

**UNIVERSIDADE ESTADUAL DE PONTA GROSSA
MESTRADO EM COMPUTAÇÃO APLICADA**

LAYS HELENA LOPES VELOSO

**ALGORITMO K-MEANS PARALELO BASEADO EM
HADOOP-MAPREDUCE PARA MINERAÇÃO DE DADOS AGRÍCOLAS**

**PONTA GROSSA
2015**

LAYS HELENA LOPES VELOSO

**ALGORITMO K-MEANS PARALELO BASEADO EM
HADOOP-MAPREDUCE PARA MINERAÇÃO DE DADOS AGRÍCOLAS**

Dissertação apresentada para obtenção do
título de Mestre em Computação Aplicada
na Universidade Estadual de Ponta Grossa,
Área de concentração: Computação para
Tecnologias Agrícolas.

Orientador: Prof. Dr. Luciano José Senger

**PONTA GROSSA
2015**

Ficha Catalográfica
Elaborada pelo Setor de Tratamento da Informação BICEN/UEPG

V433 Veloso, Lays Helena Lopes
 Algoritmo K-means paralelo baseado em
 Hadoop-mapreduce para mineração de dados
 agrícolas/ Lays Helena Lopes Veloso. Ponta
 Grossa, 2015.
 59f.

 Dissertação (Mestrado em Computação
 Aplicada - Área de Concentração:
 Computação para Tecnologias em
 Agricultura), Universidade Estadual de
 Ponta Grossa.

 Orientador: Prof. Dr. Luciano José
 Senger.

 1.K-Means paralelo. 2.MapReduce.
 3.Hadoop. 4.Dados de fluxo. 5.Mineração de
 dados. I.Senger, Luciano José. II.
 Universidade Estadual de Ponta Grossa.
 Mestrado em Computação Aplicada. III. T.

CDD: 004.3

TERMO DE APROVAÇÃO

Lays Helena Lopes Veloso

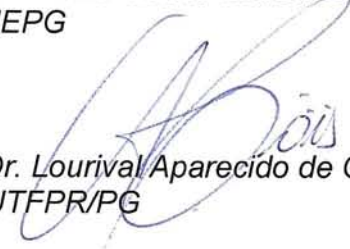
**“ALGORITMO K-MEANS PARALELO BASEADO EM HADOOP-MAPREDUCE
PARA MINERAÇÃO DE DADOS AGRÍCOLAS”**

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre no Programa de Pós-Graduação em Computação Aplicada da Universidade Estadual de Ponta Grossa, pela seguinte banca examinadora:

Orientadora:


Dr. Luciano José Senger
UEPG


Dr.ª Maria Salete Marcon Gomes Vaz
UEPG


Dr. Lourival Aparecido de Góis
UTFPR/PG

Ponta Grossa, 29 de abril 2015.

Aos meus pais Jorge e Helena e esposo Rogério, fundamentais em minha vida.

AGRADECIMENTOS

Agradeço a Deus por essa e várias oportunidades que me proporciona.

Aos meus pais Jorge e Helena por apostarem em mim.

Ao meu esposo Rogério por todo apoio e paciência.

Ao meu orientador, Prof. Dr. Luciano José Senger, pela dedicação e interesse na orientação deste trabalho.

Ao corpo docente do mestrado em Computação Aplicada da Universidade Estadual de Ponta Grossa, pela preparação, dicas e amizade.

Aos meus amigos e colegas do mestrado, pela cumplicidade e trocas de experiência.

À Capes pela bolsa de estudos.

RESUMO

Este trabalho teve como objetivo investigar a utilização de um algoritmo de agrupamento K-Means paralelo, com base no modelo paralelo MapReduce, para melhorar o tempo de resposta da mineração de dados. O K-Means paralelo foi implementado em três fases, executadas em cada iteração: atribuição das amostras aos grupos com centróide mais próximo pelos *Mappers*, em paralelo; agrupamento local das amostras atribuídas ao mesmo grupo pelos *Mappers* usando um *Combiner* e atualização dos centróides pelo *Reducer*. O desempenho do algoritmo foi avaliado quanto ao *SpeedUp* e *ScaleUp*. Para isso foram executados experimentos em modo *single-node* e em um *cluster* Hadoop formado por seis computadores de *hardware* comum. Os dados agrupados são medições de torres de fluxo de regiões agrícolas e pertencem a Ameriflux. Os resultados mostraram que com o aumento do número de máquinas houve ganho no desempenho, sendo que o melhor tempo obtido foi usando seis máquinas chegando ao *SpeedUp* de 3,25. Para apoiar nossos resultados foi construída uma tabela ANOVA a partir de repetições usando 3, 4 e 6 máquinas no *cluster*, respectivamente. Os resultados da análise ANOVA mostram que existe pouca variância entre os tempos de execução obtidos com o mesmo número de máquinas e existe uma diferença significativa entre as médias para cada número de máquinas. A partir dos experimentos para analisar o *ScaleUp* verificou-se que a aplicação escala bem com o aumento equivalente do tamanho dos dados e do número de máquinas no *cluster*, atingindo um desempenho próximo. Com os resultados conforme esperados, esse trabalho apresenta uma implementação paralela e escalável do K-Means para ser executada em um *cluster* Hadoop e melhorar o tempo de resposta do agrupamento de grandes bases de dados.

Palavras-chave: K-Means Paralelo, MapReduce, Hadoop, dados de fluxo, Mineração de Dados

ABSTRACT

This study aimed to investigate the use of a parallel K-means clustering algorithm, based on parallel MapReduce model, to improve the response time of the data mining. The parallel K-Means was implemented in three phases, performed in each iteration: assignment of samples to groups with nearest centroid by Mappers, in parallel; local grouping of samples assigned to the same group from Mappers using a Combiner and update of the centroids by the Reducer. The performance of the algorithm was evaluated in respect to SpeedUp and ScaleUp. To achieve this, experiments were run in single-node mode and on a Hadoop cluster consisting of six off-the-shelf computers. The data were clustered comprise flux towers measurements from agricultural regions and belong to Ameriflux. The results showed performance gains with increasing number of machines and the best time was obtained using six machines reaching the speedup of 3,25. To support our results, ANOVA analysis was applied from repetitions using 3, 4 and 6 machines in the cluster, respectively. The ANOVA show low variance between the execution times obtained for the same number of machines and a significant difference between means of each number of machines. The ScaleUp analysis show that the application scale well with an equivalent increase in data size and the number of machines, achieving similar performance. With the results as expected, this paper presents a parallel and scalable implementation of the K-Means to run on a Hadoop cluster and improve the response time of clustering to large databases.

Keywords: Parallel K-Means, MapReduce, Hadoop, flux data, Data Mining

LISTA DE SIGLAS

AM	Aprendizado de Máquina
API	<i>Application Programming Interface</i>
HDFS	<i>Hadoop Distributed File System</i>
IaaS	<i>Infraestructure as a Service</i>
KDD	<i>Knowledge Discovery in Database</i>
LAN	<i>Local Area Network</i>
MD	Mineração de Dados
MIND	<i>Multiple Instruction Multiple Data Stream</i>
MISD	<i>Multiple Instruction Single Data Stream</i>
MPI	<i>Message Passing Interface</i>
OV	Organização Virtual
PaaS	<i>Plataform as a Service</i>
PC	<i>Personal Computer</i>
PVM	<i>Parallel Virtual Machine</i>
SaaS	<i>Software as a Service</i>
SDK	<i>Software Development Kit</i>
SIMD	<i>Single Instruction Multiple Data Stream</i>
SISD	<i>Single Instruction Single Data Stream</i>
SSI	<i>Single System Image</i>

LISTA DE ILUSTRAÇÕES

1	Arquitetura de um <i>cluster</i> de computadores	13
2	Arquitetura de sistemas de grades computacionais em camadas	15
3	Arquitetura da Computação em Nuvem	17
4	Taxonomia de Flynn	20
5	Características do <i>Big Data</i>	24
6	<i>Cluster</i> Hadoop em camadas	26
7	Estrutura do Sistema de arquivos distribuído do Hadoop (HDFS)	26
8	Fluxo de execução do MapReduce	30
9	Imagens dos locais de estudo	36
10	Processo MapReduce para o K-Means	42
11	Gráfico de conclusão de tarefas <i>Map</i>	44
12	Gráfico de conclusão de tarefas <i>Reduce</i>	44
13	Gráfico de <i>SpeedUp</i>	46
14	Gráfico de <i>ScaleUp</i>	47
15	Informações das máquinas no <i>cluster</i> executando os <i>TaskTrackers</i>	57
16	Informações sobre os <i>jobs</i> MapReduce executados	58

LISTA DE TABELAS

1	Atributos da base de dados	38
2	Resumo dos resultados do <i>SpeedUp</i>	45
3	Tempos de execução: repetição 1, repetição 2, repetição 3	45
4	Tabela Resumo da ANOVA	45
5	Tabela ANOVA	45
6	Resumo dos resultados do <i>ScaleUp</i>	46

SUMÁRIO

1	Introdução	10
2	Revisão da Literatura	12
2.1	Computação Paralela	17
2.2	Mineração de Dados	20
2.3	K-Means	22
2.4	Mineração de Dados com Big Data	23
2.5	Hadoop	25
2.5.1	MapReduce	29
2.5.2	<i>Frameworks</i> para processamento distribuído de <i>Big Data</i>	30
2.6	Trabalhos Relacionados	31
3	Materiais e Métodos	34
3.1	Algoritmo K-Means com base em MapReduce	34
3.2	Bases de dados utilizadas	35
3.3	Avaliação do Desempenho	39
4	Resultados	40
4.1	K-Means Paralelo	40
4.2	Análise dos Tempos de Execução	43
5	Conclusões	48
	REFERÊNCIAS	50
	APÊNDICE A – Arquivos de configuração do Hadoop	54
	APÊNDICE B – Imagens extraídas da interface <i>web</i>	56
	APÊNDICE C – Publicação	59

1 INTRODUÇÃO

Com a rápida taxa de modernização nos equipamentos de aquisição e transmissão de dados, grandes volumes de dados estão sendo gerados. Com isso surge a necessidade de ferramentas capazes de lidar com esses volumes crescentes de dados e extrair informações úteis que possam ser aproveitadas no processo de tomada de decisão, isto, com um tempo de resposta aceitável e com baixo custo. Esse fenômeno é conhecido como *Big Data*.

A Mineração de Dados (MD) é uma ferramenta que vem apoiando diferentes campos da ciência, pois permite extrair padrões e relacionamentos entre dados, a partir da aplicação de algoritmos de Aprendizado de Máquina (AM). A partir desses padrões, é possível construir modelos preditivos para apoiar a tomada de decisão (WITTEN; FRANK, 2005). No entanto, muitos *softwares* disponíveis, não estão preparados para tirar vantagem de toda a capacidade de processamento dos *hardwares* atuais e melhorar o tempo de resposta da MD, que pode ser demorado quando grandes bases de dados são usadas. Outro problema é que com as bases de dados cada vez maiores, se torna difícil organizar e preparar os dados para o processo de MD. Uma saída comum para esses problemas é usar pequenas parcelas de dados, não sendo a solução mais apropriada, pois informações importantes podem ser ocultadas.

Essas dificuldades motivaram o desenvolvimento de ferramentas voltadas para *Big Data* com objetivo de armazenar, processar e extrair informações de grandes volumes de dados com alto desempenho e com baixo custo de manutenção. Algumas dessas ferramentas implementam modelos analíticos que permitem a implementação de algoritmos que são usados em MD. Desse modo, é possível usar *Big Data* para melhorar os resultados da MD, uma vez que mais dados estão disponíveis Kudyba (2014). O Hadoop é uma plataforma de computação distribuída que permite tirar vantagem das capacidades de computadores de baixo custo para aumentar o desempenho de aplicações. Ele implementa o *framework* MapReduce para a modelagem de algoritmos escaláveis. Kumar e Gupta (1994) definem a escalabilidade de um algoritmo paralelo como sendo a medida da sua capacidade de fazer uso eficaz de um número crescente de processadores em uma arquitetura paralela.

A agricultura é uma área que pode se beneficiar dos mecanismos para *Big Data*, pois as aplicações de MD geralmente envolvem processar e combinar informações de diferentes fontes de dados, como imagens de sensoriamento remoto, redes de sensores e conjuntos de dados meteorológicos, o que requer repetidos cálculos em grandes bases de dados. O agrupamento de dados é um método comumente utilizado em MD. Algumas

aplicações relatadas na área agrícola são a segmentação de imagens de sensoriamento remoto (VIEIRA *et al.*, 2012), a identificação de grupos de comportamentos dos ecossistemas (MILLS *et al.*, 2011) e o delineamento de zonas de manejo agrícola (TAGARAKIS *et al.*, 2013). O K-Means é uma técnica de agrupamento clássica, popular por ser eficaz e simples de ser implementado.

Dessa forma, este trabalho teve como objetivo geral investigar o uso do algoritmo de agrupamento *K-Means* paralelo implementado com base no modelo MapReduce para melhorar o tempo de resposta da mineração de dados agrícolas, usando grandes bases de dados. Os objetivos específicos foram: investigar o ganho de desempenho que pode ser obtido usando Hadoop e MapReduce no agrupamento de grandes bases de dados e avaliar a escalabilidade do K-Means paralelo baseado no MapReduce em um *cluster* Hadoop.

Para atingir esses objetivos o algoritmo K-Means paralelo foi avaliado com relação ao *SpeedUp* e *ScaleUp*, a partir de experimentos no *cluster*, realizando o agrupamento em bases de dados de medições de fluxo de regiões agrícolas. Essas bases de dados contêm medições de fluxo de carbono, variáveis do solo e climáticas, que permitem analisar os fatores que influenciam na dinâmica do carbono. Tais medições de longo prazo são promissoras para a agricultura sustentável, pois permitem prever os estoques de carbono sob mudanças climáticas dando suporte ao planejamento agrícola que envolve escolha de culturas, forma de preparo do solo e outras práticas para aumentar a produção com baixa emissão de carbono.

O trabalho está organizado em 5 Capítulos. No Capítulo 1 é apresentada a motivação do trabalho, os objetivos gerais e objetivos específicos. No Capítulo 2, é feita uma revisão bibliográfica sobre Computação Distribuída, apresentando os conceitos de Computação paralela, grades e *clusters* computacionais e de computação em nuvem, que estão relacionados ao estudo. Em seguida são discutidas as técnicas e as tecnologias utilizadas no trabalho, que são: Mineração de Dados, K-Means e Hadoop. Por fim, são comentadas pesquisas relacionadas.

