UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Avaliação de Desempenho das Plataformas Apache Hadoop, Apache Spark e Apache Flink Usando o Benchmark Hibench-master 7

(Versão Definitiva Após Defesa Pública)

Isabel Soqui Bongo

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Drº. Mário Marques Freire

Covilhã, Janeiro de 2019

Dissertação elaborada no Instituto de Telecomunicações - Delegação da Covilhã e no Departamento de Informática da Universidade da Beira Interior por Isabel Soqui Bongo, Licenciada em Ciências da Computação pela Universidade Katyavala Bwila (Angola), sob orientação do Doutor Mário Marques Freire Investigador Sénior do Instituto de Telecomunicações e Professor Catedrático do Departamento de Informática da Universidade da Beira Interior, e submetida à Universidade da Beira Interior para discussão em provas públicas.
Trabalho realizado no âmbito do Programa Integrado de IC&DT "C4 – Centro de Competências em Cloud Computing" com contrato CENTRO-01-0145-FEDER-000019, cofinanciado pelo Sistema de Apoio à Investigação Científica e Tecnológica – Programas Integrados de IC&DT, pelo Programa Operacional Regional do Centro (Centro 2020) e pela União Europeia através do Fundo Europeu de Desenvolvimento Regional (FEDER) e pelo Instituto Nacional de Gestão de Bolsa de Estudo (INAGBE), Luanda, Angola, através da Bolsa de Estudo nº432.

Cofinanciado por:

CENTRO 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Europeu de Desenvolvimento Regional

Agradecimentos

Ao Instituto Superior Politécnico da Universidade Katyaval Bwila (Angola) pelo apoio. Ao Instituto de Telecomunicações - Delegação da Covilhã pelo apoio. Aos professores do Departamento de Informática da Faculdade de Engenharia da UBI pelo profissionalismo e sabedoria com que ministram as aulas, apresentando as várias áreas do curso, dando uma base sólida dos conhecimentos. A Deus por ter me dado saúde e vida; aos meus pais e ao meu esposo, pelo amor, pelo apoio, pela compreensão, pelo incentivo dado e paciência tida, permitindo que a minha formação fosse um objetivo alcançado. Ao meu orientador, professor Dr. Mário Marques Freire pelo apoio e interesse. Aos meus colegas, pelo apoio e partilha dos conhecimentos. Sem este conjunto de intervenientes não seria possível a realização deste trabalho.

Dedicatórias

Dedico este trabalho aos meus pais Miguel Bongo e Teresa Cassinda Bongo, ao meu esposo Miguel Barvante Kapusu, à minha filha Alexandra Bongo Kapusu. Aos meus familiares, colegas, amigos e a todos aqueles que se preocupam com a busca e a atualização dos conhecimentos.

Resumo

Tendo em conta o forte crescimento dos dados que se observa atualmente, o conceito de big data vem ganhando popularidade, dando origem às ferramentas capazes de processar, analisar e armazenar estes grandes volumes de dados. Nesta senda, um dos desafios que se coloca aos profissionais e serviços que lidam com esse tipo de dados consiste na escolha adequada da melhor plataforma a utilizar para processamento de big data, tendo sido investigado o desempenho de Apache Hadoop, Apache Spark e Apache Flink que representam as três plataformas mais utilizadas para o processamento de big data. Nesta dissertação é avaliado o desempenho do Hadoop, do Spark e do Flink utilizando a suite de Benchmark Hibench na sua versão Hibench-master 7, tendo cido selecionado cinco cargas de trabalho nomeadamente: Sort, Terasort, Wordcount, K-means e Pagerank. Estas plataformas foram instaladas e configuradas num cluster homogéneo com quatro nós (máquinas físicas), um mestre e três escravos. Para avaliar o desempenho das plataformas, foram consideradas duas métricas: tempo de execução e a taxa de transferência dos dados, tendo-se caracterizado a utilização de recursos tais como memória, Central Processing Unit (CPU), Disco (E/S) e rede, para diferentes escalas de dados tais como*small*, *large* e *gigantic*. Foram realizadas várias experiências, tendo os respetivos resultados mostrado que o cluster do Spark ao executar as cargas de trabalho wordcount, sort e terasort obteve melhor desempenho com tamanho de dados *gigantic*, enquanto que o Hadoop apresentou melhor desempenho com tamanho de dados *small* e *large*, apesar de no wordcount a diferença ser pequena. Por outro lado, o Spark ao executar algoritmos iterativos como o k-means apresentou melhor desempenho com entradas de dados *small* e *large* e, para o pagerank, apenas com tamanho de dados *small*, enquanto que o Hadoop melhorou o seu desempenho com tamanho de dados *gigantic* para K-means e *large* para o pagerank. Os resultados obtidos mostram que o desempenhos das duas plataformas nesta experiência é relativo dependendo da carga de trabalho, do tamanho dos dados de entrada e do tamanho da memória. Foram também comparadas as plataformas Spark e o Flink executando o programa Wordcount dos seus ficheiros de exemplos, tendo-se observado que o Flink apresentou melhor desempenho que o Hadoop para todos os tipos de dados de entrada, sendo 2x mais rápido que o Spark. O Spark apresentou melhor desempenho que o Hadoop para tamanhos de dados de entrada de 2MB e 392MB, mas observou-se que o seu desempenho degradava-se com o aumento do tamanho de dados de entrada. O desempenho do Flink melhorou significativamente, sobretudo para tamanhos de dados de entrada de 8GB e 38GB, após o ajuste do valor do parâmetro de fração da memória.

Palavras-chave

Computação em nuvem, desempenho, Hadoop, Spark, Flink, benchmarks, carga de trabalho.

Abstract

Given the strong data growth that is currently occurring, the concept of big data has gained popularity, giving rise to tools capable of processing, analyzing and storing these large volumes of data. In this way, one of the challenges facing professionals and services dealing with this type of data is the adequate choice of the best platform to use for big data processing, and the performance of Apache Hadoop, Apache Spark and Apache Flink has been investigated, which represent the three most widely used platforms for big data processing. In this dissertation, the performance of Hadoop, Spark and Flink is evaluated using the Hibench Benchmark suite in its Hibench-master 7 version, having selected five workloads namely: sort, terasort, wordcount, K-means and pagerank. These platforms were installed and configured in a homogeneous cluster with four nodes (physical machines), one master and three slaves. In order to evaluate the performance of the platforms, two metrics were considered: execution time and throughput, being also characterized the resource consuption such as memory, Central Processing Unit (CPU), Disk (I/O) and network, for different scales of data such as small, large and gigantic. A number of experiments were carried out, with the respective results showing that the Spark cluster performing wordcount, sort and terasort workloads performed better with gigantic data size, while Hadoop performed better with small and large data sizes, although in wordcount the difference is small. On the other hand, Spark when executing iterative algorithms like k-means presented better performance with small and large data sizes and, for pagerank, only with small data size, while Hadoop improved its performance with gigantic data size for K-means and large for the pagerank. The results show that the performance of the two platforms in this experiment is relative depending on the workload, the size of the input data and the size of the memory. The Spark and Flink platforms were also compared by running the Wordcount program of their sample files, and it was observed that Flink performed better than Hadoop for all input data types, being 2x faster than Spark. Spark performed better than Hadoop for 2MB and 392MB input data sizes, sizes, but it was observed that its performance was degraded with the increasing of the size of input data. Flink performance improved significantly, especially for 8GB and 38GB input data sizes, after adjusting the memory fraction parameter value.

Keywords

Cloud computing, performance, Hadoop, Spark, Flink, benchmarks, workload.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

ACL	Access Control List
AM	ApplicationMaster
ANSI	American National Standards Institute
API	Application Programming Interface
BDAS	Berkeley Data Analytics Stack
CEP	Complex Event Processing
CPU	Central Process Unit
DAG	Directed Acyclic Graph
DFG	Deutsche Forschungsgemeinschaf
E/S	Entrada/Saida
GFS	Google File System
GUI	Graphical User Interface
HA	High Availability
HDFS	Hadoop Distribut File System
HTTP	Hypertext Transfer Protocol
HQL	Hibernate Query Language
IO	Input/Output
JAR	Java ARchive
JDBC	Java Database Connectivity
JVM	Java virtual Machine
ML	Machine Learning
NDFS	Nutch Distributed File System
ODBC	Open Database Connectivity
POST	power-on self-test
RDDs	Resilient Distributed Dataset
REST	Representational State Transfer
RM	ResourceManager
RPC	Remote Procedure Call
SQL	Structured Query Language

SSL	Secure Socket Layer
UC	University of California
YARN	Yet Another Resource Negotiator
YCSB	Yahoo! Cloud Serving Benchmark

Capítulo 1

Introdução

1.1 Foco da Dissertação

A computação em nuvem permite o processamento de enormes volumes de dados que as técnicas tradicionais de base de dados e softwares têm tido dificuldades em processar dentro de tempos aceitáveis [1]. Esse conjunto de dados enorme e variado que requer elevadas velocidades de processamento e análise é conhecido como Big data. A quantidade de dados que são gerados diariamente, potenciados através do uso da Internet, preocupam as empresas, governos e outras instituições no sentido de encontrar as melhores soluções de big data, pelo que foram surgindo várias ferramentas que disponibilizam serviços que lidam com grandes quantidades de dados, fornecendo aos utilizadores a capacidade de utilizar servidores comuns para efetuar processamento distribuído e eficiente. O MapReduce é um modelo de programação paralela capaz de executar aplicações em milhares de computadores simultaneamente, tendo sido proposto por um grupo de engenheiros da Google em 2004 para indexar páginas web de modo a implementar o seu motor de busca [2]. Este sistema foi muito bem sucedido, mas a sua implementação não está disponível publicamente [3]. Em 2006 foi apresentada uma implementação da plataforma MapReduce desenvolvida num projeto de código aberto da Apache designado por Hadoop [4]. Esta versão tem sido usada por muitas empresas de computação em nuvem incluindo a Amazon, a IBM, a RackSpace e a Yahoo [3]. Esta nova abordagem de computação distribuída foi originalmente implementada como parte do projeto Apache Nutch, que mais tarde tomaria forma através da comunidade *open source* e se tornariam projetos independentes, tornando-se no sistema Apache Hadoop [5] que conhecemos hoje [2]. A comunidade de investigação também contribuiu para a popularidade da plataforma explorando-a e publicando vários algoritmos MapReduce, extensões e componentes para resolver problemas ou melhorar o desempenho [3]. O Apache Hadoop fornece um serviço eficiente, confiável, escalável e tolerante a falhas para processamento de dados em larga escala. O MapReduce é responsável pelo processamento distribuído de dados no Hadoop e o HDFS (Hadoop Distributed File System) garante o armazenamento distribuído ao sistema [4]. O Hadoop não é adequado para todos os tipos de aplicações e em função disso foi proposto o Apache Spark [6] como alternativa para superar as limitações do Hadoop. O Apache Spark é um sistema que faz o processamento dos dados na memória e foi projetado para suportar aplicações que reutilizam um conjunto de dados em várias operações paralelas [7]. Contudo, o Spark apresenta a desvantagem de o desempenho ser restringido pela memória. Como alternativa para suprir as desvantagens do Spark, surgiu o Apache Flink [8] uma plataforma de código aberto, tolerante a falhas e híbrida que processa os dados em batch e em fluxo contínuo de dados distribuídos, não limitando o respetivo processamento à memória, o que permite melhorar a robustez e a velocidade através do uso da gestão da memória. Utilizando a gestão de memória [9].Esta variedade de plataformas para processamento de big data permite que as diversas instituições escolham a que melhor se adequa às suas necessidades, havendo necessidade de indentificar a plataforma que melhor se adequa para determinados tipos e tamanhos de dados. Esta dissertação é dedicada à avaliação e comparação do desempenho das

plataformas Apache Hadoop, Apache Spark e Apache Flink.

1.2 Problema e Objetivo da Investigação

O problema em questão consiste em avaliar e comparar o desempenho das plataformas Apache Hadoop, Apache Spark e Apache Flink. O principal objetivo desta dissertação consiste em avaliar e comparar o desempenho do Apache Hadoop, Apache Spark e do Apache Flink usando a suite de Benchmark Hibench-master 7, num ambiente de teste composto por um cluster com quatro máquinas físicas executando as cargas de trabalho wordcount, sort, terasort, k-means e pagerank do Hibench [10].

1.3 Abordagem Adotada para Resolver o Problema

A abordagem adotada para resolver o problema em questão consiste em investigação orientada ao problema envolvendo a experimentação que inclui a implementação e configuração de um cluster homogéneo das plataformas Hadoop, Spark e Flink no modo distribuído, com um testbed constituído por quatro máquinas físicas. Após a implementação e configuração destas plataformas serão realizados testes com benchmarks da Suite Hibench-master 7 para avaliar o desempenho das mesmas. No final dos testes será feito um estudo de comparação do desempenho, tendo em consideração o tempo de execução e o throughput.

1.4 Principais Contribuições

Existem estudos de desempenho de cada uma das plataformas consideradas nesta dissertação, assim como estudos de desempenho envolvendo a comparação entre o Apache Hadoop e Apache Spark [11], [12] e a comparação entre o Apache Spark e o Apache Flink [13]. No entanto, existem relativamente poucos estudos comparativos do desempenho das plataformas Hadoop, Spark e Flink. Em [7] é apresentada uma análise comparativa entre estas três plataformas e em [14] é apresentada uma comparação do desempenho destas três plataformas usando cargas de trabalho (workloads) representativas de Big Data. Nesta dissertação é apresentada uma análise, comparação de desempenho e caracterização de utilização dos recursos das plataformas Apache Hadoop, Apache Spark e Apache Flink usando o Benchmark Hibench-master 7.

1.5 Limitações do Trabalho

Tendo em conta a quantidade de máquinas físicas disponibilizadas e os recursos nelas existentes para a implementação e configuração das plataformas Hadoop, Spark e Flink afim de avaliarmos e compararmos os respetivos desempenhos, isto, limitou de certa forma a escala dos clusters uma vez que impossibilitou testes com a escala de dados gigantic para a carga de trabalho pagerank e big data. Contudo, ao executarmos certas cargas de trabalhos nas três plataformas, estas reportavam erros devido à falta de recursos necessários para tamanho de dados da escala gigantic e bigdata. Este caso ocorreu principalmente com o Spark quando executava o pagerank. Por isso decidimos executar as plataformas Hadoop e Spark apenas com dados da escala small e large para o pagerank. Não foi também possível avaliar o Flink executando diretamente o

Hibench-master 7 devido a dificuldades com o servidor Apache Kafka [15], uma vez que este terminava sistematicamente a conexão com os nós não permitindo a geração dos dados para os testes. A alternativa explorada nesta dissertação para a avaliação de desempenho do Flink consistiu em gerar os dados através do Hibench-master 7 e executá-los através dos exemplos desta plataforma, deixando para trabalho futuro a avaliação do Flink executando diretamente o Hibench-master 7.

1.6 Organização da Dissertação

O corpo principal desta dissertação é constituído por quatro capítulos: o Capítulo 1 dissertação, o problema em questão, os objetivos da investigação subjacentes a esta dissertação, a abordagem adotada para resolver o problema, as principais contribuições, as limitações do trabalho desenvolvido e a organização da dissertação. O Capítulo 2 apresenta uma descrição detalhada dos principais conceitos relativos às plataformas Apache Hadoop, Apache Spark e Apache Flink, incluindo os seus componentes principais, um breve enquadramento histórico e uma visão geral da comparação das três plataformas em termos de desempenho, arquitetura e funcionamento. São também apresentados relatos de trabalhos desenvolvidos relacionados com o tema desta dissertação. O Capítulo 3 descreve o ambiente de teste e os resultados experimentais obtidos para as Plataformas Hadoop, Spark e Flink. Este capítulo começa por descrever os conceitos de benchmarks, as suas categorias, assim como a categoria de benchmarking utilizada nesta dissertação (Benchmarking suite Hibench), bem como as suas respetivas cargas de trabalho. É também descrito neste capítulo a implementação dos clusters das três plataformas, incluindo a especificação das ferramentas e respetivas versões utilizadas, os recursos de software e hardware, e os procedimentos de configuração dos clusters do Hadoop, Spark e Flink. Este capítulo inclui ainda a caracterização dos recursos computacionais utilizados pelo Apache Hadoop e o Apache Spark, a análise e discussão dos resultados experimentais e a comparação detalhada do desempenho do Hadoop, Spark e o Flink. No capítulo 4 são apresentadas as principais conclusões da dissertação com base no trabalho de investigação realizado sobre o desempenho das três plataformas e são apontadas sugestões para trabalho futuro.

Capítulo 2

Background e Trabalhos Relacionados

2.1 Introdução

Este capítulo tem como objetivo a apresentação dos conceitos associados aos elementos que formam a base necessária para o melhor entendimento do trabalho. Apresentam-se aqui os conceitos mais gerais, deixando os aspetos específicos para serem apresentados quando necessário. Este capítulo é composto por 7 secções tais como: A Secção 2.2 apresenta os principais conceitos relacionados com Apache Hadoop, sua arquitetura, ecossistema, resenha histórica e seu modo de funcionamento; a secção 2.3 é dedicada ao Apache Spark, nela contem arquitetura, historial modo de funcionamento e seus modelos relacionados; a secção 2.4 é dedicada ao Apache Flink, nela contem arquitetura, historial modo de funcionamento e seus modelos relacionados a secção 2.5 apresenta a comparação do ponto de vista arquitetural e funcional, comparação do desempenho das três plataformas de big data Hadoop, Spark e Flink, a secção 2.6 faz uma breve descrição dos trabalhos relacionados ao tema; e por fim a secção 2.7 que apresenta as conclusões do capítulo.

2.2 Apache Hadoop

O Apache Hadoop é uma plataforma de código aberto, para armazenamento e processamento distribuído de grandes conjuntos de dados em clusters de computadores, construído a partir de hardware comum, utilizando um modelo de programação simples. Os serviços Hadoop referem-se a armazenamento de dados, processamento de dados, acesso à dados, gestão de dados, segurança e operações [16]. Ele é projectado para ampliar a partir de um único servidor para milhares de máquinas, cada uma oferecendo computação e armazenamento local. Ao invés de confiar em hardware para proporcionar alta disponibilidade, a própria biblioteca é projectada para detectar e lidar com falhas na camada de aplicação, de modo que a entrega de um serviço altamente disponível no topo de um cluster de computadores, cada um dos quais pode ser propenso a falhas [17].

2.2.1 Resenha Histórica

O Hadoop desde sua origem tem dado passos gigantescos no mundo do BigData. Nestas subsecção será apresentada uma breve história e evolução até a versão atual do Hadoop:

- Em fevereiro de 2003: a Google buscava aperfeiçoar seu serviço de busca de páginas Web, ferramenta pela qual ficou mundialmente conhecida - almejando criar uma melhor técnica para processar e analisar, regularmente seu imenso conjunto de dados da Web. Com esse objetivo, Jeffrey Dean e Sanjay Ghemawat, dois funcionários da própria Google desenvolveram a tecnologia MapReduce, que possibilitou otimizar a indexação e catalogação dos dados sobre as páginas Web e suas ligações [2].

- Em outubro de 2003, Sanjay Ghemawat, Howard Gobioff e Shun-Tak Leung, publicaram um artigo intitulado (The Google File System). Este artigo descreve a implementação do GFS (Google File System) um sistema de ficheiros distribuídos escalável para grandes aplicações intensivas em dados da Google utilizando a tecnologia MapReduce. Fornece tolerância a falhas enquanto funcionando em hardware de baixo custo [18].

- Em dezembro de 2004, Jeffrey Dean e Sanjay Ghemawat publicaram um artigo (MapReduce: Simplified Data Processing on Large Clusters) onde apresentam conceitos e características funcionais do modelo de programação MapReduce desenvolvido pela Google em 2003, mostrando assim a forma simplificada de processamento de dados em grande escala para permitir que programadores sem grandes experiências em sistemas paralelos e distribuídos utilizassem os recursos de um sistema distribuído [19].

- Em dezembro de 2005, Doug Cutting cria uma versão do MapReduce para o projeto Nutch, até o meio desse ano, todos os principais algoritmos Nutch foram controlados para serem executados utilizando MapReduce e NDFS (Nutch Distributed File System) [20].

- Em fevereiro de 2006 Hadoop se torna um sub-projeto do Apache Lucene [20], nesta mesma época, Doug Cutting juntou-se ao Yahoo!, que forneceu uma equipe dedicada e os recursos para transformar o Hadoop em um sistema que funcionava na escala da web. Isso foi demonstrado em abril de 2007, quando Yahoo! anunciou que seu índice de pesquisa de produção estava sendo gerado por um cluster Hadoop de 10.000 núcleos [20].

