Control-M**	**Menagerie**	**Cook**	**GoDocker**	**JobServer**
Suporte a lotes	S	S	S	S	S	S	S	S
Dependências entre lotes	S	N	N	S	N	N	N	N
Dependências entre tarefas	S	S	S	S	N	N	S	S
Execução paralela de jobs	S	S	S	S	S	S	S	S
Execução paralela de lotes	S	S	S	S	N	S	S	S
Especificação de Host para execução de jobs	S	S	S	S	N	S	S	S
Especificação de múltiplos hosts	N	N	S	S	N	S	S	S
Garantia de recursos para a execução de lotes na hora cadastrada	N	N	N	S	N	N	N	N
Módulo ECA	S	S	N	N	N	N	N	N
Eventos temporais	S	N	N	N	N	N	N	N
Condições flexíveis	S	N	N	N	N	N	N	N
Informações provenientes de logs	S	N	N	N	N	N	N	N
Formato de entrada dos dados:	Web/Rest	Yaml/Rest	Web/Rest	Web/Rest/Cliente	Http/Json	REST	Web, Client e REST	Web e REST
Requisitos mínimos de execução de lotes/jobs	S	N	S	S	N	S	S	N
Tolerância a falhas (Tarefas)	S	S	S	S	N	S	S	S
Tolerância à falhas (Sistema)	N	S	S	S	N	S	S	S
Suporte a prazos máximos de execução	S	N	N	S	N	N	N	N

Tabela 2.1: Comparação entre as principais funcionalidades encontradas no BatchKeeper e outros softwares de finalidade similar

O Apache Mesos oferece flexibilidade ao permitir que seus usuários escrevam código-fonte para o escalonamento de tarefas. O Chronos adiciona ao Mesos o cadastro e gerenciamento de lotes. Porém, não possuem uma plataforma para automatização dos fluxos que envolvem o processamento de lotes.

O OpenStack Mistral possui um módulo ECA, porém o funcionamento deste é bem distinto em relação ao ECA do BatchKeeper. Para cada execução de tarefa, pode-se especificar a execução de outras tarefas de acordo com o status da execução da última tarefa executada, funcionando de forma similar aos blocos "Try" "Catch" e "Finally", comuns em linguagens de programação. Nesse caso, os únicos eventos existentes seriam "Tarefa executada com sucesso" e "Falha na execução da tarefa". O funcionamento do ECA do BatchKeeper é muito mais amplo. As condições e ações executadas, pelo fato de serem código-fonte, podem consultar diversas informações de lotes no BatchKeeper e executar tarefas diversas, como: fazer um post em uma interface REST de outro sistema; realizar cálculos e guardar dados em um banco de dados local, como parte de uma condição; dentre outros.

Outra grande diferença do BatchKeeper em relação aos outros sistemas é a presença de um algoritmo previsor de limite superior do tempo de execução de lotes. Este algoritmo consegue, através de dados históricos de execuções, prever a violação de prazos máximos de execução de lotes, possibilitando a ativação de fluxos programados para o tratamento deste tipo de evento.

Capítulo 3

Processamento em lote com o BatchKeeper

O BatchKeeper é uma solução completa para o agendamento, execução e monitoramento de processamentos em lote. Um *lote* é um agrupamento de tarefas que podem ser executadas em uma determinada sequência. Uma *tarefa* define um programa a ser executado, parâmetros e entradas. O BatchKeeper possui um gerenciador de lotes onde todos os lotes são cadastrados. A execução dos processamentos é disparada e gerenciada por um agente local em cada máquina (virtual ou física) utilizada pelo BatchKeeper. O BatchKeeper contém um módulo ECA que funciona como uma ferramenta de *workflow* integrada que permite a automatização de decisões, em tempo de execução, relacionadas aos eventos de cada processamento. Os eventos capturados pelo módulo ECA são eventos lançados pelo próprio BatchKeeper para alertar sobre o andamento da execução de lotes ou tarefas, além de eventos que podem ser lançados por sistemas externos através de interfaces do BatchKeeper. Os eventos lançados por sistemas externos podem ser lançados por ferramentas de coleta e monitoramento de logs, por exemplo, para abastecer o BatchKeeper com informações externas. Um caso de uso seria utilizar sistemas de monitoramento para passar informações completas sobre o estado do conjunto de máquinas, por exemplo.

3.1 Arquitetura

O BatchKeeper é formado pelos módulos Gerenciador de Lotes, ECA e Executor. O BatchKeeper utiliza um banco de dados relacional para prover persistência de dados, e agentes para executar os lotes e tarefas diretamente nas máquinas de processamentos. A arquitetura do BatchKeeper é mostrada na figura 3.1.

O Gerenciador de Lotes armazena as informações dos lotes, tarefas, máquinas, grafo de dependências entre tarefas e grafo de dependências entre lotes. O Gerenciador de Lotes gerencia as prioridades de cada lote e implementa o algoritmo de escalonamento

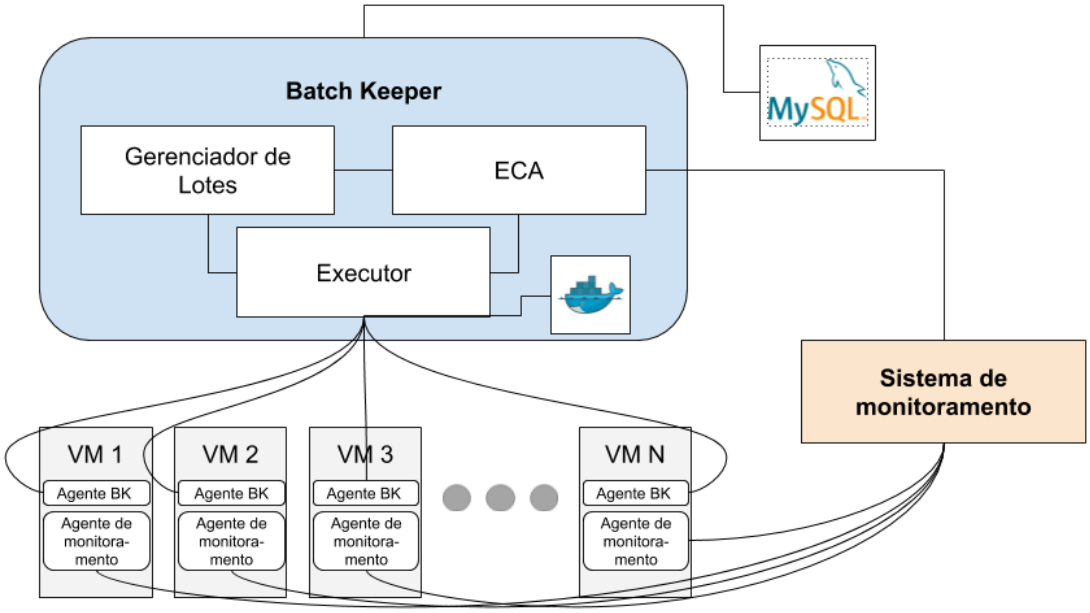


Figura 3.1: Arquitetura do BatchKeeper

do BatchKeeper.

O módulo ECA armazena as regras ECA (Evento - Condição - Ação) associadas aos lotes e as tarefas. As regras ECA especificam e automatizam decisões, durante a execução de cada lote. As condições e ações das regras ECA são descritas diretamente como código-fonte na linguagem Python, provendo flexibilidade aos operadores de implantações do BatchKeeper. O módulo ECA recebe todos os eventos lançados pelos outros módulos do BatchKeeper e os armazena em uma tabela de logs no banco de dados do BatchKeeper. É através do recebimento desses eventos que o módulo ECA consegue ter dados gerais sobre o funcionamento do BatchKeeper, incluindo execuções de lotes, condições e ações, e pode executar as regras ECA apropriadas. Ferramentas de coleta ou monitoramento de logs podem ser configurados para enviar eventos para o módulo ECA, enriquecendo a tomada de decisões feita por este módulo. O módulo Gerenciador de Lotes expõe os dados de lotes, tarefas máquinas e grafos de dependência para leitura e escrita ao módulo ECA.

O Executor é responsável por executar as tarefas, definidas pelo escalonador do Gerenciador de Lotes, no conjunto de máquinas utilizadas para os processamentos. Além disso, o executor é responsável por executar as condições e ações de regras ECA em contêineres. Na prática, o executor implanta as instruções, sobre quais lotes ou regras executar, recebidas do escalonador e módulo ECA. O executor executa condições e ações em contêineres para prover segurança por construção, através da execução do código-fonte das condições e ações em um ambiente isolado. Ao ser acionado para executar

alguma tarefa, o *executor* instancia um agente para executá-la na máquina definida pelo escalonador. O *executor* pode utilizar máquinas físicas, virtuais ou contêineres para a execução de tarefas. Cada agente instanciado recebe o comando que deve executar e atualiza o *executor* com suas informações para conexão e resultados da execução de cada tarefa. O *executor* lança eventos associados às execuções de lotes, condições e ações que são capturados pelo módulo ECA, fornecendo insumos para as tomadas de decisões do módulo ECA.

3.2 Gerenciador de lotes

O Gerenciador de Lotes é responsável por cadastrar e gerenciar as informações de lotes e tarefas. O cadastro e gerenciamento das informações de lotes e tarefas é realizado em uma interface Web. Todas as informações e operações sobre lotes e tarefas são expostas via interfaces REST, contribuindo para flexibilização no uso do BatchKeeper, permitindo automatização de sequências de operações e integração com outros sistemas. O gerenciador de lotes também possui um escalonador, responsável pelas políticas de escalonamento de lotes e tarefas do BatchKeeper.

O Gerenciador de Lotes provê a funcionalidade de se definir, explicitamente, dependências entre tarefas de mesmo lote, através do cadastro de grafo de dependências. Caso não haja o cadastro das dependências entre tarefas de um mesmo lote, o *escalonador* assume que todas as tarefas podem ser executadas paralelamente.

O gerenciador de Lotes permite o cadastro de *grupos de lotes*. Um grupo de lotes serve para agrupar lotes que devem ser executados em uma determinada sequência. Um grupo de lotes possui a maioria dos campos de um lote. Um grupo de lotes pode ser utilizado nos casos em que seja observada a necessidade de impedir a execução paralela de um conjunto de lotes. Grupos de lotes também podem ser utilizados para facilitar o gerenciamento de lotes similares que podem fazer sentido serem agrupados.

O Gerenciador de Lotes verifica, no momento do cadastro dos grafos de dependências, a existência de ciclos nestes grafos. O Gerenciador de Lotes permite apenas o cadastro de grafos direcionados acíclicos.

Atributos de Lotes	
Atributo	**Descrição**
id	Identificador do lote
nome	Nome do lote
descricao	Descrição do lote
prioridade	Prioridade de execução
prazo_maximo	Prazo máximo para execução do lote
repeticao	Repetição do lote. Exemplo: diário, semanal, mensal, anual
intervalo_repeticao	O intervalo da repetição. Exemplo: se o intervalo tiver o número 2 e a repetição estiver como diária, o lote será executado a cada 2 dias
inicio	Data e hora agendada para o início da execução
inicio_execucao	A data e hora em que o lote foi realmente executado na última vez
fim_execucao	A data e hora em que o lote terminou de ser executado na última vez
status	O status do lote. Exemplo: criado, aguardando execução, em execução, sucesso, falha ao executar,
dono	O e-mail do usuário dono do lote
nome_dono	O nome do dono do lote
ativo	Booleano que especifica se o lote está ativo ou não
última_execucao	Data e hora da última execução do lote
regressao	Booleano que especifica a execução do cálculo de limite superior de tempo de execução para o lote

Tabela 3.1: Listagem de atributos de lotes

Atributos de Tarefa	
Atributo	**Descrição**
id	Identificador da tarefa
nome	Nome da tarefa
descricao	Descrição
lote	Lote a qual esta tarefa está ligada
maquina	máquina em que esta tarefa deverá ser executada
nome_container_docker	O nome do contêiner docker em que esta tarefa deverá ser executada
comando	O comando a ser executado
argumentos	Os argumentos do comando a ser executado
status	O status da tarefa
requisitos_minimos	Requisitos mínimos para a execução desta tarefa
prazo_maximo	Prazo máximo de execução da tarefa
inicio_execucao	Data e hora do último início de execução da tarefa
fim_execucao	Data e hora do último fim de execução da tarefa
ativo	Flag que especifica se esta tarefa está ativa ou não
saida	Saída da execução da tarefa
saida_erro	Saída de erro da execução da tarefa
exit_code	Código da saída da execução da tarefa

Tabela 3.2: Listagem de atributos de tarefas

3.2.1 Atributos de lotes e tarefas

Cada lote possui atributos chave, como nome, descrição, prioridade, data e horário para início de processamento, frequência de execução, recursos mínimos computacionais (CPU, memória e armazenamento) livres necessários para a sua execução, status, prazo máximo de execução (deadline) e um campo para ativação ou desativação do lote. Todos os atributos de lotes estão descritos na tabela 3.1. Cada tarefa possui atributos chave como, lote, nome, descrição comando, argumentos, máquina, nome do contêiner em que o comando deverá ser executado (caso o comando tenha de ser executado em um contêiner),

status, requisitos mínimos e um campo para ativação ou desativação da tarefa. Os atributos de tarefas estão descritos na tabela 3.2. Estes atributos chave são preenchidos pelo operador do BatchKeeper e são necessários para que o *executor* tenha insumos suficientes para fazer as execuções dos lotes e tarefas. Os atributos restantes são preenchidos pelo próprio BatchKeeper para acompanhamento pelo operador do resultado das execuções de lotes e tarefas.

Os módulos ECA e Executor, além do escalonador, também utilizam os dados de atributos de lotes e tarefas para identificar as configurações de execução e atualizações sobre o andamento dos processamentos.

3.2.2 Escalonador

O módulo Gerenciador de Lotes possui um escalonador, responsável por decidir, em tempo de execução, quais lotes serão executados. O escalonador do BatchKeeper suporta a execução paralela de lotes e tarefas, respeitando dependências entre jobs, além de monitorar o andamento de execuções de lotes e tarefas.

O escalonador obtém o grafo de dependências de tarefas, bem como o grafo de dependências de lotes e somente envia lotes para execução caso haja recursos computacionais que atendem os requisitos mínimos de todas as tarefas do lote, nas máquinas em que as tarefas serão executadas. Um lote pode ter tarefas que sejam executadas em máquinas distintas. Caso o atributo máquina esteja configurado em uma tarefa, o escalonador não a enviará para ser executada em outra máquina. Caso contrário, o escalonador poderá enviar a tarefa para execução em qualquer máquina do conjunto de máquinas para processamento que possua os requisitos computacionais mínimos disponíveis da tarefa.

O escalonador utiliza o campo estado de cada lote para gerenciar a fila de processamentos. Cada novo lote do BatchKeeper é cadastrado com o status inicial "Aguardando processamento". O escalonador adiciona lotes à fila de lotes para execução de acordo com a configuração de início e período de execuções de cada lote. Caso a execução de uma tarefa falhe, o estado desta tarefa é atualizado para "Falha ao executar" e as tarefas dependentes desta tarefa não são executadas.

O escalonador utiliza o campo de prioridade de cada lote para ordenar a fila de lotes para execução em que lotes com maior prioridade estejam sempre à frente de lotes com menor prioridade. Após esta ordenação, estes lotes são disponibilizados ao *executor* para serem executados.

O comportamento do escalonador pode ser alterado através da definição de regras ECA que alterem os valores de prioridades de lotes e alocação de máquinas em tempo

de execução. A partir desta flexibilidade, outros algoritmos de escalonamento podem ser utilizados para reordenar a fila de lotes para execução alterando as prioridades dos lotes cadastrados. As regras ECA serão explicadas na seção 3.3.

3.3 ECA

O módulo ECA do BatchKeeper funciona como um módulo de workflow. O módulo de workflow do BatchKeeper serve para automatizar tomadas de decisões durante os fluxos de processamentos. A automatização se dá a partir da definição de ações a serem executadas de acordo com eventos disparados e condições satisfeitas, denominando uma regra ECA (Evento - Condição - Ação). Todos os eventos que são lançados pelo BatchKeeper são armazenados em uma tabela de logs do banco de dados do BatchKeeper. Sistemas externos, como sistemas de monitoramento, também podem enviar ou obter eventos do BatchKeeper através de uma interface REST ou acesso direto à tabela de logs. O módulo ECA monitora modificações na tabela de logs do BatchKeeper para obter todos os eventos que são disparados internamente ou externamente, como no caso de eventos disparados por um sistema de monitoramento. Para cada novo evento na tabela de logs, o módulo ECA verificará a existência de regras ECA que contêm o evento disparado em seu cadastro. Caso estas regras existam, as condições vinculadas às regras são executadas e as ações vinculadas às condições avaliadas como verdadeiras são executadas.