No Capítulo 3 são apresentados os materiais e métodos utilizados para o desenvolvimento do trabalho, detalhando como foram conduzidos os experimentos exploratórios.

No Capítulo 4 são detalhadas as partes do algoritmo K-Means paralelo e são apresentados os resultados obtidos com os experimentos.

Por fim, as conclusões do trabalho, juntamente com recomendações e sugestões de trabalhos futuros são apresentadas no Capítulo 5.

2 REVISÃO DA LITERATURA

Para Tanenbaum e Steen (2007), um sistema distribuído é uma coleção de computadores independentes (nós), que são vistos como um único sistema. A computação distribuída é uma classe de sistemas distribuídos usada para melhorar o desempenho de tarefas computacionais. Os sistemas de computação distribuída podem ser divididos em grades e *clusters* computacionais (TANENBAUM; STEEN, 2007):

Um **cluster de computadores** consiste em uma coleção de estações de trabalho ou computadores convencionais (PCs) interconectados por meio de uma rede local (LAN) de alta velocidade, sendo que na maioria dos casos é executado o mesmo Sistema Operacional em cada nó.

Já **grades computacionais** são sistemas distribuídos que geralmente são construídos como uma confederação de sistemas computacionais, sendo que cada sistema pode pertencer a um domínio diferente e podem ser diferentes em *hardware*, *software* e na tecnologia da implantação da rede.

Conforme Karimi (2014), os nós em uma plataforma de computação distribuída podem ser fracamente ou fortemente acoplados. Na primeira forma, os nós são fixos e a remoção e adição de novos nós requer operações mais complexas. Da segunda maneira, os nós são dinâmicos, podendo ser removidos ou adicionados *on-the-fly*.

A seguir serão discutidos os conceitos de Computação em *Cluster*, Computação em Grade e Computação em Nuvem que é uma proposta mais recente baseada em grades computacionais.

As vantagens principais da computação em *cluster* segundo (WILKINSON; ALLEN, 2005) são:

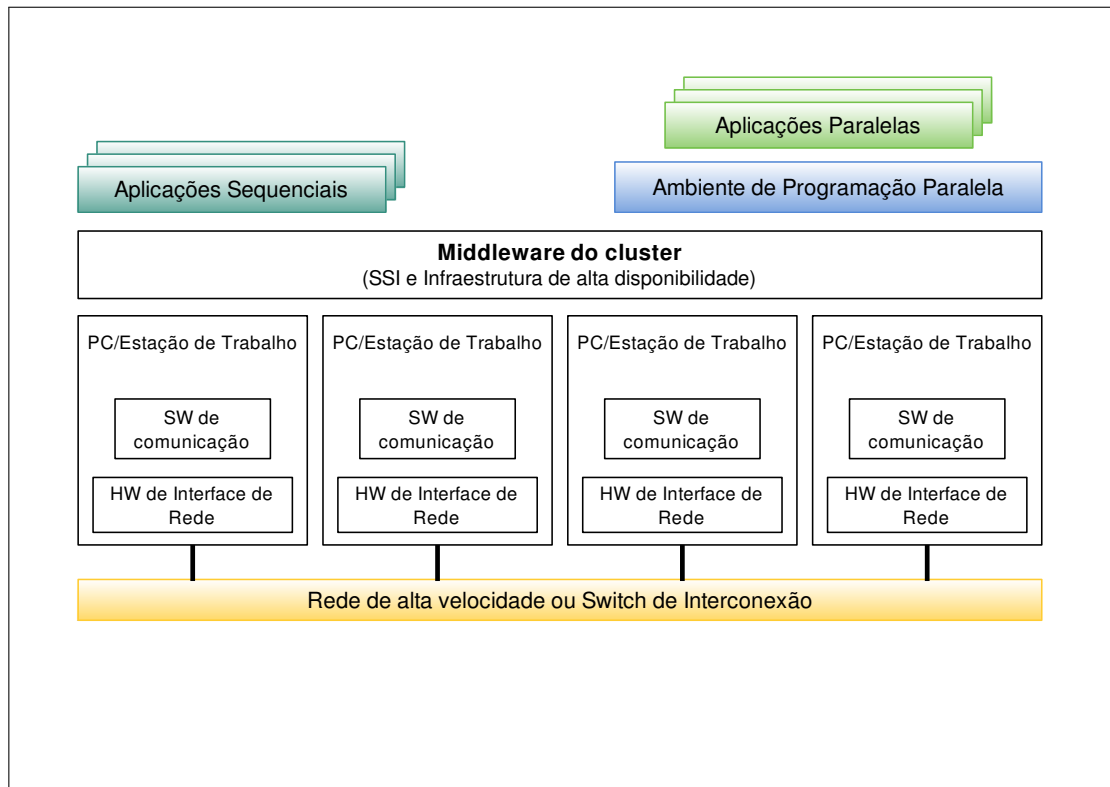
- Baixo custo: Os *clusters* de computadores podem ser configurados usando computadores convencionais (PCs) que estão disponíveis comercialmente a baixos custos;
- Expansibilidade: O *cluster* pode ser expandido facilmente pela adição de computadores, discos e outros recursos para aumentar o poder de processamento;
- Reuso: *Softwares* já existentes podem ser usados ou modificados.

A Figura 1 ilustra a arquitetura típica de um *cluster* de computadores onde computadores independentes são interconectados através de uma rede local de alta velocidade ou através de um *Switch*. Os nós do *cluster* podem ser PCs ou estações de trabalho. Um

middleware é instalado em cada computador para permitir a computação em *cluster*. Em computação em *cluster* o *middleware* é uma camada de software sobre os nós que permite que o *cluster* seja tratado como um único sistema de computação paralela utilizando o conceito de Imagem Única do Sistema (SSI) (BUYYA, 1999). Exemplos de *middlewares* já conhecidos são *Parallel Virtual Machine* (PVM) (GEIST *et al.*, 1994) e *Message Passing Interface* (MPI) (GROPP *et al.*, 2014).

Na maioria dos casos a Computação em *Cluster* é usada para programação paralela onde um programa computacionalmente intensivo é executado paralelamente em múltiplas máquinas (TANENBAUM; STEEN, 2007).

Figura 1: Arquitetura de um *cluster* de computadores



Fonte: Traduzida de (BUYYA, 1999)

PVM é um ambiente em tempo de execução e uma biblioteca de *software* que permite que uma coleção de computadores, conectados por uma rede, seja tratada como uma única Máquina Paralela Virtual. A biblioteca da PVM fornece uma interface para passagem de mensagens e gerenciamento de processos (DROZDOWSKI, 2010).

MPI é um padrão de interface para a passagem de mensagens em computadores paralelos usando um modelo de programação baseado em memória distribuída. Sua versão atual é a MPI-3.0 (GROPP *et al.*, 2014).

Em contraste com a homogeneidade dos *clusters* computacionais, os sistemas de

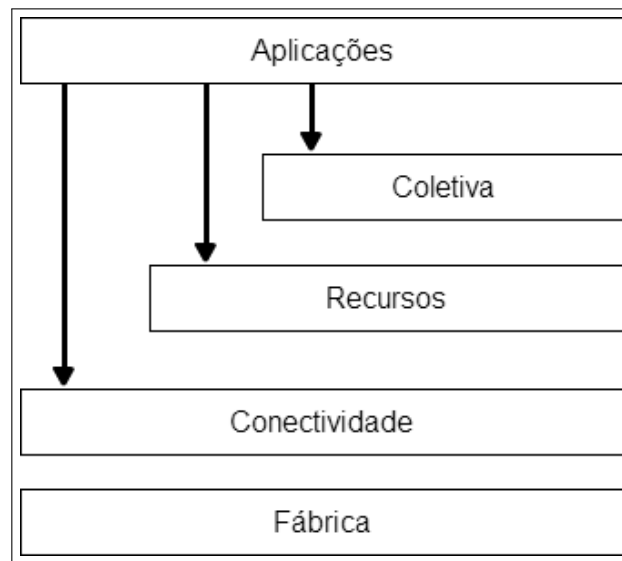
grade computacionais possuem um alto nível de heterogeneidade já que não são feitas previsões em termos de *hardware*, sistema operacional, rede, domínio administrativo, políticas de segurança e outros (TANENBAUM; STEEN, 2007).

De acordo com Foster, Kesselman e Tuecke (2001) um dos focos das grades computacionais é o compartilhamento de recursos em larga escala: uma série de recursos de diferentes organizações são compartilhados na forma de uma Organização Virtual (OV) onde cada organização possui direitos de acesso aos recursos fornecidos para tal. A maior dificuldade encontrada nesse sistema é ter o controle do compartilhamento de recursos e da resolução de problemas em organizações virtuais, multi-institucionais e dinâmicas.

Foster, Kesselman e Tuecke (2001) propôs uma arquitetura para sistemas de computação em grade (Figura 2). Seus componentes são organizados em 5 camadas:

- A camada **Fábrica** fornece recursos acessíveis por intermédio dos protocolos da grade, por exemplo: recursos computacionais, sistemas de armazenamento catálogos, recursos de rede e sensores. Os recursos também podem ser lógicos, como um sistema de arquivos distribuído ou um *cluster* de computadores;
- A camada **Conectividade** define os protocolos básicos de comunicação e os protocolos de autenticação necessários para operações de rede específicas da grade computacional. Os protocolos de comunicação permitem a troca de dados entre os recursos da camada Fábrica. Os protocolos de autenticação são construídos sobre serviços de comunicação para prover mecanismos seguros de criptografia e verificar a identidade de usuários e recursos;
- A camada de **Recursos** define protocolos (APIs e SDKs) para negociação, monitoramento, controle e pagamento de operações compartilhadas em recursos individuais de forma segura. As implementações da camada de Recursos desses protocolos chamam as funções da camada Fábrica para acessar e controlar recursos locais;
- A camada **Coletiva** possui esse nome pois captura interações entre conjuntos de recursos. Enquanto a camada de Recursos é focada em interações com recursos individuais essa camada contém protocolos e serviços associados com recursos globais;
- A camada final compreende as **Aplicações** dos usuários que operam em um ambiente de OV. Esta camada oferece aplicações como serviços definidos em qualquer uma das camadas.

Figura 2: Arquitetura de sistemas de grades computacionais em camadas



Fonte: Adaptada de (FOSTER; KESSELMAN; TUECKE, 2001)

Uma proposta mais recente de sistema de computação distribuída é a Computação em Nuvem, que teve base nas grades computacionais. A Computação em Nuvem se difere da Computação em Grade com relação a sua arquitetura e gerenciamento dos recursos. Como visto nas grades computacionais os usuários possuem acesso direto aos recursos, enquanto no ambiente de nuvem a entrega de recursos ocorre de maneira abstrata (LONEA, 2012).

O termo Computação em Nuvem é usado para definir um conjunto de recursos computacionais disponibilizados através da Internet. Dessa forma, computadores simples podem acessar uma variedade de recursos e serviços, geralmente pagos com base na utilização, reduzindo os requisitos de equipamentos do usuário (COULOURIS *et al.*, 2013). Os ambientes de Computação em Nuvem são implementados usando *clusters* computacionais para fornecer escalabilidade e o desempenho exigidos pelos serviços (COULOURIS *et al.*, 2013).

Taurion (2009), descreve as características principais de um ambiente de Computação em Nuvem, como sendo:

- Ilusão da disponibilidade de recursos infinitos, acessíveis sob demanda;
- Eliminação da necessidade de adquirir e provisionar recursos;
- Elasticidade, de forma que os recursos possam ser acessados na quantidade que forem necessários e a capacidade computacional aumente e diminua dinamicamente;
- O pagamento de serviços pela quantidade de recursos utilizados (*pay-per-use*).

As nuvens computacionais são classificadas pelo tipo de uso e modo de acesso. De acordo com Jamsa (2011), são três os modelos principais de serviço na nuvem, oferecidos na Internet:

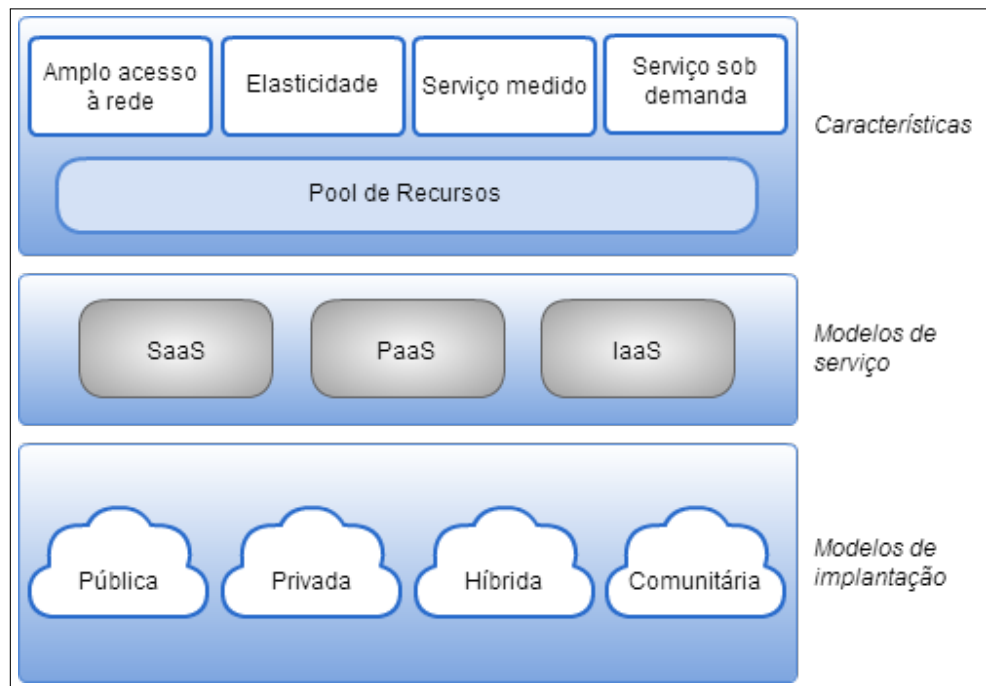
- **Software como Serviço (SaaS)** Neste modelo, os usuários finais utilizam um navegador web para acessar a aplicação, que reside juntamente com seus programas e dados, na nuvem. Como as soluções residem na nuvem, podem ser facilmente escaladas para atender as necessidades do usuário. Este modelo elimina a necessidade de manter uma infraestrutura de *data center* local;
- **Plataforma como Serviço (PaaS)** Este tipo de solução fornece uma coleção de recursos de *hardware* e *software* que desenvolvedores podem utilizar para construir e implantar aplicações dentro da nuvem. Como os recursos computacionais não residem no *data center* e sim na nuvem, podem ser escalados para aumentar ou diminuir dependendo da demanda da aplicação;
- **Infraestrutura como Serviço (IaaS)** Este modelo fornece o *hardware* necessário para executar uma aplicação e elimina a necessidade de alojar e manter *data centers* caros. Ao contrário do modelo PaaS, esta solução requer que o usuário gerencie todas as aplicações e se responsabilize em manter as atualizações do sistema.