- Em janeiro de 2008, Hadoop deixa de ser o projeto de Lucene e se transforma em um projeto de sucesso da Apache, e nesta altura já se encontrava na versão 0.15.2, tem lançamentos constante de versões com correção de erros e novas funcionalidades [21]. 2010 Facebook anuncia o maior aglomerado Hadoop do mundo (mais de 2.900 nós e 30 petabytes de dados).

- Em dezembro de 2011, o Apache Hadoop disponibiliza sua versão 1.0.0 com maior confiabilidade e estabilidade em escalonamentos. Dentre os destaques dessa nova versão, cabe mencionar o uso do protocolo de autenticação de rede Kerberos, para maior segurança de rede; a incorporação do sub-projeto HBase, oferecendo suporte a BigTable; e o suporte à interface webhdfs, que permite o acesso HTTP (Hypertext Transfer Protocol) para leitura e escrita de dados [21], nesta série tem lançamentos constante de versões com correção de erros e novas funcionalidades.

- Em 23 de maio de 2012 é lançada a primeira versão (alfa) na série Hadoop2.x, esta versão oferece características importantes sobre a série estável Hadoop-1.x, incluindo: HDFS (Hadoop Distribut File System) HA (High Availability) para NameNode (manual failover), YARN aka NextGen MapReduce, HDFS Federation, Performance, Wire-compatibility for both HDFS and YARN/MapReduce (using protobufs). Como destaque a substituição do MapReduce 1 por YARN e MapReduce 2 no Hadoop 2 . [5]

- Em 13 de dezembro de 2017 Apache Hadoop lançou a versão 3.0.0 que representa um ponto de estabilidade e qualidade da API (Application Programming Interface) prontos para produção. Após quatro versões alfa e uma versão beta, 3.0.0 está geralmente disponível. 3.0.0 consiste em 302 correções de bugs, melhorias e outros aprimoramentos desde 3.0.0-beta1. Em conjunto, 6242 problemas foram corrigidos como parte da série de versão 3.0.0 desde 2.7.0 [5]. As principais mudanças que se destacam nesta versão são: Os JARs (Java

ARchive) do Hadoop são compilados na versão 8 do JAVA, uma pré-visualização inicial (alfa 2) de uma grande revisão do YARN Timeline service v.2, reescrita de scripts de shell para correção de muitos erros, suporte para Erasure coding (método para armazenamento durável de dados com economias de espaço significativas em comparação com a replicação), Suporte para contêineres oportunistas e agendamento distribuído, otimização nativa do nível de tarefa do MapReduce, alteração das portas padrão de alguns serviços afetando o NameNode, NameNode secundário e DataNode, suporte para mais de 2 NameNode (capaz de tolerar a falta de qualquer nó no sistema), Suporte para conectores do sistema de ficheiros Microsoft Azure Data Lake e Aliyun Object Storage System e tantas outras melhorias [5].

2.2.2 Hadoop e Modelos Relacionados

Existe uma ampla gama de projetos relacionados ao Hadoop que têm contribuído para o sucesso do mesmo tendo em conta que cada um deles tem seu objetivo. A tabela 2.1, abaixo lista alguns desses modelos.

Tabela 2.1: Projeto relacionados do Hadoop (Adaptado de [4], [22]).

Modelos Relacionados do Hadoop	
Common	Um conjunto de componentes e interfaces para sistemas de ficheiros distribuídos e E/S (Entrada, Saída) gerais (sincronização, Java RPC (Remote Procedure Call), estruturas de dados persistentes)
Avro	Um sistema de sincronização para RPC eficiente e de linguagem cruzada e armazenamento dados persistentes
Pig	Uma linguagem de fluxo de dados e um ambiente de execução para explorar conjuntos de dados muito grandes. O Pig é executado em cluster HDFS e MapReduce
Hive	Um warehouse de dados distribuído para dados armazenados em HDFS; Além disso fornece uma linguagem de consulta baseada em SQL(Structured Query Language) (HiveQL)
Hue	Uma interface de administração Hadoop com ferramentas GUI(Graphical User Interface) úteis para pesquisa de ficheiros, emitindo consultas de Hive e Pig e desenvolvendo fluxos de trabalho Oozie
ZooKeeper	Um serviço de coordenação distribuído e altamente disponível. ZooKeeper oferece primitivas como bloqueios distribuídos que podem ser utilizados para a construção de aplicações distribuídas
Sqoop	Uma ferramenta para mover eficientemente dados entre bases de dados relacionais e HDFS
HBase	Uma base de dados distribuída, orientada a coluna. O HBase utiliza HDFS para armazenamento subjacente e suporta tanto cálculos de estilo em batch utilizando MapReduce e consultas de pontos (leituras aleatórias)
Oozie	Uma ferramenta de gestão de fluxo e trabalho que pode lidar com o projetos relacionados com Hadoop agendamento e encadeamento de aplicações Hadoop
Ambari	Uma ferramenta baseada na Web para o aprovisionamento, gerência e monitores de grupos, que inclui suporte para HDFS, MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig e Sqoop
Cassandra	Uma base de dados multi-mestre escalável sem pontos únicos de falhas

2.2.3 Arquitetura do Hadoop

O Hadoop consiste em três componentes principais: uma estrutura de processamento distribuído chamado MapReduce, um módulo de gestão de recursos separado YARN (Yet Another Resource

Negotiator), e um sistema de ficheiros distribuídos conhecido como o sistema de ficheiros distribuídos Hadoop (HDFS) [22]. A figura 2.1 abaixo apresenta a comparação da arquitectura do Hadoop 1.x e Hadoop 2.x, 3.x.

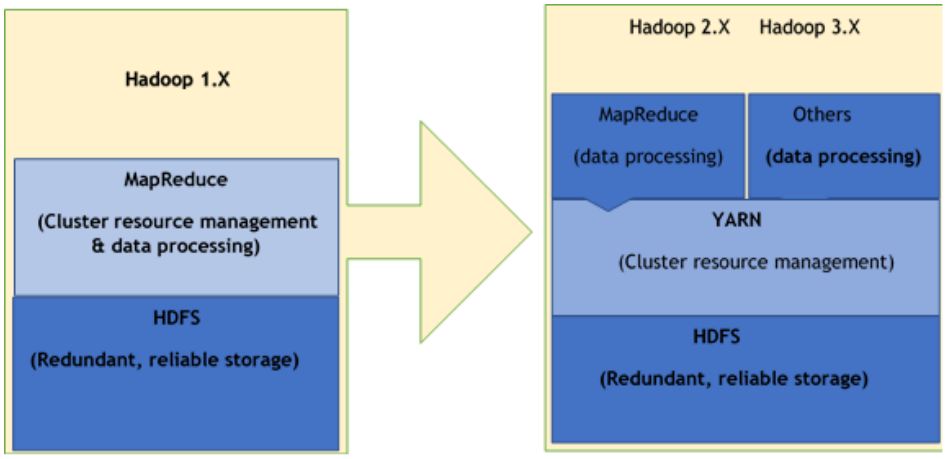

Figura 2.1: Comparação da arquitetura do Hadoop 1.x e Hadoop 2.x (adaptado de [23]).

Na figura 2.1 acima podemos observar que o Hadoop 1.x consiste em apenas dois componentes o HDFS que é administrado por um Namespace e o MapReduce (funciona no topo do HDFS) que é executado pelo JobTracker (composto por ApplicationMaster e ResourceManager) que gere os recursos do cluster, o TaskTracker assume o lançamento e a gestão das tarefas de mapa e redução em um servidor [24]. O modelo de programação MapReduce e a gestão de recursos estavam interligado. A versão 2 do Hadoop veio melhorar o desempenho das aplicações, dar suporte aos modelos de processamento. A arquitectura foi projectada de uma forma completamente diferente, foi separado o processamento de dados e a gestão de recursos, surgindo assim um novo componente: *YARN (Yet Another Resource Negotiator)* que é conhecido como sistema operacional do Hadoop assumindo a gestão e monitoramento das cargas de trabalho, também foram atualizadas as responsabilidades dos componentes HDFS e MapReduce. O Objectivo é ter um ResourceManager global e um ApplicationMaster por aplicação substituindo assim o JobTracker e o TaskTracker respectivamente [3].

HDFS

O HDFS é um sistema de ficheiros distribuídos projetado para ser executado em hardware de comum e para armazenar ficheiros grandes ou seja é adequado para aplicações que possuem grandes conjuntos de dados. É altamente tolerante a falhas e foi preparado para ser implantado em hardware de baixo custo. Construido inicialmente como infraestrutura para o projeto do mecanismo de pesquisa da Web Apache Nutch. Faz parte do projeto Apache Hadoop Core. O HDFS organiza os ficheiros hierarquicamente, com isso se pode criar diretoria, excluir, mover e renomear ficheiros. Os ficheiros no HDFS são armazenados como uma sequência de blocos de 64 MB ou 128 MB e os blocos são distribuídos entre os nós dos cluster. O ultimo bloco é menor em relação aos outros que devem ter o mesmo tamanho. O acesso aos dados é gerido em fluxo, o que significa que aplicações ou comandos são executados diretamente pelo modelo de processamento MapReduce [25]. Os ficheiros no HDFS são apenas para leitura quer dizer que uma vez gravados não permitem alteração.

Arquitetura O HDFS suporta uma arquitetura mestre/escravo, possuindo no lado mestre uma instância do NameNode e em cada escravo uma instância do DataNode [21]. Um cluster HDFS

consiste em um único NameNode, um servidor mestre que gere o Namespace do sistema de ficheiros e regula o acesso a ficheiros por clientes. Também existe uma série de DataNodes (um ou mais nós) por cluster, que gere o armazenamento aos nós conectados que rodam em HDFS, expõe um NameSpace do sistema de ficheiros e permite que os dados do utilizador sejam armazenados. Internamente um ficheiro é dividido em um ou mais blocos e estes blocos são armazenados em um conjunto de DataNodes. O NameNode executa operações de NameSpace do sistema como: abrir, fechar e renomear ficheiros e diretoria. Também determina o mapeamento de blocos para DataNodes. Os DataNodes são responsáveis pela leitura e escrita de pedidos dos clientes [5]. Os blocos de dados em HDFS são armazenado e replicados em trê por padrão, para melhorar a confiabilidade, com uma das réplicas em um rack diferente para aumentar a disponibilidade ainda mais [3]. A figura 2.2 abaixo mostra a arquitetura do HDFS.

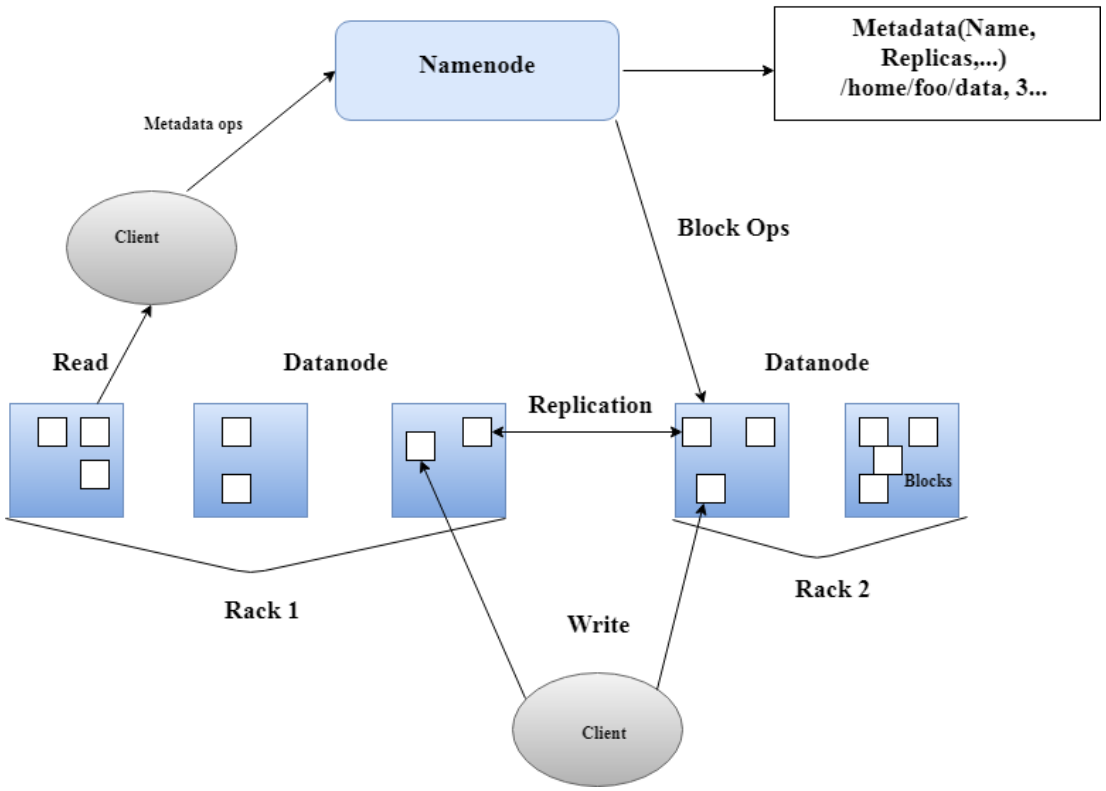

Figura 2.2: Arquitetura do HDFS (adaptado de [5]).

MapReduce

O MapReduce é um modelo de programação para escrever facilmente aplicações que processam grandes quantidades de dados, conjuntos vários terabytes distribuído, paralelo e em grandes clusters de hardware de maneira confiável e tolerante a falhas [5]. Utiliza um conjunto de pares (chave/valor) de entrada e produz um conjunto de pares de (chave/valor) de saída. Possui duas fases: mapeamento e redução. Na fase de mapeamento, o MapReduce pega os dados de entrada e envia cada um dos elementos de dados para a função mapper. Já na fase de redução, a função reduz processa todas as saídas da função mapper e chega a um resultado final. Em outros termos, a função Mapper é feita para filtrar e transformar os dados que serão agregados pela função Reducer.

Um Job MapReduce geralmente divide o conjunto de dados de entrada em blocos independentes que são processados pelas tarefas do mapa de forma completamente paralela. A estrutura classifica as saídas dos mapas, que são então inseridas nas tarefas de redução . Normalmente, tanto

a entrada como a saída dos Job são armazenadas em um sistema de ficheiros. A estrutura cuida das tarefas de agendamento, reexecução e acompanha as tarefas erradas. Conforme referido anteriormente, um cluster do Hadoop é constituído pelos seguintes tipos de nós: O NameNode (o mestre da nuvem) e os DataNodes (ou os escravos). Quando o processo MapReduce é iniciado, o MRAppMaster conecta-se ao Resource- Manager para que sejam negociados os recursos afim de que o Nodemanager execute e monitorize os jobs [16].

YARN (Yet Another Resource Negotiator)
O YARN é chamado de sistema operativo do Hadoop 2.x. A Sua arquitetura fornece uma plataforma de processamento de dados de propósito geral que não se limita apenas ao MapReduce. O YARN habilita o Hadoop a processar outro sistema de processamento de dados construído especificamente, além do MapReduce [5]. O principal objetivo do YARN é dividir as funcionalidades de administração de recursos e do planeamento de tarefas em daemons separados. A ideia é ter um ResourceManager global (RM) e ApplicationMaster (AM) por aplicações.
Arquitetura Na arquitetura YARN, um ResourceManager global é executado como um daemon mestre, geralmente em uma máquina dedicada, que gere os recursos de cluster disponíveis entre várias aplicações concorrentes. O ResourceManager rastreia quantos nós e recursos estão disponíveis no cluster e coordena quais aplicações enviadas pelos utilizadores que devem obter esses recursos e quando.O ResourceManager é o único processo que possui essas informações para que ele possa tomar as suas decisões de agendamento de forma compartilhada e segura [26].

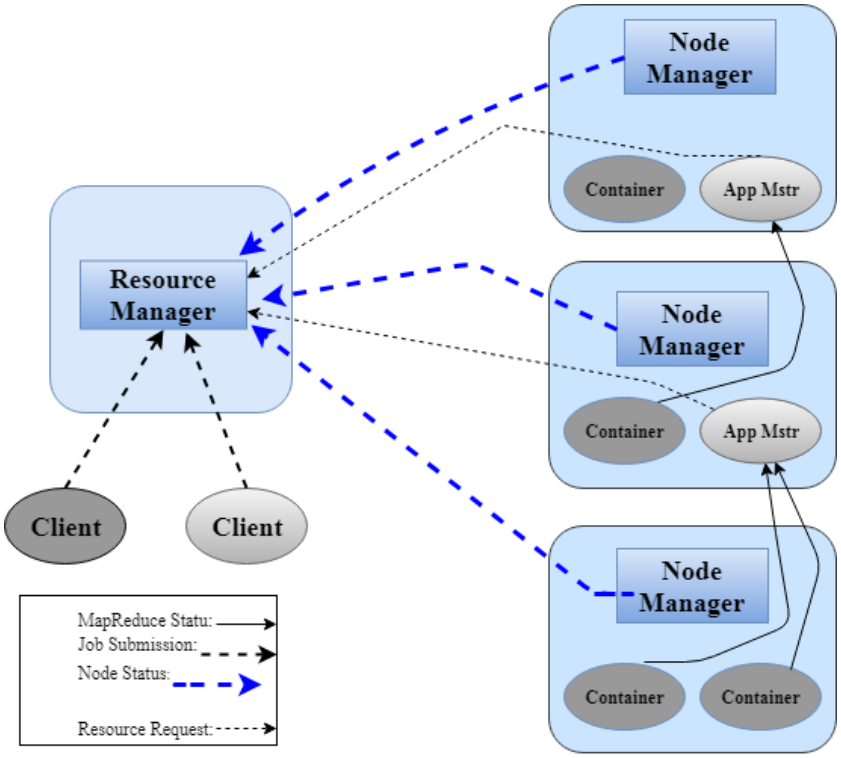

Figura 2.3: Representação esquemática da arquitetura do YARN (adaptado de [5]).

A figura 2.3 ilustra como é feita a comunicação entre os componente do YARN na gestão dos recursos das aplicações no sistema, assim: o NodeManager é o agente responsávelpelos containers e monitoriza a utilização dos recursos (CPU, memória, disco, rede) e informa o estado dos mesmos para o ResourceManager/Scheduler. O ApplicationMaster por aplicação é uma biblioteca

específica do framework que permite negociar recursos do ResourceManager e trabalhar com o NodeManager para executar e monitorizar jobs. O ResourceManager possui dois componentes principais: o Scheduler e o ApplicationsManager. O Scheduler é responsável por alocar recursos para as várias aplicações em execução, sujeitos a restrições conhecidas de capacidades, filas e etc. É um agendador puro, pois não rastrea o estado para a aplicação. Além disso não oferece garantias sobre a reinicialização de tarefas com falhas devido a falhas na aplicação ou a falhas de hardware. O Scheduler executa a sua função de agendamento com base nos requisitos de recursos das aplicações [5].

2.2.4 Funcionamento do Hadoop

O Hadoop funciona através da execução interna de 5 processos: NameNode, DataNode, Secondary NameNode, ResourceManager e NodeManager. O processamento dos dados no Hadoop é iniciado pelo cliente no momento em que ele envia os dados e o programa que são armazenados pelo Hadoop utilizando o HDFS e processados utlizando o MapReduce [20]. Quando há necessidade de um cliente armazenar um ficheiro, o Hadoop divide o ficheiro em blocos (4 blocos de 128MB) e armazena cada bloco em um DataNode, replica estes blocos por 3 (por norma) afim de garantir a tolerância a falhas (se um servidor falhar tem a réplica) e também tem a capacidade de analisar quando um bloco não foi replicado. O processamento de dados Hadoop é feito pelo MapReduce transformando um conjunto grande de dados em dados menores através de um Job de MapReduce que agrupa, soma e sintetiza os dados formando assim um novo conjunto de dados menor [17]. O Job de Mapreduce executa o trabalho em três fases: Map, Sort e Reduce. Na fase Map o job faz a leitura dos dados percorrendo cada linha do ficheiro, posteriormente passa os valores encontrados para a função Map que mapeia esses dados gerando assim uma lista intermediária de um conjunto de par (chave, valor). Na fase Sort o Hadoop ordena a lista de par (chave, valor) obtida na fase Map e agrupa os valores com a mesma chave para serem processados no mesmo nó evitando a duplicação dos mesmos e garantir o desempenho. Na fase Reduce, o Job lê os dados mapeados, ordenados e agrupados e, em cada linha dos dados agrupados, o Job envia os valores encontrados para a função Reduce. Para cada linha, a função Reduce retorna a soma, agregação, filtro ou transformações dos dados. Posteriormente os dados processados são gravados no HDFS.