O ECA já é utilizado em aplicações como bancos de dados [22]. O ECA também já foi utilizado como motor de um sistema para processamento de lotes [55]. No BatchKeeper decidimos usar o ECA para gerenciar fluxos que envolvem os processamentos de lotes, podendo ser usado também para estender o funcionamento de outros módulos do BatchKeeper, como implantar diferentes políticas de escalonamento e alterar a alocação de tarefas às máquinas do conjunto de máquinas utilizadas para processamentos.

Com o objetivo de prover flexibilidade para se tomar decisões automaticamente na execução de lotes e tarefas, as condições e ações são escritas como código Python e cadastradas na interface Web do módulo ECA. Como as condições e ações são escritas diretamente como código-fonte, é perigoso executá-las em um ambiente não isolado. Por isso, o módulo ECA aciona o módulo Executor para executar as condições e ações em um contêiner Docker. Assim, o BatchKeeper consegue isolar a execução do código-fonte de cada condição e ação, provendo segurança por construção na execução de código-fonte externo.

O contêiner utilizado para execução de condições e ações contém o banco de dados SQLite, as bibliotecas padrão do Python e uma biblioteca para acesso ao BatchKeeper

instalada. Não é permitida a instalação de outras bibliotecas e pacotes, por padrão. As informações de lotes e tarefas do BatchKeeper podem ser acessadas via interface REST (seção §3.2). A biblioteca do BatchKeeper encapsula a comunicação REST das condições e ações com o servidor do BatchKeeper, para facilitar a extração e alteração de informações de lotes e tarefas. Cada lote é associado ao usuário que o cadastrou. Em cada contêiner criado, é montado um diretório para o usuário que cadastrou o lote vinculado à regra ECA. A partir deste diretório, o usuário consegue manter dados armazenados entre execuções de regras condições e ações. Estes dados podem ser armazenados em arquivos texto ou em um arquivo de banco de dados SQLite, por exemplo. Isto é importante para armazenar informações que podem otimizar execuções de condições e ações. Em particular, um amplo espectro de condições e ações podem ser verificados de forma iterativa em tempo de execução usando estado local [33], o que evita consultas às tabelas de logs do BatchKeeper

Cada condição é atribuída a uma função Python que deve retornar verdadeiro, caso a ação vinculada deva ser executada, ou falso caso contrário. Cada ação é um script Python e apenas retorna a informação sobre se a execução ocorreu com sucesso ou não. As funções vinculadas às condições e os scripts vinculados às ações recebem, via parâmetros, o identificador do registro da tabela de logs que contém o evento disparado e o identificador do lote ou tarefa vinculada à regra ECA, caso um lote ou tarefa esteja relacionado ao evento disparado. De posse dessas informações, as condições e ações poderão adquirir as informações necessárias das tarefas ou lotes relacionados, evento disparado e informações extras que podem ser armazenadas na tabela de logs.

É possível a criação de condições e ações, no módulo ECA, que possam levar em consideração quaisquer atributos de lotes ou tarefas do BatchKeeper. O campo prazo máximo de execução, por exemplo, poderá servir para que ações possam ser configuradas para serem tomadas caso o prazo máximo de execução de um lote seja atingido; isto é necessário em empresas em que a execução de alguns de seus lotes deva ser restrita pelo tempo de execução ou a algum horário específico, ou para que sejam identificados possíveis comportamentos não esperados que possam interferir na execução de outros lotes e tarefas.

O BatchKeeper possui ações básicas predefinidas para o agendamento, reagendamento, início e reinício de tarefas; envio de e-mails, mensagens de texto e notificações via ligações telefônicas. Para adicionar novas condições e ações ao BatchKeeper, basta adicionar os arquivos com o código-fonte de cada condição e ação a um diretório definido no arquivo de configuração do BatchKeeper e cadastrar os nomes dos arquivos de condições e ações na interface web do BatchKeeper.

Alguns exemplos de eventos gerais do BatchKeeper são: perda de comunicação com uma máquina e a atualização de recursos de máquinas. Podemos citar os seguintes eventos como exemplos de eventos relacionados a lotes ou tarefas: alteração do valor de algum atributo de lote ou tarefa; prazo máximo de execução excedido; tempo esperado

de execução excedido e quantidade insuficiente de recursos disponíveis para a execução de alguma tarefa ou lote. Eventos podem ser lançados na tabela de logs e processados pelo módulo ECA. Eventos também podem ser criados através de uma interface REST no ECA. A seguinte configuração de regra ECA é um exemplo de regra possível de ser cadastrada:

- Evento: Quantidade insuficiente de recursos disponíveis para a execução de tarefa ou lote
- Condição: Verificação se a tarefa em questão é crítica para o negócio da empresa
- Ação: Interromper tarefas de lotes de menor prioridade para liberar os recursos computacionais necessários

O ECA foi escolhido para ser o principal responsável pela flexibilização do BatchKeeper por ser possível utilizá-lo para implementar a maioria das automatizações desejadas nos fluxos que envolvem os processamentos em lote e estender funcionalidades chave de outros módulos do BatchKeeper. Além disso, o armazenamento de estado local para condições e ações, presente no módulo ECA do BatchKeeper, facilita melhorias de desempenho na execução de regras ECA. Não temos conhecimento de outras ferramentas para processamento de lotes que utilizem o ECA da mesma forma que é utilizado no BatchKeeper.

3.3.1 Biblioteca para criação de condições e ações

O BatchKeeper disponibiliza interfaces REST para verificar, inserir, alterar e deletar atributos de lotes, tarefas e outras entidades do BatchKeeper. Diversas condições e ações precisam obter dados das entidades do BatchKeeper para cumprirem seus objetivos. Por exemplo, para cadastrar novo lote ou alterar a máquina de execução de alguma tarefa seria necessário fazer requisições às APIs de lote, tarefa e máquina do BatchKeeper.

Para facilitar a implementação de condições e ações às informações do BatchKeeper foi criada uma biblioteca em Python que encapsula o código-fonte necessário para se fazer requisições às APIs do BatchKeeper. Esta biblioteca também facilita a manipulação de lotes, tarefas, máquinas e dependências entre lotes ou entre tarefas, provendo classes para manipulação desses objetos. Assim, a utilização da biblioteca favorece a diminuição do tempo gasto na criação de código-fonte de condições e ações.

A biblioteca é instalada de forma offline a cada instanciamento de novo contêiner para execução de condições e ações. A biblioteca é pequena e instalada de um pacote no

volume de dados utilizado pelo contêiner (i.e., não é necessário o download da biblioteca durante a instalação). Esta abordagem pode ser mais apropriada quando existem alterações frequentes na biblioteca e deseja-se manter a versão da biblioteca sempre atualizada. Porém, a criação de uma imagem para os contêineres contendo a biblioteca já instalada pode ser útil para evitar a instalação da biblioteca à cada instanciamento do contêiner. O apêndice A possui alguns exemplos de utilização da biblioteca em condições e ações.

3.3.2 Integração com sistemas de monitoramento

O módulo ECA disponibiliza uma interface REST para receber eventos externos. Sistemas de monitoramento podem utilizar esta interface para enviar ou obter eventos do BatchKeeper, por exemplo. Há também a possibilidade de sistemas de monitoramento terem acesso direto à tabela de logs do banco de dados do BatchKeeper.

Qualquer sistema de monitoramento que consiga ler dados de um banco de dados ou enviar dados através de uma interface REST poderia ser integrado ao módulo ECA. Geralmente, sistemas de monitoramento executam um agente de monitoramento em cada máquina para coletar os logs do cluster. Estes logs são enviados para o sistema de monitoramento onde são aplicados filtros e transformação de dados. Uma opção para o envio de eventos ao BatchKeeper é utilizar ferramentas de mensageria ou uma plataforma de fluxo (*streaming*) distribuído, como o Apache Kafka e o RabbitMQ. Estas ferramentas poderiam ser utilizadas também para o envio de eventos do BatchKeeper para sistemas de monitoramento. No caso dessas ferramentas citadas, existem conectores que podem conectá-las a bancos de dados ou sistemas através de interfaces REST para o envio e recebimento de dados.

3.4 Executor

O módulo *executor* é responsável por aplicar as instruções recebidas do *escalonador* do módulo *gerenciador de lotes*, isto é, realizar a execução dos lotes e tarefas de acordo com as informações cadastradas no módulo *gerenciador de lotes*, dadas restrições e disponibilidade de recursos computacionais. O *executor* também envia dados de execução de lotes e tarefas para o *gerenciador de lotes*, para que os atributos de lotes e tarefas fiquem atualizados. O *executor* também é responsável por criar contêineres para cada usuário

do BatchKeeper e executar as condições e ações do módulo ECA em seus respectivos contêineres Docker. O *executor* cria os contêineres na máquina do BatchKeeper.

O *executor* distribui as tarefas para execução a agentes, segundo as instruções do escalonador. O *executor* cria um agente para cada tarefa, na máquina escolhida para execução. O agente envia para o *executor* a porta utilizada pelo agente e o IP da máquina em que o agente estiver sendo executado no início de sua execução. Assim o *executor* pode se conectar ao agente e enviar o comando da tarefa a ser executada, além de outros comandos como o de interromper a execução da tarefa em execução, caso seja necessário. Após receber o comando da tarefa a ser executada, o agente é responsável por informar ao *executor* a respeito do progresso de execução. Cada agente também lida com falhas de conexão com o *executor* repetindo a tentativa de conexão periodicamente a cada 10 segundos, enviando dados assim que a conexão for restabelecida. Após enviar o resultado da execução da tarefa ao *executor*, o agente termina sua execução.

O *executor* é responsável por armazenar as informações de execução de cada lote e tarefa, como a data e hora do início da execução de lotes ou tarefas, estado de cada execução dentre outras informações. O *executor* também armazena os lançamentos de eventos relacionados à execução dos lotes ou tarefas. Estas informações também são armazenadas em uma série temporal (*time series*), onde pode-se verificar o momento do lançamento de cada evento da execução de cada lote ou tarefa.

O módulo *executor* mantém uma relação atualizada das máquinas disponíveis para processamentos. O executor faz uma conexão ssh a cada máquina cadastrada no BatchKeeper periodicamente, em um intervalo de tempo parametrizável, sendo o padrão definido como 10 segundos. Ao ocorrer uma falha de conexão a alguma máquina cadastrada, esta máquina é marcada como inativa e não é escolhida pelo *escalonador* para novas execuções de tarefas. Caso não haja falha de conexão, a máquina é marcada como ativa. O executor verifica e armazena as seguintes informações de cada máquina ativa: quantidade de núcleos do processador, memória RAM e espaço disponível em disco.

3.4.1 Previsor de tempo de execução

Um lote pode ter um prazo máximo (*deadline*) de execução definido. Uma situação em que os processamentos em uma empresa são executados apenas na madrugada, aproveitando-se de uma utilização mais baixa de recursos computacionais, é um exemplo em que lotes poderiam ter um prazo máximo de execução definido. Nestes casos, é interessante que o sistema utilizado para o processamento de lotes possa prever, com base em dados históricos de execução de cada tarefa e dados de execução corrente, seu tempo de

execução. Previsões podem ser utilizadas para identificar execuções anormais e antecipar possíveis violações de prazo máximo de execução definidas.

O *executor* possui um previsor de tempo de execução de lotes. O previsor utiliza dados históricos de execução e dados de execução corrente para estimar o término de execução de cada lote. Estas estimativas são usadas para alimentar o módulo ECA através do lançamento de eventos de possíveis perdas de prazo máximo de execução e de atrasos em execuções de lotes ou tarefas.

Utilização de regressão

O previsor utiliza um algoritmo de regressão no cálculo da estimativa do tempo de execução de lotes. O algoritmo de regressão é executado para calcular a função de distribuição, bem como seus respectivos parâmetros, que melhor se assemelha com o comportamento observado dos dados históricos de tempo de execução de cada tarefa. Esta execução é realizada a cada início de execução de cada lote, executando a regressão para cada tarefa do lote em execução. A função de distribuição, e seus respectivos parâmetros, encontrada pelo algoritmo de regressão é utilizada para capturar o comportamento histórico de execução de lotes ou tarefas.

O Algoritmo 1 contém o pseudo-código do algoritmo de regressão. Este algoritmo utiliza o *ScyPy* para a execução das regressões. Para isso, o BatchKeeper envia os dados históricos de uma tarefa e uma lista de distribuições a serem testadas. O *ScyPy* é usado para testar cada função de distribuição recebida pelo BatchKeeper, variando seus respectivos parâmetros, e retorna a distribuição e seus respectivos parâmetros que mais se assemelha à curva do histórico de execução de cada tarefa, segundo o teste Kolmogorov-Smirnov [42]. Todas as funções de distribuições disponíveis na biblioteca *ScyPy* [63] podem ser utilizadas na regressão das tarefas. Por padrão, o BatchKeeper utiliza as seguintes funções para a execução do algoritmo de regressão: Log normal; Normal; Weibull exponencial; Weibull máximo; Weibull mínimo; Pareto; Valor extremo generalizado; Exponencial; Gamma e Uniforme. Estas funções foram escolhidas por serem funções mais clássicas e terem se mostrado adequadas à maioria das distribuições de tempo de execução no histórico dos lotes utilizados na avaliação. Além disso, utilizar distribuições clássicas evita possíveis overfittings quando existem poucos dados de execução.

Cálculo da estimativa de tempo de execução de lotes

O previsor de tempo de execução prevê o término de execução de cada lote, durante as execuções destes lotes. O Algoritmo 2 contém o pseudo-código do algoritmo do previsor de tempo de execução de lotes. Este algoritmo utiliza um grafo para mapear a execução de tarefas. Este algoritmo atualiza os pesos das arestas de acordo com a duração real ou estimativa de duração de cada tarefa de acordo com o algoritmo de regressão. A

Algoritmo 1: Pseudo-código do algoritmo de regressão

Entrada: lista de dados historicos de execução de uma tarefa e conjunto $\mathcal{D}$ de funções estatísticas a serem consideradas

Saída : Melhor distribuição estatística e seus respectivos parâmetros, que melhor se aproxima do comportamento histórico da execução da tarefa de entrada

```
1  melhorDistribuicao ← ∅
2  parametrosMelhorDistribuicao ← ∅
3  melhorScore ← ∅
4  para cada funcao f ∈ D faça
5      parametros ← FIT(f, dadosHistoricosTarefa)
6      score ← SCORE(dadosHistoricosTarefa, f, parametros)
7      se score > melhorScore então
8          melhorDistribuicao ← f
9          melhorScore ← score
10         parametrosMelhorDistribuicao ← parametros
11 retorna melhorDistribuicao, parametrosMelhorDistribuicao
```

estimativa de término de execução de um lote é calculada como a maior duração entre os caminhos possíveis no grafo. O algoritmo do previsor lança eventos no módulo ECA para avisar sobre prováveis violações de prazos máximos de execução de lotes ou em caso de confirmação destas violações.

O grafo de execução utilizado neste algoritmo é o grafo direcionado de dependência entre tarefas, cadastrado no *gerenciador de lotes*, acrescido de um vértice raiz e um vértice final. O vértice raiz possui arestas para os vértices referentes às tarefas que não são dependentes, isto é, que podem ser executadas no início do processamento do lote. Todas os vértices de tarefas que não são dependentes de outras tarefas, ou seja, que não possuem arestas para outros vértices, são ligados ao vértice final.

Os pesos das arestas dos grafos referentes aos lotes em execução são calculados continuamente. Os pesos das arestas incidentes em vértices referentes às tarefas com execução já finalizada são atualizados com a duração exata, em segundos, da execução. Os pesos das arestas incidentes em vértices referentes às tarefas em execução são atualizados com o valor máximo entre o tempo de execução da tarefa e o 80^{o} percentil da função de distribuição parametrizada através do algoritmo 1. Os pesos das arestas incidentes em vértices referentes às tarefas ainda não executadas são atualizados com o valor do 80^{o} percentil da função de distribuição gerada. A previsão do término de execução de um lote é dada pela soma entre a data e hora do início da execução do lote e o caminho entre o vértice inicial e o vértice final com maior peso, calculado através de um algoritmo modificado de busca em profundidade.