As classes de nuvens, conforme o modo de acesso são:

- **Pública** - Uma Nuvem Pública é fornecida por um fabricante para diferentes clientes;
- **Privada** - A Nuvem Privada pertence ao próprio utilizador ou a uma empresa e seus usuários são somente internos;
- **Híbrida** - Uma Nuvem Híbrida é a junção de serviços executados em uma Nuvem Pública e em uma Nuvem Privada;
- **Comunitária** - A infraestrutura de uma Nuvem Comunitária é compartilhada por diferentes empresas.

A Figura 3 mostra a arquitetura geral da Computação em Nuvem.

Figura 3: Arquitetura da Computação em Nuvem



Fonte: Traduzida de (AMINI *et al.*, 2013)

2.1 Computação Paralela

Conforme Wilkinson e Allen (2005) a Computação Paralela consiste na utilização de múltiplos processadores em um único computador, e/ou no uso de múltiplos computadores operando simultaneamente em um único problema, que é dividido em partes, onde cada uma é executada por um processador diferente, paralelamente. O uso da Computação Paralela é especialmente fundamental em problemas de maior complexidade, que geralmente necessitam executar cálculos repetidas vezes em grandes quantidades de dados para apresentar resultados válidos (WILKINSON; ALLEN, 2005).

Para que um programa obtenha vantagem da computação paralela é necessário escrever programas que executem fragmentos em paralelo. A prática de escrever programas para funcionar desta forma é conhecida como Programação Paralela (WILKINSON; ALLEN, 2005).

Existem diferentes medidas para avaliar o desempenho de programas paralelos. As mais comuns são: *SpeedUp*, Eficiência e *ScaleUp*.

A medida que define o quanto o uso do paralelismo melhorou o desempenho de um programa é conhecida como *SpeedUp*. O *SpeedUp* em p processadores é dado pela

equação 1 (WILKINSON; ALLEN, 2005):

$$S(p) = \frac{T_{seq}}{T_{par}} \quad (1)$$

onde T_{seq} é o tempo de execução do algoritmo sequencial correspondente e T_{par} é o tempo de execução do algoritmo paralelo em p processadores.

O *SpeedUp* máximo possível é geralmente p em p processadores (*SpeedUp* linear). O *SpeedUp* será obtido se a computação puder ser dividida em processos de duração igual (WILKINSON; ALLEN, 2005).

Conforme Taniar *et al.* (2008), um *SpeedUp* superlinear (quando $S(p) > p$), é difícil de ser atingido. Porém, quando ocorre, geralmente é devido ao uso de um algoritmo sequencial sub-ótimo ou a uma característica particular da arquitetura paralela, por exemplo, memória extra no sistema.

A eficiência paralela de um algoritmo define por quanto tempo os processadores estão sendo utilizados durante a computação. A eficiência, E é dada pela equação 2 (WILKINSON; ALLEN, 2005):

$$E = \frac{T_{seq}}{T_{par} \times p} \quad (2)$$

o que leva a:

$$E = \frac{S(p)}{p} \quad (3)$$

Se o *SpeedUp* for linear ($S(p) = p$), a eficiência será máxima e terá valor 1 equivalente a 100%, se dada em porcentagem.

O *ScaleUp* se refere a capacidade de processar tarefas maiores no mesmo período de tempo, aumentando o número de elementos de processamento, ou aumentando o nível de paralelismo. Ou seja, o objetivo de medir o *ScaleUp* é investigar se a adição de mais recursos quando a carga de trabalho é aumentada, mantém um bom desempenho. O *ScaleUp* é calculado pela equação 4 (TANIAR *et al.*, 2008):

$$ScaleUp = \frac{\text{tempo gasto com um único processador em um sistema menor}}{\text{tempo gasto com multiprocessador em um sistema maior}} \quad (4)$$

De acordo com Shroff (2010), uma implementação paralela é escalável quando:

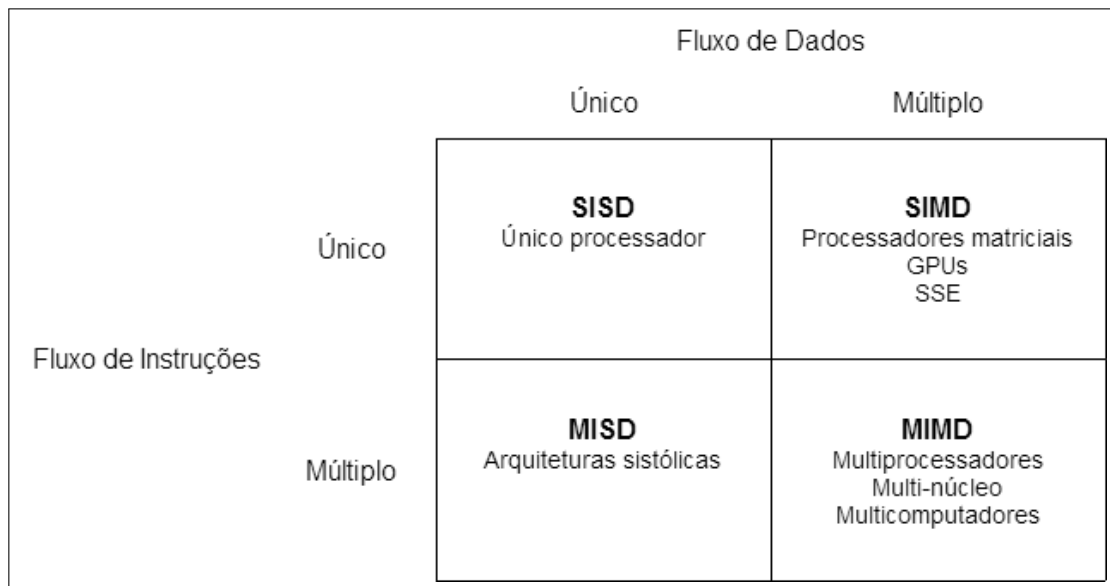
- A eficiência paralela permanece constante a medida em que o tamanho dos dados aumenta com um aumento correspondente da quantidade de processadores e;
- A eficiência paralela aumenta com o tamanho dos dados para um número fixo de processadores.

Um Computador Paralelo é uma coleção de elementos de processamento que trabalham em conjunto para resolver problemas grandes em alta velocidade (ALMASI; GOTTLIEB, 1989). Na taxonomia de Flynn e Rudd (1996) tem-se quatro arquiteturas paralelas classificadas de acordo com as métricas **fluxo de dados** e **fluxo de instruções** (Figura 4):

- SISD (*Single Instruction Single Data Stream*) - Nesta arquitetura um único processador executa um único fluxo de instruções sobre dados armazenados em uma única memória;
- SIMD (*Single Instruction Multiple Data Stream*) - Esta classe descreve computadores com múltiplos elementos de processamento que executam uma operação em comum em múltiplos dados simultaneamente. Arquiteturas SIMD possuem uma unidade de controle centralizada. As unidades de processamento gráfico (GPUs) modernas são comumente baseadas em SIMD;
- MISD (*Multiple Instruction Single Data Stream*) - Os computadores MISD têm múltiplos fluxos de instruções operando em um mesmo fluxo de dados;
- MIMD (*Multiple Instruction Multiple Data Stream*) - Nesta arquitetura processadores diferentes podem executar instruções independentes em partes de dados independentes. A maioria das arquiteturas paralelas modernas são MIMD. São exemplos os computadores com múltiplos processadores, computadores multi-núcleo e sistemas multicomputadores (*clusters*).

A Computação paralela e distribuída pode ser usada para resolver problemas computacionais em alta velocidade e melhorar o desempenho das aplicações. Os algoritmos de MD podem obter vantagem dessas formas de computação pois muitas vezes envolvem repetidos cálculos em grandes bases de dados. Para isso é necessário escrevê-los de forma que sua fase mais custosa possa ser dividida em partes para serem executadas em paralelo. Na próxima Seção será tratado sobre a MD e como ela pode obter vantagem da Computação paralela e distribuída.

Figura 4: Taxonomia de Flynn



2.2 Mineração de Dados

A Mineração de dados surgiu pela necessidade de agilizar o processo de descoberta de conhecimento em dados, que anteriormente só era feita através da análise e interpretação humana. Com o aumento nas quantidades de dados que são coletadas, fica impraticável uma análise manual com o mínimo de erro e em tempo razoável (FAYYAD; PIATETSKY-SHAPIO; SMYTH, 1996).

A MD é a etapa em Descoberta de Conhecimento em Bases de Dados (KDD), em que algoritmos inteligentes são aplicados para extrair padrões em dados que auxiliem a descoberta de conhecimento (HAN; KAMBER; PEI, 2011). O KDD é um processo composto por etapas sequenciais. O processo geral de KDD consiste em cinco etapas, descritas a seguir (FAYYAD; PIATETSKY-SHAPIO; SMYTH, 1996):

- Seleção - Esta etapa é dedicada à compreensão do domínio da aplicação e dos objetivos da tarefa e à criação do conjunto de dados com os atributos e exemplos a serem usados na tarefa;
- Pré-processamento - Esta etapa consiste na limpeza dos dados: remoção de ruídos e *outliers*¹, tratamento de falta de dados e reconfiguração dos dados a fim de assegurar formatos consistentes;
- Transformação - Consiste na transformação dos dados em formatos utilizáveis. Ex:

¹ *Outliers*: Dados que possuem valores atípicos ou com características bastante distintas dos demais registros podendo influenciar negativamente os resultados da MD

redução da dimensionalidade e discretização dos dados;

- Mineração de Dados - Nesta etapa é feita a escolha da tarefa de MD (p.e. classificação, regressão, agrupamento, etc.) e a escolha e execução do algoritmo de MD adequado à tarefa para a extração de padrões, que podem ser apresentados de uma forma particular, como regras de associação, árvores de decisão, modelos de regressão, entre outras.
- Interpretação e avaliação - Por fim, é feita a interpretação dos padrões extraídos e a consolidação do conhecimento: incorporação e documentação do conhecimento e comunicação aos interessados.

São candidatas à MD, qualquer fonte de dados que possam conter padrões interessantes (ex: bases de dados, *data warehouses*², a *Web* e dispositivos que colem e transmitam dados em tempo real). Os padrões descobertos na MD podem auxiliar à tomada de decisão em vários domínios e melhorar a precisão de previsões em dados futuros (WITTEN; FRANK, 2005).

A MD pode ser categorizada em tipos de tarefas. A seguir são descritos tipos de tarefas comumente utilizadas (WITTEN; FRANK, 2005):

- Classificação (*classification*) - Essa tarefa visa identificar a qual classe um determinado registro pertence. O aprendizado por classificação é também chamado supervisionado pois analisa o conjunto de entrada, com cada registro contendo uma indicação da classe ao qual o mesmo pertence, e aprende como classificar um novo registro;
- Associação (*association*) - A tarefa de associação visa encontrar relações entre os atributos do conjunto de dados. Essa técnica de aprendizado é não-supervisionada pois encontra qualquer regra nos dados e não apenas as que predizem uma classe em especial;
- Agrupamento (*clustering*) - Como associação o agrupamento também se trata de uma técnica não-supervisionada que pode ser usada quando não há classe especificada. Seu objetivo é identificar estruturas características (grupos) presentes em um conjunto de dados.

²*Data warehouse* ou depósito de dados, é um banco de dados organizado para dar suporte à tomada de decisões estratégicas de uma empresa

O agrupamento de dados têm aplicações relevantes em diferentes campos da ciência. Entre as aplicações relatadas na área agrícola estão: segmentação de imagens de sensoriamento remoto (VIEIRA *et al.*, 2012), identificação de grupos de comportamentos dos ecossistemas (MILLS *et al.*, 2011) e delineamento de zonas de manejo agrícola (TAGARAKIS *et al.*, 2013). O K-Means é uma das técnicas de agrupamento mais populares por ser eficaz e simples de ser implementado.

2.3 K-Means

O Método K-Means consiste em reunir n amostras de dados em k grupos de maneira que os elementos em um mesmo grupo sejam similares entre si e diferentes daquelas em outros grupos (SAKR; GABER, 2014). A distância entre os elementos em um mesmo grupo é chamada distância **intra-grupo** e deve ser a mais baixa possível e a distância entre os elementos de um grupo e outro é chamada distância **inter-grupo** e deve ser a mais alta possível.

O algoritmo K-Means sequencial pode ser descrito resumidamente em 4 passos:

1. Seleção de k amostras como centróides iniciais;
2. Atribuição de cada amostra ao grupo com centróide mais próximo;
3. Cálculo de novos k centróides;
4. Os passos 2 e 3 são repetidos até que se atinja o ponto de convergência, ou seja, até que os centróides não mudem mais.

Para calcular a similaridade entre os elementos é usada alguma técnica de medida de distância. A distância euclidiana é a medida mais utilizada. No espaço bidimensional, a distância euclidiana entre dois pontos (a, b) e (x, y) pode ser calculada pelo teorema de Pitágoras que fornece o segmento de reta que une os dois pontos (BORTOLOSSI, 2002):

$$d((x, y), (a, b)) = \sqrt{(x - a)^2 + (y - b)^2} \quad (5)$$

Já no espaço n -dimensional, a distância euclidiana entre dois pontos $p = (p_1, p_2, \dots, p_n)$ e $x = (x_1, x_2, \dots, x_n)$ é calculada por (BORTOLOSSI, 2002):

$$d(x, p) = \sqrt{(x_1 - p_1)^2 + \dots + (x_n - p_n)^2} = \sqrt{\sum_{i=1}^n (x_i - p_i)^2} \quad (6)$$

A complexidade de tempo por iteração do K-Means é linear ao tamanho da base de dados n , ao número de grupos k e à dimensionalidade da base de dados. São necessários $n \times k$ cálculos em cada iteração (BEKKERMAN; BILENKO; LANGFORD, 2012). O K-Means pode tirar vantagem do paralelismo. As amostras na base de dados podem ser distribuídas em cada processador e então atribuídas ao grupo com o centróide mais próximo, em paralelo (DEAN, 2014).