2.3 Apache Spark

O Spark é uma plataforma para processamento de Big Data, construído com foco em velocidade, facilidade de uso e análises sofisticadas. O Spark utiliza a infraestrutura do Hadoop Distributed File System (HDFS). O Spark também suporta um conjunto de ferramentas de nível superior, incluindo Spark SQL para SQL (Structured Query Language) e processamento de dados estruturados, MLlib para aprendizagem de máquina, GraphX para processamento de gráficos e Spark Streaming que possibilita o processamento de fluxo em tempo real [6].

O Spark foi projetado para ser tolerante a falhas (suporta a perda de qualquer conjunto de nós de trabalho), uma vez que executará novamente todas as tarefas que esses nós estavam executando e recompilará todos os dados armazenados neles. O Spark também foi projetado para funcionar quando o volume de dados exceder a capacidade da memória, nesse caso ele coloca os dados no disco [27].

Para superar as limitações do MapReduce, o Spark faz uso do RDD (Resilient Distributed Datasets), uma abstracção para uma colecção de elementos que podem ser operados em paralelo e tolerante a falhas, o qual implementa estruturas de dados em memória e que são utilizadas para armazenar em cache os dados existentes entre os nós de um cluster [28]. A reutilização dessas estruturas na memória pelo Spark faz com que este seja adequado para operações iterativas de aprendizagem máquina, bem como para querys interativas [29]. As aplicações iterativas e ferramentas interativas de mineração de dados permitem construir aplicações com as quais as atuais estruturas de computação trabalham de forma ineficiente e que originaram os RDDs [30]. Os RDDs são resilientes, isto é: se um nó que executa uma operação for perdido, o conjunto de dados poderá ser reconstruido. Isso acontece porque o Spark conhece a linhagem de cada RDD (sequência de etapas para criar o RDD). Os dados em RDDs são divididos em uma ou várias partições e distribuídos como grupos de objetos de memória nos nós de trabalho do cluster. os RDDs fornecem uma forma efetiva de memória compartilhada para trocar dados entre processos (executores) em diferentes nós [29]. Os RDDs possuem propriedades importantes que devem ser levadas em conta [29]:

- RDDs são particionados, cada partição contém um conjunto exclusivo de registos e pode ser operada de maneira independente.

- RDDs são imutáveis, após serem criados e preenchidos com dados, eles não poderão ser atualizados. Em vez disso, novos RDDs são criados executando transformações, como mapeamento ou funções de filtro em RDDs existentes.

Um DAG (Directed Acyclic Graph) é uma construção matemática usada geralmente nas ciências da computação para representar fluxos de dados e suas dependências, que contêm vértices ou nós e arestas. O DAG consiste em tarefas e etapas. Num contexto de fluxo de dados, os nós são etapas no fluxo do processo e as tarefas são a menor unidade de trabalho agendável num programa Spark. Os Estágios são conjuntos de tarefas que podem ser executadas em conjunto e as etapas são dependentes umas das outras [29]. A Figura 2.4 apresenta um DAG na execução de um programa.

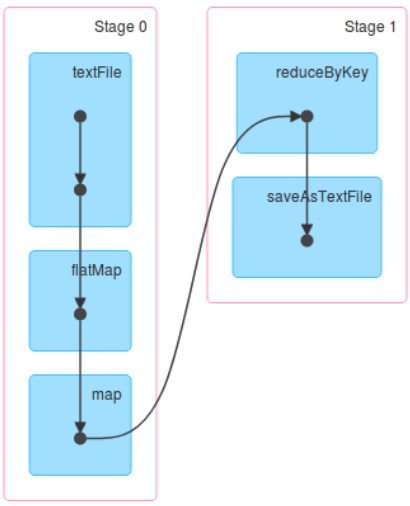

Figura 2.4: Directed Acyclic Graph (DAG) na execução do Wordcount.

2.3.1 Resenha Histórica

O Apache Spark surgiu em 2009 como um projeto de investigação no AMPLa da Universidade de Califórnia em Berkeley e o seu código foi tornado aberto no início de 2010. A maior parte das ideias que. A maior parte das ideias que suportam o sistema foram apresentadas em vários trabalhos de investigação ao longo dos anos. O projeto é atualmente desenvolvido de forma colaborativa por uma comunidade de centenas de programadores e organizações [31]. da Universidade da California em Berkeley, incluindo investigadores de aprendizagem máquina, como o projeto Mobile Millennium, que utilizou o Spark para monitorizar e prever o congestionamento de tráfego na área da Baía de São Francisco. Num curto intervalo de tempo, algumas organizações externas começaram também a utilizar o Spark e atualmente o número de organizações que utliza o Spark tem crescido. Além da Universidade da California em Berkeley, os principais contribuidores para o Spark incluem a Databricks, Yahoo e Intel [27]. A seguir são apresentados alguns marcos históricos sobre o Spark:

- O Spark tornou-se numa plataforma de código aberto pela primeira vez em março de 2010 e foi transferido para a Apache Software Foundation em junho de 2013, onde atualmente é um projeto de alto nível [31].
- Em 2011, o AMPLab começou a desenvolver componentes de nível superior no Spark, como o Shark (Hive on Spark) e o Spark Streaming. Esses e outros componentes são, às vezes, chamados de *BDAS (Berkeley Data Analytics Stack)* [31]. Desde a sua criação, o Spark tem sido um projeto muito ativo e a comunidade continua a lançar versões atualizadas do Spark em uma programação regular. Em maio de 2014 o Spark lançou a versão 1.0 [31].
- Para permitir testes do próximo lançamento do Spark 2.0 em grande escala na comunidade, a equipa do Apache Spark publicou em maio de 2016 uma versão de pré-lançamento do Spark 2.0. Em julho de 2016 a equipa disponibilizou a versão 2.0 que é a primeira na série de lançamento 2.x [6]. Como principais atualizações, a *API (Application Programming Interface)* DataFrame foi integrada com a API do Conjunto de Dados, unindo, assim, os recursos de processamento de dados nas bibliotecas Spark 2.3.2 na arquitetura do Spark. Além disso, o Spark 2.0.0 amplia os recursos do Spark *SQL (Structured Query Language)* incluindo um novo analisador ANSI SQL com suporte a sub-querys e à norma SQL: 2003 [32].
- Na série de lançamento 2.x, foram disponibilizadas muitas versões de manutenção e de correções de estabilidades. A versão atual do Spark é 2.3.0 (momento em esta dissertação está a ser escrita), que corresponde ao quarto lançamento da série 2.x, lançada a 28 de fevereiro de 2018. Em termos de alterações, destacam-se: suporte para o processamento contínuo em fluxo estruturado e um novo *back-end do Kubernetes Scheduler*. Outras atualizações importantes incluem as novas *APIs DataSource e Structured Streaming* v2 e vários aprimoramentos de desempenho do *PySpark* [6].

Esta dissertação centra-se principalmente na versão 2.2.0.

2.3.2 Componentes do Spark

O Spark suporta um conjunto de ferramentas de nível superior, que o tornam eficientes no processamento dos dados. Cada uma das ferramentas tem um objetivo. A Tabela 2.2 lista os componentes do Spark.

Tabela 2.2: Componentes do Spark(adaptado de [27], [33]).

Componentes do Spark	
Spark SQL	É o módulo do Spark para processamento de dados estruturados, fazer consultas de dados via SQL, bem como a variante Apache Hive do SQL, chamada de Hive Query Language (HQL)
Spark Streaiming	Permite o processamento de fluxos de dados como: ficheiros de logs gerados por servidores da Web de produção ou filas de mensagens contendo atualizações de status postadas por utilizadores de um serviço da web
Machine Learning Lib	É uma biblioteca de aprendizagem de máquina escalonável que fornece um algoritmo de alta qualidade e velocidade
Graphx	É um sistema de computação gráfica que permite a manipulação e o processamento paralelo de gráficos
Spark Core Component	Contém as funcionalidade básica do Spark, incluindo componentes para escalonamento de tarefas, gestão de memória, recuperação de falhas, interação com sistemas de armazenamento e fornece também APIs para criar e manipular os RDDs

2.3.3 Arquitetura do Spark

A arquitetura de uma plataforma Spark é constituída pelos seguintes componentes: Driver program, cluster manager e os executores, que são executados num worker Node. Estes componentes existem independentemente do Spark estar a ser executado num único nó ou num cluster com milhares de nó. Cada um destes componentes tem uma função específica na execução de um programa Spark. Algumas dessas funções são passivas durante a execução, como os componentes do cliente, e outras funções estão ativas na execução do programa, incluindo componentes que executam funções de computação [29]. Para enviar aplicações no spark os clientes usam um processo. Este processo que é conhecido como drive, mantém a vida útil da aplicação Spark dando o início e a conclusão. Também tem a função de planear e coordenar a execução do programa Spark e retornar o estado e/ou os resultados (dados) para o cliente. O Driver cria o Sparkcontext que é o objeto da aplicação que representa a conexão com o mestre (cluster manager) do Spark. O Sparkcontext é instanciado no início de uma aplicação Spark (incluindo os shells interativos) e é usado para o programa completo. Os executores do Spark são os processos do host nos quais as tarefas de um Spark DAG são executadas. Os executores reservam recursos de CPU e memória no nó worker no cluster do Spark. Os executores são dedicados a uma aplicação específica do Spark e terminados quando a aplicação é concluída. Um executor pode executar centenas ou milhares de tarefas dentro de um programa Spark. O gestor do cluster é o processo responsável pelo monitoramento dos workers e pela reserva de recursos nesses nós, a pedido do mestre [29]. A Figura 2.5 mostra a arquitetura do Spark.

2.3.4 Funcionamento do Spark

As aplicações em Spark são executadas como conjuntos independentes de processos num cluster coordenados pelo objeto SparkContext no respetivo programa principal (designado por program driver). Para executar num cluster, o SparkContext pode conectar-se a vários tipos de gestores de cluster (o próprio gestor de cluster autónomo do Spark, Mesos ou YARN) que aloca recursos em aplicações. Uma vez conectado, o Spark adquire executores em nós no cluster, que são processos que executam cálculos e armazenam dados para a sua aplicação. Em seguida, ele envia o código da aplicação (definido por ficheiros JAR ou Python passados para SparkContext)

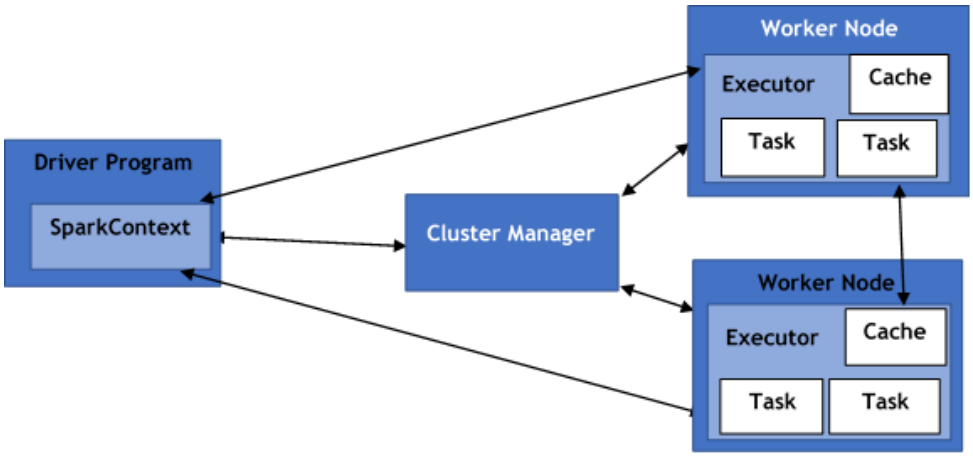

Figura 2.5: Representação esquemática da arquitetura do Spark (adaptado de [6]).

para os executores. No fim, o SparkContext envia tarefas. No fim, o SparkContext envia tarefas aos executores para executar. A Figura 2.5 mostra como é feita a execução das aplicações [6].

2.4 Apache Flink

O Apache Flink É uma plataforma de código aberto, desenvolvido pela Apache Software Foundation, projetada para realizar cálculos tirando partido da velocidade da memória e em qualquer escala. O Flink integra-se com todos os gestores de recursos de cluster comuns, tais como o Hadoop YARN, Apache Mesos e Kubernetes, mas também pode ser configurado para ser executado na nuvem ou localmente e como um cluster autónomo [8]. A principal característica do Flink reside na sua capacidade de processar fluxos de dados em tempo real [13]. O Apache Flink oferece um mecanismo de elevada tolerância a falhas para recuperar consistentemente o estado das aplicações de fluxo de dados. Esse mecanismo gera snapshots (registos instantâneos) consistentes do fluxo de dados distribuídos e do estado do operador. Em caso de falha, o sistema pode retornar a esses registos instantâneos [13]. Este mecanismo de tolerância a falhas permite que o sistema mantenha elevadas taxas de transferência e baixa latência [8].

2.4.1 Resenha Histórica

O Flink que se tornou num projeto da Apache Incubator em 2014, nasceu a partir do projeto de investigação "Stratosphere": Gestão da Informação na Nuvem (financiado pela Fundação Alemã de Investigação (DFG)), que foi iniciado como uma colaboração da Universidade Técnica de Berlim, *Humboldt-Universität zu Berlin e Hasso. Plattner-Institut Potsdam*. O Flink partiu de uma derivação do mecanismo de execução distribuído do *Stratosphere* [34], [35]. A seguir são apresentados alguns marcos históricos sobre o Flink:

- Em dezembro de 2014, o Flink foi aceito como um projeto de nível superior da Apache.
- Em março de 2016 foi lançada a versão 1.0.0 do Flink que marca o início da série de distribuições 1.xx, que manterá a compatibilidade retroativa com a versão 1.0.0. Além da compatibilidade com versões anteriores, o Flink 1.0.0 traz uma variedade de novos recursos para o utilizador, bem como: anotações de estabilidade da interface, suporte se estado fora do núcleo, pontos de gravação e atualizações de versão, biblioteca para

processamento de eventos complexos (CEP), interface de monitoramento aprimorada, incluindo submissão de trabalhos, estatísticas de pontos de verificação e monitoramento de contrapressão, melhor controlo e monitoramento de pontos de verificação, conector Kafka melhorado e suporte para Kafka 0.9. Esta versão resolveu mais de 450 problemas, incluindo correções de bugs, melhorias e novos recursos [36]. Depois desta, foram feitas dentro desta série novas distribuições com melhorias, adição de novos recursos e correções de erro: Apache Flink 1.0 (04/2016: v1.0.1 ; 04/2016: v1.0.2 ; 05/2016 v1.0.3).

- Lançamento de várias versões do Flink. 12/2017: Apache Flink 1.4 (02/2018: v1.4.1, 03/2018: v1.4.2), 06/2017: Apache Flink 1,3 (06/2017: v1.3.1, 08/2017: v1.3.2, 03/2018: v1.3.3), 02/2017: Apache Flink 1.2 (04/2017: v1.2.1), 08/2016: Apache Flink 1.1 (08/2016: v1.1.1, 09/2016: v1.1.2, 10/2016: v1.1.3, 12/2016: v1.1.4)) estas versões foram lançadas antes da verão actual [9].

- A versão actual foi lançada a 31 de julho de 2018 e é a segunda versão do bugfix da série Apache Flink 1.5. Esta versão inclui mais de 20 correções e pequenas melhorias para o Flink 1.5.1. Foi adicionado um novo recurso, que permiti que a API REST para a execução de uma tarefa forneça a configuração da tarefa como o corpo da solicitação POST. Foram corrigidos vários erros e feitas várias melhorias [9]. Nesta dissertação foi usada a versão 1.0.3.

2.4.2 Flink e Modelos Relacionados

O Apache Flink tem quatro grandes bibliotecas para fins específicos, construídas nas suas principais *APIs (DataStream API, DataSet API, Table API e Streaming SQL)*. A tabela 2.3 descreve essas bibliotecas integradas no Apache Flink e que contribuem para o melhor funcionamento da plataforma.

Tabela 2.3: Componentes do Flink (adaptado de [13]).

Componentes do Flink	
Gelly	É o sistema de processamento de grafos no Flink. Contém métodos e utilitários para o desenvolvimento de aplicações de análise de gráficos
Flink *ML (Machine Learning)*	Fornece um conjunto de algoritmos ML escaláveis e uma API intuitiva. Contém algoritmos para aprendizagem supervisionada, aprendizagem não supervisionada, pré-processamento de dados
API de tabela e SQL	É uma linguagem de expressão semelhante a SQL para fluxo relacional e processamento em batch que pode ser incorporada nas APIs de dados do Flink
Flink *CEP(Complex Event Processing)*	É a biblioteca de processamento de eventos complexos. Permite detetar padrões de eventos complexos em fluxos

2.4.3 Arquitetura do Flink

A figura 2.6 mostra a arquitectura do Flink como uma pilha de software, composta por quatro camadas principais construidas para aumentar o nível de abstração: instalação, núcleo, APIs e bibliotecas. bibliotecas. Podemos observar na Figura 2.6 que o Flink pode ser executado na nuvem ou no modo local e num cluster independente ou num cluster gerido por YARN ou Mesos.

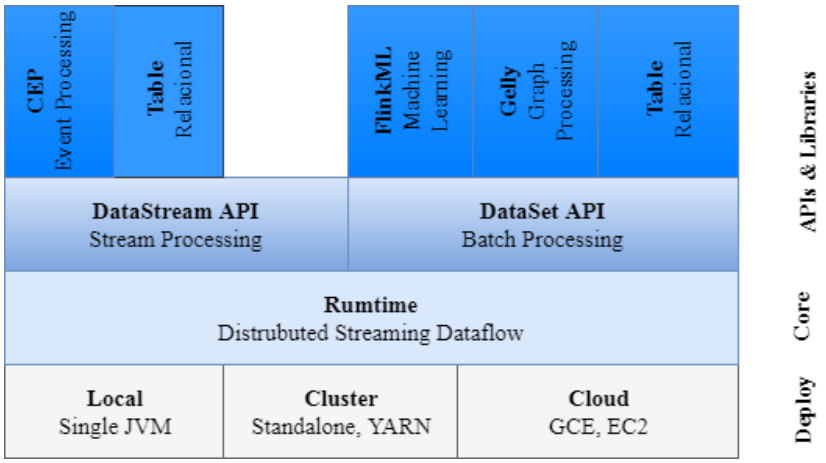

Figura 2.6: Arquitetura do Flink (adaptado de [35]).

O núcleo do Flink (Runtime) é um mecanismo de fluxo de dados de fluxo distribuído [8], que executa programas de fluxo de dados. O runtime do Flink é um DAG que contem estados de operadores conectados com fluxos de dados. Podem-se distinguir duas APIs principais na pilha do Flink, a DataSet e a DataStream, ambas criam programas executados pelo núcleo. A API DataSet processa conjuntos de dados limitados (processamento em batch) e a API DataStream processa fluxos de dados potencialmente ilimitados (processamento de fluxo) [37]. Além das APIs encontramos bibliotecas específicas do domínio: Flink ML, Gelly e tabela (para operações semelhantes a SQL), descritas na tabela 2.3. O Flink não fornece seu próprio sistema de armazenamento de dados, fornece conectores de fontes e coletores de dados para sistemas como o Amazon Kinesis, o Apache Kafka, o HDFS, o Apache Cassandra e o ElasticSearch **Wiki2018Flink**

2.4.4 Funcionamento do Flink

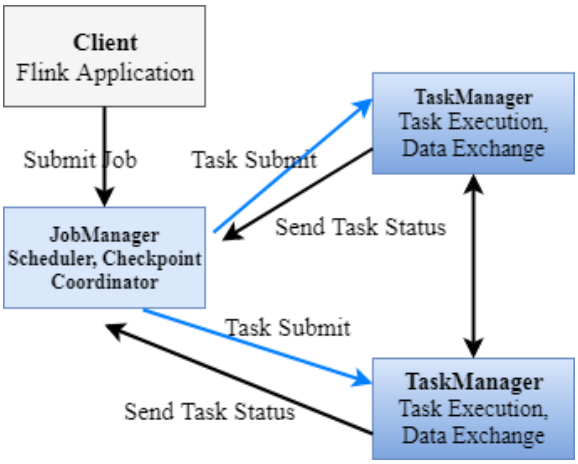

Figura 2.7: Representação esquemática do funcionamento do Flink (adaptado de [38], [37]).