A figura 3.2 mostra um exemplo de grafo utilizado pelo previsor em um lote de 6 tarefas. Considere que, em um determinado momento, os vértices 2 e 4 tiveram suas

Algoritmo 2: Pseudo-código do algoritmo de previsão de tempo de execução de lotes em execução

Entrada: Lote, grafo G de tarefas do lote, conjunto D de funções estatísticas a serem consideradas, data hora atual

```
1  para cada tarefa ∈ G faça
2      dadosHistoricosTarefa ← BuscaDadosHistoricos(tarefa)
3      distr, params ← regressao(tarefa, dadosHistoricosTarefa, D)
4      se tarefa.statusFinalizada então
5          tarefa.peso ← tarefa.fim − tarefa.inicio
6      senão
7          tarefa.peso ← max(tarefa.fim − tarefa.inicio, p80(distr, params))
8      se tarefa.prazoMaximo < dataHoraAtual então
9          EnviaEventoPrazoMaximoExcedido(tarefa)
10 duracaoEstimadaLote ← calculaMaiorCaminho(G)
11 se lote.prazoMaximo < dataHoraAtual então
12     EnviaEventoPrazoMaximoExcedido(lote)
13 se lote.prazoMaximo < dataHoraAtual então
14     EnviaEventoPrazoMaximoProvavelmenteExcedido(lote)
15
```

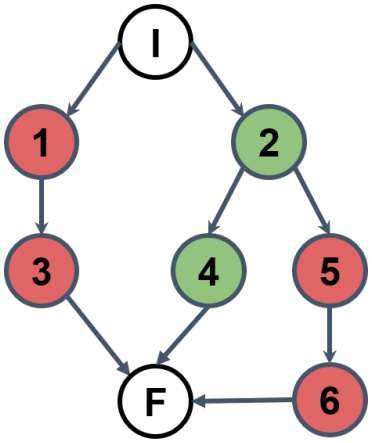


Figura 3.2: Exemplo de grafo de execução utilizado pelo previsor

execuções finalizadas (verde) e os vértices 1 e 5 estejam aguardando execução ou em execução (vermelho). Neste caso, os pesos das arestas incidentes nos vértices 2 e 4 serão iguais à duração real da execução destas respectivas tarefas, em segundos. Os pesos das outras arestas serão atualizadas de acordo com o valor máximo entre a duração corrente e o 80º percentil da função de distribuição calculada pelo algoritmo de regressão.

Utilizamos o 80º percentil por padrão pois, com esse valor, o previsor conseguiu prever o término das execuções de lotes na implantação em produção do BatchKeeper de forma mais próxima ao real término das execuções. Porém, a escolha deste percentil é parametrizável e deve variar de acordo com o comportamento da duração da execução de lotes de cada implantação. É importante a escolha de um valor de percentil que

permita que a estimativa do tempo de execução de cada lote funcione como um limite superior do tempo de execução de cada lote. Porém, esse percentil é parametrizado Caso essa estimativa ultrapasse o prazo máximo de execução de algum lote o *previsor* lança o evento "O prazo máximo de execução provavelmente será excedido" para que possa ser possível programar ações para lidar com esse cenário.

3.4.2 Eventos lançados pelo Executor

O módulo ECA verifica todos os novos registros da tabela de logs, podendo-se criar regras para executar condições e ações no lançamento de qualquer evento lançado. O *executor* lança eventos na tabela de logs do BatchKeeper ao constatar o estouro do prazo de execução de algum lote ou tarefa, ou quando existe uma estimativa de limite superior para o término de execução que exceda o prazo máximo de execução. Além disso, pode ser configurado o lançamento de eventos pelo *executor* ao detectar a violação do tempo de execução referente a um grupo de percentis. Isso pode ser útil caso seja desejável o envio de alertas sobre o andamento de processamentos que possuem poucos dados históricos e, por isso, tem estimativa de limite superior do tempo de execução menos precisa, por exemplo. Assim, a equipe responsável poderia receber alertas anteriores de possível violação do prazo máximo de execução de alguma tarefa.

O *executor* também lança eventos de controle de execução de lotes e tarefas, como o início, término e quaisquer alterações de status de execução de lotes ou tarefas. O *executor* também lança eventos de alteração de estado da comunicação entre os agentes de execução de tarefas, ou máquinas pertencentes ao cluster de execução, e o *executor*. Quaisquer atualizações de recursos computacionais livres no conjunto de máquinas utilizadas para processamentos, também são lançadas como eventos na tabela de logs do banco de dados do BatchKeeper.

3.4.3 Tolerância a falhas

O BatchKeeper é transacional, armazenando todos os seus dados em um banco de dados. Por isso, caso a execução do servidor do BatchKeeper seja interrompida ou o BatchKeeper trave por algum motivo, o BatchKeeper é capaz de continuar a escalonar a mesma fila de execução e se comunicar com os agentes de execução da mesma forma que

antes de ser reiniciado.

O BatchKeeper precisa que o banco de dados esteja sempre disponível para funcionar normalmente. Caso o banco de dados fique indisponível, o BatchKeeper também ficará indisponível. Por isso, é importante a replicação distribuída do banco de dados do BatchKeeper. A utilização de um banco de dados local, em caso de falha do banco de dados principal está prevista como um trabalho futuro.

Os agentes de execução de lotes e tarefas são executados nas máquinas de processamentos. Caso ocorra algum problema de conexão entre estes agentes e o servidor do BatchKeeper durante a execução de alguma tarefa, tão logo esta conexão seja restabelecida, as informações de execuções dos processamentos serão enviadas pelos agentes para o servidor do BatchKeeper, através de uma interface REST do *executor*. Comandos de interrupção de execução também serão executados pelos agentes assim que houver conectividade entre estes e o *executor*, caso haja algum problema de conexão. Cada agente só termina sua execução após enviar os resultados dos processamentos ao *executor*. Caso um agente tenha sua execução interrompida por alguma falha ou desligamento da máquina virtual, esse não poderá recuperar o seu estado. Nesse último caso, o operador do BatchKeeper deverá analisar o impacto na interrupção da execução da tarefa devido ao desligamento da máquina.

3.4.4 Segurança

O *executor* executa o código-fonte de condições e ações em contêineres Docker para isolar o ambiente de execução de código-externo do próprio ambiente de execução do BatchKeeper, trazendo segurança ao ambiente de execução. Toda a comunicação entre condições ou ações e os demais módulos do BatchKeeper ocorrem através de interfaces REST. Estas interfaces podem ser protegidas através de diversos mecanismos de autenticação, como OAuth1 e OAuth2, impedindo que condições ou ações acessem interfaces ou operações destas interfaces indevidamente [36].

Contudo, devido a alta flexibilidade trazida pelo BatchKeeper, é recomendável que todo código-fonte de condição ou ação seja inspecionado para a correta definição das permissões de acesso às interfaces necessárias para cada finalidade de cada condição ou ação. É necessário avaliar cada condição ou ação para que suas execuções não tragam comportamentos indesejáveis ao sistema. Por exemplo, caso uma condição e ação seja escrita com o intuito de alterar apenas dados de um grupo específico de lotes, deve-se verificar se esse código-fonte possa alterar dados de outros grupos de lotes pois o BatchKeeper não provê mecanismos de segurança relacionado a inspeção de código-fonte,

existindo mecanismos de segurança na granularidade de autenticação das interfaces REST e isolamento da execução de código-fonte.

O *executor* provê diretórios para armazenamento de estado por condições ou ações para cada usuário do BatchKeeper. Estes diretórios são construídos como volumes Docker, sendo possível para cada usuário o acesso apenas ao seu diretório. Assim, não há como condições ou ações de um usuário acessar arquivos armazenados em diretórios de outros usuários.

3.5 Limitações

O algoritmo de escalonamento do BatchKeeper possui o foco principal de ordenar a execução de lotes segundo prioridades manualmente definidas. Porém, não há garantias de que, em tempo de execução dos lotes, haverão recursos suficientes livres para executar todas as tarefas cadastradas. O escalonador do BatchKeeper só toma decisões em relação a execução de um lote no momento em que esse tiver sido agendado para ser executado, podendo não haver recursos computacionais livres o suficiente para todas as tarefas do lote. O BatchKeeper foi pensado para ser utilizado em ambientes com máquinas sem exclusividade de recursos computacionais para processamentos em lote, podendo suas máquinas serem utilizadas para outros fins durante os processamentos de lotes.

Apesar das imagens para execução de condições e ações serem modificáveis, o módulo *executor* não permite a personalização da imagem do contêiner utilizado para executar condições e ações através das interfaces do BatchKeeper. Por isso, caso alguma condição ou ação seja dependente de alguma biblioteca que não seja a do BatchKeeper ou do conjunto padrão de bibliotecas do Python, o código-fonte não poderá ser executado sem modificação da imagem utilizada por todas condições e ações.

O BatchKeeper não provê alta disponibilidade durante a ocorrência de falhas pois não funciona em múltiplas instâncias simultaneamente. Estas limitações não são fundamentais e podem ser sanadas em futuras atualizações do BatchKeeper

Capítulo 4

Avaliação

Um dos objetivos da avaliação é avaliar o algoritmo previsor de tempo de execução de lotes e tarefas para empresas que possuem um parque computacional heterogêneo e um conjunto de lotes com prazos máximos de execução definidos e cujas tarefas podem possuir dependências. Este algoritmo é responsável por detectar possíveis violações de prazo máximo de execução de lotes ou tarefas. A partir dessa detecção, eventos são lançados na tabela de logs do banco de dados do BatchKeeper, permitindo a configuração de regras ECA que possam lidar com regras de negócio relacionadas aos prazos dos processamentos de lotes.

A avaliação do algoritmo previsor de limite superior de tempo de execução de lotes e tarefas foi realizada utilizando dados reais (seção 4.1) de processamentos de lotes de uma empresa. Foi utilizada a técnica validação cruzada (k-fold cross-validation) para atestar a eficiência deste algoritmo (seção 4.2.1).

O módulo ECA também foi avaliado de forma qualitativa através de demonstrações sobre a flexibilidade em ser utilizado para adaptar o BatchKeeper à diversas situações envolvidas em processamentos em lote (seção 4.4).

4.1 Conjunto de dados

Desenvolvemos o BatchKeeper como uma solução genérica, que pode ser aplicada em vários contextos, e o aplicamos em uma empresa do mercado financeiro que presta serviço a outras empresas. Esta empresa permitiu a coleta de metadados de processamentos de seus lotes. Cada lote refere-se ao processamento de dados de determinada empresa cliente da empresa avaliada. Cada empresa cliente da empresa avaliada é denominada aqui como “empresa cliente”. Os metadados dos processamentos contêm informações de duração de cada processamento, ocorrência de falhas, data e hora de início e fim de cada execução, nome da empresa cliente e categoria de cada tarefa executada. A descrição dos metadados encontra-se na tabela 4.1.

Coluna	Descrição
Empresa cliente	O nome da empresa cliente (anonimizado)
Categoria do lote	Os lotes podem ser categorizados em duas categorias: “Exportação” e “Importação”
Início	Data e hora do início da execução da tarefa
Fim	Data e hora do fim da execução da tarefa
Erro	Flag para indicar se ocorreu algum erro crítico no processamento
Duração	A duração da execução da tarefa em segundos

Tabela 4.1: Descrição dos metadados do conjunto de dados utilizado para a avaliação do BatchKeeper

A empresa considerada para avaliação do BatchKeeper possui centenas de empresas que são suas clientes e processa os dados destas empresas todos os meses. São processados dois lotes distintos, em diferentes dias de cada mês, para cada empresa cliente. A coluna “categoria do lote” contém a categoria da execução do lote. Os nomes das categorias foram alterados para anonimizar a atividade da empresa avaliada. Definimos os lotes como “exportação” e “importação”. O lote de exportação possui duas tarefas. O lote de importação possui cinco tarefas.

Para a avaliação, os lotes de exportação e importação, referentes a cada empresa cliente presente no conjunto de dados, foram cadastrados no BatchKeeper através de um script. As execuções das tarefas dos lotes de cada empresa cliente, definidas por cada registro do conjunto de metadados, foram importadas no BatchKeeper.

4.1.1 Caracterização

A base de dados utilizada para avaliação possui 28159 registros de lotes executados com sucesso, contendo 92381 execuções de tarefas. Apenas os registros referentes a lotes executados com sucesso foram utilizados na avaliação. Assim, foi possível treinar o previsor de tempo de execução de lotes para identificar um modelo de processamentos que foram executados corretamente. A partir deste modelo, foi possível detectar prováveis violações de prazo máximo de execução.

Existem registros de processamentos relacionados a 342 empresas clientes. Das 342 empresas clientes, há registros de processamentos de lotes de importação de 63 empresas, além de processamentos de lotes de exportação de 103 empresas, por pelo menos 5 anos (60 meses).

Existem processamentos que demoram poucos minutos e outros que demoram muitas horas. Mais especificamente, 96% dos processamentos realizados pela empresa avaliada duram menos do que 1 hora. A grande disparidade no tempo de execução deve-se à variação da quantidade de informações processadas e ambiente de execução dos pro-

cessamentos. Os processamentos mais demorados possuem grande quantidade de dados processados e são executados em máquinas mais lentas, controladas pelas respectivas empresas clientes. Os processamentos de lotes de importação são sempre mais demorados do que processamentos de lotes de exportação. Isso ocorre pois o processamento de exportação consiste numa exportação de informações do banco de dados com poucos cálculos envolvidos, enquanto o processamento de importação consiste na importação de arquivos, criação de registros, e cálculos complexos de grande parte dos usuários do sistema da empresa avaliada. A figura 4.1 mostra a distribuição do tempo de execução (total, incluindo todas as tarefas) entre lotes de importação e exportação.

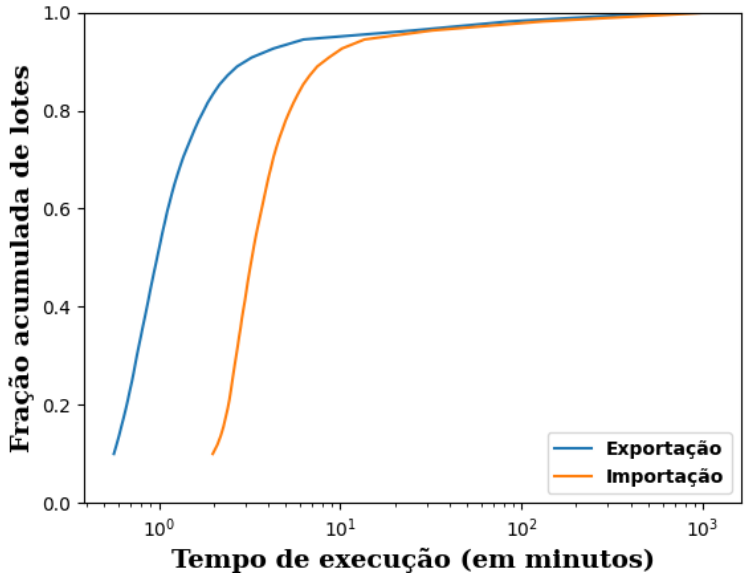


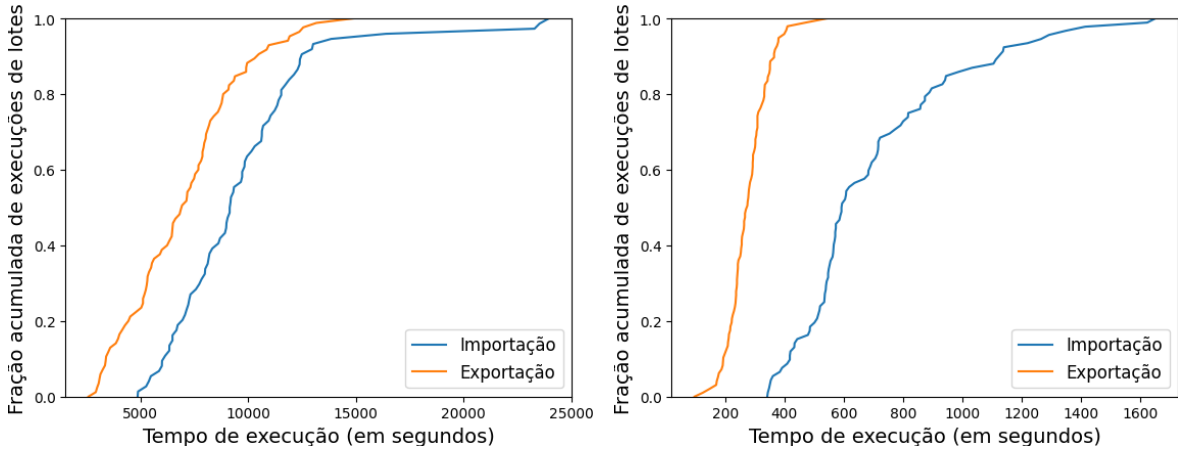
Figura 4.1: Distribuição acumulada do tempo de execução de processamentos de lotes de importação e exportação. As curvas não foram suavizadas.