O K-Means apresenta algumas limitações:

- Os grupos finais são sensíveis aos centróides iniciais e estes são geralmente escolhidos de maneira aleatória. Pequenas mudanças na escolha inicial pode resultar em grupos completamente diferentes. Além disso, em um domínio desconhecido é uma tarefa difícil determinar o número de grupos que reflitam uma divisão adequada das observações;
- Pode ser necessária alguma normalização antes que os dados sejam agrupados. Por exemplo garantir que os valores de atributos equivalentes sejam expressos na mesma unidade de medida.

Uma forma de estimar o número de grupos em uma base de dados sem possuir conhecimento do domínio é medir a distância entre os elementos nos grupos resultantes de tal forma que os grupos mais adequados serão aqueles que seus elementos sejam diferentes dos elementos em outros grupos, e os elementos em um mesmo grupo sejam semelhantes entre si. Diferentes técnicas foram propostas (DUNN, 1974; HUBERT; SCHULTZ, 1976; DAVIES; BOULDIN, 1979).

O algoritmo de agrupamento *Canopy* (MCCALLUM; NIGAM; UNGAR, 2000), pode ser utilizado como uma etapa de pré-processamento antes de executar o K-Means para selecionar os primeiros grupos centróides.

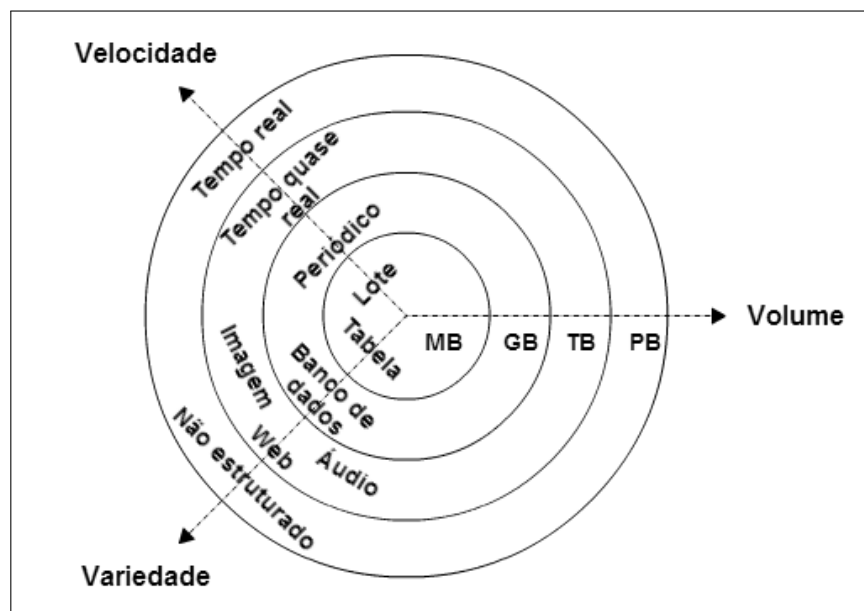
2.4 Mineração de Dados com Big Data

A McKinsey Global Institute define *Big Data* como "Um conjunto de dados cujo crescimento é exponencial e cuja dimensão está além da habilidade das ferramentas típicas de capturar, gerenciar e analisar dados".

De acordo com Taurion (2013), o *Big Data* é composto basicamente pelas três seguintes variáveis (Figura 5):

1. Volume - As organizações estão coletando o máximo de dados possíveis, para melhorar análises preditivas e a tomada de decisão;
2. Variedade de dados - Os dados são coletados de diferentes fontes, de sensores, a ERPs e comentários nas mídias sociais resultando em uma ampla variedade de formatos, estruturados e não estruturados;
3. Velocidade - Se refere a velocidade na qual os dados são capturados. Muitas vezes é preciso analisar os dados e tomar decisões em tempo real.

Figura 5: Características do *Big Data*



Fonte: Traduzida de (BHOOLA *et al.*, 2014)

As soluções *Big Data* foram desenvolvidas principalmente para lidar com o grande volume de dados gerado continuamente na *Web*, por exemplo, através das redes sociais. Porém, elas podem ser usadas para lidar com dados gerados por outros meios digitais, como redes de sensores, dispositivos GPS e torres de medição. Essas soluções são usadas para uma série de tarefas que permitam extrair informações úteis nos dados, tais como: mineração de texto, indexação *Web* em larga escala e aprendizado de máquina em larga escala.

Conforme (KUDYBA, 2014) o uso de *Big Data* na MD pode melhorar a tomada de decisão, pois o uso de todos os dados possíveis pode permitir encontrar relações entre os dados anteriormente desconhecidas, que não poderiam ser encontradas quando é usada uma quantidade reduzida dos dados.

Os atuais *frameworks* para tratamento de *Big Data* estão em pleno desenvolvimento e evolução. A próxima Seção irá tratar sobre o Hadoop que foi o escolhido para ser utilizado nesse trabalho. O Hadoop é atualmente um projeto *Open Source* da Apache para processamento e análise de *Big Data*. O Hadoop está sendo desenvolvido e utilizado por uma ampla comunidade de colaboradores devido à sua facilidade de instalação e configuração e o baixo custo envolvido.

2.5 Hadoop

O Hadoop (HADOOP, 2014; BORTHAKUR, 2007) é um *framework Open Source* baseado na linguagem de programação Java para o processamento de grandes volumes de dados em ambiente de computação distribuída. O Hadoop foi inspirado nos projetos *Google MapReduce* e *Google File System* (GFS)³. Atualmente o Hadoop é um projeto de alto nível da fundação Apache usado por grandes empresas como Amazon, Google, Yahoo e IBM.

O Hadoop implementa um sistema de arquivos distribuído, o *Hadoop Distributed File System* (HDFS), e o *framework* MapReduce, modelo de programação para análise de dados em paralelo. Para obter vantagem do processamento paralelo do Hadoop o programa deve ser escrito como um *job* MapReduce. Um *job* MapReduce consiste dos dados de entrada, do programa MapReduce e de informações de configuração. O Hadoop executa o *job* dividindo-o em tarefas, geralmente de dois tipos: tarefas *Map* e tarefas *Reduce* (WHITE, 2012).

Um *cluster* Hadoop é dividido entre as camadas do HDFS e do MapReduce e em cada uma são executados *daemons*⁴ específicos. O papel desses *daemons* serão comentados a seguir. A Figura 6 apresenta essa abstração em um exemplo de *cluster* com uma máquina mestre e uma máquina trabalhadora.

O HDFS tem uma arquitetura cliente/servidor. Um *cluster* do Hadoop consiste de até dois nós mestres: um *Namenode* e um *JobTracker* e múltiplos *Datanodes*, geralmente um por nó, que atendem solicitações de diversos clientes (BORTHAKUR, 2007). O *JobTracker* e o *Namenode* também podem ser executados em uma única máquina mestre ao mesmo tempo.

Uma representação da estrutura do HDFS é mostrada na Figura 7.

³GFS: Sistema de arquivos distribuído criado pela Google

⁴Daemons: Processos que são executados no Sistema Operacional em plano de fundo

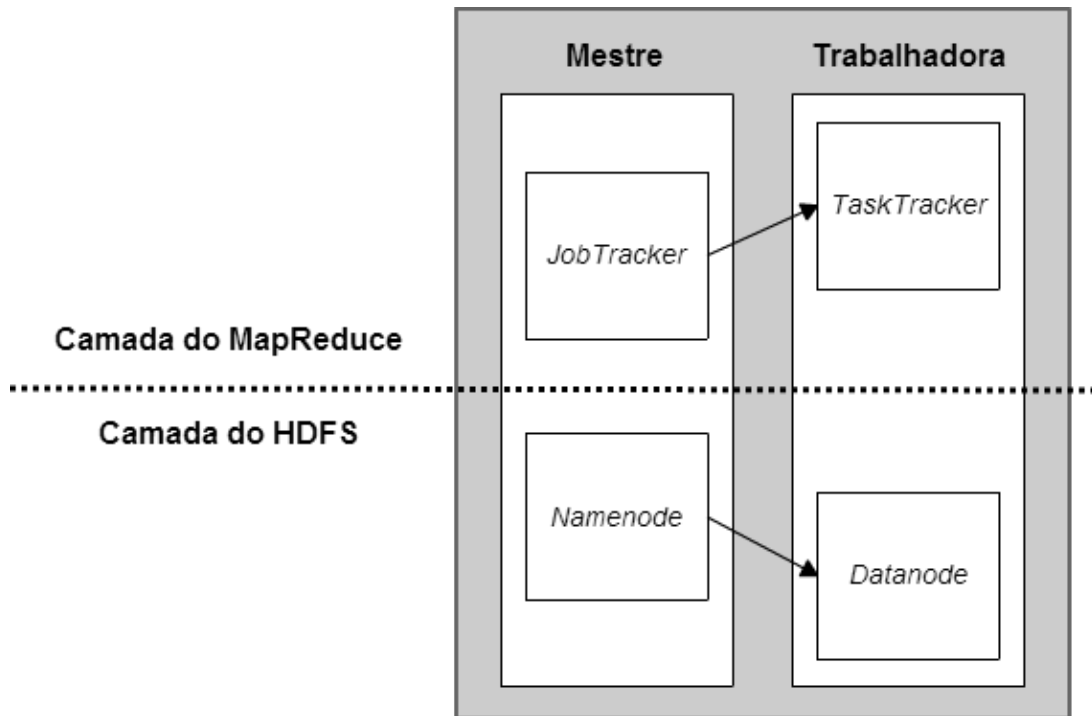
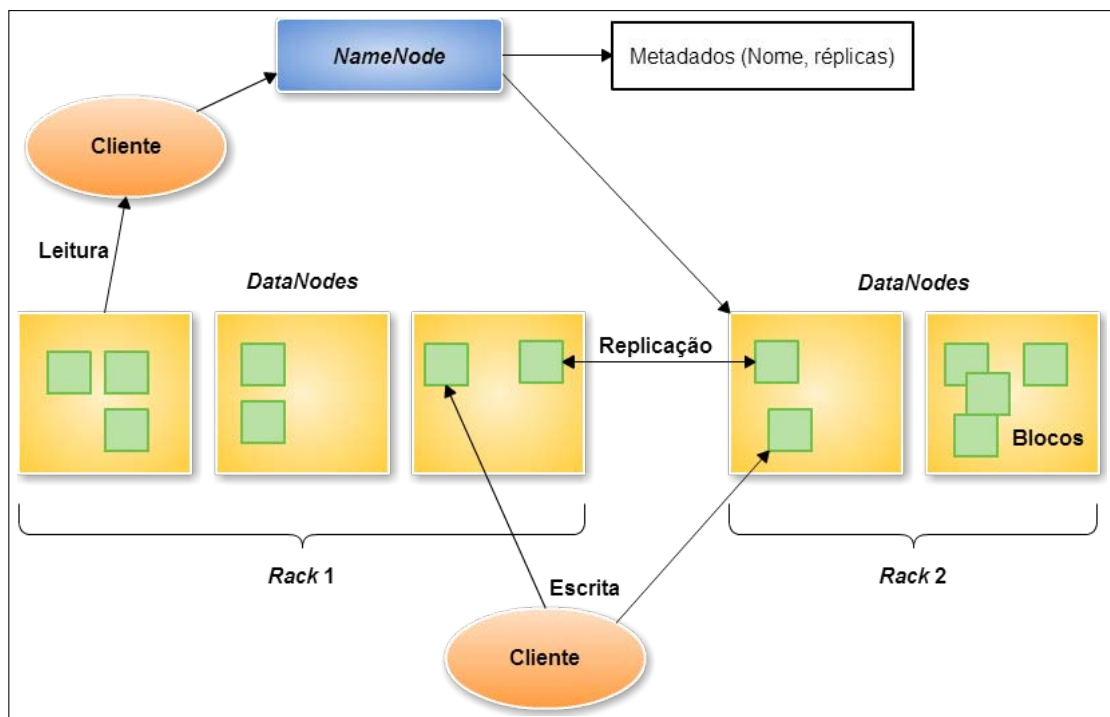
Figura 6: *Cluster* Hadoop em camadas

Figura 7: Estrutura do Sistema de arquivos distribuído do Hadoop (HDFS)



Fonte: Traduzida de (BORTHAKUR, 2007)

A aplicação cliente se comunica com o *Namenode* e faz a requisição dos metadados. Após receber os metadados, executa operações diretamente nos *Datanodes*. Se a operação for um *job* MapReduce, o cliente cria um *job* e o envia para uma fila. O *JobTracker* é que gerencia essa fila.

O *Namenode* mantém todos os metadados do sistema de arquivos. Esses metadados contêm o *namespace*, informação de controle de acesso, o mapeamento dos arquivos para os blocos e a localização atual dos blocos. O *Namenode* também executa operações como abrir, fechar e renomear arquivos e diretórios e envia instruções aos *Datanodes* para que executem operações de sistema, ex., criação, remoção e replicação de blocos.

Com base nos recursos do sistema e no tamanho do arquivo de entrada, o *Namenode* decide a quais *Datanodes* os clientes devem se conectar e fornece essa informação para o cliente.

Os *Datanodes* armazenam os dados do HDFS no sistema de arquivos local e executa as operações dos clientes ou operações do sistema a partir de instruções do *Namenode*. O HDFS é projetado para processar grandes quantidades de dados, portanto, os conjuntos de dados são divididos entre blocos com o mesmo tamanho, exceto o último bloco. Esses blocos são armazenados pelos *Datanodes* e replicados para tolerar falhas. Um *Blockreport* contém uma lista de todos os blocos em um *Datanode*. O tamanho padrão de um bloco do HDFS é 64MB (WHITE, 2012).

Um *TaskTracker* é um nó no *cluster* que aceita tarefas MapReduce do *JobTracker*. Cada *TaskTracker* é configurada com um conjunto de *slots* que indicam quantas tarefas o mesmo pode aceitar. Quando o processo é terminado, obtendo sucesso ou não, o *TaskTracker* notifica o *JobTracker*. O *TaskTracker* também envia mensagens à cada poucos minutos, para verificar se o *JobTracker* ainda está sendo executado. Estas mensagens também atualizam o *JobTracker* do número de *slots* ainda disponíveis.

De uma maneira geral, o funcionamento do *JobTracker* pode ser descrito pelos seguintes passos (WHITE, 2012):

- As aplicações cliente submetem *jobs* para o *JobTracker*.
- O *JobTracker* se comunica com o *Namenode* para determinar o local dos dados.
- O *JobTracker* aloca nós *TaskTracker* com *slots* disponíveis próximos ao local dos dados para reduzir o tráfego de rede.
- O *JobTracker* submete o *job* aos nós *TaskTracker* escolhidos.
- Os nós *TaskTracker* são monitorados através de sinais de vida enviados em intervalos de tempo. Quando uma tarefa falha, o *JobTracker* decide o que fazer em seguida: este pode reenviar o *job* para outros nós, marcar o registro específico como corrompido ou marcar o *TaskTracker* como não confiável.