A figura 2.7 mostra a execução de uma aplicação de streaming no Flink. O Flink segue o modelo mestre-escravo composto pelos seguintes componentes: o JobManager, os TaskManagers e o cliente. O cliente pega o código do programa e o transforma em um gráfico de fluxo de dados lógico não paralelos (Job-Graph) e o envia para o JobManager, este por sua vez converte o Job-Graph em um grafo de fluxo de dados físico paralelo (o Execution Graph consiste em pelo menos uma tarefa). O JobManager atribui cada tarefa a um TaskManager que inicia imediatamente a

execução da tarefa recebida [38]. O JobManager é o coordenador do sistema, ele rastreia o estado e o progresso de cada operador e fluxo, programa novos operadores e coordena os pontos de verificação e a recuperação, enquanto que os TaskManagers são os trabalhadores que executam um ou mais operadores que produzem fluxos e reportam o seu estado ao JobManager, dado que nele ocorre o processamento de dados reais [37]. As tarefas em execução trocam dados com suas tarefas anteriores e posteriores.Se as tarefas de envio e receção forem executadas em diferentes TaskManagers, os dados serão enviados pela rede. Caso contrário os dados são trocados localmente. Em ambos os casos, o TaskManager é responsável pela transferência de dados entre as tarefas [38]. A figura 2.8 mostra um exemplo do Flink executando um programa.

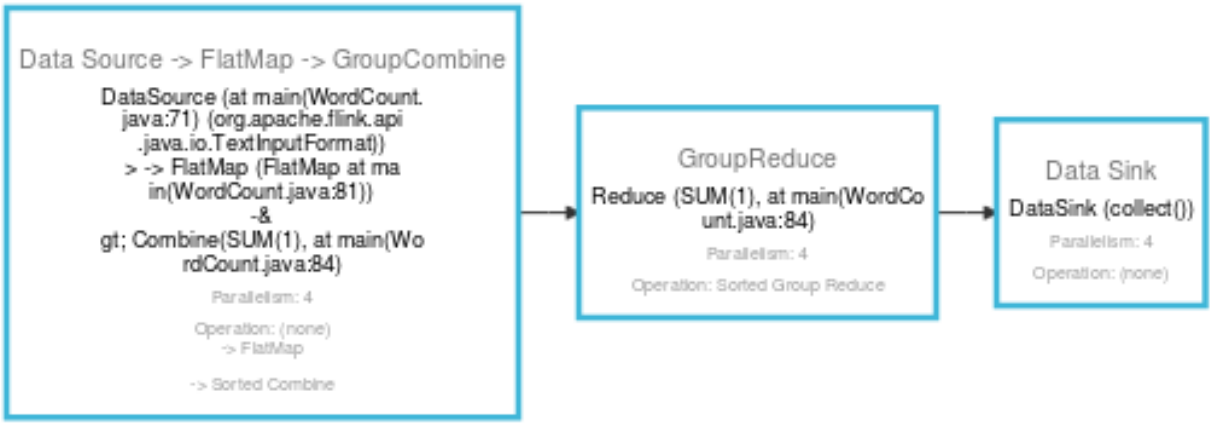

Figura 2.8: Representação esquemática ilustrando o Flink a executar o programa wordcount.

2.5 Hadoop Versus Spark Versus Flink

Estas tecnologias são vistas por vezes como concorrentes, mas é crescente o consenso de que são melhores quando conjugadas. O Hadoop, o Spark e o Flink são plataformas de Big Data, mas foram projetadas para fins e casos de usos diferentes. A Apache Software Foundation agrupa-as em diferentes categorias sendo: o Hadoop uma plataforma de processamento distribuídos de dados em batch, enquanto que Spark e o Flink são uma plataforma de processamento distribuídos de dados em fluxo e em batch. O Hadoop e o Spark podem trabalhar de forma independente (um sem o outro), mas muitos afirmam que funcionam melhor quando conjugadas. Por outro lado, o Spark não substitui o Hadoop mas completa-o nos casos em que é necessário fazer análise em fluxos de dados quase em tempo real, já que o Hadoop MapReduce faz o processamento de dados em batch [39]. Já o Flink é um substituto para o Hadoop MapReduce que funciona nos modos de batch e fluxo contínuo (é uma plataforma híbrida) [40].

2.5.1 Comparação Arquitetural e Funcional

O Spark, o Hadoop e o Flink apesar de serem plataformas de big data divergem na sua arquitetura e funcionamento. A Tabela 2.4 resume uma comparação de alguns pontos considerados relevantes entre as três plataformas.

Tabela 2.4: Comparação arquitetural e funcional do Hadoop, Spark e Flink (adaptado de [41], [7]).

Comparação Arquitetural e Funcional			
Pontos	**Apache Hadoop**	**Apache Spark**	**Apache Flink**
Modelo de Computação	MapReduce suporta apenas o modelo orientado em batch	Adotou o micro-batch (coletar e depois processar)	Fluxo contínuo baseado no operador, processa os dados quando eles chegam, sem atrasos na coleta dos mesmos
Requisitos de Hardware	Funciona bem no hardware commodity	Hardware de médio a alto nível	Hardware de médio a alto nível
Linguagem	Escrito em JAVA, suporta C, C++, Ruby, Groovy, Perl e Python	Escrito no Scala, e fornece API em JAVA, Python e R	Suporta Scala, Python e R. O Flink é implementado em Java. Ele fornece a API do Scala
Processamento	Processamento em batch, não é iterativo	Processamento em batch, fluxo contínuo e gráfico, os dados são iterados em batch	Fornece um único tempo de execução para o fluxo e o processamento em batch
Fluxo de Dados	MapReduce não possui loop no seu fluxo de cálculo. Cada fase é avançada utilizando como entrada, a saída da fase anterior, Utiliza o Apache Mahout para a aprendizagem de máquina	Possui um algoritmo de aprendizagem máquina (fluxo de dados cíclicos); é representado como um gráfico acíclico direto (DAG)	Suporta gráfico de dependência cíclica controlada em tempo de execução.
Mecanismo de Transmissão	O MapReduce lê um grande conjunto de dados de entrada numa única vez processa e produz o resultado	O Spark Streaming processa fluxos de dados em micro-batchs. os RDDs permitem executar várias operações do mapa na memória	É o verdadeiro mecanismo de streaming. Ele usa fluxos para cargas de trabalho: fluxo contínuo, SQL, micro-batch e batch.
Velocidade	Mapreduce produz muitos dados intermediários trocados entre os nós, isso aumenta a latência da E/s do disco	É mais rápido que o MapReduce armazena em cache grande parte dos dados de entrada na memória por RDD e mantém os dados intermediários na memória. Quando necessário guarda os dados no disco	Processa mais rapidamente que o Spark devido à sua arquitetura de streaming, processa parte dos dados alterados para aumentar o seu desempenho.
Abstração	No MapReduce não há nenhum tipo de abstração.	Abstração do Spark RDD para batch e o DStreaming para String (internamente é o RDD)	Abstração de DataSet para batch e DataStreams para aplicação de streaming
Tolerância a Falha	MapReduce é altamente tolerante a falha. Quando há falha no Hadoop, ele não precisa reiniciar a aplicação. Suporta a tolerância a falha por meio da replicação	O Spark Streaming recupera a tarefa perdida sem configuração extra, utiliza o RDD e vários modelos de armazenamento de dados para tolerância a falha, minimizando a E/S da rede	Mecanismo baseado em snapshots distribuídos da Chandy-Lamport
Escalabilidade	MapReduce é escalável, foi utilizado na produção de dezenas de milhares de nós.	É altamente escalável, em função das necessidades é possível adicionar o número de nós. Um grande cluster Spark conhecido é de 8000 nós	É altamente escalável
Segurança	Suporta a autenticação kerberos (protocolo de rede para comunicações seguras em uma rede insegura).	Suporta autenticação via segredo compartilhado (autenticação por palavra passe). Se o Spark for utilizado no HDFS poderá fazer o uso do *ACLs (Access Control List)*. Também pode ser executado no YARN para beneficiar da autenticação kerberos	Suporta autenticação de utilizadores por meio da infraestrutura do Hadoop/ Kerberos, os programas de streaming podem se autenticar como intermediários de fluxo via *SSL(Secure Socket Layer)*.
Escalonadores	Possui dois escalonadores para carga de trabalho multi-utilizador: *Fair Scheduler e Scheduler capacity*. Para fluxos complexos, precisa de um escalonador de tarefa externo como o Oozie.	Atua em seu próprio escalonador de fluxo	Pode usar o *YARN Scheduler*, mas também possui seu próprio escalonador
Análise em Tempo Real	MapReduce Falha no processamento de dados em tempo real por ter sido projetado para processar grande volume de dados em batch.	Pode processar dados em tempo real, isto é, dados provenientes de fluxos de eventos em tempo real	Usado principalmente para análise de dados em tempo real
Compatibilidade	Hadoop MapReduce e Spark são compatíveis. O Spark compartilha todas as compatibilidades do MapReduce para origens de dados, formatos de arquivos e ferramentas de *business intelligence* via *JDBC (Java Database Connectivity) e ODBC (Open Database Connectivity)*.	Pode ser executado em cluster do Hadoop por meio do modo Stand-alone do YARN e pode processar dados em HDFS, HBase, Cassandra e Hive	Fornece um pacote de compatibilidade do Hadoop para agrupar as funções implementadas nas interfaces do Hadoop MapReduce e incorporá-las aos seus programas.

2.5.2 Comparação em Termos de Desempenho

O Spark e o Hadoop, embora sejam projetos separados, estão intimamente relacionados entre si como componentes críticos do cenário de big data [29]. O Hadoop armazena os dados intermédios num sistema de ficheiros (HDFS) baseado em disco [42]. Isso afeta seu desempenho e torna-o inadequado para aplicações em tempo real [11],enquanto que o Spark armazena estes dados nas memórias dos nós de computação distribuída como (RDD), o que o torna mais rápido [42]. O micro-batch usado pelo Spark pode não ser rápido o suficiente em sistemas que exigem latência muito baixa, pelo que o Flink se encaixa perfeitamente nesses sistemas, pois usa nativamente fluxos para todos os tipos de cargas de trabalho [13]. O MapReduce lê dados do disco

e após uma iteração específica, envia resultados para o HDFS e lê novamente os dados do HDFS para a próxima iteração. Este processo aumenta a latência e torna o processamento de gráficos lento enquanto que o Spark possui uma biblioteca de computação gráfica (GraphX). O suporte gráfico e a computação na memória permitem que o algoritmo tenha um bom desempenho [41]. O Apache Spark é outro sistema de processamento em batch, mas é relativamente mais rápido que o Hadoop MapReduce, guarda em cache grande parte dos dados de entrada na memória por RDD e mantém dados intermédios na própria memória, grava os dados no disco sempre que necessário. Em contraste, o Flink tem um processamento completamente iterativo em seu mecanismo baseado em fluxos de dados cíclicos (uma iteração, uma programação) [13]. O Spark é 100 vezes mais rápido que o MapReduce [7], mas o desempenho do Apache Flink é excelente em comparação com qualquer outro sistema de processamento de dados pois usa operadores nativos de iteração de loop fechado que tornam a aprendizagem máquina e o processamento de gráficos mais rápidos em comparação com Hadoop e o Spark [7].

2.6 Trabalhos Relacionados

Muitos estudos têm sido realizados por diversos investigadores para comparar o desempenho das principais plataformas de big data: Hadoop, Spark e Flink. Esta subsecção é dedicada a trabalhos de investigação sobre análise e comparação do desempenho do Hadoop, Spark e Flink. Uma investigação reportada em [42] por Han et al. mostra o impacto do tamanho da memória no processamento distribuído de grande volume de dados, comparando o o tempo de execução do algoritmo K-means do benchmark HiBench em clusters Hadoop e Spark, com diferentes tamanhos de memórias alocadas aos nós. Han et al. fizeram experiências usando o algoritmo K-means, que é um dos algoritmos de aprendizagem de máquina representativos fornecidos pelo benchmark HiBench. Aumentaram o tamanho dos dados de 1 a 8 GB usando o benchmark Generator do HiBench. Os resultados das experiências por eles realizadas demonstraram que o cluster do Spark é mais rápido que o cluster do Hadoop, desde que o tamanho da memória seja grande o suficiente para o tamanho dos dados. Mas, com o aumento do tamanho dos dados, o cluster do Hadoop supera o cluster Spark, pois com esse aumento dos dados, o Spark precisa substituir os dados do disco por dados em cache de memória e isso requer mais tempo e diminui rapidamente a taxa de processamento dos dados. Por outro lado, Mavridis e Karatza [11] compararam o desempenho de aplicações de análise de logs realistas no Hadoop e no Spark. Eles Compararam experimentalmente o Hadoop e o Spark e avaliaram o desempenho, investigando o tempo de execução, a escalabilidade, a utilização de recursos, o custo e o consumo de energia. Realizaram várias experiências com diferentes números de nós escravos, tamanhos de ficheiro de entrada e tipos de aplicação. Estudaram o tempo total de execução das aplicações de análise de log do Hadoop e Spark. Mavridis e Karatza relataram que o Spark devido à exploração efetiva da memória principal e o uso de técnicas de otimização de eficiência era mais rápido do que o Hadoop em todos os casos. Samadi et al. [12] apresentaram uma comparação de desempenho entre duas plataformas populares de big data implantadas em máquinas virtuais. Para comparar o desempenho, os autores utilizaram o pacote de benchmark HiBench, com base em três critérios: tempo de execução, taxa de transferência e aumento de velocidade. Testaram a carga de trabalho do Wordcount com diferentes tamanhos de dados para resultados mais precisos. Os resultados experimentais por eles obtidos demonstraram que o desempenho dessas plataformas varia significativamente com base na implementação do caso de uso. Além disso, a partir dos resultados concluíram que o Spark é mais eficiente que o Hadoop para lidar com

uma grande quantidade de dados e que necessita de alocação de memória mais alta. Garcia-Gil et al. [13] no seu artigo reportaram um estudo comparativo sobre a escalabilidade do Apache Spark e Apache Flink, usando as bibliotecas de Machine Learning correspondentes para o processamento de dados em batch. disso, analisaram o desempenho das duas bibliotecas de aprendizagem máquina que o Spark possui atualmente, MLlib e ML. Para as experiências foram usados os mesmos algoritmos e o mesmo conjunto de dados. conjunto de dados. Os resultados obtidos mostraram que o Spark MLlib tem melhor desempenho e tempos de execução mais baixos do que o Flink. Para Marcu et al. [43], nenhuma das duas plataformas supera a outra para todos os tipos de dados, tamanhos e padrões de trabalho, tendo mostrado isso num artigo onde avaliaram diretamente o desempenho do Spark e do Flink, onde desenvolveram uma metodologia para correlacionar as configurações dos parâmetros e o plano de execução dos operadores com o uso dos recursos. Em [14], Veiga et al. Veiga et al. abordam o problema da avaliação comparativa do Hadoop, Spark e Flink usando cargas de trabalho de Big Data representativas e considerando fatores como desempenho e escalabilidade. Veiga et al. caracterizaram o comportamentos das plataformas pela modificação de alguns dos principais parâmetros das cargas de trabalho, como tamanho do bloco HDFS, tamanho dos dados de entrada, rede de interconexão ou configuração de encadeamento. Os resultados experimentais obtidos mostram que substituir o Hadoop por Spark ou Flink pode levar a melhorias significativas no desempenho. Existem estudos de desempenho de cada uma das plataformas consideradas nesta dissertação, assim como estudos de desempenho envolvendo a comparação entre o Apache Hadoop e Apache Spark e a comparação entre o Apache Spark e o Apache Flink. No entanto, existem relativamente poucos estudos comparativos do desempenho das plataformas Hadoop, Spark e Flink. Nesta dissertação é apresentada uma comparação do desempenho das plataformas Hadoop, Spark e Flink usando o Benchmark Hibench-master 7.

2.7 Conclusão

Este capítulo forneceu o background que servem de base para tornar claro o tema em estudo, foram apresentados os conceitos necessários para a compreensão do Hadoop, Spark e Flimk e trabalhos relevantes feito na área de big data. O Hadoop, o Spark e o Flink são projetos de código aberto da Apache Software Foundation e representam as plataformas mais utilizados na análise de grande conjunto de dados. A principal diferença entre os três está no processamento: o Spark faz o processamento dos dados na memória, já o Hadoop MapReduce lê e escreve os dados no disco. Isto conduz a diferentes desempenhos em termos de velocidade de processamento. Em quanto que o Flink fornece um único tempo de execução para o fluxo e o processamento em batch, o Spark pode ser até 100 vezes mais rápido do que o Hadoop se o volume de dados for baixo. O Hadoop MapReduce é capaz de processar conjuntos de dados maiores do que Spark, mas o Flink processa dados mais rapidamente que o Spark devido à sua arquitetura de streaming. A escolha de uma das três platformas depende muito das necessidades das aplicações: para o processamento de grandes volumes de dados, uma boa alternativa é o Hadoop MapReduce; para processamento iterativo e processamento gráfico, o Spark apresenta vantagens; mas para análise em tempo real, processamento de fluxo de dados contínuo, alta taxa de transferência e baixa latência, o Flink apresenta vantagens. Grande parte da análise e comparação foi feita entre o Spark, Hadoop MapReduce e Flink já que todas são para processamento de dados.

Capítulo 3

Ambiente de Teste e Resultados Experimentais

3.1 Introdução

Neste capítulo são apresentadas as características do ambiente de teste bem como a configuração das plataformas e os resultados experimentais obtidos. Este capítulo está organizado em quatro secções. Depois desta secção introdutória, a Secção 3.2 apresenta uma introdução dos benchmarks, categorias de benchmarks para big data e uma descrição do Hibench benchmark suite que é o benchmark escolhido para avaliar os clusters; a secção 3.3é dedicada às características dos clusters onde foram instalados o Hadoop, o Spark e o Flink; a secção 3.4 é dedicada à configuração das três plataformas Hadoop, Spark e Flink e aos resultados experimentais.

3.2 Benchmark

Em computação, um benchmark é um programa de computador que tem como objetivo avaliar o desempenho relativo de um objeto normalmente executando uma série de testes padrões e ensaios. O benchmarking é associado à avaliação de características de desempenho de hardware, mas a técnica também é aplicável a software [44].

3.2.1 Categorias de Benchmarks para Big Data

os benchmarks existentes são agrupados em três categorias: Micro Benchmarks, End-to-End Benchmarks e Benchmarks suites. A Tabela 3.1 mostra os benchmarks de big data para sistemas relacionados com o Hadoop.

Tabela 3.1: Benchmarks agrupados em categorias (Adaptado de [45]).

Benchmark agrupados em categoria		
Categoria	**Descrição**	**Benchmark**
Micro benchmarks	É usada para avaliar componentes individuais do sistema ou comportamentos específicos do sistema.	Hadoop built-in micro benchmarks, NNBench, TestDFSIO, DFSCIOTest, ThreadedMapBench, HiBD, TPCx-HS, AMPLab benchmark
End-to-End Benchmarks	É projetada para avaliar sistemas inteiros usando cenários de aplicações típicas, cada cenário corresponde a uma coleção de cargas de trabalho relacionadas.	GridMix, SWIM, MRBench, PigMix, YCSB, YCSB++
Benchmark suites	São combinações de diferentes benchmarks micro e/ou end-to-end e essas suítes visam fornecer soluções abrangentes de benchmarking.	HiBench, PUMA, MRBS, CloudSuite, BigDataBench, HcBench.

Neste trabalho foi utilizado o benchmark suite HiBench, que foi desenvolvido pela Intel e que consiste num conjunto de shell scripts publicados sob a licença Apache 2 [46].

3.2.2 HiBench

O HiBench é um conjunto de benchmark de big data que ajuda a avaliar diferentes estruturas de big data em termos de tempo de execução, throughput e utilização de recursos do sistema. Contém um conjunto de cargas de trabalho para Hadoop, Spark, streaming e Flink, incluindo Sort, WordCount, TeraSort, Sleep, SQL, PageRank, indexação Nutch, Bayes, K-means, NWeight e DFSIO aprimorado. A versão atual do Hibenche, HiBench-master 7 [10] possui um total 19 cargas de trabalho, divididas em 6 categorias de benchmarks que são: micro, ml (aprendizagem de máquina), sql, gráfico, pesquisa na web e streaming [10]. A Tabela 3.2 mostra a constituição dos benchmarks da suite Hibench-master 7 e suas respetivas cargas de trabalhos disponíveis para o Hadoop, o Spark e o Flink. Na Tabela 3.2 vamos descrever apenas as cargas de trabalhos utilizadas

Tabela 3.2: Constituição dos Benchmarks por Framework.

Categorias	Workloads	Hadoop	Spark	Flink
Micro	dfsioe	✓	✓	
	sleep	✓	✓	
	sort	✓	✓	
	terasort	✓	✓	
	wordcount	✓	✓	
ML	als		✓	
	bayes		✓	
	k-means	✓	✓	
	gbt	✓	✓	
	lda		✓	
	linear		✓	
Sql	aggregation	✓	✓	
	join	✓	✓	
	scan	✓	✓	
Graph	nweight		✓	
Streaming	fixwindow		✓	✓
	identity		✓	✓
	repartition		✓	✓
	wordcount		✓	✓
Websearch	nutchindexing	✓	✓	
	pagerank	✓		

Micro benchmarks:

- Sort (ordenar)

  O Sort é uma carga de trabalho que classifica os dados de entrada de texto que são gerados usando *RandomTextWriter*. [10]. O programa sort depende da plataforma do Hadoop para classificar os resultados finais e tanto as suass funções map e reduce são funções que emitem diretamente a entrada, pares de chave-valores como saída. A classificação dos dados é feita automaticamente durante o Shuffle and Merge. Este processo esta ligado a I/O-bound [47].