Todos os processamentos de lotes de curto prazo ocorreram em duas máquinas de alto desempenho. A maioria dos processamentos de longo prazo ocorreram em máquinas mais lentas, porém dedicadas para cada empresa cliente. Não há isolamento de recursos computacionais nas máquinas utilizadas para os processamentos.

Os processamentos de lotes de importação possuem o prazo máximo de execução de 14 horas. Os processamentos de lotes de exportação possuem o prazo máximo de execução de 8 horas. Todas as execuções de lotes e eventuais intervenções humanas, que possam ser necessárias na ocorrência de erros nas execuções de lotes, devem ser realizadas dentro destes prazos. Quando é identificada uma provável violação do prazo máximo de execução de algum lote, esse é interrompido e inicia-se uma investigação sobre as causas da demora na execução do lote para que esta demora não ocorra em uma nova tentativa de processamento no dia seguinte.

Cliente	Tipo do lote	Média - duração	Mediana - duração	Qtd de execuções
3	Exportação	6957.97	6901.5	86
	Importação	9674.69	9143	75
6	Exportação	276.81	272.5	98
	Importação	698.20	592	93

Tabela 4.2: Quantidade de dados históricos, média e mediana de lotes de exportação e importação de duas empresas clientes.



(a) Empresa 3 (tarefas longas)

(b) Empresa 6 (tarefas curtas)

Figura 4.2: Distribuição acumulada da duração de processamentos dos lotes de importação e exportação

A tabela 4.2 mostra dados estatísticos de lotes de duas empresas clientes. As empresas estão identificadas por um número identificador. A empresa 3 é um exemplo de empresa cliente cujos lotes processam grandes quantidades de dados, possuindo um longo tempo de execução. Os lotes da empresa 6 são considerados lotes de curto tempo de execução.

Todos os processamentos dos lotes da empresa 3 ocorreram em uma máquina mais lenta. Esta máquina não era utilizada para servir processamentos de outras empresas clientes. Os scripts de processamentos, uma instância de software web e a instância do banco de dados estavam presentes em uma única máquina, não havendo isolamento de recursos computacionais entre estes itens. Por isso, durante as execuções dos lotes, poderiam haver variações significativas no tempo de execução dos lotes, caso o banco de dados fosse utilizado fortemente por usuários ou aplicações não relacionadas aos processamentos. Um exemplo deste caso seria a utilização do banco de dados para a geração de relatórios diversos ou execução de consultas computacionalmente pesadas. Enquanto a maioria dos processamentos desta empresa cliente tiveram duração semelhante, houve alguns casos em que esta duração foi bem superior à duração média, conforme mostra a figura 4.2(a).

Lotes de curto tempo de execução são executados em máquinas onde existem centenas de instâncias do software da empresa avaliada referentes às empresas clientes diversas.

Nestes casos, o uso das instâncias de bancos de dados é compartilhado, armazenando dados de centenas de empresas clientes em cada instância. Não há execução paralela de lotes de curto tempo de execução nessas máquinas, porém, as instâncias de bancos de dados continuam sendo utilizadas por outras instâncias do software web durante a execução de algum lote. Existem execuções muito mais demoradas do que as médias de execuções desses lotes por causa do não isolamento de recursos computacionais entre as aplicações que rodam nessas máquinas e pelo compartilhamento das instâncias do banco de dados. Os lotes de curto tempo de execução possuem um comportamento no tempo de execução semelhante entre si; esses lotes duram poucos minutos, mas existe uma quantidade significativa de execuções cujos tempos de execução estão muito acima de suas respectivas médias ou medianas. Um exemplo de comportamento do tempo de execução de um lote de curto tempo de execução pode ser visto na figura 4.2(b).

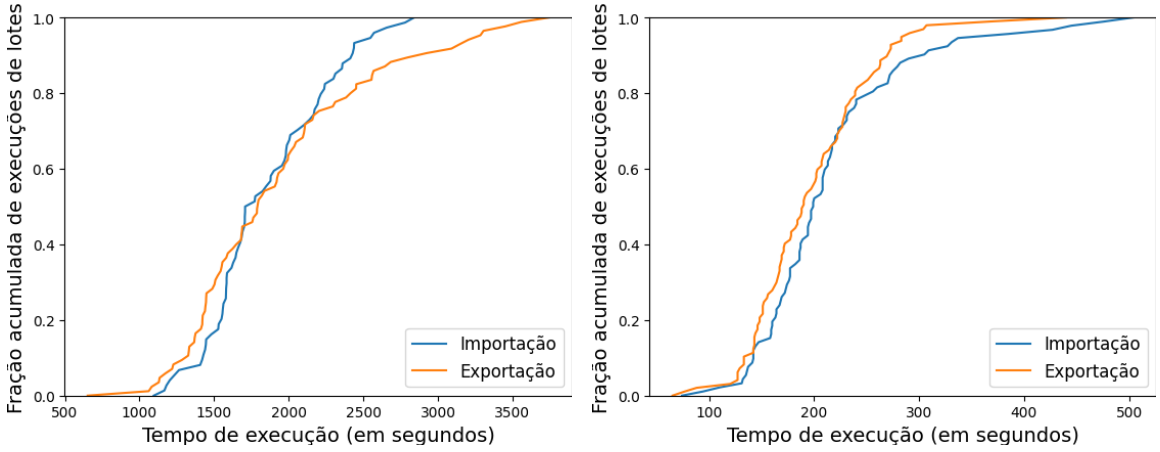
As tarefas 0 dos lotes de exportação e importação constituem um backup dos dados, da empresa cliente associada à tarefa, do banco de dados. Esse backup é realizado para que, caso haja algum problema no processamento dos dados, seja possível retornar o banco ao seu estado anterior.

O código-fonte executado nas tarefas 0 é quase idêntico em ambos os tipos de lotes. Por isso, o tempo de execução dessa tarefa é parecido nos lotes de exportação e importação tanto em lotes de longo tempo de execução quanto em lotes de curto tempo de execução. A figura 4.3(a) mostra o tempo de execução da tarefa 0 em um lote de exportação e de longo tempo de execução e de um lote de importação e longo tempo de execução da empresa 3. Já a figura 4.3(b) mostra o tempo de execução da tarefa 0 em lotes de importação e exportação, e de curto tempo de execução, da empresa 6. A figura 4.3(a) mostra que existiram algumas execuções de lotes de exportação que demoraram significativamente mais do que as execuções de lotes de importação, porém acreditamos que essa variação ocorreu pela falta de isolamento de recursos computacionais no ambiente de execução.

A tarefa 1 do lote de exportação realiza cálculos financeiros e uma exportação de dados no formato csv. Esta tarefa possui a maior carga de trabalho do lote de exportação. Esta tarefa teve a mediana do tempo de execução de 3601 segundos em lotes da empresa 3, sendo quase o dobro da mediana do tempo de execução da tarefa 0. A figura 4.4(a) mostra a distribuição acumulada da duração de processamentos da tarefa 1 do lote de exportação da empresa 3.

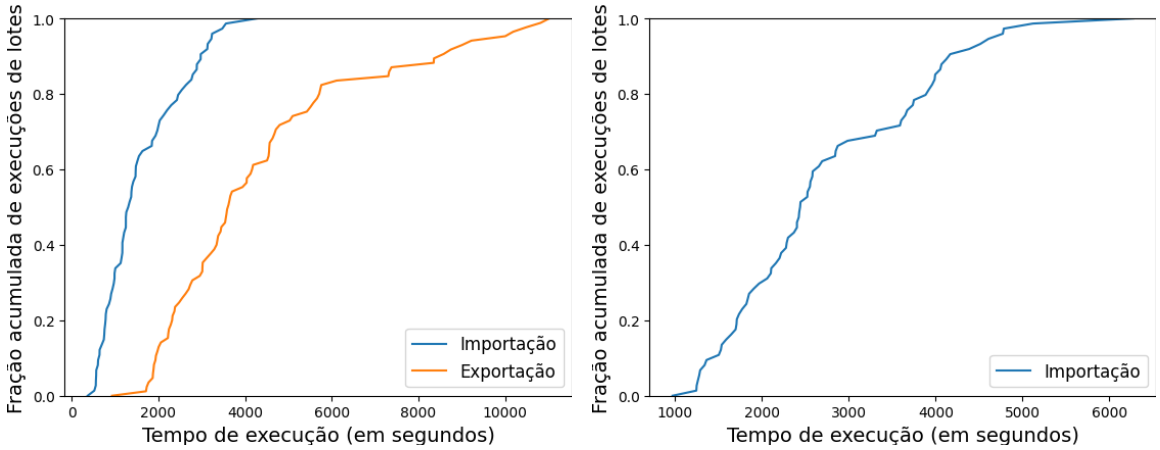
A tarefa 1 do lote de importação realiza cadastros de usuários no software da empresa avaliada. Durante os cadastros, são executados cálculos pouco complexos. Por isso, esta tarefa possui uma menor variância no tempo de execução quando comparado à execução de outras tarefas deste lote. A figura 4.4(a) também mostra a distribuição do tempo de execução da tarefa 1 do lote de importação da empresa 3.

A tarefa 2 do lote de importação realiza os cálculos mais complexos e utiliza muito



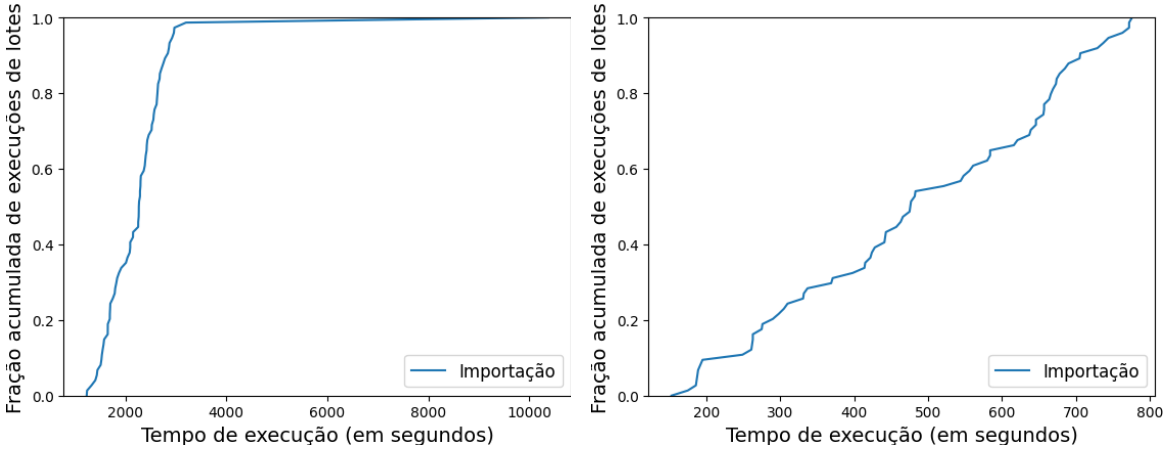
(a) Empresa 3 - Tarefa 0 (b) Empresa 6 - Tarefa 0

Figura 4.3: (a) Distribuição acumulada da duração de processamentos das tarefas 0 dos lotes de importação e exportação da empresa cliente 3; (b) Distribuição acumulada da duração de processamentos das tarefas 0 dos lotes de importação e exportação da empresa cliente 6.



(a) Empresa 3 - Tarefa 1 (b) Empresa 3 - Tarefa 2

Figura 4.4: (a) Distribuição acumulada da duração de processamentos das tarefas 1 dos lotes de importação e exportação da empresa cliente 3; (b) Distribuição acumulada da duração de processamentos da tarefa 2 do lote de importação da empresa cliente 3.



(a) Empresa 3 - Tarefa 3 (b) Empresa 3 - Tarefa 4

Figura 4.5: (a) Distribuição acumulada da duração de processamentos da tarefa 3 do lote de importação da empresa cliente 3; (b) Distribuição acumulada da duração de processamentos da tarefa 4 do lote de importação da empresa cliente 3.

o banco de dados, sendo a tarefa mais demorada deste lote. Por isso, a tarefa 2 é a tarefa que possui a maior variação no tempo de execução em lotes de importação. A figura 4.4(b) mostra a distribuição do tempo de execução da tarefa 2 do lote de importação da empresa 3.

A tarefa 3 é considerada uma extensão da tarefa 2. Porém, sua carga é menor do que a carga da tarefa 2 e possui baixa variância do tempo de execução. A figura 4.5(a) mostra a distribuição do tempo de execução da tarefa 3 do lote de longo prazo da empresa 3.

A tarefa 4 constitui-se da execução de scripts adicionais de processamento; não existe um padrão de conjunto destes scripts, existindo lotes de empresas em que nenhum script é executado nesta tarefa e outros em que scripts muito demorados são executados. Por isso, é a tarefa com a maior variação no tempo de execução entre lotes. A figura 4.5(b) mostra a distribuição do tempo de execução da tarefa 4 do lote de importação da empresa 3.

A alta variação no tempo de execução de muitos lotes, a baixa quantidade de registros de execução de alguns lotes e a falta de isolamento de recursos computacionais no ambiente de execução de lotes faz com que o conjunto de dados seja desafiador na avaliação do algoritmo do previsor.

4.1.2 Frequência de erros (intervenções humanas)

A quantidade de intervenções humanas necessárias é igual ao número de processamentos com falhas. Isso ocorre porquê não existia, até a implantação do BatchKeeper, automatizações para lidar com falhas nos processamentos, exigindo uma intervenção humana. A base de dados cedida pela empresa avaliada só possui informações sobre intervenções humanas em lotes de importação e de longo tempo de execução. A quantidade de intervenções humanas por lote está na tabela 4.3.

Cliente	Tipo do lote	Qtd de execuções	Intervenções humanas	Intervenções/Execuções
0	Importação	68	4	5,88%
1	Importação	48	9	18,75%
2	Importação	31	8	25,81%
3	Importação	75	3	4,00%
4	Importação	27	15	55,56%

Tabela 4.3: Taxas de intervenções humanas em lotes de importação e de longo tempo de execução

Através da tabela 4.3, pode-se observar que há lotes de alto tempo de execução com altas taxas de intervenções humanas. As altas taxas de intervenções humanas justificam a necessidade de se utilizar o módulo ECA para a automatização dos fluxos de processamentos em lote.

4.2 Previsor de tempo de execução

Como os processamentos desta empresa são mensais, demoraria muito tempo para se obter uma amostra recente grande o suficiente de cada lote para a realização da avaliação do algoritmo do previsor. Por isso, o conjunto de dados cedidos pela empresa avaliada foi inserido diretamente nas tabelas de histórico de processamentos do BatchKeeper.

A partir do histórico de processamentos nas tabelas do BatchKeeper, utilizamos avaliação cruzada para avaliar a eficiência do algoritmo do previsor em criar previsões que atuam como um limite superior de tempo de execução. Dizemos que o previsor é eficiente quando a previsão de término de execução de um lote for maior ou igual à duração real deste lote, variando os conjuntos de teste e treino na avaliação cruzada.

4.2.1 Validação cruzada de k vias

K-Fold cross validation é uma técnica muito utilizada para validar a eficiência de algoritmos, particularmente em recuperação de informação e aprendizado de máquina [61]. Esta técnica consiste em separar um conjunto de dados $\mathcal{D}$ em K subconjuntos de mesmo tamanho; para cada subconjunto $d_i \subset \mathcal{D}$, o algoritmo utiliza os dados em $\mathcal{D} \setminus d_i$ para treino e utiliza os dados em d_i para teste. Após a separação dos conjuntos de teste e treino, o modelo a ser avaliado é treinado com os elementos do conjunto de treino e depois avaliado nos elementos do conjunto de teste. Em cada iteração deste algoritmo o subconjunto de teste é diferente de todos os outros subconjuntos de teste anteriores, utilizando cada subconjunto de teste somente uma vez. À cada vez em que o modelo é avaliado, a acurácia é armazenada para posterior cálculo da média e mediana.