- Quando o *job* é completado, o *JobTracker* atualiza seu status.
- A aplicação cliente pode fazer uma consulta ao *JobTracker* para obter informações.

A partir do conhecimento sobre o Hadoop e sua estrutura podem ser relacionadas algumas vantagens e desvantagens:

Vantagens do Hadoop

- Código aberto - O Hadoop é *Open Source* e há uma comunidade ativa a suportá-lo composta por organizações e programadores independentes que partilham seus conhecimentos, melhorias e documentação;
- Economia - Além de o Hadoop ser *Open Source* e livre de custos de licença, ele é baseado em *hardware* comum, logo, é possível realizar o processamento de dados utilizando computadores convencionais de baixo custo. O Hadoop também é oferecido como *Platform as a Service* (PaaS) em ambientes de computação em nuvem como a Amazon Elastic MapReduce (EMR) que permite a execução de aplicações Hadoop sem a necessidade de implantar o próprio *cluster* computacional;
- Robustez - O Hadoop fornece tolerância a falhas e alta disponibilidade através de mecanismos de replicação de dados. A fim de se proteger contra blocos corrompidos e falhas de *hardware*, cada bloco de dados é replicado para um número de máquinas fisicamente separadas. Se um bloco fica indisponível, uma cópia deste pode ser lida a partir de outro local de uma maneira transparente para o usuário. Ainda, o bloco que ficou indisponível pode ser novamente replicado a partir de suas localizações alternativas a fim de normalizar o fator de replicação;
- Escalabilidade - É possível aumentar rapidamente a quantidade de computadores utilizados no processamento em resposta ao aumento da quantidade de dados modificando apenas arquivos de configuração, sem a necessidade de alteração na codificação;
- Simplicidade - O Hadoop fornece o *framework* MapReduce para executar algoritmos paralelos com uma curva de aprendizado relativamente fácil. Ele permite que o programador mantenha o foco na abstração do problema, retirando do mesmo a responsabilidade de gerenciar questões relativas à computação paralela, tais como tolerância a falhas, escalonamento e balanceamento de carga.

Desvantagens do Hadoop

- Servidor Centralizado - O *Namenode* armazena todos os metadados e realiza todas as operações importantes do sistema de arquivos. Essa centralidade pode estabelecer limites de escalabilidade, disponibilidade e desempenho do sistema;
- Dificuldade de gerenciamento - É encontrada certa dificuldade no gerenciamento do *cluster* em operações como depuração e análise de *logs*;
- Modelo de programação restritivo - Problemas mais complexos podem ser difíceis de serem abstraídos utilizando somente as funções do MapReduce.

A próxima Sessão irá tratar sobre o MapReduce que é implementado no Hadoop como o modelo de análise de dados.

2.5.1 MapReduce

O MapReduce (DEAN; GHEMAWAT, 2004) é um modelo de programação proposto pela Google em 2004 para o processamento de grandes conjuntos de dados. Este é o *framework* aplicado no provedor de buscas na *Web* da Google. O MapReduce funciona basicamente dividindo o processamento em duas fases: *Map* e *Reduce* (WHITE, 2012). Para isso, o programador determina os passos da aplicação nas funções *map()* e *reduce()*. Desta forma, o sistema de execução irá automaticamente paralelizar a aplicação através do *cluster* de computadores. Essa abstração é inspirada nas primitivas *map* e *reduce* empregadas na linguagem Lisp e em outras linguagens de programação funcionais (DEAN; GHEMAWAT, 2004).

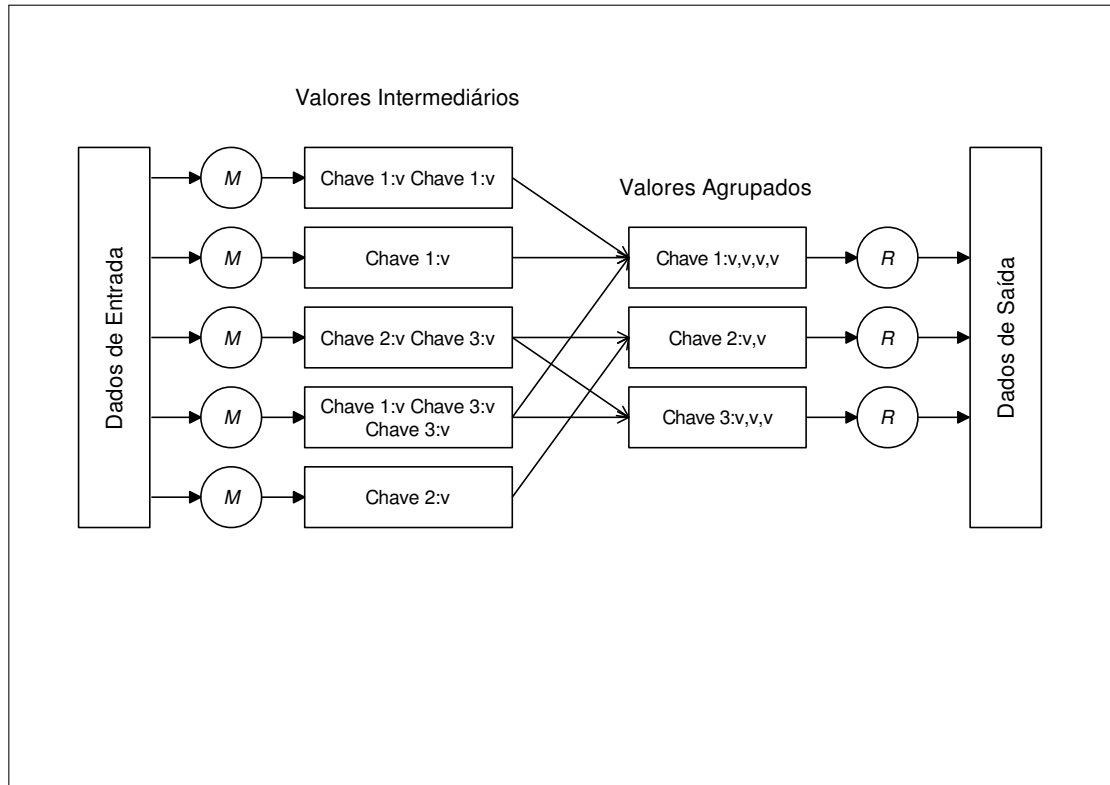
A fase *Map* processa um par de chave e valor e produz um conjunto de pares de chave e valor intermediários. O valor se refere aos dados relacionados a tarefa e a chave se refere ao número do conjunto ao qual o valor pertence. O MapReduce agrupa os valores intermediários associados à mesma chave intermediária e os envia para a função *reduce*. O Hadoop e o MapReduce produzem uma tarefa *Map* para cada parcela de dados em que o conjunto de entrada foi dividido. A fase *Reduce* processa os pares de chave e valor intermediários e agrupa todos os valores associados à mesma chave para obter conjuntos menores de valores.

Tarefas *Reduce* são separadas em três fases: Quando o processamento é feito em modo distribuído os arquivos de entrada são copiados para o sistema de arquivos local (fase *copy*). Após todos os dados estarem disponíveis localmente são associados a sua chave (fase *sort*). Então os dados são enviados a fase *reduce* que é a função opcional

escrita pelo programador que executa o último processamento nos dados para que enfim estes sejam escritos no HDFS.

As etapas de um processo MapReduce são apresentadas na Figura 8. Processos MapReduce podem ser executados repetidas vezes até que seja alcançado um critério de parada. As execuções de um processo e o próximo processo são feitas de maneira serial.

Figura 8: Fluxo de execução do MapReduce



Fonte: A autora

2.5.2 Frameworks para processamento distribuído de *Big Data*

Além do Hadoop, outros *frameworks* estão sendo desenvolvidos que implementam abstrações diferentes para o processamento de *Big Data*.

- Spark (ZAHARIA *et al.*, 2010): O Spark é um *framework* para computação em *cluster* semelhante ao Hadoop. A diferença é que o Spark é baseado no processamento de dados na memória e por isso pode apresentar um melhor desempenho em aplicações que compartilham dados entre diferentes processos. Por outro lado, abordagens baseadas em memória podem não escalar devido ao limite de capacidade das máquinas individuais. O Spark é uma proposta mais recente e novos trabalhos ainda são baseados no Hadoop.

- Trinity (SHAO; WANG; LI, 2013): O Trinity é um *framework* para o processamento de aplicações baseadas em grafos através de uma nuvem de memória distribuída. O Trinity organiza a memória de vários computadores em um espaço de endereçamento de memória distribuído globalmente endereçável. Dessa forma, é possível armazenar grandes grafos na memória de um *cluster* de computadores de baixo custo.

Exemplos de modelos alternativos ao MapReduce para processamento paralelo são:

- Pregel (MALEWICZ *et al.*, 2010): O Pregel foi criado para modelar problemas *Big Data* que podem ser expressos por meio de estruturas em grafos. O Pregel é baseado no modelo *Bulk Synchronous Parallel*. Os algoritmos são implementados iterativamente e cada iteração é chamada de superpasso. Durante um superpasso o *framework* invoca uma função definida pelo programador para cada vértice, em paralelo.
- PowerGraph (GONZALEZ *et al.*, 2012): Semelhante ao Pregel, o PowerGraph fornece um modelo alternativo de processamento por meio de grafos. A diferença é que o PowerGraph é totalmente assíncrono.

2.6 Trabalhos Relacionados

Em razão do aumento contínuo do tamanho dos conjuntos de dados a paralelização de técnicas de mineração de dados tem ganho atenção. Algumas implementações paralelas do algoritmo de agrupamento K-Means foram propostas:

O estudo de Kucukyilmaz (2014) se trata da implementação e na análise de um algoritmo K-Means paralelo para ser executado em multiprocessadores com memória compartilhada. Para maximizar o desempenho do K-Means paralelo, cada instância e cada grupo/centróide é atribuído a um dos processadores disponíveis. A fim de reduzir o tempo de espera quando os processadores são sincronizados foi assegurado um balanceamento da carga computacional mantendo tamanhos próximos do número de instâncias e centróides atribuídos a cada processador. Esta análise mostrou que em termos de complexidade computacional, o número de instâncias, número de grupos e a dimensionalidade da base de dados tem praticamente a mesma importância. Os resultados mostram que o algoritmo escala de forma semelhante variando tanto o número de grupos como o número de atributos.

Apesar dos bons resultados obtidos o uso de memória compartilhada pode apresentar limitações de escalabilidade impostas pelos limites físicos das máquinas utilizadas. O Hadoop é uma proposta que pode ser altamente escalável. Outro fator que é simplificado com o Hadoop é o balanceamento de carga que pode ser feito modificando um parâmetro de configuração eliminando do programador a preocupação apontada pelos autores.

Golghate e Shende (2014) apresentam uma implementação paralela do algoritmo K-Means baseado no *framework* Hama e fazem uma comparação com uma implementação semelhante baseada em Hadoop/MapReduce como a apresentada no presente trabalho. O Hama implementa o modelo de computação paralela BSP (*Bulk Synchronous Parallel*). Ambos os algoritmos foram executados em um computador *stand-alone* para o mesmo número de bases de dados. O algoritmo baseado no Hama obteve um melhor desempenho por não ser necessário submeter um novo *job* para uma nova iteração como no MapReduce e o superpasso no BSP é menos custoso que executar um *job* do MapReduce.

Apesar de os autores apontarem que submeter um *job* MapReduce consome um tempo maior que um superpasso do modelo BSP, a diferença entre os desempenhos obtidos foi pequena. O K-Means pode ser implementado com o modelo MapReduce que possui um nível maior de abstração simplificando a codificação e ainda assim conseguir um bom desempenho.

Semelhante ao presente trabalho, Zhou, Lei e Ye (2011) implementaram o algoritmo K-Means paralelo baseado no MapReduce para ser executado em um *cluster* Hadoop. Os autores demonstraram a eficiência e acurácia do algoritmo na classificação de documentos em larga escala. Os resultados experimentais mostram a melhoria no tempo de resposta na execução em 10 nós em comparação com a execução em 1 único nó usando o mesmo número de documentos e a melhoria linear no tempo de resposta conforme mais nós são adicionados no *cluster*. Os resultados mostram que a classificação foi feita de maneira satisfatória, com precisão média de 89%.

O algoritmo K-Means paralelo apresentado por Ramesh, Ramar e Babu (2013) foi baseado no modelo de programação em memória distribuída (SPMD) utilizando a linguagem Java e o padrão de passagem de mensagem (*Message Passing Interface* - MPI). No sistema proposto o conjunto de dados a ser agrupado foi dividido entre vários processos sendo que o centróide inicial foi selecionado por um único nó e enviado por *broadcast* para todo o sistema. Cada processo opera em sua parte correspondente do conjunto de dados, calcula a distância entre os pontos e os centróides atribuindo os pontos ao centróide mais próximo. Cada processo também calcula a soma parcial dos pontos atribuídos a cada grupo

na sua parte dos dados. Ao final da iteração, uma operação de redução global é realizada a fim de calcular os novos centróides. O desempenho do algoritmo K-Means paralelo foi validado agrupando conjuntos de dados de solo. O algoritmo K-Means paralelo teve um melhor desempenho em comparação com o algoritmo equivalente sequencial.

Kumar *et al.* (2011) implementaram o algoritmo K-Means paralelo para o delineamento de regiões ecológicas usando grandes bases de dados. A implementação foi feita na linguagem C usando o padrão MPI. O desempenho do algoritmo foi avaliado usando 3 bases de dados de tamanhos diferentes para diferentes números de grupos a partir de experimentos executados no supercomputador Jaguar Cray XT5. Os autores apontaram observações interessantes. Foi identificado que a leitura e escrita de arquivo é intensificada com o aumento do número de processadores utilizados e é gasto um tempo maior nestas tarefas. No entanto, o tempo gasto para os cálculos das distâncias é reduzido significativamente com a computação distribuída entre um alto número de processadores, diminuindo o tempo total para a solução do algoritmo.

Usar o MapReduce para modelar o K-Means é mais prático comparado ao modelo MPI citado em Kumar *et al.* (2011) e Ramesh, Ramar e Babu (2013) visto que o MPI requer mais mudanças na codificação para adaptar a execução sequencial para a execução paralela que o MapReduce.