- WordCount (contagem de palavras)

  O Wordcount conta a ocorrência de cada palavra nos dados de entrada que são gerados usando *RandomTextWriter*. Ele é representativo de outra classe típica de tarefas MapReduce do mundo real - extrai uma pequena quantidade de dados interessantes de um grande conjunto de dados. Este processo está ligado à CPU [10]. No WordCount, cada tarefa de

mapa emite (word, 1) para cada palavra na entrada, o combinador calcula a soma parcial de cada palavra em uma tarefa map e a tarefa de redução simplesmente calcula a soma final para cada palavra [47].

- TeraSort

  O TeraSort é uma referência padrão criada por Jim Gray. Os seus dados de entrada são gerados pelo programa de exemplo Hadoop TeraGen [10]. Os dados de entrada para essa carga de trabalho são gerados ao executar o script de preparação que cria 10 bilhões de registros de 100 bytes gerados pelo programa TeraGen contido na Distribuição do Hadoop. TeraGen usa map ou reduz para produzir dados, dividindo o número desejado de linhas pelo número desejado de tarefas e atribui intervalos de linhas a cada mapa. O TeraSort coleta amostras dos dados de entrada e usa o map ou reduz para classificar os dados em uma ordem total. O processo é limitado pela CPU durante o fase Mapeamento e I/O durante a fase de Redução [47].

Machine Learning - As cargas de trabalho contidas nesta categoria de benchmarks são incluídas no HiBench porque representam um dos usos importantes da aprendizagem máquina em grande escala [47]:

- K-means clustering (Kmeans)

  O K-means (um algoritmo de agrupamento para descoberta de conhecimento e mineração de dados) testa o armazenamento em cluster. O conjunto de dado de entrada é gerado pelo GenKMeansDataset com base na Distribuição Uniforme e na Distribuição Guassiana [10]. Essa carga de trabalho, primeiro calcula o centróide de cada cluster executando um trabalho do Hadoop de forma iterativa, até o número máximo de iterações ser atingido. Depois disso, ele executa um trabalho de cluster atribuindo cada amostra a um cluster. Esse processo é limitado pela CPU durante a iteração e ligado à E/S durante o agrupamento [47].

Benchmarks Websearch - As cargas de trabalho Nutch Indexing e PageRank estão incluídas no HiBench para avaliar as plataformas de big data porque representam um dos usos mais significativos de MapReduce (sistemas de indexação de pesquisa em grande escala) [47]:

- PageRank (pagerank)

  Esta carga de trabalho faz o benchmark do algoritmo PageRank implementado nos exemplos de referência do Spark-MLLib / Hadoop. A fonte de dados é gerada a partir de dados da Web cujos hiperlinks seguem a distribuição Zipfian [10]. Ele calcula as fileiras das páginas da web de acordo com o número de links de referência. A carga de trabalho do PageRank consiste em uma cadeia de tarefas do Hadoop, entre os quais vários trabalhos são iterados até a condição de convergência ser satisfeita. Este processo esta ligado a CPU-bound [48].

As cargas de trabalho dos benchmarks possuem vários perfis de escalas de dados: *tiny, small, large, huge, gigantic e bigdata*. Entre estes selecionamos as escalas *small, large e gigantic* [49], devido à limitação de recursos para big data. Na nossa experiência, utilizamos cinco cargas de trabalho do HiBench-master 7 incluindo Pagerank, Sort, Terasort e Wordcount e K-means. O tamanho dos dados de entrada são variáveis, pois são gerados automaticamente executando o scripts de preparação em cada referência. Para entender a diferença e comparar as plataformas, utilizamos as seguintes métricas essenciais: tempo de execução das tarefas, throughput

(número de tarefas concluídas por minuto), consumo de recursos (CPU, memória e E/S). A escolha do Hibench para avaliação das plataformas foi motivada pela disponibilidade, a facilidade de configuração e por possuir cargas de trabalhos que suportam as características das três plataformas como exemplo padrão e a similaridade da implementação utilizando Hadoop, Flink e o Spark.
Para avaliação do desempenho das plataformas Hadoop, Spark e Flink foram usadas as métricas tempo de execução e Throughput, as quais são definidas da seguinte forma:

- Tempo de Execução: É o tempo necessário que uma plataforma leva para executar uma carga de trabalho. É a diferença entre o tempo de início e a hora de término. Considera-se que tem melhor desempenho o que tiver menos tempo decorrido e esse tempo é medido em segundos.
- Throughput: É a quantidade de trabalho que pode ser executada em um determinado período de tempo. É medido em byte/segundo. Tem melhor desempenho o que executar mais trabalho.

3.3 Ambiente de Teste

Esta secção apresenta as caraterísticas de software e hardware onde foram instaladas as plataformas, bem como as configurações das mesmas.

3.3.1 Instalação do Cluster do Hadoop, Spark e Flink

A experiência foi realizada em um cluster homogéneo com quatro máquinas físicas, sendo um nó master e três nós slaves, todos conectados a um switch Fast Ethernet 10/100 Base-TX. As configurações detalhadas de hardware e software do servidor são mostradas na Tabela 3.3. Todas as máquinas estão conectadas à mesma rede (10.0.5.0/25) através de um switch com oito portas em 10/100 Base-TX para permitir a comunicação entre elas. Também foi configurado o protocolo SSH para permitir o acesso remoto das mesmas. As três plataformas foram configuradas no modo distribuído. Tanto o Spark (utiliza o YARN para gestão de recursos) como o Flink utilizam o HDFS para armazenamento dos dados. Fizemos os benchmarkings do Hadoop e Spark utilizando os benchmarks, micro, machine learning e websearch com geradores de dados das cargas de trabalhos correspondentes contidos na versão atual Hibench-master 7. Utilizamos as configurações recomendadas nos ficheiros de configuração das plataformas. Todas as ferramentas utilizadas foram escolhidas devido à disponibilidade, abertura de código facilidade de implementação, acessibilidade, uso frequente por empresas que utilizam tecnologias de big Data e por terem uma comunidade de desenvolvimento ou contribuidores ativos e disponíveis para dar suporte para quaisquer erros.

3.3.2 Configuração das Plataformas

Os Clusters do Hadoop, do Spark e do Flink foram instalados ao descompactar o software em todos os nós no cluster. Um nó no cluster foi designado de hadoop-master que é o nó principal no qual estão em execução o Namenode e o ResourceManager (para o Hadoop), O Master e o Resource Manager (para o Spark), e o JobManager (para o Flink). As três máquinas restantes no cluster são chamadas de Slaves (Slave1, Slave2 e Slave3) nas quais estão em execução o

Tabela 3.3: Configuração de Hardware e Software das máquinas.

Descrição de Software e Hardware Utilizados	
Processador	Intel(R)Core(TM) i7-7700 CPU @ 3.60GHz*8 4 cpu-core
Placa Gráfica	Intel® HD Graphics 630 (Kaby Lake GT2)
Memória	15,5 GB
Sistema Operativo	Ubuntu 16.04.1
Disco	983,4 GB
JVM	jdk-1.8.0
Hadoop versão	Hadoop-2.9.1
Spark versão	Spark -2.2.0
Flink versão	Flink -1.0.3
Hibench versão	Hibench-master -7.0

NodeManager, Datanode (Hadoop), worker (Spark) e o taskmanager (Flink). Antes da instalação e configuração das plataformas, foi instalado o java e configurado o SSH em todo o cluster. As etapas de instalação e configuração encontram-se no Apêndice A. As figuras 3.1, 3.2 e 3.3 abaixo apresentam os clusters do Hadoop, do Spark e do Flink configurados.

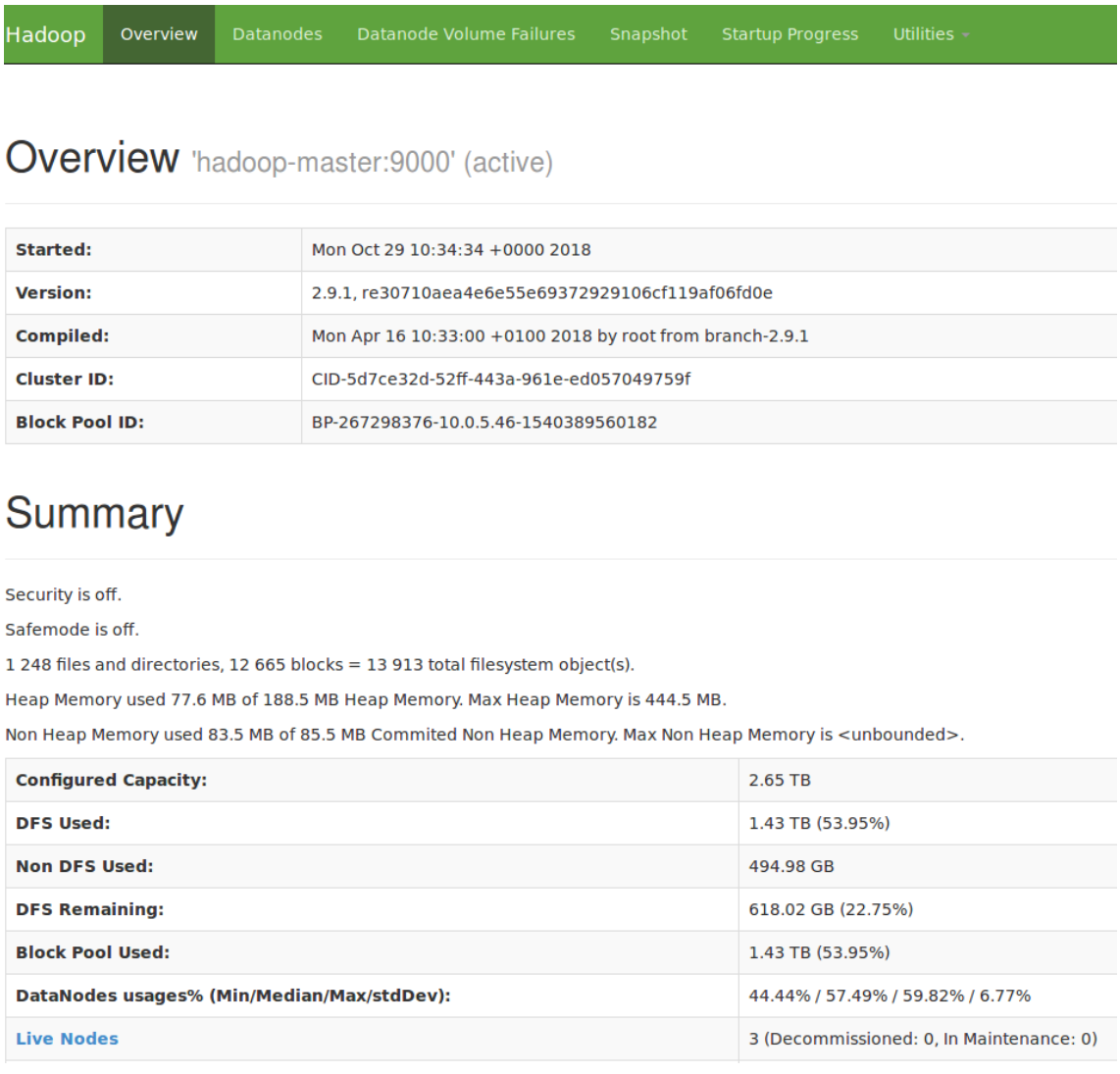

Figura 3.1: Cluster do Hadoop configurado.

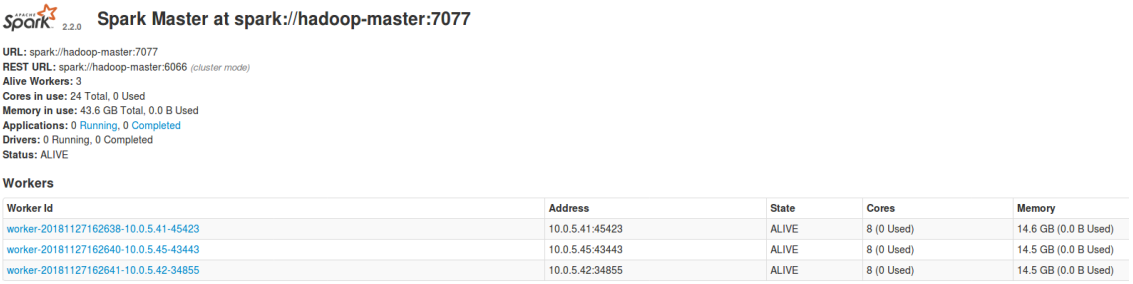

Figura 3.2: Cluster do Spark configurado.

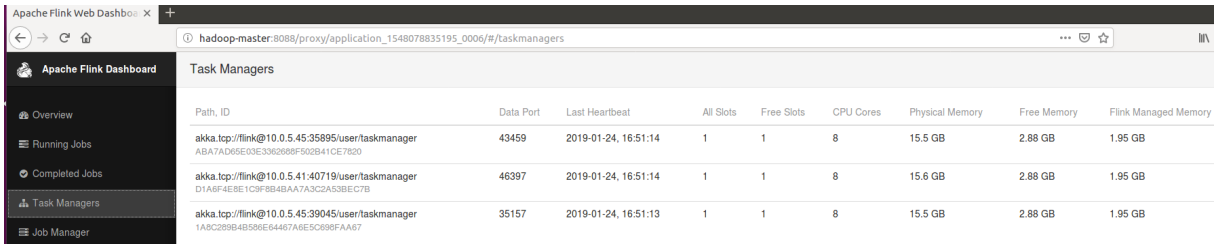

Figura 3.3: Cluster do Flink Configurado.

3.4 Análise dos Resultados Experimentais

Nesta secção apresentamos os resultados das experiências que avaliam e caracterizam as plataformas do Hadoop, Spark e Flink executando os benchmarks do HiBench-master 7.

3.4.1 Caracterização dos Recursos Utilizados pelo Hadoop e pelo Spark

As figuras de 3.4- 3.13 mostram os recursos utilizados pelos clusters do Hadoop e Spark ao executarem as cargas de trabalho (sort, terasort, wordcount, pagerank e k-means) do Hibench-master 7: memória, rede, CPU e Disco E/S. A análise dos recursos foi feita depois de serem executadas as cargas de trabalho com a escala de dados *gigantic* que foi o maior volume de dados utilizados no teste.

A carga de trabalho Sort simplesmente transforma os dados (de uma representação para outra), em função disso os dados de entrada, os dados aleatórios (da fase shuffle) e a sáida das tarefas de sort geralmente possuem os mesmos tamanhos. O processo de classificação dos dados (feito automaticamente durante o Shuffle and Merge) está ligado à E/S do disco. Nas figuras 3.4-Hadoop e 3.5-Spark podemos observar que o Hadoop ao executar o sort tem maior utilização da CPU durante a fase map e o princípio da fase reduz, envolvendo até 50% as tarefas do utilizador, as tarefas do sistema e a espera no processo E/S tem envolvimento de até 25%; têm maior envolvimento na fase reduz para o Spark as tarefas do utilizador envolvem cerca de 50% do tempo da CPU e as tarefas do sistema e a espera E/S têm um envolvimento de mais ou menos 25%. Para E/S do Disco nota-se que no Hadoop são lidos e escritos aproximadamente 228.88 MiB/s e são realizadas mais ou menos 750 iops operações de leitura e escrita, já no Spark mais ou menos 228.88MiB/s são lidos e escrito e realizadas cerca de 1800 iops operações de leitura e escrita. Durante o início do job e a fase reduz a Memória principal é utilazada até mais ou menos 50% e tendo ocupação do buffer cache de quase 100% tanto para o Hadoop quanto para o Spark. Na rede, o número de pacotes enviado é o mesmo que recebidos com um valor 30370

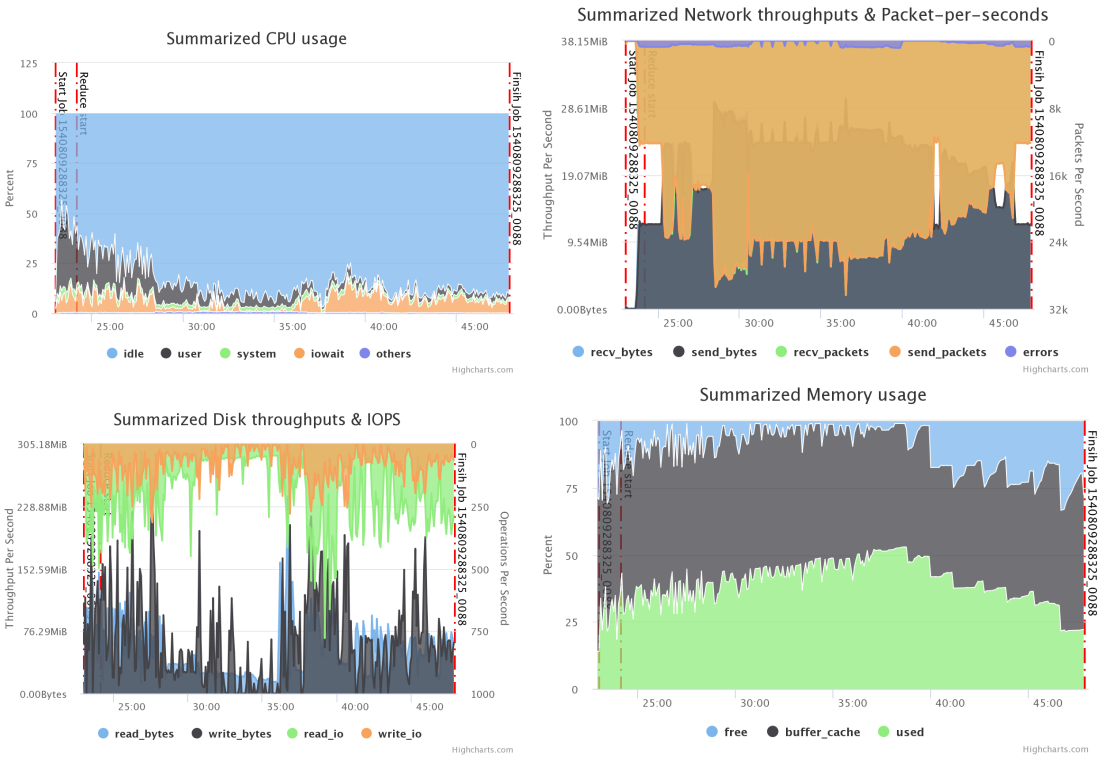

Figura 3.4: Utilização do sistema Hadoop executando o Sort.

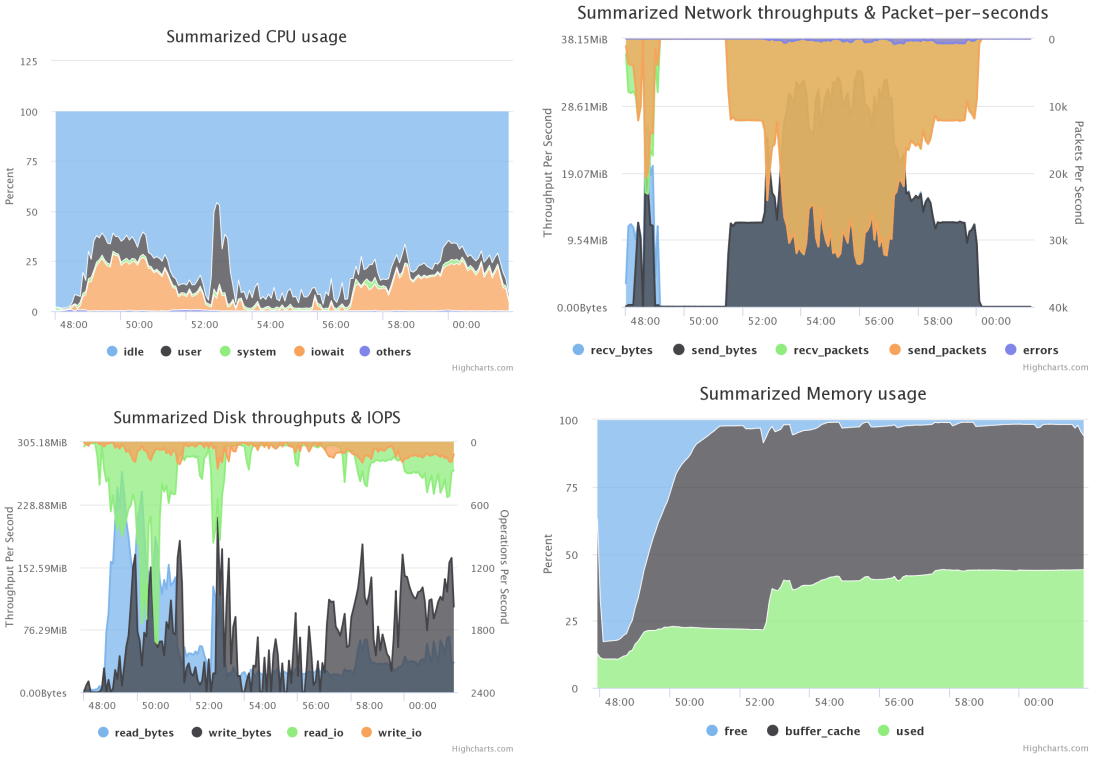

Figura 3.5: Utilização do sistema Spark executando o Sort.

pacotes, 316 pacotes falhados para o Hadoop enquanto isso no Spark há oscilação nesses valore onde são enviados e recebidos 32334 pacotes contra 825 pacotes falhados. O Spark usa mais a rede durante a fase intermediaria.