O K-Fold cross validation foi utilizado para avaliar a exatidão do algoritmo do previsor. O algoritmo K-Fold cross validation foi executado com o parâmetro k = 10. Apenas os dados de execuções de lotes e tarefas cujas execuções tenham sido terminadas com sucesso foram utilizadas na validação do algoritmo do previsor. Em cada etapa de execução do algoritmo de validação cruzada, foram simuladas as execuções do previsor como se estivessem sendo executadas antes da execução de cada tarefa presente no lote em avaliação. Assim, pôde-se obter estimativas próximas das que seriam obtidas na execução real do lote em vários momentos da execução do mesmo. Nesta avaliação, foi calculada a diferença entre a estimativa e o real tempo de execução de cada tarefa de cada lote, sendo considerada esta diferença como o erro do previsor.

O algoritmo do previsor é mais útil em processamentos de lotes de longo tempo de execução porque a alta duração de lotes aumenta a probabilidade de violações de prazos máximos de execução. Por isso, executamos o algoritmo K-Fold cross validation para todas as empresas com lotes de longo tempo de execução e para apenas uma empresa com lotes de curto tempo de execução.

4.2.2 Regressão

O algoritmo do previsor executa o algoritmo de regressão para identificar as distribuições estatísticas, e seus respectivos parâmetros, que mais se aproximam do comportamento histórico dos lotes em execução. O algoritmo de regressão utilizou o conjunto padrão de distribuições estatísticas utilizadas pelo BatchKeeper (que utiliza o *ScyPy*), durante a avaliação do algoritmo do previsor.

Cliente	Tipo do lote	Qtd de execuções
0	Exportação	67
0	Importação	68
1	Exportação	92
1	Importação	48
2	Exportação	51
2	Importação	31
3	Exportação	86
3	Importação	75
4	Exportação	54
4	Importação	27
5	Exportação	66

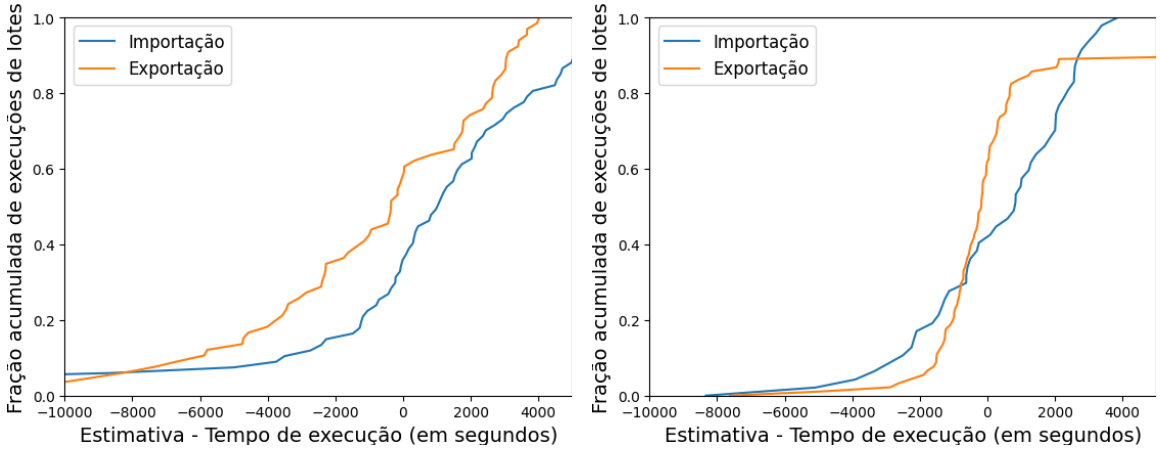
Tabela 4.4: Quantidade de execuções de lotes de longo tempo de execução no conjunto de dados da avaliação

4.3 Resultados

A eficácia do algoritmo do previsor de tempo de execução de lotes está diretamente ligada à quantidade de dados históricos bem como ao cálculo da distribuição estatística, e seus respectivos parâmetros, mais adequada aos dados históricos de execução de tarefas por lote.

A partir da execução do algoritmo de validação cruzada, obtivemos 55699 estimativas de limite superior de tempo de execução de lotes da empresa avaliada. Para cada estimativa, computamos a eficácia do algoritmo como a diferença entre a estimativa do limite superior do tempo de execução e o tempo real do término da execução de cada lote. Se esta diferença for positiva, isso significa que a estimativa de limite superior foi maior do que a duração real do lote. Como o algoritmo do previsor é executado durante toda a execução dos lotes e as tarefas dos lotes da empresa avaliada são executadas sequencialmente, separamos os dados obtidos na validação cruzada por rodadas. Cada rodada significa que uma tarefa a mais foi marcada como já executada na execução do algoritmo do previsor. Exemplo: Os resultados obtidos na rodada 0 foram obtidos em um cenário em que nenhuma tarefa havia sido executada. Já em um lote com cinco tarefas em que o previsor foi executado na rodada três, o previsor levou em consideração que as três primeiras tarefas já haviam sido executadas, restando apenas duas para execução.

O algoritmo do previsor de tempo de execução de lotes utiliza o 80^o percentil no cálculo do limite superior, conforme a seção 11. Por isso, esperávamos que o limite superior fosse maior do que aproximadamente 80 % das execuções reais, nos resultados do algoritmo de validação cruzada. Computamos a CDF de lotes de longo tempo de execução a partir dos dados obtidos na rodada 0 do algoritmo do previsor. Nesse caso, pôde ser observado que houve lotes de longo tempo de execução em que o limite superior foi maior que aproximadamente 80% das execuções reais, conforme as figuras, 4.7(a) (lotes de importação e exportação) e 4.7(b) (lote de importação). Porém, houve casos em que



(a) Empresa 0 (b) Empresa 1

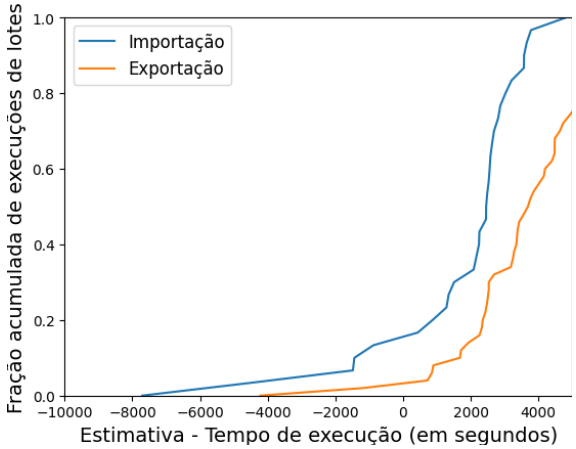
Figura 4.6: (a) Distribuição acumulada dos erros em previsões em lotes de importação e exportação da empresa cliente 0; (b) Distribuição acumulada dos erros em previsões em lotes de importação e exportação da empresa cliente 1.

mais de 20% dos lotes tiveram um tempo de execução maior do que o tempo calculado pelo algoritmo do previsor, conforme pode ser visto nas figuras 4.6(a) (lotes de importação e exportação) e 4.6(b) (lotes de importação e exportação). Acreditamos que esse comportamento ocorre devido à quantidade de dados históricos ser inferior ao necessário para que o comportamento destes dados convirja para uma distribuição estatística. A tabela 4.4 mostra a quantidade de dados históricos de lotes de longo tempo de execução por empresa cliente. Como o algoritmo do previsor utiliza o 80^o percentil das funções de distribuições das tarefas não executadas em seu cálculo, os erros acumulados de estimativas referente a cada tarefa ainda não executada pode reduzir a exatidão da previsão do tempo de execução do lote.

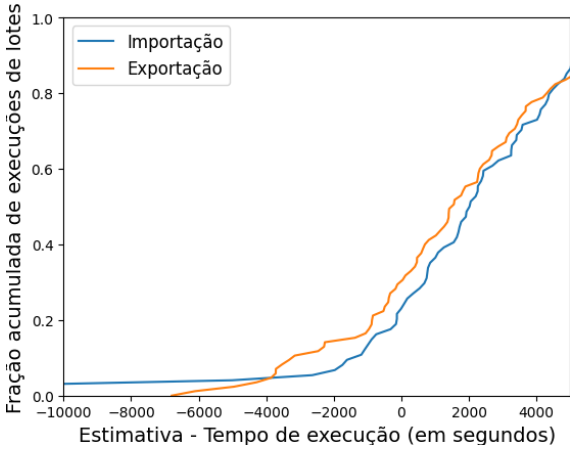
Também realizamos o mesmo experimento com dados da empresa 6, que possui lotes de curto tempo de execução. A figura 4.7(c) mostra o comportamento do tempo de execução dos dois lotes dessa empresa. Nesse caso, o previsor subestimou o tempo de execução em cerca de 21% dos casos em lotes de importação, com a mediana em 301 segundos, e 25% em lotes de exportação, com a mediana em 44 segundos.

Observamos que o previsor subestimou o tempo de execução de todos os lotes avaliados em 63,9% dos casos. Em lotes com menos de 50 execuções no histórico de execuções, esse número chegou a 77,68% dos casos. Em lotes com mais de 50 execuções, esse número caiu para 54,29%. Em lotes de longo tempo de execução, 33,5% das execuções totais foram subestimadas, tendo o previsor atuado como um limite superior de tempo de execução com sucesso em 66,5% dos casos. Não houve diferença significativa na eficácia do previsor em lotes de longo tempo de execução entre lotes com menos de 50 execuções e lotes com mais de 50 execuções.

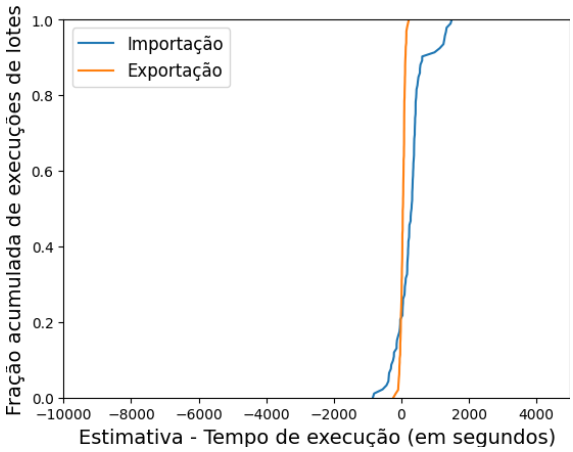
Observamos que a partir do conjunto de dados utilizado na avaliação, a quantidade



(a) Empresa 2



(b) Empresa 3



(c) Empresa 6

Figura 4.7: (a) Distribuição acumulada dos erros em previsões em lotes de importação e exportação da empresa cliente 2; (b) Distribuição acumulada dos erros em previsões em lotes de importação e exportação da empresa cliente 3; (c) Distribuição acumulada dos erros em previsões em lotes de importação e exportação da empresa cliente 6;

de vezes em que o previsor subestimou o tempo de execução de lotes tendeu a aumentar, à medida que as tarefas foram executando, em lotes de longo tempo de execução. A figura 4.8 mostra a quantidade de subestimativas do algoritmo previsor em lotes de importação de longo tempo de execução. O código-fonte executado na última tarefa de lote de importação varia muito para cada lote de importação. Em alguns casos, este código-fonte apresenta alta variação em seu tempo de execução e na maioria dos casos corresponde à tarefa de menor duração. A última tarefa dos lotes de importação é a tarefa em que o algoritmo de regressão menos conseguiu encontrar uma distribuição e parâmetros aproximados o suficiente do comportamento real da execução da tarefa.

A execução da última tarefa do lote de importação do cliente 0 é muito rápida em relação ao tempo de execução das outras tarefas. Além disso, o algoritmo de regressão foi eficiente ao aproximar o comportamento real da duração de execuções das outras tarefas.

Pode-se observar nesse caso que a quantidade de subestimações tendeu a cair conforme o andamento da execução do lote. Porém, na execução da última tarefa o previsor obteve um resultado ruim devido a baixa eficiência do algoritmo de regressão para os dados desta tarefa.

Esse mesmo comportamento também foi observado durante a execução de lotes de exportação de longo prazo. Porém, os lotes de exportação possuem apenas duas tarefas e a segunda tarefa contém alta variação no tempo de execução. A figura 4.9 mostra a quantidade de subestimativas por rodada do algoritmo do previsor em lotes de exportação de longo prazo.

Estes resultados mostram que a eficiência do algoritmo de regressão interfere fortemente na eficiência do previsor.

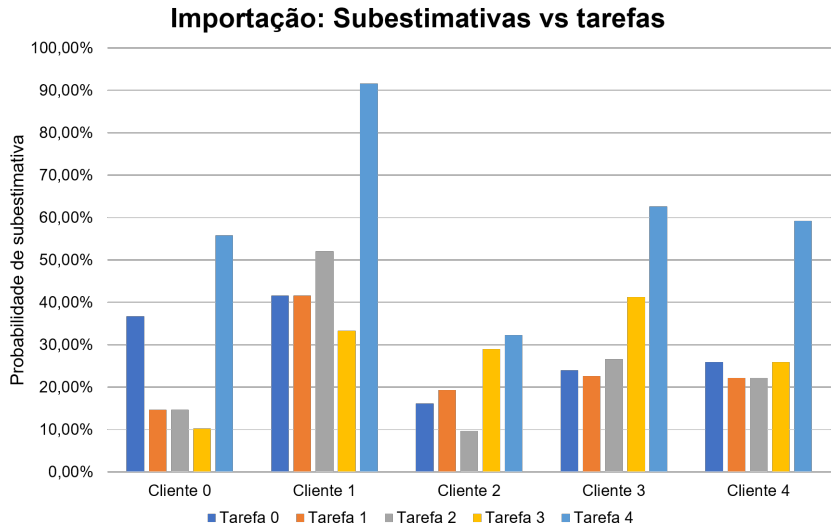


Figura 4.8: Quantidade de vezes em que o previsor subestimou o tempo de execução de lotes de importação

Comparamos os valores da mediana das previsões de término de execução dos lotes de longo prazo, antes da execução de cada tarefa, com a duração real de cada tarefa. As figuras 4.10 e 4.11 mostram o percentual de erro mediano relativo do previsor ao longo das execuções dos lotes de importação e exportação, respectivamente. Pode-se observar que o erro do previsor tende a diminuir ao longo das execuções dos lotes de importação. Nos casos dos lotes de exportação, o erro tende a aumentar para alguns lotes pois a tarefa 1 é a mais longa e variável tarefa do lote de exportação.

Pode-se observar que, apesar de a última tarefa de lotes de importação ter as maiores taxas de subestimações do término de execução de lotes de importação, como a duração dessa tarefa possui duração frequentemente curta, o erro do previsor tende a ser menor também nas previsões obtidas antes da execução da última tarefa.