3 MATERIAIS E MÉTODOS

3.1 Algoritmo K-Means com base em MapReduce

Neste trabalho uma versão paralela do algoritmo K-Means com base no modelo MapReduce foi analisada a partir de experimentos executados em um *cluster* Hadoop. O algoritmo foi organizado em 4 partes: fase *Map*, *Combine* e *Reduce* e um código para submeter o *job* MapReduce. O *Combiner* foi utilizado para antecipar parte do processamento feito pelo *Reducer* e assim reduzir o tráfego de rede.

Para cada iteração do K-Means é submetido um novo *job* MapReduce. Para tanto, o *job* é invocado em um laço que é executado até que a convergência seja atingida ou até que se atinja o número máximo de iterações. As fases *Map*, *Combine* e *Reduce* são executadas a cada iteração e têm como entrada e saída pares chave/valor.

Os experimentos foram executados em um *cluster* Hadoop formado por seis máquinas sendo uma delas mestre e trabalhadora e as demais trabalhadoras. Em todas as máquinas foi executado o Sistema operacional Linux, a versão 1.2.1 do Hadoop e a mesma versão do Java. As máquinas usadas tem configurações equivalentes, cada uma com um processador AMD Phenom(tm) II X4 B97 com 4 núcleos e 7,4 GB de memória principal.

A configuração do *cluster* Hadoop foi feita por arquivos de configuração localizados no diretório de instalação do Hadoop. Os arquivos e seus parâmetros que foram modificados são listados a seguir:

- *masters*: Neste arquivo foi inserido o nome de *host* da máquina mestre. A máquina designada executa o *secondaryNameNode*;
- *slaves*: Neste arquivo foram inseridos os nomes de *host* das máquinas trabalhadoras. As máquinas indicadas executam os *daemons* *Datanode* e *TaskTracker*;
- *core-site.xml*: Neste arquivo foi inserido o parâmetro *fs.default.name*. Seu valor é o endereço URI da máquina mestre que executa o *Namenode*;
- *mapred-site.xml*: O parâmetro modificado neste arquivo foi o *mapred.job.tracker* e seu valor é o nome de *host* da máquina mestre que executa o *daemon* *JobTracker*;
- *hdfs-site.xml*: Neste arquivo foi modificado o parâmetro *dfs.replication* referente a configuração do fator de replicação do Hadoop. Seu valor foi configurado para 3 que é o padrão do Hadoop.

Os arquivos *masters* e *slaves* foram modificados apenas na máquina mestre, que é de onde são lançados os *jobs*. Os demais arquivos foram modificados em todos os nós. Exemplos desses arquivos são encontrados no Apêndice A.

3.2 Bases de dados utilizadas

O desempenho do K-Means paralelo foi investigado a partir de experimentos usando bases de dados de medições de torres de fluxo de diferentes regiões agrícolas.

O fluxo pode ser definido como a quantidade de uma grandeza que passa através de uma superfície por unidade de tempo. Os dados de fluxo são obtidos utilizando a técnica de covariância de vórtices turbulentos ou *Eddy Covariance* (EC). O princípio geral das medições de EC é a covariância entre a concentração da grandeza de interesse e a velocidade vertical do vento (BURBA, 2013).

Atualmente mais de 500 torres de fluxo estão sendo operadas em todo o mundo. Dados de longo prazo podem ser obtidos em conjuntos de dados consistentes e documentados disponibilizados na Internet, o que facilita estudos com o objetivo de entender os controles sobre os fluxos de carbono.

As medições dos fluxos de carbono são relevantes não somente para estudos em mudanças climáticas mas são também fatores chave na produtividade agrícola sustentável, biodiversidade e processos ecossistêmicos e florestais. A seguir são listadas algumas finalidades dessas medições (CIHLAR; DENNING; GOSZ, 2002):

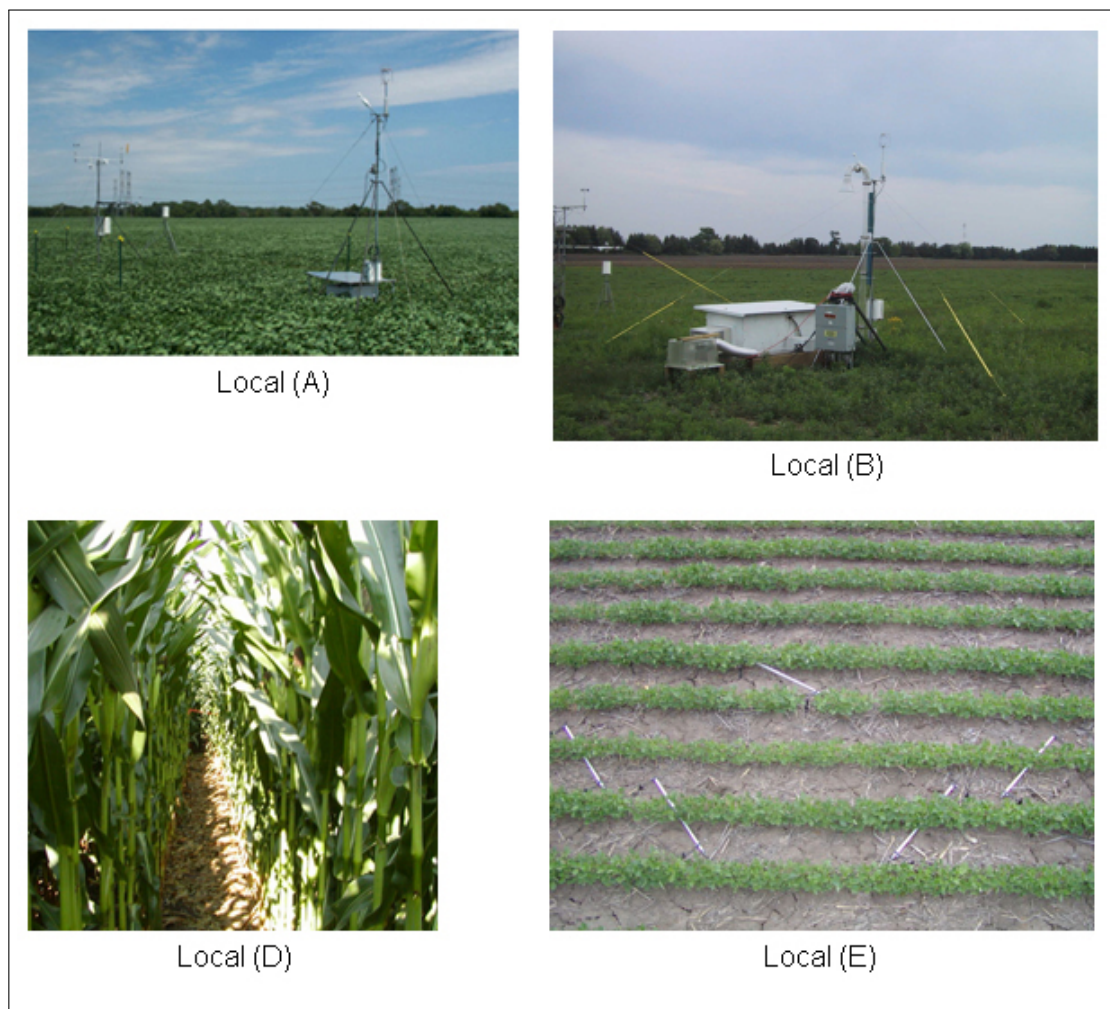
- Entender o ciclo global do carbono para: identificar fontes e sumidouros⁵ de carbono e suas variações através do tempo e prever como podem mudar no futuro;
- Criar indicadores sobre o atual estado do sistema climático global;
- Avaliar as mudanças em nível regional e global no ciclo de carbono e seus impactos;
- Suporte a tomada de decisões economicamente mais eficazes e a avaliação de suas consequências;
- Prover informações para desenvolver melhores estratégias mitigatórias e adaptativas;
- Prover informações para finalidades gerais públicas, educacionais, etc.

⁵Sumidouro - Os sumidouros realizam o sequestro de carbono, retirando da atmosfera mais carbono do que emitem. São exemplos as florestas e os oceanos.

Os dados utilizados nos experimentos pertencem a rede AmeriFlux⁶ e foram obtidos em (<http://ameriflux.ornl.gov/>). A rede AmeriFlux é um conjunto de torres de fluxo localizadas principalmente nos EUA, mas com locais no Canadá, América Central e América do Sul. A rede cobre uma ampla variedade de tipos de ecossistemas, incluindo florestas, pradarias, áreas agrícolas, vegetação rasteira, zonas úmidas, savanas e outros (BALDOCCHI, 2003).

Os dados foram extraídos dos locais: (A) Fermi Agricultural / US-IB1 na latitude 41.8593 e longitude -88.2227, (B) Fermi Prairie / US-IB2 na latitude 41.8406 e longitude -88.2410, (C) Mead Irrigated / US-Ne1 na latitude 41.1650 e longitude -96.4766, (D) Mead Irrigated Rotation / US-Ne2 na latitude 41.1649 e longitude -96.4701 e (E) Mead Rainfed / US-Ne3 na latitude 41.1797 e -96.4396 que ficam no centro-norte dos Estados Unidos. A Figura 9 apresenta imagens dos locais estudados.

Figura 9: Imagens dos locais de estudo



Fonte: AmeriFlux

⁶AmeriFlux - Rede de torres de fluxo das Américas

O local (A) se trata de uma região agrícola atualmente cultivada com a rotação das culturas de trigo e soja, o local (B) é uma pradaria recuperada que já foi usada para plantio agrícola, o local (C) é uma área agrícola com cultivo contínuo de milho utilizando a técnica agrícola de sequeiro e os locais (D) e (E) são áreas com a rotação das culturas de milho e soja utilizando agricultura irrigada.

Foram utilizados dados de medições de fluxo do período de 2004 à 2011 totalizando 455.712 linhas de registros. Cada base de dados contendo 34 atributos que estão descritos na Tabela 1.

Tabela 1: Atributos da base de dados

Atributo	Medição	Descrição
YEAR	-	Ano
DOY	-	Dia do ano
HRMIN	-	Hora e minuto
UST	m/s	Força de atrito
TA	deg C	Temperatura do Ar
WD	deg	Direção do vento
WS	m/s	Velocidade do vento
NEE	umol/m2/s	Trocas líquidas do ecossistema (medida como $FC + SFC$)
FC	umol/m2/s	Fluxo de CO ₂
SFC	umol/m2/s	Fluxo de armazenamento de CO ₂
H	W/m2	Fluxo de calor sensível
LE	W/m2	Fluxo de calor latente
FG	W/m2	Fluxo de calor do solo
TS1	deg C	Temperatura do solo na profundidade 1
TSdepth1	cm	Profundidade do solo 1
TS2	deg C	Temperatura do solo na profundidade 2
TSdepth2	cmm	Profundidade do solo 2
PREC	mm	Precipitação
RH	%	Umidade relativa do ar
PRESS	kPa	Pressão barométrica (medida como $RE - NEE$)
CO2	umol/mol	Concentração de CO ₂
VPD	kPa	Déficit de pressão de vapor
SWC1	%	Umidade volumétrica do solo na profundidade 1
SWC2	%	Umidade volumétrica do solo na profundidade 2
Rn	W/m2	Saldo de radiação
PAR	umol/m2/s	Radiação fotossinteticamente ativa emergente
Rg	W/m2	Radiação solar global emergente
PARout	umol/m2/s	Radiação fotossinteticamente ativa incidente
RgOut	W/m2	Radiação solar global incidente
CO2top	umol/mol	Concentração de CO ₂ no topo do dossel
CO2height	m	Altura do CO ₂
APAR	umol/m2/s	Radiação fotossinteticamente ativa absorvida pelo dossel
APARpct	%	Porcentagem de radiação fotossinteticamente ativa absorvida
ZL	-	Parâmetro de estabilidade atmosférica

As medições que foram feitas porém foram consideradas corrompidas ou errôneas foram substituídas pelo valor -9999. Já as ocorrências em que as medidas não foram feitas foram substituídas pelo valor -6999.

3.3 Avaliação do Desempenho

A avaliação do desempenho do algoritmo K-Means paralelo foi feita com relação ao *SpeedUp* e *ScaleUp* a partir de experimentos variando o número de máquinas utilizadas no processamento. Para medir o *SpeedUp*, foram executados 4 testes aumentando o número de máquinas disponíveis no sistema e para todos foram usadas as mesmas bases de dados. A primeira execução foi feita localmente em modo *single-node* (Com único nó). Para assegurar que o programa fosse executado de forma completamente sequencial foi definido na codificação para ser executada apenas 1 tarefa *Map*.

Os testes seguintes foram executados em *cluster* usando a partir de 3 nós. Não foi considerado usar apenas 2 nós pois o Hadoop usa um fator de replicação, que por padrão é igual a 3. Caso o número de nós seja menor que o fator de replicação o Hadoop considera os blocos como sub-replicados e tenta constantemente convertê-los para o fator esperado, o que pode afetar o desempenho do Hadoop. Nestes testes não foi definido o número de tarefas *Map* de forma que estas fossem criadas automaticamente pelo Hadoop conforme o tamanho da base de dados e a quantidade de recursos disponíveis. Os testes foram executados com 3, 4 e 6 máquinas, respectivamente.

Para que os dados de entrada sejam divididos entre os *Datanodes* é necessário adicioná-los ao sistema de arquivos distribuído do Hadoop. Então, antes de iniciar a execução dos *jobs* foram copiados os arquivos utilizados para diretórios do HDFS. Esses diretórios são passados como parâmetro para o programa.

Para medir o *ScaleUp*, os testes foram executados aumentando proporcionalmente o tamanho das bases de dados e o número de computadores no sistema. O *ScaleUp* demonstra a habilidade de um sistema n vezes maior executar uma tarefa igualmente maior em um tempo equivalente.

4 RESULTADOS

4.1 K-Means Paralelo

Nesta Seção será detalhada a implementação do algoritmo K-Means paralelo que foi baseado no MapReduce. A implementação foi separada em quatro partes: a primeira para configurar e submeter o *job*, e as funções *Map*, *Combine* e *Reduce*.