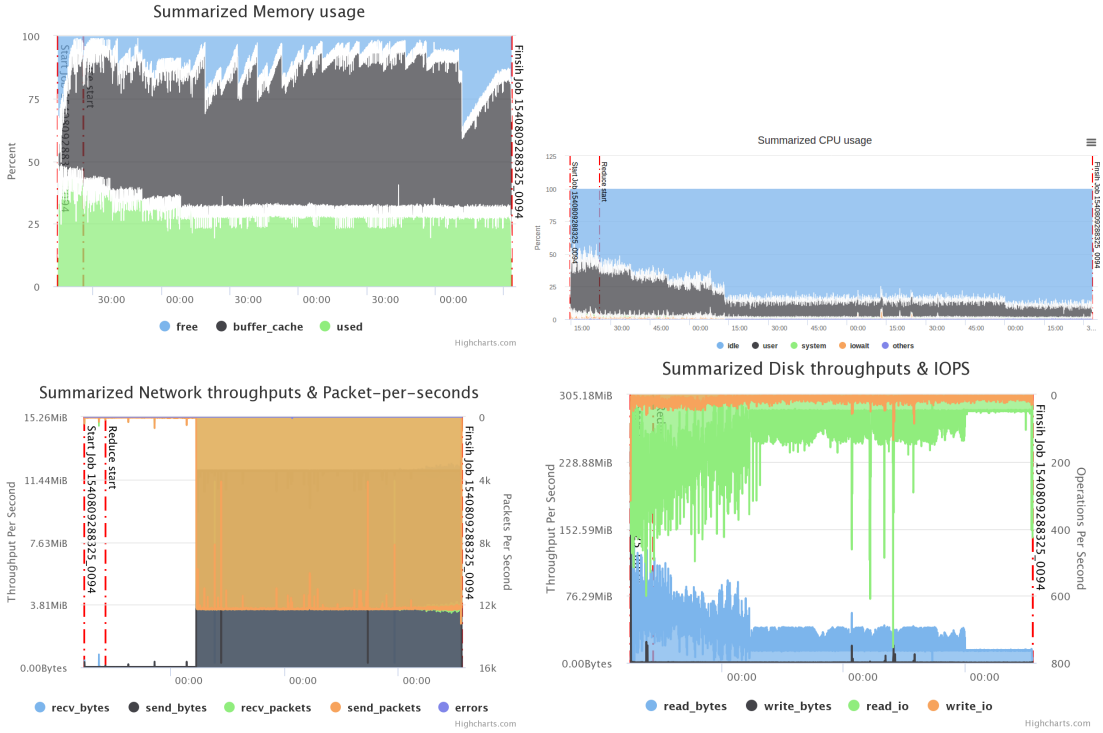

Figura 3.6: Utilização do sistema Hadoop executando o Wordcount.

Figura 3.7: Utilização do sistema Spark executando o Wordcount.

As figuras 3.6-Hadoop e 3.7-Spark mostram o consumo dos recursos ao executar a carga de trabalho Wordcount. Como ela extrai uma pequena quantidade de dados de um grande conjunto de dados, o shuffle e a saída dos dados são menores do que a entrada. Como consequência, ao ser executado no Hadoop consome aproximadamente 50% da memória principal e mais de 90% do buffer cache, enquanto que o Spark consome aproximadamente 25% da memória e quase 100% do buffer-cache. o Hadoop utiliza até aproximadamente 50% do tempo da CPU nas tarefas do utilizador (no início do job e da fase reduz), o mesmo valor é utilizado no Spark do início do job até quase o final do Job onde a percentagem é reduzida, e 6% na espera E/S para Hadoop e cerca de 25% para o Spark. O Hadoop lê cerca de 246.04 MiB/s e escreve cerca de 143.05 MiB do início do job até um pouco mais da fase reduz. Estes valores são mais baixos no final do job, sendo realizadas aproximadamente 880 IO operações de leitura e 316 IO operações de escrita no disco. Já no Spark são cerca de 228.88 MiB lidos e 25.94 MiB escritos, mais de 1500IO operações de leitura. Enquanto isso, em termos de rede no Hadoop, um máximo de até 13220 pacotes são enviados e recebidos, 83 pacotes com erros, cerca de 10159 pacotes enviados e recebidos, não se nota pacotes com erros no Spark.

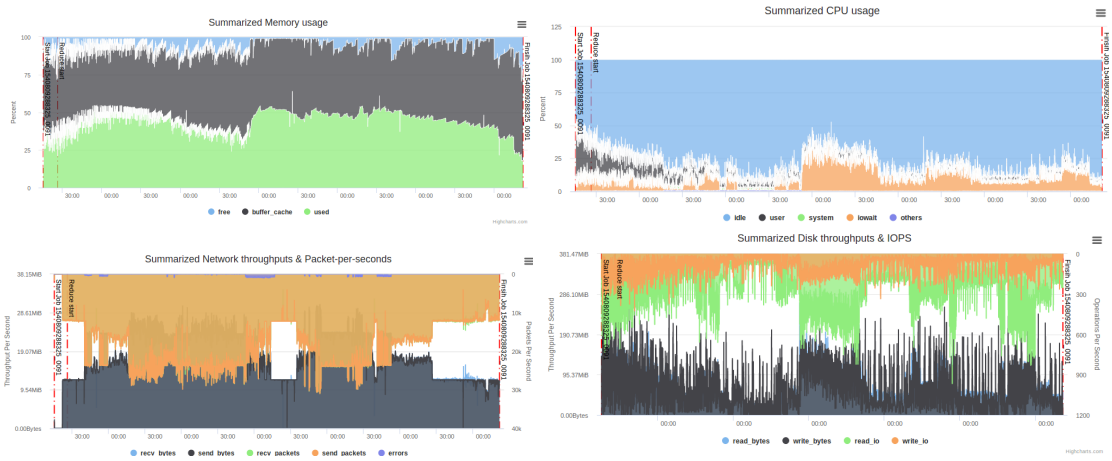

Figura 3.8: Utilização do sistema Hadoop executando o Terasort.

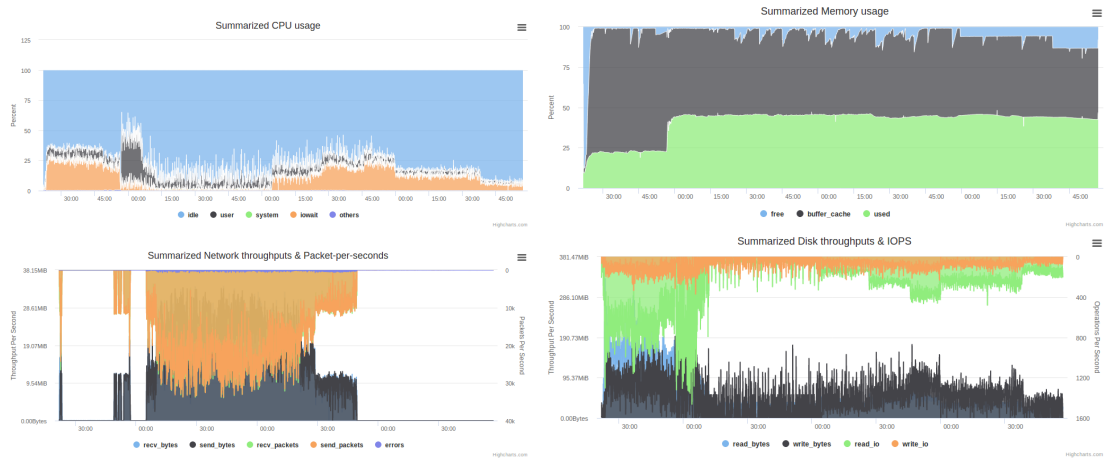

Figura 3.9: Utilização do sistema Spark executando o TeraSort.

As figuras 3.8-Hadoop e 3.9-Spark apresentam o consumo dos recursos ao executar a carga de trabalho TeraSort (semelhante a sort ou à sua melhoria), tem processo limitado pela CPU du-

rante a fase Map e no disco E/S durante a fase Reduce. No entanto, tanto o Hadoop como o Spark consomem aproximadamente 50% da CPU nas tarefas do utilizador e na espera de E/S, mais de 50% da memória principal e aproximadamente 100% do buffer cache é utilizado no Hadoop enquanto que o Spark utiliza aproximadamente 50% da memória principal e próximo de 100% o buffer cache. Aproximadamente 206.10 MiB são lidos e escritos, e mais de 900IO operações de escrita e 300IO operações de leituras realizadas no Hadoop, enquanto que no Spark a maior utilização do disco E/S acontece na operação de leitura (1200IO) e aproximadamente 150.73MiB escritos. Tanto para o Hadoop como para o Spark, o maior consumo da rede acontece na fase intermediária.

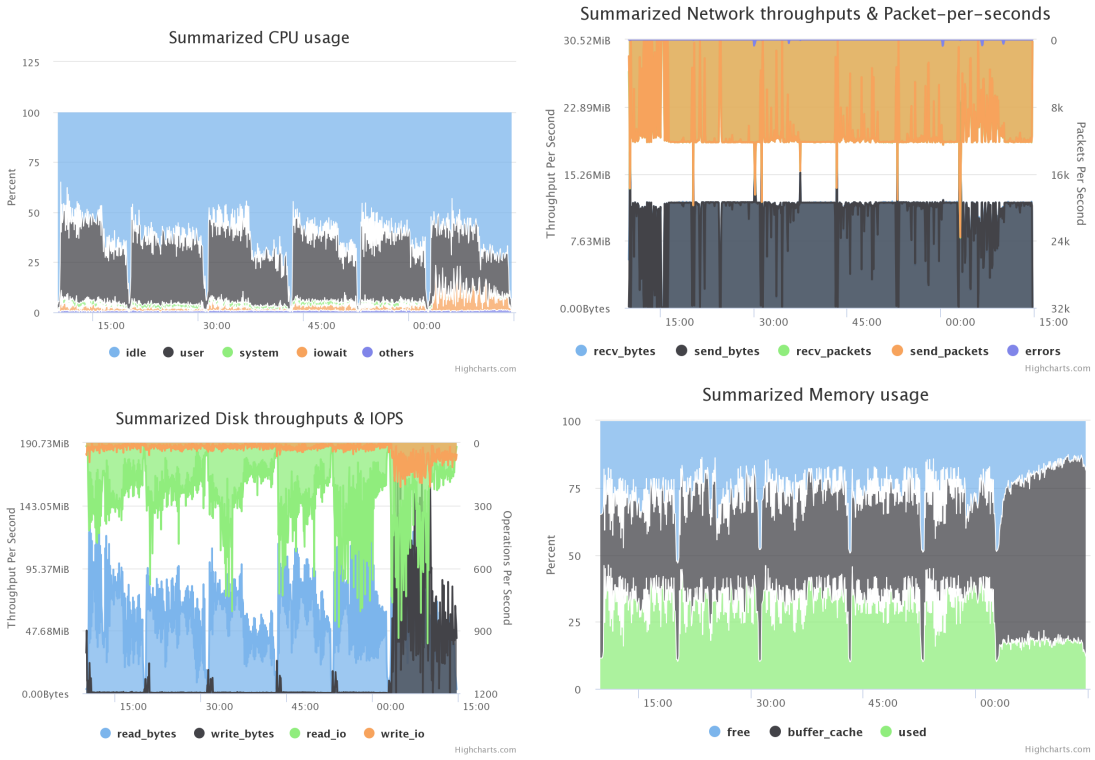

Figura 3.10: Utilização do sistema Hadoop executando o K-means.

A carga de trabalho K-means clustering consome a maior parte do tempo em iterações no cálculo do centroide do cluster. O seu processo é limitado pela CPU durante a iteração e ligado à E/S do disco durante o agrupamento. Nas figuras 3.10-Hadoop e 3.11-Spark pode-se observar que tanto o Spark como o Hadoop consomem aproximadamente 50% da CPU nas tarefas do utilizador, mas a espera E/S é maior no Spark (próximo de 25%). O Hadoop realiza mais de 900IO operações de leitura em comparação ao Spark que atinge 750IO, ainda 143.05MiB lidos e escritos no Hadoop contra aproximadamente 228.88 MiB lidos no Spark. Aproximadamente 50% da memória principal é usada tanto no Hadoop como no Spark. No Hadoop o buffer-cache é usado a cerca de 75% contra aproximadamente 100% no Spark. Para o Spark há muita oscilação no consumo da rede mas pode enviar e receber pacotes maiores e apresenta um número muito reduzido de erros em relação ao Hadoop.

Portanto, o Spark ao executar o PageRank utiliza aproximadamente 90% do CPU nas tarefas do utilizador, enquanto que o Hadoop utiliza cerca de 50%. As tarefas do sistema e a espera E/S têm utilização baixa para ambas. São mais ou menos 90IO operações de escrita e leitura no

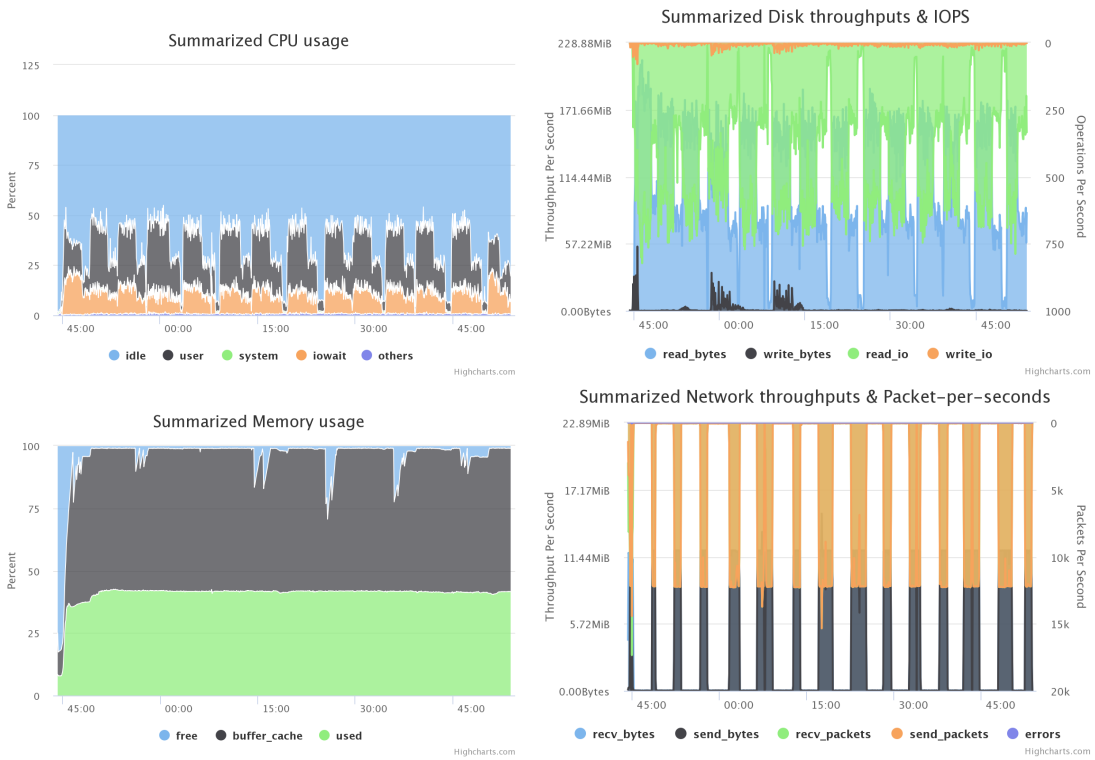

Figura 3.11: Utilização do sistema Spark executando o K-means.

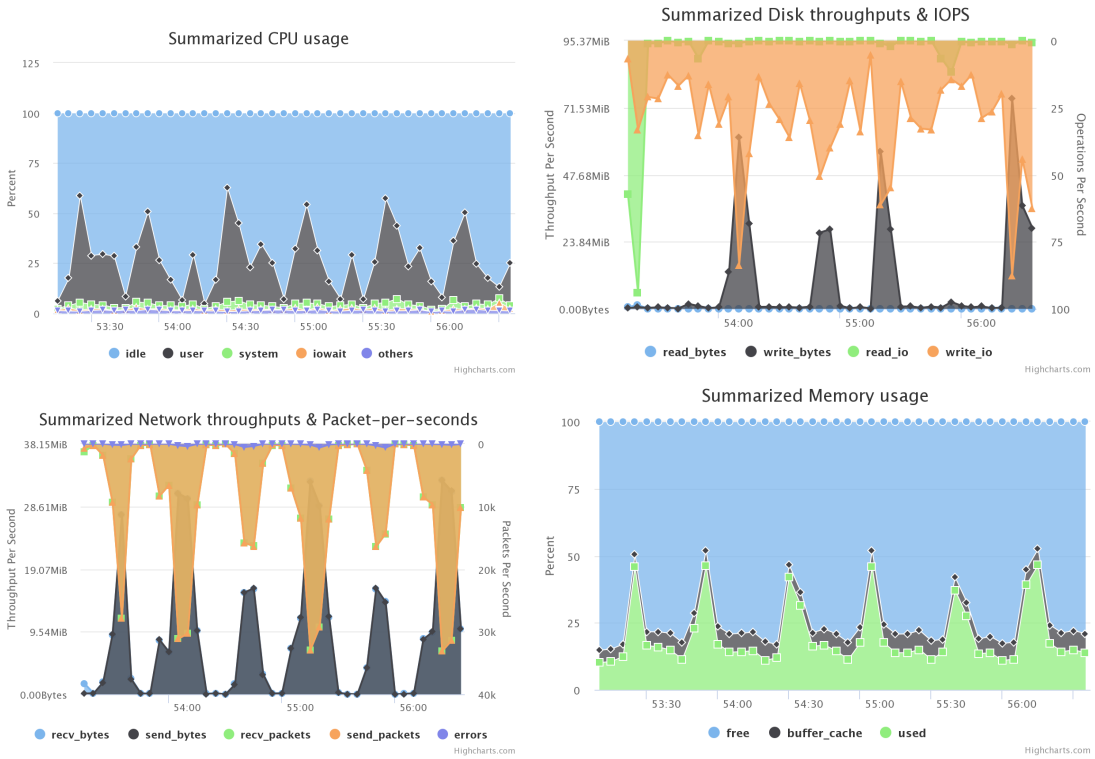

Figura 3.12: Utilização do sistema Hadoop executando o PageRank.