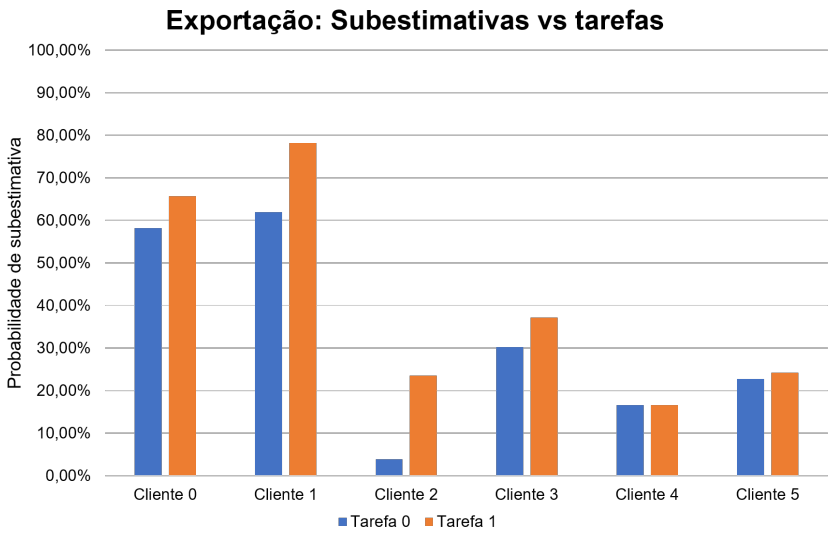


Figura 4.9: Quantidade de vezes em que o previsor subestimou o tempo de execução de lotes de exportação

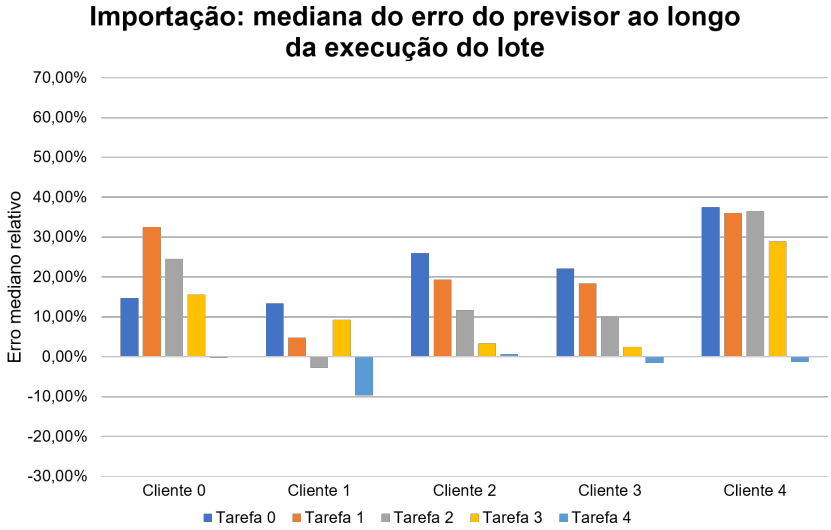


Figura 4.10: Percentual de erro relativo mediano do previsor ao longo da execução de lotes de importação

4.4 ECA

Fizemos uma avaliação qualitativa do módulo ECA, discutindo sobre diversos cenários em que este módulo pode ser aplicado para automatizar a tomada de decisões de fluxos em processamentos em lote e até estender as funcionalidades do escalonador.

O módulo ECA pode ser utilizado para automatizar fluxos de tomada de decisão, prover integração com outros sistemas e até estender o escalonamento de lotes. Esta seção destina-se a apresentar exemplos sobre como o ECA pode ser útil.

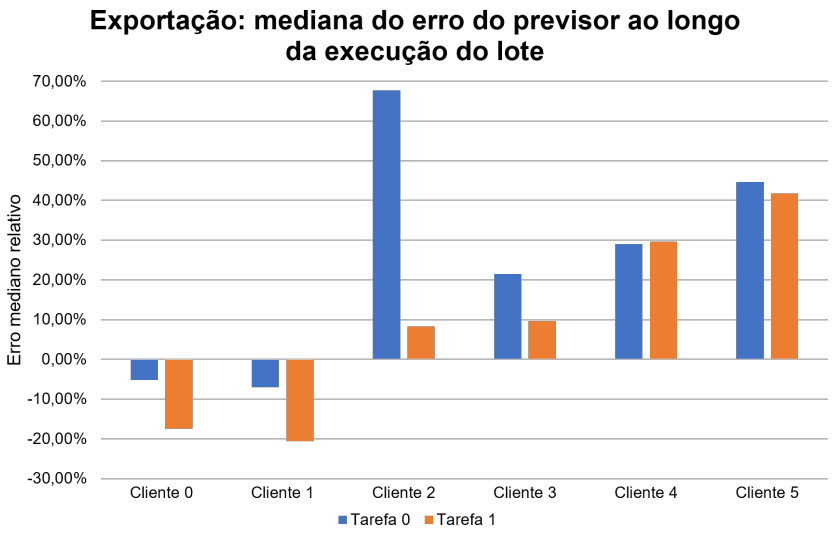


Figura 4.11: Percentual de erro relativo mediano do previsor ao longo da execução de lotes de exportação

O módulo ECA pode se integrar facilmente com sistemas de monitoramento através do envio e recebimento de logs por uma interface REST ou por uma conexão direta com a tabela de logs do banco de dados do BatchKeeper. A partir dessa integração, o módulo ECA pode obter maior visibilidade a respeito do estado das máquinas utilizadas para processamentos. Ao constatar algum problema crítico com alguma máquina, como a presença de falhas de hardware, o sistema de monitoramento poderia enviar um evento ao ECA avisando de um problema com a máquina em questão. Poderíamos criar uma regra com esse evento, uma condição para identificar o tipo de problema e uma ação que poderia desativar essa máquina no *gerenciador de lotes*. Assim, novos processamentos não seriam escalonados para a máquina com problemas.

O módulo ECA poderia ser usado para automatizar o fluxo de tomada de decisão relacionado aos processamentos em lote, reduzindo intervenções humanas. O ECA poderia ser configurado para notificar equipes sobre o andamento dos processamentos, por e-mail, ligações telefônicas, SMS, dentre outros. Para isso, bastaria configurar condições para cada estado da execução de lotes e ações notificando as equipes pelos meios adequados. Uma ação que fizesse ligações telefônicas poderia ser executada apenas quando um lote que fosse crítico falhasse, por exemplo. Caso o lote não fosse crítico, a notificação poderia ocorrer via SMS ou e-mail. Para fazer ligações telefônicas e enviar SMS, pode ser necessário personalizar a imagem do contêiner do BatchKeeper para incluir bibliotecas especializadas nesse tipo de notificação ou poderia ser criado um software para disparar as ligações e mensagens de texto e a ação apenas faria uma requisição a esse software, como através de uma interface REST por exemplo.

Caso haja um cenário em que, de acordo com algum evento e condição, um lote precise ser criado, o ECA poderia ser configurado com uma regra em que uma ação

pudesse se conectar às interfaces REST de lote e tarefa para criar o novo lote. O cenário em que um lote de longo tempo de execução falhe e não haja automatização de tomada de decisões e esse deva ser reagendado para ser executado no próximo dia é um exemplo de necessidade de criação de lotes por uma ação.

O ECA pode ser utilizado para integrar o BatchKeeper a outros sistemas. Em um cenário em que haja um sistema responsável por provisionar máquinas virtuais ao conjunto de máquinas utilizadas para processamentos, poderia haver uma condição para identificar a existência de um número baixo de recursos computacionais disponíveis no conjunto de máquinas e acionar uma ação para enviar uma requisição ao sistema provisionador de máquinas para provisionar uma nova máquina para processamento.

O ECA também pode ser usado como uma forma de estender o algoritmo de escalonamento do BatchKeeper. Caso um evento de falta de recursos computacionais livres para a execução de um lote seja lançado, poderia haver uma condição para identificar a criticidade e a prioridade do lote em questão e dos outros lotes em execução. Caso necessário, uma ação que interrompesse lotes de menor prioridade poderia ser executada para liberar recursos computacionais para a execução do lote de maior prioridade.

Poderia haver um sistema responsável por gerenciar manutenções futuras em máquinas do conjunto de máquinas utilizadas para processamentos. Esse sistema poderia enviar um evento ao ECA para avisar de uma manutenção futura em uma máquina com um horário definido. Poderia-se criar uma condição para verificar a quantidade de lotes agendados para o horário da manutenção associados à máquina em questão. Uma ação poderia ser associada para enviar uma notificação aos operadores, caso houvessem muitos lotes agendados para execução na máquina em questão. Outro caso de uso seria criar uma ação para realocar as execuções dos lotes para outras máquinas que estariam disponíveis na data e hora da manutenção.

Em casos de falha ao executar um lote crítico, o ECA poderia aumentar a prioridade deste lote para a próxima execução do mesmo, por meio de uma condição e ação.

4.5 Conclusão da avaliação

O *previsor* pode ser útil em casos em que lotes possuam prazos máximos de execução definidos. Em particular, o *previsor* de tempo de execução provou-se útil quando utilizado para prever a duração de lotes de longo prazo. A avaliação do *previsor* foi feita utilizando um conjunto de dados desafiador, devido a alta variação do tempo de execução e escassez de dados históricos. As previsões do *previsor* atuaram como limite superior do tempo de execução em 66% dos casos em lotes de longo prazo. Acreditamos que a

eficiência do previsor é muito maior quando utilizados em maiores conjuntos de dados com menor variação no tempo de execução de tarefas.

Para obter bons resultados de eficiência com o *previsor* é importante ajustar o parâmetro de percentil padrão para uso na previsão de tempo de execução para um valor apropriado para a variação histórica do tempo de execução de lotes ou tarefas. Além disso, é importante verificar se o comportamento do histórico de execuções é bem aproximado por alguma função de distribuição padrão utilizada pelo BatchKeeper. Caso esta aproximação seja ruim em relação ao histórico de execução das últimas tarefas, o erro do previsor pode aumentar conforme o andamento da execução do lote, embora o erro normalmente tenda a diminuir ao longo da execução. Caso o comportamento do histórico de execução possa ser melhor aproximado por outra função de distribuição que não pertença ao conjunto padrão de funções de distribuições do BatchKeeper, outras funções presentes no *ScyPy* podem ser adicionadas ao BatchKeeper de forma fácil e utilizadas no algoritmo de regressão.

A arquitetura de solução do BatchKeeper permite flexibilização do funcionamento do BatchKeeper e maior flexibilização na automatização dos fluxos que envolvem o processamento em lote. Garantir alta flexibilidade nas automatizações destes fluxos é importante para aumentar a confiabilidade dos processamentos através da redução de intervenções humanas. Esta flexibilidade presente no projeto do módulo ECA permite que o BatchKeeper possa ser utilizado em diversos cenários.

Capítulo 5

Implantação em Produção

O BatchKeeper é uma solução genérica que pode ser aplicada em vários contextos. O BatchKeeper foi aplicado na mesma empresa que cedeu os dados utilizados na seção de avaliação (seção 4) para gerenciar e executar os processamentos em lote. O BatchKeeper também foi utilizado para automatizar alguns dos fluxos de execução de processamentos, através da utilização de regras ECA. Esta empresa possui grandes quantidades de processamentos de lotes que são críticos para a sua operação, sendo executadas centenas de processamentos de grandes quantidades de dados a cada mês. Além disso, a empresa possui lotes que possuem prazos para processamento e prioridades distintas. A carga de trabalho de cada lote é diferente, devido à distinção da quantidade de informações (tamanho da base de dados) processadas em cada lote. A falta de isolamento de recursos computacionais para os processamentos em lote também favorece a variação no tempo de execução dos processamentos.

A empresa avaliada possui um software web utilizado por várias empresas clientes. A empresa avaliada disponibiliza uma instância do software web para cada empresa cliente. Cada instância do software web é executada em uma única máquina. Existem duas máquinas em que são executadas centenas de instâncias. Existem outras máquinas em que apenas uma instância é executada. Os processamentos desta empresa ocorrem nas mesmas máquinas em que se encontram as respectivas instâncias do software web para cada empresa cliente. Os dados processados provêm dos bancos de dados associados às respectivas instâncias. Em vários momentos dos processamentos há alta utilização de banco de dados.

O BatchKeeper foi implantado nesta empresa com o objetivo de automatizar o gerenciamento de processamentos de lotes, minimizando a quantidade de intervenções humanas em todo o processo relacionado aos processamentos.

5.1 Funcionamento anterior à implantação do BatchKeeper

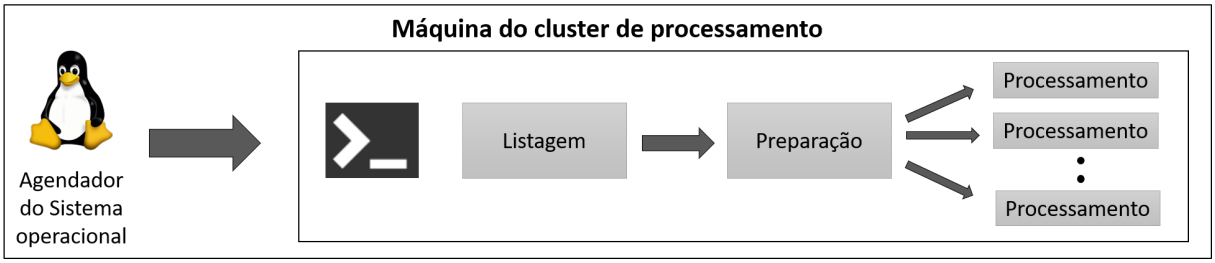


Figura 5.1: Fluxo de processamento em lote anterior a implantação do BatchKeeper

A figura 5.1 mostra o fluxo de processamento em lote anterior a implantação do BatchKeeper. Esse fluxo consistia na execução de três tipos de scripts, aqui nomeados *listagem*, *preparação* e *processamento*. O script de listagem era responsável por iniciar o fluxo de processamentos, sendo executado diariamente pelo agendador de tarefas do sistema operacional da máquina do processamento (*cron* do Linux). Esse script era responsável por criar uma lista de todas as empresas clientes da empresa avaliada que deveriam ter seus dados processados naquele dia. Após a criação da lista de empresas clientes, o script de listagem executava o script de preparação. O script de preparação era responsável por calcular variáveis que seriam utilizadas como parâmetros para a execução dos processamentos, como o período dos processamentos. Existia um script de processamento para cada empresa cliente. Os scripts de processamento faziam o processamento dos lotes relacionados às suas respectivas empresas clientes e eram executados sequencialmente ao final da execução do script de preparação. Cada máquina, do conjunto de máquinas utilizadas para processamentos, hospedava os scripts de listagem, preparação e processamento referentes às instâncias do software web hospedados na mesma máquina. Os scripts de listagem, preparação e processamento foram escritos em shell script e foram inicialmente desenvolvidos para lidar com poucos processamentos de curta e média duração (até 1 hora de tempo de execução). Como não houve grandes refatorações nestes scripts e nem foram escritos em uma linguagem de programação mais moderna que pudesse dar suporte a testes unitários, diminuiu-se a confiabilidade na execução dos mesmos.

Não havia automatização nas tomadas de decisão em relação aos processamentos, exigindo acompanhamento e intervenção humana na ocorrência de erros em processamentos, principalmente em processamentos com duração maior que 3 horas. A necessidade de acompanhamento e intervenções humanas representava um aumento de custos e falhas humanas ocasionais. Além disso, era difícil c

5.2 Solução implantada com o BatchKeeper

Houveram duas implantações coexistentes do BatchKeeper na mesma empresa utilizada na avaliação para realizar o gerenciamento de dois conjuntos distintos de processamentos de lotes. Cada implantação correspondia a processamentos de lotes de produtos distintos da empresa avaliada. Cada um destes produtos possuía um conjunto de servidores (cluster) para a execução dos processamentos. A figura 5.2 mostra o fluxo de processamento em lote da primeira implantação do BatchKeeper.