As partições de dados foram distribuídas entre as máquinas escravas no *cluster*. Cada máquina executa *Mappers* que têm a função de computar k grupos para cada amostra na sua partição e encontrar o centróide mais próximo: cada amostra é associada a uma chave referente ao grupo com centróide mais próximo. Essa chave e os valores na instância são passados ao *Combiner* que é executado no mesmo nó do *Mapper* e faz o agrupamento das amostras associadas ao mesmo grupo no *Mapper* respectivo. Ao final da execução todos os *Combiners*, um *Reducer* faz o agrupamento total das amostras atribuídas ao mesmo grupo nas fases anteriores, calcula os novos centróides e emite os novos centróides juntamente com o número de amostras atribuídas ao mesmo grupo. Os grupos resultantes serão usados na próxima iteração do K-Means. Ao final de cada iteração, no *Reducer*, os novos centróides são comparados aos anteriores. Se os centróides não mudaram é atingida a convergência e a execução do programa é finalizada. O processo MapReduce para o K-Means é ilustrado na Figura 10.

A implementação da função *map* encontra-se no Código 4.1. A função *map* tem como entrada um conjunto de pares chave/valor onde chave é o índice da amostra na base de dados e valor é uma *String* formada pelos valores da amostra.

Na linhas 2 e 3 foram inicializadas as variáveis auxiliares *minDis* para armazenar a mínima distância calculada e *closestCenter* para armazenar o índice do centróide mais próximo. Para cada k grupo é calculada a distância euclidiana entre o centróide e a amostra a fim de descobrir o centróide mais próximo.

A função *combine* tem como entrada o conjunto de pares chave/valor retornados na função *map*. Nela serão agrupadas as amostras atribuídas ao mesmo centróide. A função *combine* é executada no mesmo nó da função *map* respectiva. Sua implementação encontra-se no Código 4.2.

O par chave/valor retornado pela função *combine* contém o índice do centróide e a String formada pelos valores das amostras atribuídas ao mesmo grupo e o número total de amostras atribuídas ao mesmo grupo.

Código 4.1: map (chave, valor)

```

1 Constrói o ponto a partir de "valor" contendo todos os valores da amostra;
2 double minDis = Double.MAX_VALUE;
3 int closestCenter = -1;
4
5 Para i = 0 até centroides.size() faça
6     double dis = calcDist(ponto, centroides.get(i));
7     if (dis < minDis) {
8         minDis = dis;
9         closestCenter = i;
10    }
11 fim para
12
13 Toma chave como closestCenter;
14 Toma valor como ponto;
15 Emite par chave/valor com o índice do grupo com centróide mais próximo e os
    valores da amostra atribuída ao grupo;

```

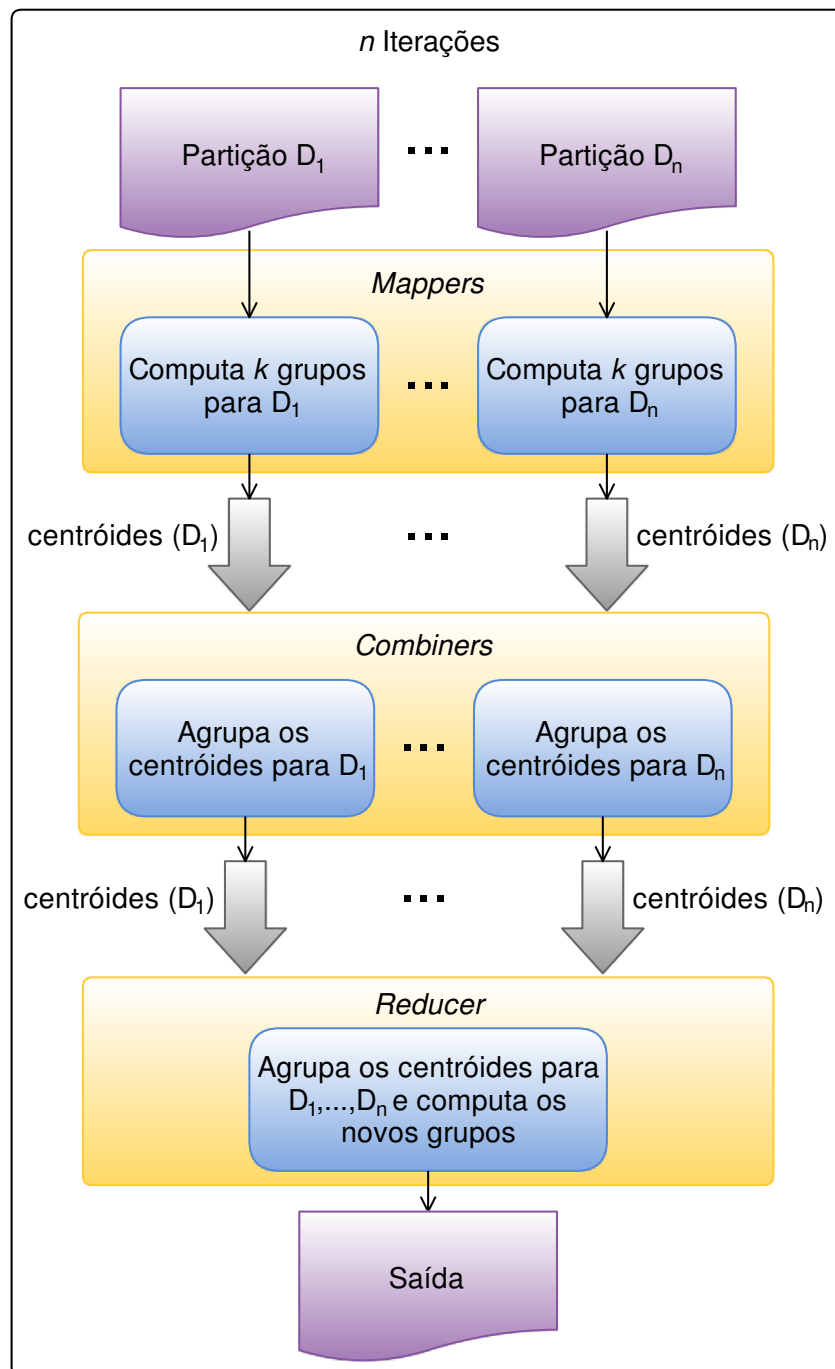
Código 4.2: combine (chave, valores)

```

1 Constrói o ponto "sum" para armazenar a soma dos valores das amostras;
2 Inicializa num = 0 para armazenar o número de amostras atribuídas ao mesmo
    centro;
3
4 while (valores.hasNext()) {
5     Constrói a amostra a partir de valores.next();
6     Adiciona os valores da amostra à "sum"
7     num++;
8 }
9
10 Toma chave como chave;
11 String valor = sum.toString() + ":" + String.valueOf(num);
12 Emite par chave/valor;

```

Figura 10: Processo MapReduce para o K-Means



Fonte: A autora

A função *reduce* é executada ao término das etapas anteriores e tem como entrada um conjunto de pares chave/valor retornados pelos *Combiners*.

O par chave/valor retornado pela função *reduce* contém o índice do centróide e

Código 4.3: reduce (chave, valores)

```

1  Constrói o ponto "sum" para armazenar a soma dos valores das amostras;
2  Inicializa NUM = 0 para armazenar o número total de amostras atribuídas ao
    mesmo centro;
3
4  while (valores.hasNext()) {
5      Constrói a amostra a partir de valores.next();
6      Adiciona os valores da amostra à "sum"
7      NUM += num;
8  }
9
10 Divide as entradas em "sum" por NUM para obter os novos centróides;
11 Toma chave como chave;
12 Constrói a String com os valores dos novos centróides e NUM;
13 Emite par chave/valor;
14 Se os novos centróides forem iguais aos anteriores
15     Incrementa o Counter para finalizar a execução;

```

a *String* formada pelos valores dos novos centróides juntamente com o número total de amostras atribuídas ao mesmo grupo.

Para verificar a convergência do algoritmo foi utilizado um contador enumerado⁷ denominado *Counter* que acompanha o progresso de um *job* map/reduce. Se a convergência tiver sido atingida o *Counter* é incrementado na fase *reduce*. Se o *Counter* não for incrementado retornará o valor 0. As iterações são repetidas até que o *Counter* retorne um valor diferente de 0.

4.2 Análise dos Tempos de Execução

Esta Seção mostra os experimentos executados para avaliar o K-Means paralelo apresentado neste trabalho. Verificou-se o ganho de desempenho (*SpeedUp*) que o algoritmo pode obter adicionando máquinas em um *cluster* Hadoop e como o algoritmo escala com o aumento equivalente do tamanho dos dados de entrada e da quantidade de máquinas no *cluster* (*ScaleUp*). Para medir o *SpeedUp* foram executados experimentos no modo *single-node* e aumentando o número de nós do *cluster*.

O modo *single-node* foi feito de maneira pseudo-distribuída sendo que os *daemons* do Hadoop foram executados em uma única máquina. Os demais testes foram feitos de maneira totalmente distribuída aumentando o número de nós no *cluster*. Para todas as execuções o número máximo de iterações do K-Means foi limitado para 50.

As Figuras 11 e 12 foram extraídas da interface *web* de administração do Map-

⁷Enumerações - Estruturas de dados que armazenam uma coleção de valores fixos predefinidos e imutáveis

Reduce. A primeira Figura mostra um gráfico das tarefas *Map* que foram completadas para uma iteração. O eixo Y indica a porcentagem de conclusão das tarefas e o eixo X indica o número da tarefa *Map* atual. Verificou-se que o Hadoop criou uma tarefa *Map* para cada arquivo analisado (39 tarefas) pois cada arquivo não chegou a armazenar 64MB que é o tamanho padrão dos blocos do Hadoop. Caso algum dos arquivos ultrapassasse esse tamanho o mesmo seria dividido e seria criada mais uma tarefa *Map*. A segunda Figura mostra o estado das tarefas *Reduce* que são separadas de acordo com as suas fases *copy* (identificada pela cor verde), *sort* (em vermelho), e *reduce* (em azul). Foi executada apenas uma tarefa *Reduce* visto que as iterações do K-Means dependem dos grupos encontrados na iteração anterior.

Figura 11: Gráfico de conclusão de tarefas *Map*

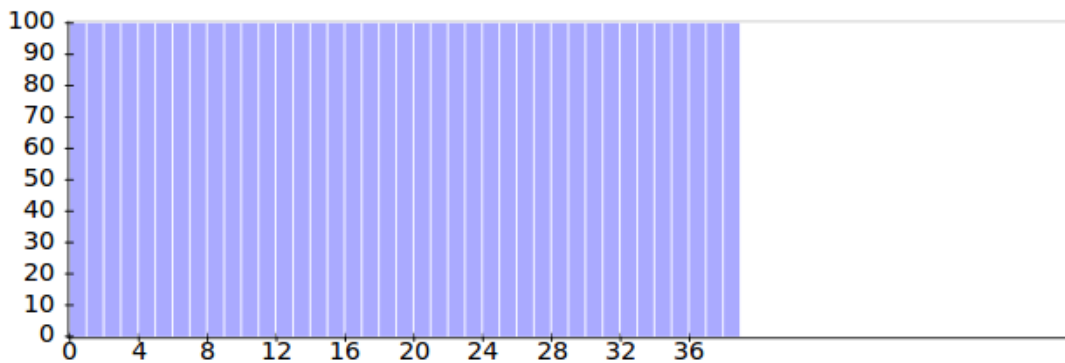


Figura 12: Gráfico de conclusão de tarefas *Reduce*



Ainda nessa interface é possível verificar o estado do *cluster* como os *daemons* que estão sendo executados em cada máquina e é possível analisar individualmente os *jobs* por meio de *logs* que mostram informações como o tempo que levaram para serem executados.

A Tabela 2 mostra os tempos de execução dos experimentos, o *SpeedUp* que foi obtido distribuindo o processamento e a Eficiência paralela.

Para apoiar nossos resultados foi construída uma tabela ANOVA (Tabela 5) com

Tabela 2: Resumo dos resultados do *SpeedUp*

	<i>single-node</i>	3 nós	4 nós	6 nós
Tempo de execução:	72,14	35,11	29,29	22,14
<i>SpeedUp</i>		2,054	2,46	3,25
Eficiência		0,68	0,61	0,54

um fator. A Análise de Variância ANOVA é uma técnica para comparar mais que dois experimentos. Neste trabalho foram comparadas as médias dos tempos de execução do K-Means. Para isso foram executadas três repetições com cada número de máquinas que foram 3, 4 e 6 máquinas, respectivamente. Foi considerado 95% de grau de confiança ($\alpha = 0,05$). A Tabela 3 contém os tempos de execução de cada repetição.

Tabela 3: Tempos de execução: repetição 1, repetição 2, repetição 3

Nº de nós:	3	4	6
repetição 1	35,11	29,29	22,14
repetição 2	34,52	29,32	22,03
repetição 3	34,54	29,44	22,16

Na Tabela 4 é mostrado o resumo da ANOVA com a soma, média e a variância entre os valores de cada grupo.

Tabela 4: Tabela Resumo da ANOVA

Nº de nós	Nº de repetições	Soma	Média	Variância
3	3	104,17	34,72	0,1122
4	3	88,05	29,35	0,0063
6	3	66,33	22,11	0,0049

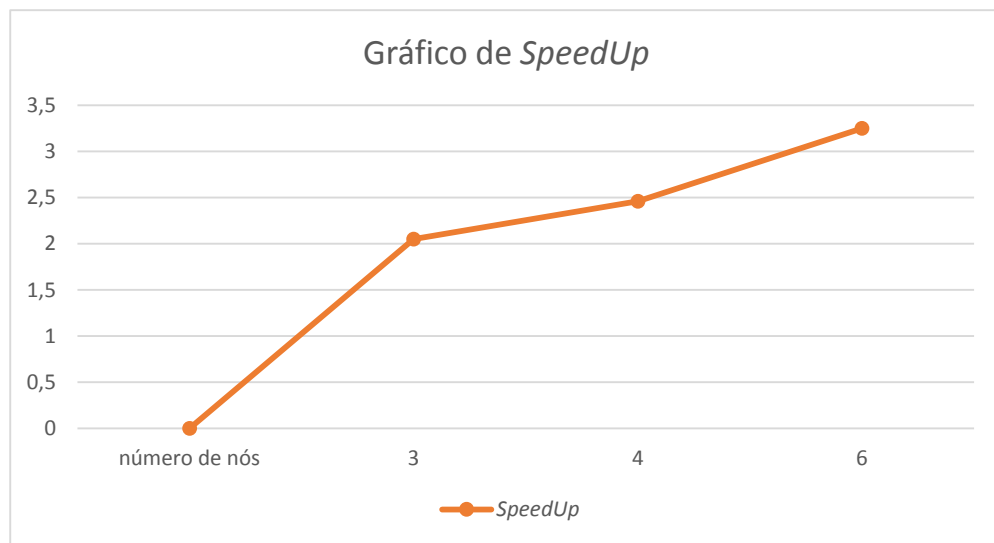
Tabela 5: Tabela ANOVA

<i>Fonte da variação</i>	<i>SQ</i>	<i>gl</i>	<i>MQ</i>	<i>F</i>	<i>valor-P</i>	<i>F crítico</i>
Entre grupos	240,3864	2	120,1932	2921,2508	0,00000000107974	5,1432
Dentro dos grupos	0,2468	6	0,04114			
Total	240,6333	8				

Os resultados da análise ANOVA mostram que há diferença significativa entre os grupos com diferentes números de máquinas pois o valor-P é menor que 10^{-4} .