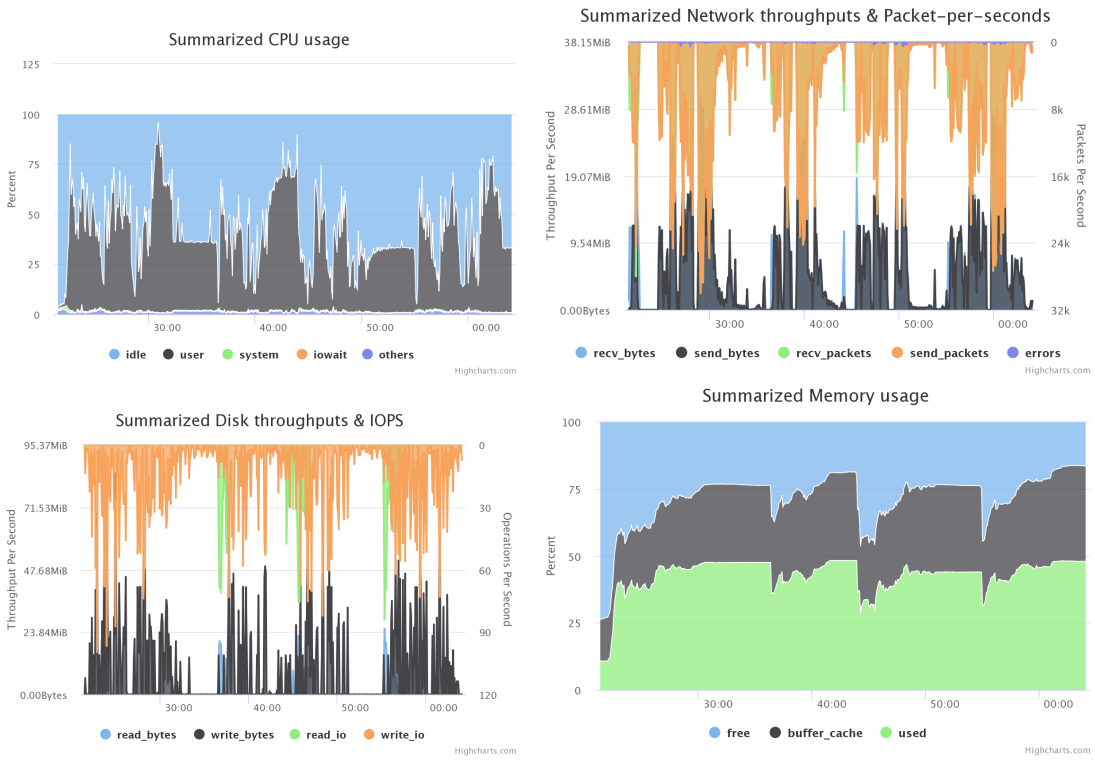

Figura 3.13: Utilização do sistema Spark executando PageRank.

Hadoop e no Spark e mais de 71.53 MiB escritos e lidos no Hadoop contra cerca de 47.68 MiB escritos e lidos no Spark. O Hadoop e o Spark utilizam aproximadamente 50% da memória principal, fazendo mais uso do buffer cache o Spark com mais 75% contra 50% do Hadoop. Portanto, tanto para Hadoop como para o Spark, em todas as cargas de trabalho, a maior parte da utilização da memória acontece no buffer cache para não sobrecarregar o disco, e a CPU tem maior envolvimento com as tarefas do utilizador, mas, de um modo geral, o Hadoop utiliza mais a rede. Para o Wordcount e K-means, o Spark faz maior utilização da E/S e do disco devido ao tamanho dos dados de entrada quando a memória não é suficiente.

3.4.2 Comparação de Desempenho Entre o Hadoop e o Spark

Nesta subsecção é apresentada a análise dos resultados obtidos da avaliação das plataformas Hadoop e Spark. Para melhor comparação e para obter resultados precisos, usamos os mesmos tamanhos de dados de entrada para as duas plataformas. A tabela 3.4 abaixo mostra os resultados da execução de todas as cargas de trabalho para o Apache Spark e o Apache Hadoop, incluindo tamanho de dados de entrada, tempos de execução, throughput (byte/s) e a taxa de transferência por nó. Os dados apresentados na tabela 3.4 representam os resultados das várias experiências realizadas, com vários tipos de dados e vários tamanhos de dados de entrada.

Foi avaliado o desempenho do Hadoop e do Spark utilizando as métricas: tempos de execução e o throughput para as cargas de trabalho já referidas. Foram utilizadas três iterações para o Pagerank e cinco iterações para o K-means respetivamente. Os tamanhos de entrada dos dados de todas as cargas de trabalho bem como o resultado das suas execuções em termos de tempo e bytes carregados são apresentados na tabela 3.4. Para todos os gráficos apresentados a seguir: nos gráficos à esquerda, o eixo vertical apresenta o tempo de execução (em segundo) das plataformas para as cargas de trabalhos, o eixo horizontal são apresentadas todas as cargas

Tabela 3.4: Resultado da execução de todas as cargas de trabalho para o Hadoop e para o Spark.

Plataformas	Cargas de Trabalho	Tamanho de Dados de Entrada			Tempo de Execução (s)			*Troughput (byte/s)*			*Troughput/nó*		
		small	*large*	*gigantic*	*small*	*large*	*gigantic*	*small*	*large*	*gigantic*	*small*	*large*	*gigantic*
Hadoop	Sort	3.3MB	328.49MB	32.85GB	17.323	26.084	1495.167	189835	12593846	21970227	63278	4197948	7323409
	Terasort	320MB	3.2GB	320GB	26.674	134.233	21982.894	11996700	23839145	14556773	3998900	7946381	4852257
	Worcount	328.5MB	3.285GB	328.45GB	23.422	102.303	11941.524	14025050	32109630	27508333	4675016	10703210	9169444
	Kmeans	602.46MB	4GB	40.2GB	194.062	504.693	3894.154	3104485	7958049	10313830	1034828	2652683	3437943
	PageRank	5000 pages	500000pages		110.511	207.658		16389	1251807		5463	417269	
Spark	Sort	3.3MB	328.49MB	32.85GB	60.646	68.134	840.163	54224	4821350	39098555	18074	1607116	13032851
	Terasort	320MB	3.2GB	320GB	86.423	188.236	12915.405	3702718	16999936	24776613	1234239	5666645	8258871
	Worcount	328.5MB	3.285GB	328.45GB	69.117	105.759	1777.753	4752734	31060349	184779002	1584244	10353449	61593000
	Kmeans	602.46MB	4GB	40.2GB	82.386	129.611	4183.474	7312681	30987892	9600548	2437560	10329297	3200182
	PageRank	5000pages	500000pages		45.252	840.163		40024	2391776		13341	797258	

de trabalhos. No gráfico à direita o eixo vertical apresenta o throughput e o eixo horizontal as cargas de trabalho. A cor azul representa o cluster do Hadoop e a cor cinzenta o cluster do Spark executando as cargas de trabalho com escala de dados: *small, large e gigantic*.

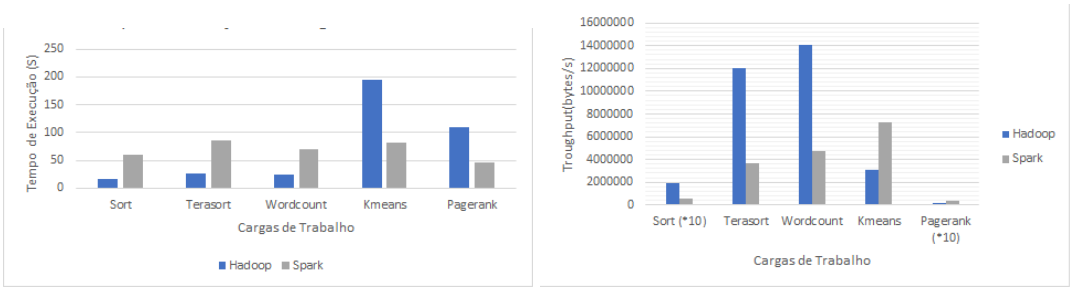

Figura 3.14: Tempo de Execução e Taxa de Transferência das plataformas para a escala de dados *small*.

A figura 3.14 acima representa o tempo de execução e a taxa de transferência das plataformas Hadoop e Spark as cargas de trabalho executadas com tamanhos de dados *small*. Como podemos observar nos gráficos da figura 3.14 o Hadoop apresentou melhor desempenho e maior rendimento ao executar as cargas de trabalhos sort, terasort e wordcount, pois levou menos tempo do que o Spark. Enquanto que o Spark teve melhor desempenho maior rendimento ao executar o pagerank e o k-means pois fez menos tempo de execução em relação ao Hadoop. O Hadoop mostrou ser entre 3,5x a 2x mais rápido do que o Spark ao executar o sort, o terasort e o wordcount em contrastes com o Spark que para o k-means e o pagerank foi 2,4x mais rápido.

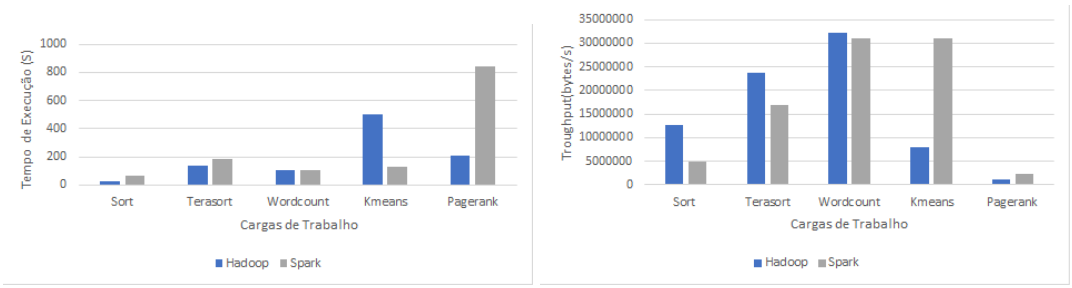

Figura 3.15: Tempo de Execução e Taxa de Transferência das plataformas para a escala de dados *large*.

A figura 3.15 acima representa o tempo de execução e o throughput das plataformas Hadoop e Spark para as cargas de trabalho executadas com a escala de dados *large*. Como podemos observar nos gráficos da figura 3.15, o Hadoop apresentou melhor desempenho ao executar as cargas de trabalhos sort, terasort, wordcount (a diferença do tempo de execução aqui foi pequena) e pagerank, pois levou menos tempo do que o Spark. O Hadoop apresentou desemepenho inferior ao executar o pagerank e, em contrapartida, o Spark apresentou melhor desempenho

para o pagerank e melhor desempenho ao executar o k-means pois apresentou menor tempo de execução. O Hadoop mostrou ser entre 4x a 1x mais rápido do que o Spark ao executar o sort, terasort, wordcount e o pagerank, em contraste com o k-means, o Spark foi 3,9x mais rápido do que o Hadoop.

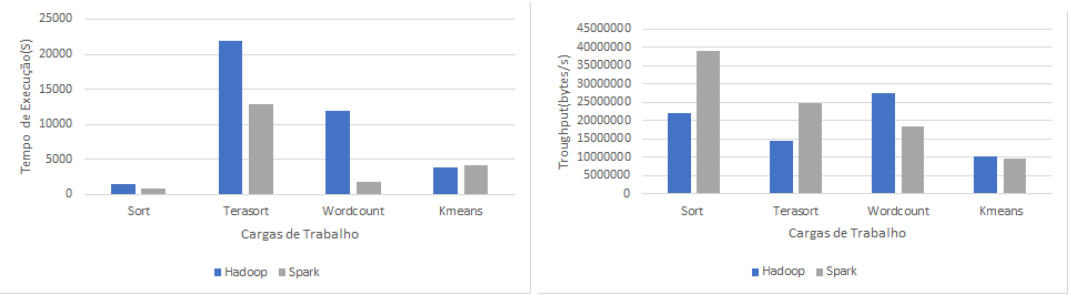

Figura 3.16: Tempo de Execução e Taxa de Transferência das plataformas para a escala de dados *gigantic*.

A figura 3.16 acima representa o tempo de execução e o throughput das plataformas Hadoop e Spark para as cargas de trabalho executadas com a escala de dados *gigantic*. Como podemos observar nos gráficos da figura 3.16, os resultados são completamente contrários ao que acontece com os dados *small e large*. O Spark apresentou melhor desempenho ao executar as cargas de trabalhos sort, terasort, wordcount, mas apresentou um desempenho inferior ao executar o wordcount e o k-means, enquanto que o Hadoop teve melhor desempenho ao executar o k-means (apesar de a diferença ser muito pequena), teve menor tempo de execução e apresentou melhor desempenho no wordcoun e k-means. O Spark carregou menos bytes ao executar o wordcount e o kmeans mas mostrou ser entre 6x a 1x mais rápido do que o Hadoop ao executar o sort, terasort e o wordcount, em contraste, para o k-means, o Hadoop teve quase o mesmo desempenho que o Spark. Nesta experiência não foi possível executar o pagerank com este tamanho de dados, pois o Spark reclamou memória insuficiente.

Para as cargas de trabalho wordcount, sort e terasort, o Hadoop apresentou melhor desempenho em relação ao Spark com as escalas de dados *small e large*, mas quando mudamos para a escala gigantic o seu desempenho baixou, enquanto que para o K-means e o pagerank o Hadoop apresentou melhor desempenho com tamanho dos dados maior e o Spark o seu desempenho melhorou com dados menores. Portanto, observou-se que o desempenho das duas plataformas é relativo dependendo de vários fatores tais como: o tamanho da memória, parâmetros de configuração, tipo de aplicação e o tamanhos dos dados. Podemos dizer que embora o Spark seja em geral mais rápido que o Hadoop em operações iterativas, precisa de consumir mais memória. Além disso, para essas cargas de trabalho iterativas como k-means e pagerank, o desempenho do Spark enfraqueceu à medida que o tamanho dos dados de entrada ia aumentando e no momento em que a memória não era suficiente para armazenar resultados intermediários recém-criados e criar novos RDDs necessitando de substituição no disco. O pagerank e o K-means foram executados com iterações que requerem muito uso de memória. Por outro lado, nesta experiência para as cargas de trabalho sort, worcount e terasort com a escala de dados *small e large* o Spark apresentou fraco desempenho devido ao tipo de associação sortMerge que o spark utiliza por definição pois é mais adequado para conjunto de dados grandes. Mantivemos o sortMerge pois precisávamos também de trabalhar com dados grandes.

3.4.3 Comparação de Desempenho Entre o Hadoop, o Spark e o Flink Para o Wordcount

Nesta experiência comparamos as três plataformas Hadoop, Spark e Flink executando o programa wordcount dos seus respetivos ficheiros de exemplos. Os dados de entrada foram gerados pelo script de preparação da carga de trabalho wordcount do Hibench-master. O Spark e o Flink nesta fase executaram a carga de trabalho no modo autónomo, de forma distribuída, utilizaram o sistema de ficheiros HDFS. Os tamanhos dos dados de entrada e os resultados do de execução (em segundo) das três plataformas executando o wordcount são apresentados na tabela 3.5 abaixo.

Tabela 3.5: Resultado da execução das plataformas Hadoop, Spark e Flink para o Wordcount.

Tamanho dos Dados	Tempo de Execução (s)		
	Hadoop	Spark	Flink
2 MB	12	1	1
392MB	37	36	22
8GB	318	720	741
38GB	1670	3585	1957

Wordcount é uma boa ferramenta para avaliar o componente de agregação em cada plataforma, já que tanto o Spark quanto o Flink usam um combinador do lado do mapa para reduzir os dados intermediários.

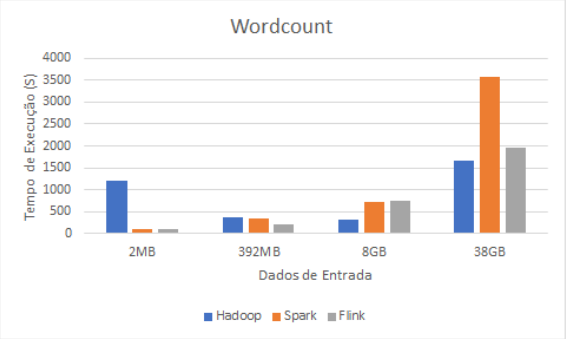

Figura 3.17: Tempo de execução das plataformas Hadoop, Spark e Flink para o Wordcount.

A figura 3.17 mostra uma comparação do desempenho do Hadoop, Spark e Flink executando a carga de trabalho Wordcount obtida dos seus ficheiros de exemplos, com diferentes tamanhos de entrada dos dados de entrada que vão de 2MB a 38GB e as diferentes cores nas barras do gráfico apresentam as plataformas (Hadoop, Spark e Flink). Os dado utilizados no gráfico estão apresentados na tabela 3.5. Ao representarmos os tamanhos de dados de 2MB no gráfico tivemos de multiplicar por 100 para permitir a visualização e por 10 o tamanho de dados de 392MB. Podemos observar no gráfico acima que o Spark e o Flink tiveram melhor desempenho em relação ao Hadoop quando executaram o wordcount com um tamanho de 2MB e 392MB. O Flink teve melhor desempenho sendo 1,7x mais rápido que o Hadoop e o Spark. A medida que o tamanho dos dados aumentava de 8GB a 38GB, o Flink e o Spark baixaram o desempenho enquanto que o Hadoop melhorou o desempenho sendo mais rápido do que o Spark e o Flink. O Flink mostrou-se mais rápido que o Spark na execução dos dados com tamanho de 38GB. O Flink processa os dados de uma vez ao contrário do Spark que processa os dados em micro-batch. Alguns nós no

cluster são muito mais lentos que outros influenciando assim o desempenhos das plataformas. Ao longo das experiências, os nós no cluster do Flink eram perdidos (tendo em conta que o Flink não assume máquinas uniformes no cluster) e tendo sido executado o wordcount na maior parte das vezes com menos uma máquina, isto, de certa forma influenciou no desempenho do Flink.

Tabela 3.6: Resultado da execução após ajustar o Flink.

Tamanho de Dados	Tempo de Execução(s)		
	Hadoop	Spark	Flink
2 MB	12	1	0,698
392MB	37	36	22
8GB	318	720	316
38GB	1670	3585	1497

Alteramos o valor padrão do parâmetro taskmanager.memory.fraction para 0,8 (está relacionado com a gestão da memória, é a quantidade de memória reservada pelo taskmanager para armazenamento em cache e classificação de tabelas hash) que têm maior influência no desempenho do Flink. Para o Spark não foi necessário alterar o valor deste parâmetro, pois apesar de ser relevante é recomendado manter o valor padrão 0,6 (esses valores são aplicados à maioria das cargas de trabalho). Após o ajuste desse parâmetro, observamos que houve melhoria no desempenho do Flink. A tabela 3.6 apresenta o tempo de execução do Flink ao executar o Wordcount após o ajuste do paramêtro

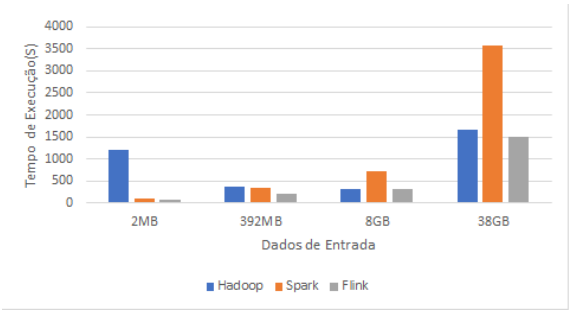

Figura 3.18: Tempo de execução das plataformas Hadoop, Spark e Flink para o wordcount.

A figura 3.18 apresenta a comparação das plataformas Hadoop, Spark e Flink após o ajuste do Flink. Podemos observar que o Flink melhorou o seu desempenho e mostrou quase o mesmo desempenho que o Hadoop (para todos tamanhos dos dados exceto para 2MB onde se mostrou 17x mais rápido) e mais rápido do que o Spark. O Spark para tamanhos de entrada de 2MB e de 392MB mostrou melhor desempenho em relação ao Hadoop, mas, à medida em que os tamanhos iam aumentando, o seu desempenho degradava-se. Aumentar o tamanho da fração de memória para o Flink melhorou significativamente o seu desempenho sobretudo para os dados de 8GB e 38GB.

Capítulo 4

Conclusões

4.1 Principais Conclusões

As plataformas Hadoop, Spark e Flink são projetos de código aberto da Apache Software Foundation e são as mais utilizadas na análise de grandes conjuntos de dados. A principal diferença que conduz a diferentes desempenhos entre as plataformas Apache Hadoop, Apache Spark e Apache Flink está no processamento: o Spark faz o processamento dos dados na memória, já o Hadoop MapReduce lê e escreve os dados no disco. Isto influencia significativamente a velocidade de processamento dado que o Flink fornece um único tempo de execução para o fluxo e o processamento em batch. O Hadoop MapReduce é capaz de processar conjuntos de dados maiores do que Spark, mas o Flink processa mais rapidamente que o Spark devido à sua arquitetura de streaming. Neste trabalho foi avaliado o desempenho das plataformas Hadoop e Spark executando as cargas de trabalho sort, terasort, wordcount, pagerank e k-means dos benchmarks: micro, Machine Learning e Websearch num cluster homogéneo com quatro máquinas físicas compondo três nós escravos e um mestre. Foi analisado a utilização dos recursos computacionais das três plataformas tendo-se observado que o Spark na maior parte dos casos fez mais uso dos recursos de CPU, disco e memória. Por outro lado, o desempenho também foi analisado chegando-se à conclusão que para as cargas de trabalho wordcount, sort e terasort, o Hadoop apresentou melhor desempenho em relação ao Spark com a escala de dados *small e large*, mas quando mudamos para dados gigantic o seu desempenho baixou. Enquanto que para o K-means e para o pagerank o Hadoop apresentou melhor desempenho com tamanho de dados de entrada maior ou a escala *gigantic* e o Spark o seu desempenho melhorou com os dados da escala *small e large*. Os resultados obtidos mostram que o desempenhos das duas plataformas nesta experiência é relativo dependendo da carga de trabalho, do tamanho dos dados de entrada, tamanho da memória e da velocidades de processamento das máquinas. Foram também comparadas as plataformas Spark e Flink executando o programa Wordcount dos seus ficheiros de exemplos, tendo-se observado que o Flink apresentou melhor desempenho que o Hadoop para todos os tipos de dados de entrada, sendo 2x mais rápido que o Spark. O Spark apresentou melhor desempenho que o Hadoop para tamanhos de dados de entrada de 2MB e 392MB, mas observou-se que o seu desempenho degradava-se com o aumento do tamanho dos dados de entrada. O desempenho do Flink melhorou significativamente, sobretudo para tamanhos de dados de entrada de 8GB e 38GB, após ser ajustado o valor do parâmetro de fração da memória.