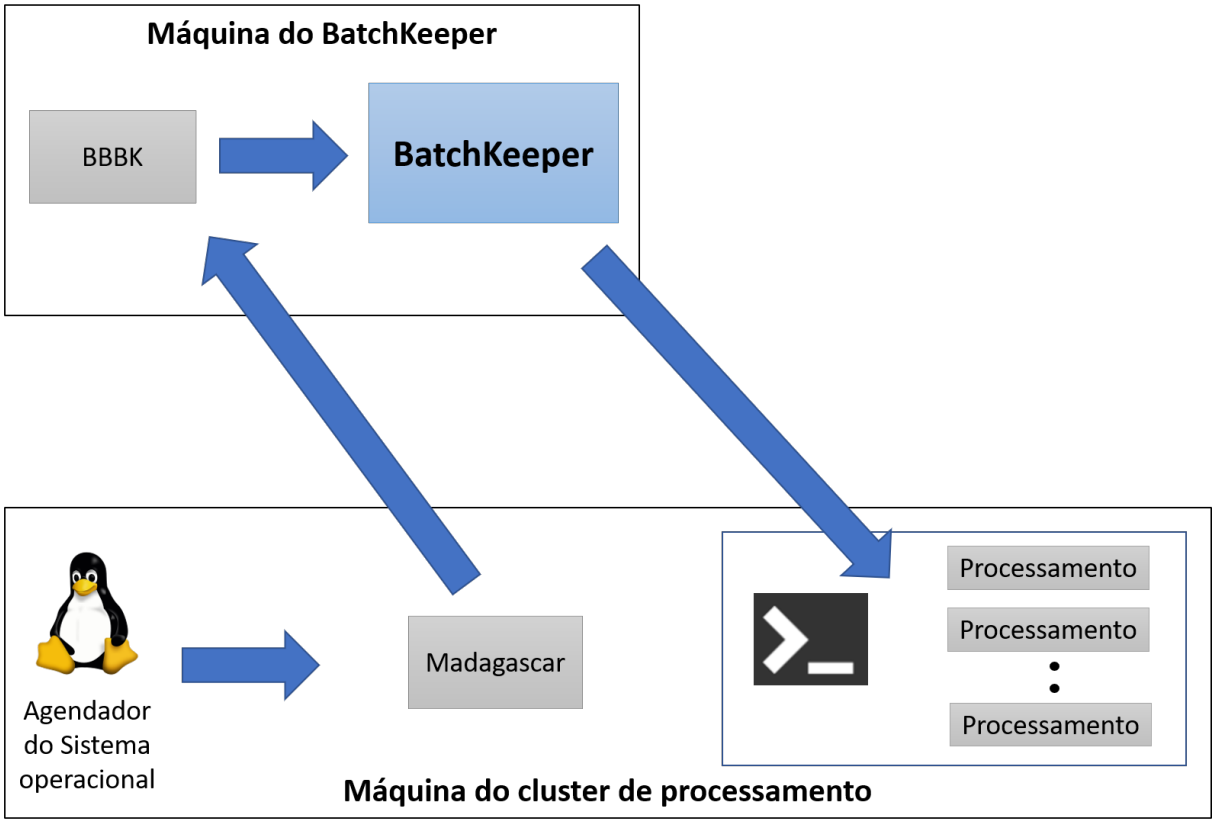


Figura 5.2: Fluxo de processamento em lote da primeira implantação do BatchKeeper

Na primeira implantação, era comum ocorrer alterações nas datas de execução dos processamentos de algumas empresas clientes. Buscou-se minimizar a intervenção humana para alterações de configurações de lotes no BatchKeeper. Por isso, foi construído um segundo software, executado periodicamente, nomeado de Madagascar, para criar uma lista de lotes que deveriam ser processados pelo BatchKeeper no mesmo dia de sua execução, no caso de lotes de importação, ou no dia seguinte, no caso de lotes de exportação. O Madagascar envia esta lista, junto com parâmetros necessários à execução de cada lote, a um terceiro sistema, nomeado BBBK. O BBBK é responsável por centralizar a comunicação com o BatchKeeper, fazendo a transformação de solicitações

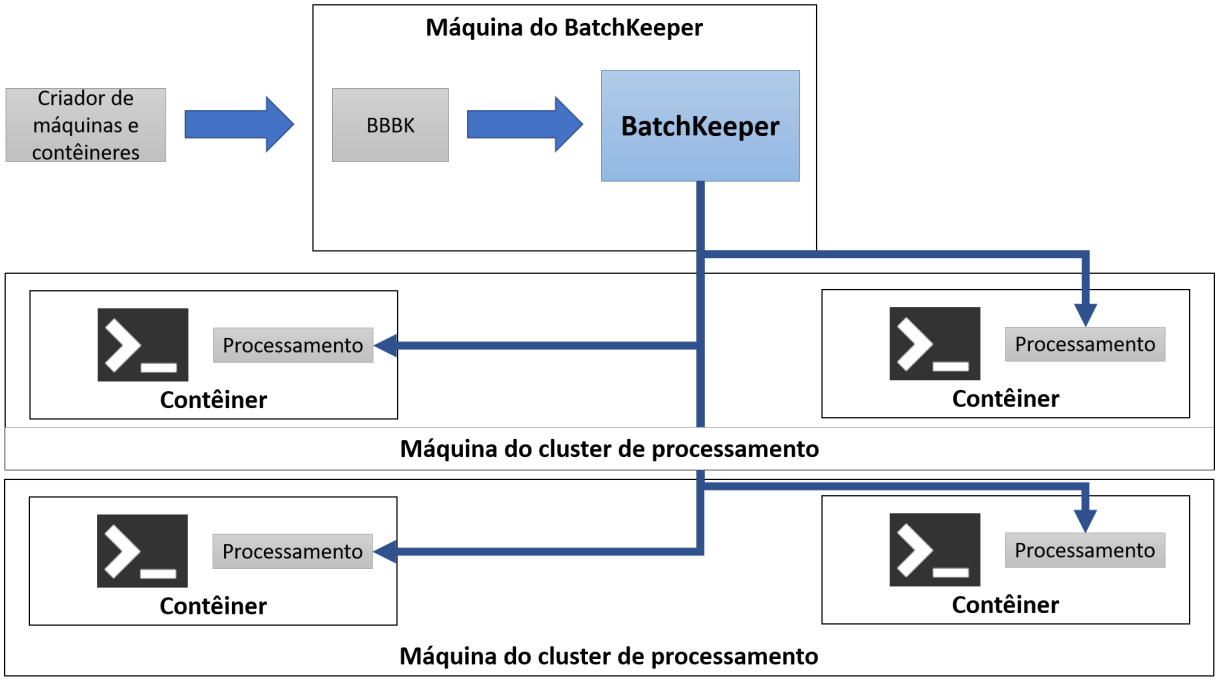


Figura 5.3: Fluxo de processamento em lote da segunda implantação do BatchKeeper

de cadastros de processamentos em cadastros de lotes, tarefas, dependências entre lotes e dependências entre tarefas. O BatchKeeper recebe do BBBK todos os comandos que deverão ser executados na realização dos processamentos e as tomadas automatizadas de decisões na forma de cadastro de lotes (seção 3.2), tarefas, dependências entre lotes e entre tarefas, condições, ações e regras ECA. O Madagascar e o BBBK foram construídos como sistemas separados do BatchKeeper porquê são sistemas auxiliares, com regras de negócio específicas da empresa utilizada na avaliação, mantendo o BatchKeeper como um sistema genérico.

Foi provisionada uma máquina para a execução do BatchKeeper e BBBK. Esta máquina possui conectividade, via ssh, a todas as máquinas do cluster de processamento. Cada máquina do cluster possui uma instância do Madagascar. Esta instância é responsável por lidar apenas com os dados das instâncias do software da empresa avaliada que estejam em execução naquela máquina.

O BatchKeeper também foi implantado para processar lotes de exportação em um segundo produto da empresa, em uma arquitetura utilizando contêineres. A figura 5.3 mostra o fluxo de processamento em lote desta implantação do BatchKeeper.

Nesse segundo produto, a alteração de datas de execução de processamentos não era necessária, não existindo um sistema como o Madagascar. Cada instância do sistema da empresa avaliada foi implantada em um contêiner Docker. Cada contêiner Docker foi implantado em um grupo de máquinas virtuais, sem replicação. O BatchKeeper recebe os lotes, tarefas, dependências entre tarefas e entre lotes, e os nomes dos contêineres

onde serão executados os processamentos, a partir do BBBK à cada cadastro de lote de exportação. O BatchKeeper é responsável por acessar as máquinas virtuais e executar os processamentos de lotes em seus respectivos contêineres.

Em ambas implantações, foram criadas regras ECA para automatizar a tomada de algumas decisões e processos nos fluxos de processamentos. Foram criadas as seguintes ações para serem utilizadas em regras ECA nessas implantações do BatchKeeper:

- Efetuar ligações
- Re-executar tarefa
- Enviar e-mails
- Interromper execução de lote
- Interromper execução de tarefas

Estas ações foram criadas para automatizar as tarefas dos funcionários que acompanhavam as execuções dos processamentos. A ação "efetuar ligações" foi criada para notificar, via ligações telefônicas, os funcionários da equipe de suporte aos processamentos em casos em que a intervenção humana seja necessária. Esta ação é utilizada para apresentar informações de erros ocorridos, ou atrasos anormais em processamentos de alto tempo de execução, via sintetizador de voz de uma biblioteca de terceiros. A ação "re-executar tarefa" foi criada para habilitar a reexecução automática de tarefas em caso de intermitência no acesso de algum serviço que a tarefa seja dependente, como banco de dados. Foi identificada grande importância em enviar e-mails ao final de cada processamento, informando os status destas execuções à equipe de suporte aos processamentos. No caso específico da empresa avaliada, lotes não podem ser executados após o prazo máximo de execução. As ações de interrupção de lotes e tarefas foram criadas para serem executadas caso o BatchKeeper, por meio do módulo executor, identificasse alta probabilidade de violação de prazo máximo de execução de lotes.

Há planos de se criar ações para lidar com a restauração automática de dumps de bancos de dados na ocorrência de erros que possam trazer inconsistência às informações processadas.

Capítulo 6

Conclusão e Trabalhos Futuros

O processamento em lote é aplicado por empresas e governos em diferentes cenários, como em geração de relatórios e identificação de hábitos de compra em bancos de dados de websites de comércio eletrônico, por exemplo. Em alguns cenários, o processamento em lote pode pertencer a uma etapa em um fluxo de negócio maior e a integração do processamento em lote com outros sistemas pode trazer ganhos em produtividade, eficiência e qualidade de serviço no fluxo de negócio. Além disso, o processamento em lotes pode conter um fluxo de tomada de decisões de acordo com resultados da execução de lotes, que pode ser automatizado para reduzir intervenções humanas e atrasos nas execuções de lotes.

O BatchKeeper é um sistema para processamento em lote que possibilita identificação de atrasos na execução de processamentos; automatizações no fluxo de execução de processamentos; integração com sistemas de monitoramento; e integração com outros sistemas para automatizar o gerenciamento de lotes e implementar o fluxo de negócio em que o processamento em lote está inserido.

O BatchKeeper provê um previsor de tempo de execução para auxiliar na identificação de violações de prazos máximos de execuções de lotes. A avaliação do previsor, utilizando dados reais de execuções de lotes, mostrou que o previsor pode ser útil em execuções de lotes de longo prazo. As execuções do histórico real de execuções de tarefas foram feitas em um ambiente de execução de lotes sem isolamento de recursos computacionais, havendo variação frequente no tempo de execução das tarefas. Mesmo nesse cenário e com poucos dados para treino do previsor, o previsor atuou como um limite superior de tempo de execução com sucesso em 66% dos casos. Acreditamos que a taxa de sucesso do previsor de tempo de execução em lotes de longo tempo de execução pode ser muito maior em cenários com isolamento de recursos computacionais ou com mais dados para treinamento. O previsor apresentou baixa eficiência em atuar como um limite superior de tempo de execução de lotes de curto tempo de execução, principalmente devido à alta variação no tempo de execução desses lotes. Porém, não se justifica a utilização do previsor em lotes de curto prazo, já que atrasos não são tão significativos quanto atrasos na execução de lotes de longo prazo.

O BatchKeeper possui um módulo ECA (Evento - Condição - Ação) flexível o

suficiente para automatizar os fluxos de execuções de processamentos. Tal flexibilidade é provida pela utilização de código-fonte na especificação de regras ECA e pela existência de funções nas interfaces no BatchKeeper que possibilitam obtenção e alteração de dados de lotes, tarefas, máquinas e grafo de dependências entre tarefas ou lote. Além disso, o ambiente de execução das regras ECA é isolado, através da utilização de contêineres Docker, provendo segurança por construção.

O BatchKeeper foi implantado em uma empresa do mercado financeiro em duas implantações coexistentes. Em ambas as implantações, o BatchKeeper contribuiu para melhorar o gerenciamento de processamentos em lote e automatizar os fluxos de execução e negócio envolvidos nos processamentos, reduzindo intervenções humanas e atrasos nos processamentos.

6.1 Trabalhos Futuros

Consideramos integrar várias melhorias ao algoritmo de escalonamento do BatchKeeper. O algoritmo de escalonamento pode ser aperfeiçoado para agendar o horário de novos processamentos automaticamente, de acordo com o histórico de tempo de execução, prioridades, prazos máximos de execução de lotes já agendados e recursos computacionais disponíveis. Assim, os operadores do BatchKeeper poderiam enviar lotes ao BatchKeeper para serem executados e o BatchKeeper poderia escolher quando executar cada lote segundo diferentes objetivos.

O escalonador poderia ser integrado a serviços de computação em nuvem e levar em consideração informações advindas de um sistema de monitoramento. Considere o cenário em que um sistema de monitoramento detecte falhas contínuas de *hardware* em uma máquina utilizada para processamentos. O escalonador poderia utilizar esta informação para requisitar uma nova máquina em um serviço de computação em nuvem para substituir temporariamente a máquina com falhas. Esta funcionalidade também poderia ser implementada como uma regra ECA; uma ação poderia se conectar a um serviço de computação em nuvem e solicitar a criação de uma nova máquina virtual; esta ação poderia ser vinculada a um evento específico lançado pelo sistema de monitoramento.

O *gerenciador de lotes* poderia levar em consideração os recursos computacionais de cada máquina no momento do agendamento de lotes, não permitindo a criação de lotes caso a falta de recursos computacionais mínimos para a execução do novo lote, fosse detectada. Uma outra opção seria solicitar a um serviço externo a adição de novas máquinas virtuais ao conjunto de máquinas utilizadas para processamentos.

O *previsor* consegue estimar o término da execução de lotes. Porém, este algoritmo

não considera os eventos lançados na execução de cada lote e seus impactos no tempo de execução. O BatchKeeper armazena todas as ocorrências de eventos durante a execução de lotes em uma tabela no banco de dados. Estas ocorrências são armazenadas em um formato de linha do tempo. Um possível trabalho futuro é utilizar os dados de eventos da linha do tempo, fazer uma correlação com o tempo de execução das tarefas relacionadas e utilizar correlações fortes para aprimorar a estimativa do término da execução de lotes.

O módulo *executor* poderia permitir o uso de imagens de contêiner personalizadas, por lote. Assim, seria possível especificar imagens com bibliotecas ou pacotes de terceiros para a execução de condições e ações específicas. Os agentes de execução de tarefas poderiam enviar dados sobre o andamento da execução de tarefas para o *gerenciador de lotes* para possibilitar o acompanhamento, em tempo de execução, por parte de operadores do BatchKeeper.

Referências Bibliográficas

[1] Amazon. Amazon Kinesis. `https://aws.amazon.com/kinesis/`, 2020. Accessed: 2020-01-27.

[2] James Bailey, George Papamarkos, Alexandra Poulovassilis, and Peter T Wood. An event-condition-action language for XML. In *Web Dynamics*. 2004.

[3] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):1–154, 2013.

[4] Nesrine Berjab, Hieu Hanh Le, Chia-Mu Yu, Sy-Yen Kuo, and Haruo Yokota. Hierarchical Abnormal-Node Detection Using Fuzzy Logic for ECA Rule-Based Wireless Sensor Networks. In *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2018.

[5] Paul A Buhler and José M Vidal. Towards adaptive workflow enactment using multiagent systems. *Information technology and management*, 6(1):61–87, 2005.

[6] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *Communications of the ACM*, 59(5):50–57, 2016.

[7] Chronos. Chronos. `https://mesos.github.io/chronos/`, 2018. Accessed: 2018-01-28.

[8] Edward Chuah, Arshad Jhumka, Sai Narasimhamurthy, John Hammond, James C Browne, and Bill Barth. Linking resource usage anomalies with system failures from cluster log data. In *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on*, 2013.

[9] Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. Event logs for the analysis of software failures: A rule-based approach. *IEEE Transactions on Software Engineering*, 39(6):806–821, 2013.

[10] Confluent. KSQL. `https://www.confluent.io/product/ksql/`, 2020. Accessed: 2020-01-27.

[11] Adriano B de Sousa, José R Torres Neto, PR Geraldo Filho, and Jó Ueyama. Uma plataforma de IoT para integração de dispositivos baseada em nuvem com Apache

Kafka. In *Anais Estendidos do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2018.

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

[13] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*, 2015.

[14] Apache Software Foundation. Kafka Streams. `https://kafka.apache.org/documentation/streams/`, 2020. Accessed: 2020-01-27.

[15] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Nsdi*, 2011.

[16] Angela Goh, Y-K Koh, and Dragan S Domazet. ECA rule-based support for workflows. *Artificial intelligence in engineering*, 15(1):37–46, 2001.

[17] Esin Gokkoca, Mehmet Altinel, R Cingil, E Nesime Tatbul, Pinar Koksal, and Asuman Dogac. Design and implementation of a distributed workflow enactment service. In *Cooperative Information Systems, 1997. COOPIS'97., Proceedings of the Second IFCIS International Conference on*, 1997.

[18] Joseph E Gonzalez, Reynold S Xin, Ankur Dave, Daniel Crankshaw, Michael J Franklin, and Ion Stoica. GraphX: Graph Processing in a Distributed Dataflow Framework. In *OSDI*, 2014.

[19] Google. Cloud Pub/Sub. `https://cloud.google.com/pubsub/`, 2020. Accessed: 2020-01-27.

[20] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[21] Taylor Groves, Jeff Knockel, and Eric Schulte. BFS vs. CFS scheduler comparison. *The University of New Mexico*, 11, 2009.

[22] Thomas Heimrich and Günther Specht. Enhancing ECA rules for distributed active database systems. In *Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World*, 2002.

[23] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, 2011.

[24] David Hollingsworth et al. The workflow reference model: 10 years on. In *Fujitsu Services, UK; Technical Committee Chair of WfMC*, 2004.

[25] Jyothish Jose, Oravanpadath Sujisha, Malayamparambath Gilesh, and Thayyil Bindima. On the fairness of linux o (1) scheduler. In *2014 5th International Conference on Intelligent Systems, Modelling and Simulation*, 2014.

[26] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Íñigo Goiri, Subru Krishnan, Janardhan Kulkarni, et al. Morpheus: Towards Automated SLOs for Enterprise Clusters. In *OSDI*, 2016.

[27] Kafka. Apache Kafka. `https://kafka.apache.org/`, 2018. Accessed: 2017-11-23.

[28] Minseo Kang and Jae-Gil Lee. An experimental analysis of limitations of MapReduce for iterative algorithms on Spark. *Cluster Computing*, 20(4):3593–3604, 2017.

[29] Jacek Kobus and Rafal Szklarski. Completely Fair Scheduler and its tuning, 2009.

[30] C Kolivas. "Brain Fuck Scheduler, 2009.

[31] Władysław Kozaczuk. *Enigma: how the German machine cipher was broken, and how it was read by the Allies in World War Two*. Univ Pubns of Amer, 1984.

[32] Grafana Labs. Grafana. `https://grafana.com/`, 2020. Accessed: 2020-01-27.

[33] Wei Le and Mary Lou Soffa. Marple: a demand-driven path-sensitive buffer overflow detector. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008.

[34] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[35] Jianxun Liu and Jinmin Hu. Dynamic batch processing in workflows: Model and implementation. *Future Generation Computer Systems*, 23(3):338–347, 2007.

[36] Encode OSS Ltd. Django Rest Framework. `https://www.django-rest-framework.org/`, 2020. Accessed: 2020-01-27.

[37] Chenyang Lu, John A Stankovic, Gang Tao, and Sang Hyuk Son. Design and evaluation of a feedback control EDF scheduling algorithm. In *Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE*, 1999.

[38] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010.

[39] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[40] Ming Mao, Jie Li, and Marty Humphrey. Cloud auto-scaling with deadline and budget constraints. In *2010 11th IEEE/ACM International Conference on Grid Computing*, 2010.

[41] Marathon. Marathon. `https://mesosphere.github.io/marathon/`, 2018. Accessed: 2018-01-28.

[42] Frank J Massey Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[43] Daniel A Menascé, Debanjan Saha, SCD Porto, Virgilio AF Almeida, and Satish K Tripathi. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. *Journal of Parallel and Distributed Computing*, 28(1):1–18, 1995.

[44] Microsoft. Azure EventHubs. `https://azure.microsoft.com/en-us/services/event-hubs/`, 2020. Accessed: 2020-01-27.

[45] Microsoft. Microsoft Service Bus. `https://azure.microsoft.com/pt-br/services/service-bus/`, 2020. Accessed: 2020-01-27.

[46] Ingo Molnar. Completely Fair Scheduler, 2007.

[47] Monasca. Monasca. `http://monasca.io/`, 2018. Accessed: 2018-12-14.

[48] Nagios. Nagios. `https://www.nagios.org/`, 2018. Accessed: 2018-12-14.

[49] Radheshyam Nanduri, Nitesh Maheshwari, A Reddyraja, and Vasudeva Varma. Job aware scheduling algorithm for mapreduce framework. In *2011 Third IEEE International Conference on Coud Computing Technology and Science*, 2011.

[50] Adam Oliner, Archana Ganapathi, and Wei Xu. Advances and challenges in log analysis. *Communications of the ACM*, 55(2):55–61, 2012.

[51] George Papamarkos, Alexandra Poulovassilis, and Peter T Wood. Event-condition-action rule languages for the semantic web. In *Proceedings of the First International Conference on Semantic Web and Databases*, 2003.

[52] Massoud Pedram. Energy-efficient datacenters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(10):1465–1484, 2012.

[53] Pivotal. Cloud Pub/Sub. `https://www.rabbitmq.com/`, 2020. Accessed: 2020-01-27.

[54] Prometheus. Prometheus. `https://prometheus.io/`, 2018. Accessed: 2018-12-14.

[55] Luise Pufahl, Nico Herzberg, Andreas Meyer, and Mathias Weske. Flexible batch configuration in business processes based on events. In *International Conference on Service-Oriented Computing*, 2014.

[56] Ying Qiao, Kang Zhong, HongAn Wang, and Xiang Li. Developing event-condition-action rules in real-time active database. In *Proceedings of the 2007 ACM symposium on Applied computing*, 2007.

[57] Ariel Rabkin and Randy H Katz. Chukwa: A system for reliable large-scale log collection. In *LISA*, volume 10, pages 1–15, 2010.

[58] Gourav Rattihalli, Pankaj Saha, Madhusudhan Govindaraju, and Devesh Tiwari. Two stage cluster for resource optimization with Apache Mesos. *Workshop: MTAGS17: 10th Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers*, 2019.

[59] Gemma Reig, Javier Alonso, and Jordi Guitart. Prediction of job resource requirements for deadline schedulers to manage high-level SLAs on the cloud. In *Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on*, 2010.

[60] Riemann. Riemann. `http://riemann.io/`, 2018. Accessed: 2018-12-14.

[61] Juan D Rodriguez, Aritz Perez, and Jose A Lozano. Sensitivity analysis of k-fold cross validation in prediction error estimation. *IEEE transactions on pattern analysis and machine intelligence*, 32(3):569–575, 2009.

[62] Martin Schatzoff, R Tsao, and R Wing. An experimental comparison of time sharing and batch processing. *Communications of the ACM*, 10(5):261–265, 1967.

[63] SciPy. SciPy. `https://www.scipy.org/`, 2020. Accessed: 2020-01-27.

[64] Colin Scott, Vjekoslav Brajkovic, George Necula, Arvind Krishnamurthy, and Scott Shenker. Minimizing faulty executions of distributed systems. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016.

[65] Colin Scott, Andreas Wundsam, Barath Raghavan, Aurojit Panda, Andrew Or, Jefferson Lai, Eugene Huang, Zhi Liu, Ahmed El-Hassany, Sam Whitlock, et al. Troubleshooting blackbox SDN control software with minimal causal sequences. *ACM SIGCOMM Computer Communication Review*, 44(4):395–406, 2015.

[66] Vijay Subramani, Rajkumar Kettimuthu, Srividya Srinivasan, and S Sadayappan. Distributed job scheduling on computational grids using multiple simultaneous requests. In *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, 2002.

[67] Byung-Chul Tak, Bhuvan Urgaonkar, and Anand Sivasubramaniam. To Move or Not to Move: The Economics of Cloud Computing. In *HotCloud*, 2011.

[68] Andrew S Tanenbaum and Herbert Bos. *Modern operating systems*. Pearson, 2015.

[69] Tibco. Tibco Enterprise Message Service. `https://www.tibco.com/products/tibco-enterprise-message-service`, 2020. Accessed: 2020-01-27.

[70] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ Twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014.

[71] JC van Winkel and Betsy Beyer. *The Production Environment at Google, from the Viewpoint of an SRE*. 2016.

[72] Claudia Vannucchi, Michelangelo Diamanti, Gianmarco Mazzante, Diletta Romana Cacciagrano, Flavio Corradini, Rosario Culmone, Nikos Gorogiannis, Leonardo Mostarda, and Franco Raimondi. virony: A tool for analysis and verification of ECA rules in intelligent environments. In *2017 International Conference on Intelligent Environments (IE)*, 2017.

[73] WhatIsBatch. What is batch processing? `https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zconcepts/zconc_whatisbatch.htm`, 2018. Accessed: 2018-02-03.

[74] Philipp Wieder, Jan Seidel, Oliver Wäldrich, Wolfgang Ziegler, and Ramin Yahyapour. Using SLA for resource management and scheduling-a survey. In *Grid Middleware and Services*. 2008.

[75] Han Wu, Zhihao Shang, and Katinka Wolter. Performance Prediction for the Apache Kafka Messaging System. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019.

[76] N Sudhakar Yadav, B Eswara Reddy, and KG Srinivasa. Cloud-Based Healthcare Monitoring System Using Storm and Kafka. In *Towards Extensible and Adaptable Methods in Computing*. 2018.

[77] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013.

[78] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

Apêndice A

Exemplos de uso da biblioteca do BatchKeeper

A biblioteca provê os principais objetos utilizados no BatchKeeper, como Batch (lote), Job (Tarefa), Host (Máquina) e outros. O BatchKeeper disponibiliza uma interface REST para cada objeto utilizado pela biblioteca do BatchKeeper. Esta biblioteca serve para encapsular os detalhes de conexão entre as ações e condições e as interfaces REST do BatchKeeper.

A.1 Operações básicas utilizando a interface REST de lote

O código-fonte abaixo mostra exemplos da aplicação de operações básicas em um lote. o código da linha 6 cria um objeto que representa um lote (Batch). Na linha 14, verifica-se a existência do lote criado, no *gerenciador de lotes*. Caso este lote já tenha sido cadastrado anteriormente, o atributo `nickname` é alterado e atualizado no *gerenciador de lotes* através das linhas 20 a 22. Caso contrário, esse objeto é adicionado no *gerenciador de lotes* através do código da linha 18. Ao final deste exemplo, o lote criado é removido do *gerenciador de lotes*.

```
1  from escalonador.escalonador import Batch
2  from datetime import datetime
3  from session import BKSession
4
5
6  b1 = Batch(id=1,nickname="teste", description="sdfjkasda",
7      priority=1, deadline=None, repetition=99,
8      repetition_interval=1, start_time=datetime.now(),
9      execution_start_time=None, execution_finish_time=None,
10     status=None, owner="teste@gmail.com",
```

```
11      owner_name="Bruno Machado", rule=None,
12      enabled=False, last_execution=None)
13
14  bk_session = BKSession("username","password",
15      "http://localhost:8030")
16  bk_response = bk_session.get(b1)
17  if not bk_response:
18      bk_response = bk_session.post(b1)
19  else:
20      b1 = Batch(**bk_response)
21      b1.nickname="teste 2"
22      bk_response = ss.post(b1)
23
24  ss.delete(b1)
```

A.2 Interrupção de lote

O código-fonte a seguir mostra um exemplo de interrupção de um lote. A interrupção de um lote se dá a partir da alteração do status de um lote para o status `KILL` (linhas 8 a 11). Periodicamente, o BatchKeeper envia uma requisição aos executores de tarefas para interromper todas as tarefas em execução cujos lotes estejam com o status `KILL`. Todas as tarefas dependentes da tarefa interrompida são retiradas da fila de execução do escalonador. Caso o lote em questão não esteja em execução, a requisição para alteração de status é ignorada.

```
1  from util.util import *
2
3  batch_id = get_batch_id()
4  bk_session = init_bk_session()
5
6  batch_data = bk_session.get(Batch(id=batch_id))
7
8  batch = Batch(**batch_data)
9  batch.status = BatchStatus.KILL
10
11 bk_session.patch(batch)
```

A.3 Interrupção de tarefa

O código-fonte a seguir demonstra como interromper uma tarefa. Esta interrupção se dá a partir da alteração do status da tarefa para `KILL` (linhas 8 a 11). Periodicamente, o BatchKeeper interrompe a execução de todas as tarefas cujo status tenha código igual a `KILL`. Todas as tarefas que sejam dependentes da tarefa interrompida são retiradas da fila de execução do escalonador de tarefas. Caso esta tarefa não esteja em execução, a requisição para alteração de status é ignorada.

```
1  from util.util import *
2
3  job_id = get_job_id()
4  bk_session = init_bk_session()
5
6  job_data = bk_session.get(Job(id=job_id))
7
8  job = Job(**job_data)
9  job.status = JobStatus.KILL
10
11 bk_session.patch(job)
```

A.4 Realização de ligação telefônica (utilizando o Twilio)

O código-fonte abaixo mostra um exemplo de ação com notificação via ligação telefônica utilizando a biblioteca externa Twilio. Esta ação é executada sempre que ocorrer um erro no processamento de um lote do tipo `exportação`. O código-fonte entre as linhas 17 e 42 configuram o que deve ser dito ao receptor da ligação telefônica através da criação de arquivos XML de configuração e da instanciação de objetos da biblioteca Twilio. A partir da linha 46, são feitas N tentativas de ligação para os receptores configurados no arquivo importado `util`.

```
1  import os
2  import sys
3  import stat
4  from shutil import copyfile
5  from twilio.rest import Client
6  from twilio.twiml.voice_response import VoiceResponse
```

```
7  from util.util import *
8
9
10 batch = get_batch_data()
11 system = batch["description"].splitlines()[2][len("Sistema: "):]
12 current_sector = batch["nickname"].split(" | ")[2]
13
14 print("Erro no processamento de exportação de '%s'\n" % system)
15
16
17 # TwiML com descrição do áudio da ligação
18 response = VoiceResponse()
19 response.play("/twilio/acorda.mp3")
20 response.play("/twilio/erro_lote_saida.mp3")
21 # O replace foi colocado para que o text-to-speech do Twilio
22 # não pronuncie o underline
23 response.say("de %s" % system.replace("_", " "), voice="alice",
24     language="pt-BR")
25
26 xml_file_name = "%s.xml" % generate_random_file_name()
27 xml_file = "%s/%s" % (PUBLIC_DIR, xml_file_name)
28
29 with open(xml_file, "w") as xml:
30     xml.write(str(response))
31
32 # Copia os arquivos de áudio para PUBLIC_DIR
33 copyfile("%s/acorda.mp3" % AUDIOS_DIR,
34     "%s/acorda.mp3" % PUBLIC_DIR)
35 copyfile("%s/erro_lote_saida.mp3" % AUDIOS_DIR,
36     "%s/erro_lote_saida.mp3" % PUBLIC_DIR)
37
38 # Ligação para avisar que houve um erro em um
39 # processamento de importação
40 account_sid = "00000000000000000000000000000000000000"
41 auth_token = "00000000000000000000000000000000"
42 client = Client(account_sid, auth_token)
43
44 exit_code = 0
45
46 for i in range(1, MAX_CALLS+1):
47     print("Tentativa %s..." % i)
48
49     call = client.calls.create(url="%s/%s" % (PUBLIC_URL,
50         xml_file_name), to=main_tel, from_=TESTE_TWILIO_TEL,
51         timeout=TIMEOUT)
52     call_status = client.calls(call.sid).fetch().status
53
```

```
54      while True:
55          if call_status in {"completed", "canceled", "busy",
56              "failed", "no-answer"}:
57              break
58
59          call_status = client.calls(call.sid).fetch().status
60
61      if call_status == "completed":
62          print("A ligação foi completada com sucesso.")
63          break
64  else:
65      print("""Não foi possível comunicar com o telefone corporativo.
66           Ligando para o gerente da área""")
67
68      for i in range(1, MAX_CALLS+1):
69          print("Tentativa %s..." % i)
70
71          call = client.calls.create(url="%s/%s" % (PUBLIC_URL,
72              xml_file_name), to=alternative_tel,
73              from_=TESTE_TWILIO_TEL, timeout=TIMEOUT)
74          call_status = client.calls(call.sid).fetch().status
75
76          while True:
77              if call_status in {"completed", "canceled",
78                  "busy", "failed", "no-answer"}:
79                  break
80
81              call_status = client.calls(call.sid).fetch().status
82          if call_status == "completed":
83              print("A ligação foi atendida e completada com sucesso.")
84              break
85      else:
86          print("Não foi possível comunicar com o telefone do gerente.")
87          exit_code = 1
88
89  # Remove os arquivos inseridos no diretório PUBLIC_DIR
90  os.remove(xml_file)
91  os.remove("%s/acorda.mp3" % PUBLIC_DIR)
92  os.remove("%s/erro_lote_saida.mp3" % PUBLIC_DIR)
93
94  sys.exit(exit_code)
```