4.2 Direções para Trabalho Futuro

Como trabalho futuro sugerimos aumentar a escala do cluster utilizando o testb Grid‘5000 com maiores recursos computacionais e avaliar o desempenho do Apache Hadoop, Apache Spark com os outros benchmarks contidos no Hibench-master 7 que não foram executados. Pretendemos avaliar a plataforma Apache Flink executando os benchmarks do Hibench-master 7 diretamente e compará-la com a plataforma Apache Spark já que ambas têm quase o mesmo foco.

Referências

[1] C. Snijders, U. Matzat e U.-D. Reips, «" Big Data": big gaps of knowledge in the field of internet science,» *International Journal of Internet Science*, vol. 7, n.º 1, pp. 1–5, 2012.

[2] J. Dean e S. Ghemawat, «MapReduce: simplified data processing on large clusters,» *Communications of the ACM*, vol. 51, n.º 1, pp. 107–113, 2008.

[3] P. A. R. S. da Costa, «Dependable MapReduce in a Cloud-of-Clouds,» pp. 1–123, 2017.

[4] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[5] A. S. F. Apache Hadoop. (2018). Apache Hadoop Releases, YARN, HDFS. accessed 12-02-2018, URL: `https://hadoop.apache.org/`.

[6] A. Spark. (2018). Apache Spark overview. accessed 18-03-2018, URL: `https://spark.apache.org/`.

[7] S. Data Flair. (2016). Hadoop vs Spark vs Flink – Big Data Frameworks Comparison. accessed 18-06-2018, URL: `https://data-flair.training/blogs/hadoop-vs-spark-vs-flink-comparison/`.

[8] A. Flink. (2017). Apache Flink overview. accessed 02-08-2018, URL: `https://flink.apache.org/`.

[9] ——, (2018). Apache Flink. accessed 06-08-2018, URL: `https://flink.apache.org`.

[10] s.-s. carsonwang heyu1 sophia-sun heyu1. (2017). Intel-hadoop/HiBench. accessed 25-10-2018, URL: `https://github.com/intel-hadoop/HiBench`.

[11] I. Mavridis e H. Karatza, «Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark,» *Journal of Systems and Software*, vol. 125, pp. 133–151, 2017.

[12] Y. Samadi, M. Zbakh e C. Tadonki, «Performance comparison between Hadoop and Spark frameworks using HiBench benchmarks,» *Concurrency and Computation: Practice and Experience*, vol. 30, n.º 12, e4367, 2018.

[13] D. Garcı□a-Gil, S. Ramı□rez-Gallego, S. Garcı□a e F. Herrera, «A comparison on scalability for batch big data processing on Apache Spark and Apache Flink,» *Big Data Analytics*, vol. 2, n.º 1, p. 1, 2017.

[14] J. Veiga, R. R. Expósito, X. C. Pardo, G. L. Taboada e J. Tourifio, «Performance evaluation of big data frameworks for large-scale data analytics,» em *Big Data (Big Data), 2016 IEEE International Conference on*, IEEE, 2016, pp. 424–431.

[15] A. Kafka. (2018). Apache Kafka® is a distributed streaming platform. accessed 25-10-2018, URL: `https://kafka.apache.org/`.

[16] R. C. de Mattos e H. Senger, «Serviço Open Source de Bigdata para Openstack,» *Revista TIS*, vol. 4, n.º 2, 2016.

[17] C. Uzunkaya, T. Ensari e Y. Kavurucu, «Hadoop ecosystem and its analysis on tweets,» *Procedia-Social and Behavioral Sciences*, vol. 195, pp. 1890–1897, 2015.

[18] S. Ghemawat, H. Gobioff e S.-T. Leung, *The Google file system*, 5. ACM, 2003, vol. 37.

[19] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2015.

[20] A. Goldman, F. Kon, F. P. Júnior, I. Polato e R. de Fátima Pereira, «Apache Hadoop: conceitos Teóricos e Práticos, Evolução e novas possibilidades,» *XXXI Jornadas de atualizações em informática*, pp. 88–136, 2012.

[21] D. DeRoos, P. Zikopoulos, B. Brown, R. Coss e R. B. Melnyk, *Hadoop for dummies*. John Wiley & Sons, Incorporated, 2014.

[22] L. Marco Garcia. (2015). Visão Geral do Hadoop e Ecossistesma. 20-03-2018, URL: `https://pt.linkedin.com/pulse/o-que-%C3%A9-hadoop-marco-garcia`.

[23] D. D. Gutierrez. (2014). Yarn all Rage Hadoop Summit. accessed 07-06-2018, URL: `https://www.kdnuggets.com/2014/06/yarn-all-rage-hadoop-summit.html`.

[24] R. POSA. (2018). Differences Between Hadoop1 and Hadoop2. accessed 07-06-2018, URL: `https://www.journaldev.com/8806/differences-between-hadoop1-and-hadoop2`.

[25] J. J. Hanson. (2011). An introduction to the Hadoop Distributed File System. accessed 07-06-2018, URL: `https://www.ibm.com/developerworks/library/wa-introhdfs/index.html`.

[26] T. Jie, G. Junlei e W. Gangshan, «Improving Scheduling Efficiency of Hadoop YARN Using AFSA Algorithm,» em *Ubiquitous Computing and Communications (ISPA/IUCC), 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on*, IEEE, 2017, pp. 919–924.

[27] M. Zaharia, P. W. A. Konwinski e H. Karau, *Learning Spark*. O'Reilly Media, Inc., 2015.

[28] A. Verma, A. H. Mansuri e N. Jain, «Big data management processing with Hadoop MapReduce and spark technology: A comparison,» em *Colossal Data Analysis and Networking (CDAN), Symposium on*, IEEE, 2016, pp. 1–4.

[29] J. Aven, *Apache Spark in 24 Hours, Sams Teach Yourself*. Sams Publishing, 2016.

[30] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker e I. Stoica, «Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,» em *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012, pp. 2–2.

[31] A. Spark. (2018). Apache Spark History. accessed 02-04-2018, URL: `http://spark.apache.org/history.html`.

[32] A. Sarkar, *Learning Spark SQL: Architect streaming analytics and machine learning solutions*. Packt Publishing, 2017.

[33] Databricks. (2018). Apache Spark. accessed 17-04-2018, URL: `https://databricks.com/spark/about`.

[34] Sally. (2015). A Apache Software Foundation anuncia o Apache ™ Flink ™ como um projeto de nível superior. accessed 03-08-2018, URL: `https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces69`.

[35] A. Flink. (2017). Apache Flink Introduction. accessed 03-08-2018, URL: `https://flink.apache.org/introduction.html`.

[36] ——, (2016). Announcing Apache Flink 1.0.0. accessed 08-08-2018, URL: `https://flink.apache.org/news/2016/03/08/release-1.0.0.html`.

[37] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi e K. Tzoumas, «Apache flink: Stream and batch processing in a single engine,» *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, n.º 4, 2015.

[38] F. Hueske e V. Kalavri, *Streaming Processing with Apache Flink*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2017.

[39] IDGNS. (2015). Cinco pontos de comparação entre Hadoop e Spark. accessed 15-12-2015, URL: https://www.computerworld.com.pt/2015/12/15/cinco-pontos-de-comparacao-entre-hadoop-e-spark/.

[40] I. Ian Pointer. (2015). Apache Flink: New Hadoop contender squares off against Spark. accessed 07-08-2018, URL: https://www.infoworld.com/article/2919602/hadoop/flink-hadoops-new-contender-for-mapreduce-spark.html.

[41] G. Gupta. (2017). Hadoop MapReduce vs Apache Spark. accessed 12-06-2018, URL: https://dzone.com/articles/apache-hadoop-vs-apache-spark.

[42] S. Han, W. Choi, R. Muwafiq e Y. Nah, «Impact of Memory Size on Bigdata Processing based on Hadoop and Spark,» em *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, ACM, 2017, pp. 275–280.

[43] O.-C. Marcu, A. Costan, G. Antoniu e M. S. Pérez-Hernández, «Spark versus flink: Understanding performance in big data analytics frameworks,» em *Cluster Computing (CLUSTER), 2016 IEEE International Conference on*, IEEE, 2016, pp. 433–442.

[44] M. Webster. (2018). Benchmark (computação). accessed 13-08-2018, URL: https://www.merriam-webster.com/Benchmark.

[45] R. Han, L. K. John e J. Zhan, «Benchmarking big data systems: A review,» *IEEE Transactions on Services Computing*, vol. 11, n.º 3, 2018.

[46] Y. Samadi, M. Zbakh e C. Tadonki, «Comparative study between Hadoop and Spark based on Hibench benchmarks,» em *Cloud Computing Technologies and Applications (CloudTech), 2016 2nd International Conference on*, IEEE, 2016, pp. 267–275.

[47] S. Huang, J. Huang, J. Dai, T. Xie e B. Huang, «The HiBench benchmark suite: Characterization of the MapReduce-based data analysis,» em *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, IEEE, 2010, pp. 41–51.

[48] P. Castagna, «Having fun with PageRank and MapReduce,» *Hadoop User Group UK talk. Available: http://static. last. fm/johan/huguk-20090414/paolo_castagna-pagerank. pdf*, 2009.

[49] u. Ivsoft. (2015). Intel-hadoop/HiBench. accessed 25-10-2018, URL: https://github.com/IMCG/HiBench/blob/master/conf/10-data-scale-profile.conf.

Apêndice A

Anexos

A.1 Pré-requisitos de Instalação dos 3 Frameworks

O software Java JDK é o pré-requisito a ser instalado antes do Hadoop, Spark e Flink que pode ser baixado do site da Oracle

1. Instalar os comandos do Java JDK:

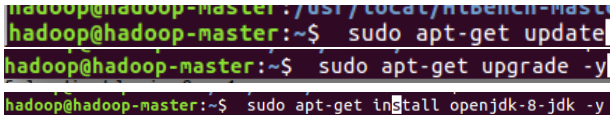

Figura A.1: Instalação do Java

2. Configurar variáveis de ambiente para java:
3. Para configurar nomes de host e endereços IP no arquivo /etc/hosts foram adicionados os nomes dos hosts e seus respetivos IP:

A configuração do cluster distribuído do Hadoop, Spark e Flink requer que o nó master acesse e comunica com os nós slaves sem exigir qualquer senha. Portanto foi instalado SSH na porta 22 em todos os nós do cluster, para gerar a chave SSH sem frase secreta. Para que o nó master acedesse os nós slaves sem senha, foi necessário copiar a chave pública do master para o ficheiro authorized_keys em todos os slaves:

1. Instalar o SSH

A.2 Instalação e configuração do Hadoop

1. *Download* do arquivo de pacotes do Hadoop.
2. Configuração das variáveis de ambiente para o Hadoop:
3. Configuração dos ficheiros XML que se encontram na diretoria $HADOOP_HOME/etc/hadoop/conf: nomeadamende core-site.xml, hdfs-site.xml, mapred-site.xml e yarn-site.xml e o hadoop-env.sh, slaves e master.

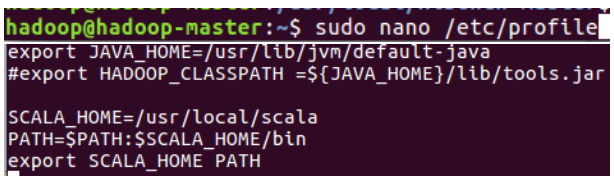

Figura A.2: Configuração da variável de ambiente para java

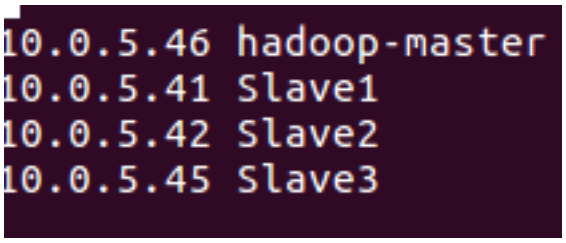

Figura A.3: Configuração dos Hosts

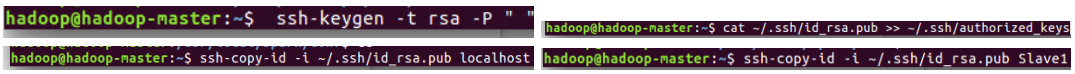

Figura A.4: Configuração do protocolo SSH

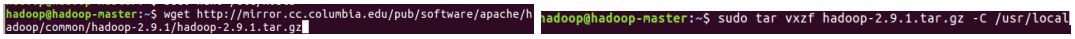

Figura A.5: *Download Hadoop*.

Configuração das propriedades do ficheiro Core-site.xml Configuração das propriedades do ficheiro hdfs-site.xml Configuração das propriedades do ficheiroo Mapred-site.xml Configuração das propriedades do ficheiro YARN-site.xml

4. Iniciar o cluster

Este processo é feito em todos os nós do cluster.

A.3 Instalação e Configuração do Spark

1. *Download* do ficheiro de pacotes do Spark.
2. Configuração dos ficheiros que se encontram na diretoria $Spark_HOME/conf: nomeadamende, slaves e master, spark . Configuração das propriedades do arquivo spark-conf

3. Iniciar o cluster do Spark

A.4 Instalação e configuração do Flink

1. *Download Flink*
2. Configuração do Ficheiro Flink-Conf.yaml

```
hadoop@hadoop-master:~$ sudo nano /home/hadoop/.bashrc
```

Figura A.6: Variáveis de Ambiente para Hadoop

```
<configuration>
<property>
 <name>fs.default.name</name>
 <value>hdfs://hadoop-master:9000/</value>
</property>
</configuration>
```

Figura A.7: Core site

```
<configuration>
<property>
 <name>dfs.replication</name>
 <value>1</value>
 <description>o valor por defeito é 3 o numero de replica do hdfs</description>
</property>
<property>
 <name>dfs.namenode.name.dir</name>
 <value>file:/home/hadoop/mystorage/hdfs/namenode</value>
</property>
<property>
 <name>dfs.datanode.data.dir</name>
 <value>file:/home/hadoop/mystorage/hdfs/datanode</value>
</property>
<property>
 <name>dfs.http.address</name>
 <value>hadoop-master:50070</value>
 <description>Primary NameNode hostname for http access.</description>
</property>
<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>
</configuration>
```

Figura A.8: HDFS site

```
<configuration>
<property>
 <name>mapreduce.framework.name</name>
 <value>yarn</value>
</property>
<property>
        <name>yarn.app.mapreduce.am.resource.mb</name>
        <value>4096</value>
</property>

<property>
        <name>mapreduce.map.memory.mb</name>
        <value>4096</value>
</property>

<property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>4096</value>
</property>
<property>
        <name>mapreduce.map.cpu.vcores</name>
        <value>1</value>
</property>
<property>
        <name>mapreduce.reduce.cpu.vcores</name>
        <value>1</value>
</property>
<property>
        <name>mapreduce.reduce.java.opts</name>
        <value>-Xmx3276m</value>
</property>
<property>
        <name>mapreduce.map.java.opts</name>
        <value>-Xmx3276m</value>
</property>
<property>
        <name>mapreduce.task.io.sort.mb</name>
        <value>1638</value>
</property>
```

```
<property>
    <name>mapreduce.jobhistory.address</name>
    <value>${yarn.resourcemanager.hostname}:10020</value>
  </property>

  <property>
    <name>mapreduce.jobhistory.admin.address </name>
    <value>${yarn.resourcemanager.hostname}:10033</value>
  </property>

  <property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>${yarn.resourcemanager.hostname}:19888</value>
  </property>
</configuration>
```

Figura A.9: MapReduce site

```
    <name>yarn.resourcemanager.address</name>
    <value>hadoop-master:8032</value>
  </property>
 <property>
    <name>yarn.nodemanager.localizer.address</name>
    <value>${yarn.nodemanager.hostname}:8040</value>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>hadoop-master:8088</value>
  </property>
  <property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>hadoop-master:8031</value>
  </property>
  <property>
    <name>yarn.resourcemanager.admin.address</name>
    <value>hadoop-master:8033</value>
  </property>
<property>
  <name>yarn.timeline-service.version</name>
  <value>2.0f</value>
</property>
<property>
 <name>yarn.nodemanager.aux-services</name>
 <value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
  <name>yarn.system-metrics-publisher.enabled</name>
  <value>true</value>
</property>
<property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>12288</value>
</property>
<property>
        <name>yarn.scheduler.maximum-allocation-mb</name>
        <value>12288</value>
</property>
<property>
        <name>yarn.scheduler.minimum-allocation-mb</name>
        <value>4096</value>
</property>
<property>
        <name>yarn.nodemanager.resource.cpu-vcores</name>
        <value>4</value>
</property>

<property>
        <name>yarn.scheduler.minimum-allocation-vcores</name>
        <value>1</value>
</property>
 <property>
```

Figura A.10: YARN site

```
hadoop@hadoop-master:/usr/local/hadoop$ jps
14307 SecondaryNameNode
14789 Jps
11670 JobHistoryServer
14073 NameNode
14491 ResourceManager
```

```
hadoop@Slave1:/usr/local/hadoop/etc/hadoop$ jps
19988 NodeManager
19861 DataNode
20108 Jps
```

Figura A.11: Elementos em Execução no Master e nos Slaves

```
hadoop@hadoop-master:/usr/local$ https://archive.apache.org/dist/spark/spark-2.2.0/spark-2.2.0-bin-hadoop2.7.tgz
```

```
hadoop@hadoop-master:/usr/local$ tar -xvf spark-2.2.0-bin-hadoop2.7.tgz
```

```
hadoop@hadoop-master:/usr/local$ mv spark-2.2.0-bin-hadoop2.7/ spark
```

Figura A.12:
Download Spark.

```
hadoop-master
```

```
hadoop@hadoop-master: /usr/local/hadoop/etc/hadoop
 GNU nano 2.5.3

Slave1
Slave2
Slave3
```

```
spark.master yarn
spark.driver.memory     6g
spark.yarn.am.memory    6g
spark.executor.memory   6g
spark.eventLog.enabled  true
spark.eventLog.dir hdfs://hadoop-master:9000/spark-logs
spark.history.provider            org.apache.spark.deploy.history.FsHistoryProvider
spark.history.fs.logDirectory     hdfs://hadoop-master:9000/spark-logs
spark.history.fs.update.interval  10s
spark.history.ui.port             18080
spark.yarn.archive hdfs://spark-jars/jars/*
```

Figura A.13: Configuração do Spark

```
hadoop@hadoop-master:~$ wget https://archive.apache.org/dist/flink/flink-1.0.3/flink-1.0.3-bin-hadoop2-scala_2.11.tgz
```

```
:~$ tar -vxzf flink-1.0.3-bin-hadoop2-scala_2.11.tgz /usr/local/flink
```

Figura A.14:
Download Flink

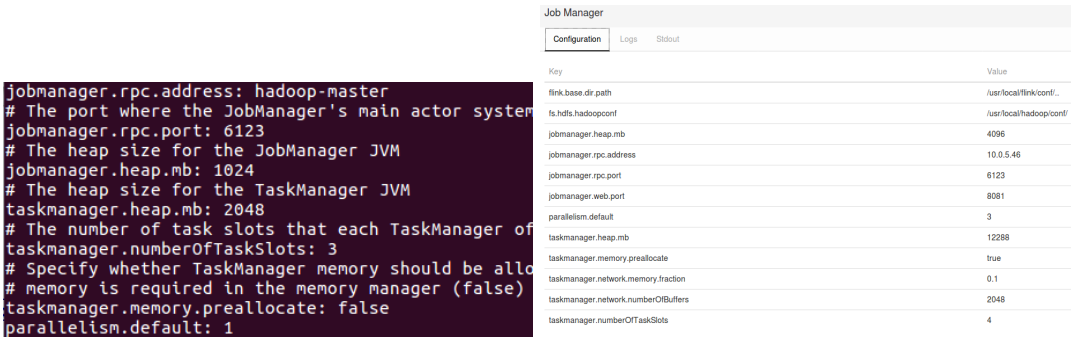

Figura A.15: Configuração do Flink