Com o aumento de computadores no *cluster* o *SpeedUp* aumentou de forma praticamente linear (Figura 13). Foi obtido um menor tempo de resposta usando 6 máquinas chegando a 3,25 de *SpeedUp*. Foi observado a diminuição da eficiência paralela com o aumento de máquinas no *cluster* relacionada ao aumento no custo de comunicação.

Figura 13: Gráfico de *SpeedUp*

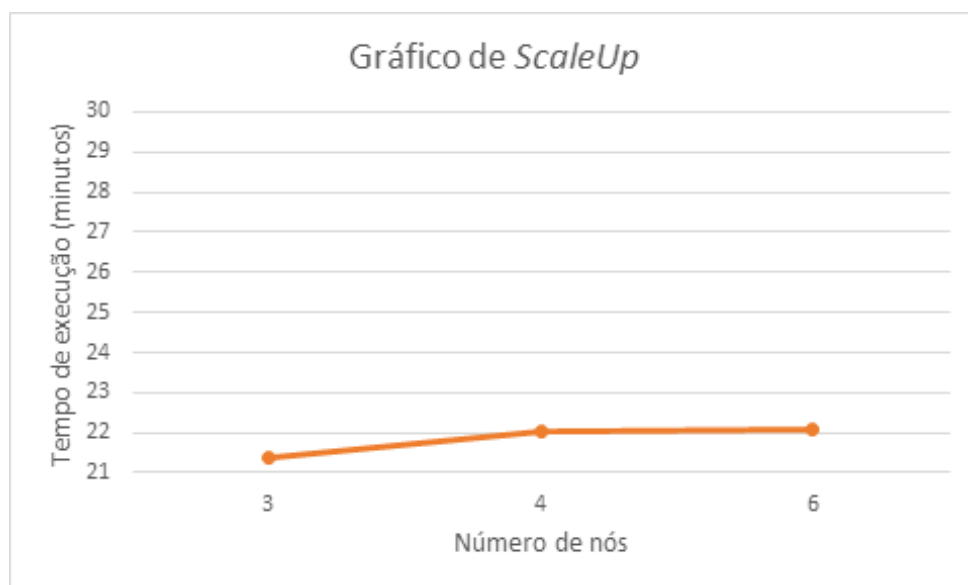


A Tabela 6 mostra os tempos de execução obtidos aumentando o número de máquinas e o tamanho das bases de dados.

Tabela 6: Resumo dos resultados do *ScaleUp*

Nº de nós	Nº de tarefas <i>Map</i>	Tamanho da base de dados	Tempo de execução
3	13	30 MB	21,38
4	18	40 MB	22,04
6	32	60 MB	22,06

Os resultados mostraram que é possível aumentar o número de máquinas no *cluster* Hadoop em resposta ao crescimento das bases de dados e obter desempenho computacional equivalente. Os tempos de resposta das execuções ficaram muito próximos como pode ser visualizado na Figura 14 então o Hadoop foi capaz de escalar bem com a configuração utilizada.

Figura 14: Gráfico de *ScaleUp*

5 CONCLUSÕES

Neste trabalho foi apresentada uma implementação paralela detalhada do algoritmo de agrupamento K-Means usando o modelo MapReduce. O seu desempenho foi investigado a partir de experimentos executados em um *cluster* Hadoop usando conjuntos de dados de medições de fluxo de regiões agrícolas.

O K-Means paralelo foi avaliado com relação ao ganho de desempenho (*SpeedUp*) e escalabilidade (*ScaleUp*). Verificou-se que o tempo para a execução das iterações do K-Means diminuiu com o aumento do número de máquinas no *cluster* apresentando um *SpeedUp* praticamente linear. O melhor tempo obtido foi usando seis máquinas, chegando ao *SpeedUp* de 3,25. Para suportar os resultados foram executadas repetições variando o número de máquinas no *cluster* e a partir dos grupos foi aplicada a análise de variância ANOVA. A Tabela ANOVA mostrou que o modelo é significativo visto que o valor-P foi menor que 10^{-4} . No experimento para verificar o *ScaleUp* se obteve tempos de resposta bem próximos aumentando proporcionalmente o tamanho das bases de dados e o número de máquinas no *cluster* sendo que os tempos variaram em poucos segundos de 21,38 até 22,06. Logo o algoritmo foi capaz de escalar bem com a configuração do *cluster* utilizada.

Alguns aspectos sobre Hadoop e MapReduce foram observados no desenvolvimento do trabalho:

- O Hadoop apresenta vantagens econômicas pois é baseado em computadores com *hardware* acessível, é um projeto de código aberto e pode ser contratado como serviço em ambientes de Computação em Nuvem não sendo necessário manter o próprio *cluster*;
- Foram poucas as dificuldades encontradas em usar o Hadoop e o MapReduce visto que não é necessário se preocupar em codificar os passos envolvidos em um programa paralelo, tais como de dividir os dados em um número de partes compatível com o número de processadores disponíveis, dividir a computação em pequenas computações, distribuí-las entre as máquinas e combinar os dados de saída em cada computação. O Hadoop automaticamente gerencia a tarefa de particionar e distribuir os dados, bastando copiá-los para seu sistema de arquivos distribuído e o *cluster* pode ser escalado modificando apenas um arquivo de configuração. No entanto, a configuração do *cluster* Hadoop é criteriosa. Para que a comunicação ocorra todas as máquinas usadas no *cluster* precisam ter uma relação de confiança através do serviço SSH, então, foi importante garantir que os nomes de *host* das máquinas não

fossem alterados e nem seus endereços IP se não as trocas de chaves teriam que ser feitas novamente;

- O Hadoop fornece interfaces *web* que facilitam o monitoramento do *cluster* sendo possível verificar o estado das tarefas *Map* e *Reduce* e analisar informações sobre os *jobs* MapReduce;
- O K-Means pôde ser facilmente paralelizado usando o MapReduce porém a implementação de algoritmos com partes difíceis de serem executadas em paralelo podem exigir um modelo menos restritivo com mais baixo nível de abstração.

Os resultados obtidos ficaram conforme esperados: Esse trabalho apresentou uma implementação paralela escalável do K-Means baseada no modelo MapReduce que pode ser executada em um *cluster* Hadoop e melhorar o tempo de resposta do agrupamento de grandes bases de dados. Estes resultados podem apoiar aplicações não só na área agrícola mas em diferentes domínios.

Durante o desenvolvimento do trabalho foram encontrados vários estudos na área agrícola utilizando mineração de dados em larga escala derivados de diferentes fontes de dados. Contudo, estudos de soluções *Big Data* são pouco exploradas com bases de dados agrícolas. Isso pode estar relacionado a baixa taxa de atualização de conjuntos de dados disponíveis e a alta ocorrência de perdas (*gaps*) nos dados. Dessa maneira, propõe-se que em trabalhos futuros os experimentos sejam feitos usando bases de dados que contenham períodos mais longos de observações.

Podem ser relacionados, ainda, os trabalhos futuros a seguir:

- Executar experimentos em plataformas de Nuvem tais como a Microsoft Azure, Google Cloud Platform e Amazon Elastic MapReduce (EMR) que executam Hadoop, para avaliar o desempenho em escalas maiores;
- Explorar outros modelos e soluções *Big Data* e verificar se existe ganho de desempenho sobre o Hadoop e o MapReduce;
- Uma limitação do K-Means é que o usuário deve determinar k (o número de grupos a serem gerados) e o algoritmo é sensível aos valores iniciais: valores iniciais diferentes podem gerar resultados diferentes. O K-Means poderá ser combinado com técnicas de pré-processamento presentes na literatura para determinar os k grupos iniciais automaticamente. Pode ser usada a técnica *Canopy* citada no trabalho.

REFERÊNCIAS

- ALMASI, G. S.; GOTTLIEB, A. *Highly parallel computing*. [S.l.]: Benjamin-Cummings Publishing Co., 1989.
- AMINI, M. *et al.* Agricultural development in iran base on cloud computing theory. *International Journal of Engineering Research & Technology (IJERT)*, v. 2, p. 796–801, 2013.
- BALDOCCHI, D. D. Assessing the eddy covariance technique for evaluating carbon dioxide exchange rates of ecosystems: past, present and future. *Global Change Biol.*, v. 9, p. 479–492, 2003.
- BEKKERMAN, R.; BILENKO, M.; LANGFORD, J. *Scaling Up Machine Learning: Parallel and Distributed Approaches*. [S.l.]: Cambridge University Press, 2012.
- BHoola, K. *et al.* Big data analytics. In: *Actuarial Society of South Africa's 2014 Convention*. [S.l.: s.n.], 2014.
- BORTOLOSSI, H. J. *Cálculo Diferencial a Várias Variáveis*. [S.l.]: Edições Loyola, 2002. 619 p.
- BURBA, G. *Eddy Covariance Method for Scientific, Industrial, Agricultural and Regulatory Applications: A Field Book on Measuring Ecosystem Gas Exchange and Areal Emission Rates*. [S.l.]: LI-COR Biosciences, 2013. 343 p.
- BUYYA, R. Parallel programming models and paradigms. *High Performance Cluster Computing*, v. 1, p. 4–26, 1999.
- CIHLAR, J.; DENNING, A. S.; GOSZ, J. R. *Terrestrial Carbon Observation: The Ottawa Assessment of Requirements, Status and Next Steps*. [S.l.]: Food & Agriculture Org., 2002. (Environment and natural resources series, 2).
- COULOURIS, G. *et al.* *Sistemas Distribuídos: Conceitos e Projeto*. 5. ed. [S.l.]: Bookman, 2013.
- DAVIES, D. L.; BOULDIN, D. W. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1, p. 224 – 227, 1979.
- DEAN, J. *Big Data, Data Mining, and Machine Learning: Value Creation for Business Leaders and Practitioners*. [S.l.]: John Wiley & Sons, 2014. 288 p.
- DEAN, J.; GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. In: *In Proc. of the 6th Symposium on Operating Systems Design and Implementation*. [S.l.: s.n.], 2004.
- DROZDOWSKI, M. *Scheduling for Parallel Processing*. [S.l.]: Springer Science & Business Media, 2010. 386 p.

- DUNN, J. Well separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, p. 95–104, 1974.
- FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery in databases. *American Association for Artificial Intelligence*, v. 17, n. 3, p. 37–54, 1996.
- FLYNN, M. J.; RUDD, K. W. Parallel architectures. *ACM Computing Surveys*, v. 28, n. 1, p. 67–70, 1996.
- FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. *Intl J. Supercomputer Applications*, 2001.
- GEIST, A. *et al. PVM: Parallel Virtual Machine - User's Guide and tutorial for Networked Parallel Computing*. [S.l.]: MIT, 1994.
- GOLGHATE, A. A.; SHENDE, S. W. Parallel k-means clustering based on hadoop and hama. *IJCAT International Journal of Computing and Technology*, v. 1, 2014.
- GONZALEZ, J. E. *et al.* Powergraph: Distributed graph-parallel computation on natural graphs. *OSDI*, v. 12, n. 1, 2012.
- GROPP, W. *et al. Using Advanced MPI: Modern Features of the Message-Passing Interface*. [S.l.]: MIT Press, 2014. 392 p.
- HADOOP. *Apache Hadoop*. 2014. Disponível em: <<http://hadoop.apache.org/>>.
- HAN, J.; KAMBER, M.; PEI, J. *Data Mining: Concepts and Techniques: Concepts and Techniques*. Elsevier Science, 2011. (The Morgan Kaufmann Series in Data Management Systems). ISBN 9780123814807. Disponível em: <<http://books.google.com.br/books?id=pQws07tdpjoC>>.
- HUBERT, L.; SCHULTZ, J. Quadratic assignment as a general data-analysis strategy. *British Journal of Mathematical and Statistical Psychology*, v. 29, p. 190–241, 1976.
- JAMSA, K. *Cloud Computing*. Jones & Bartlett Learning, 2011. ISBN 9781449647391. Disponível em: <<http://books.google.com.br/books?id=msFk8DPZ7noC>>.
- KARIMI, H. A. (Ed.). *Big Data: Techniques and Technologies in Geoinformatics*. [S.l.]: CRC Press, 2014. 312 p.
- KUCUKYILMAZ, T. Parallel k-means algorithm for shared memory multiprocessors. *Computer Science & Communications*, v. 2, n. 11, 2014.
- KUDYBA, S. *Big Data, Mining, and Analytics: Components of Strategic Decision Making*. [S.l.]: CRC Press, 2014. 325 p.
- KUMAR, J. *et al.* Parallel k-means clustering for quantitative ecoregion delineation using large data sets. *Procedia Computer Science*, v. 4, p. 1602–1611, 2011.
- KUMAR, V. P.; GUPTA, A. Analyzing scalability of parallel algorithms and architectures. *Journal of Parallel and Distributed Computing*, v. 22, p. 379–391, 1994.

- LONEA, A. M. Private cloud set up using eucalyptus open source. *Soft Computing Applications*, v. 195, p. 381–389, 2012.
- MALEWICZ, G. *et al.* Pregel: a system for large-scale graph processing. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. [S.l.: s.n.], 2010. p. 135–146.
- MCCALLUM, A.; NIGAM, K.; UNGAR, L. H. Efficient clustering of high-dimensional data sets with application to reference matching. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2000. p. 169–178.
- MILLS, R. T. *et al.* Cluster analysis-based approaches for geospatiotemporal data mining of massive data sets for identification of forest threats. In: *International Conference on Computational Science, ICCS 2011*. [S.l.]: Procedia Computer Science, 2011. v. 4, p. 1612–1621.
- RAMESH, V.; RAMAR, K.; BABU, S. Parallel k-means algorithm on agricultural databases. *IJCSI International Journal of Computer Science Issues*, v. 10, n. 1, 2013.
- SAKR, S.; GABER, M. *Large Scale and Big Data: Processing and Management*. [S.l.]: CRC Press, 2014. 636 p.
- SHAO, B.; WANG, H.; LI, Y. Trinity: A distributed graph engine on a memory cloud. In: *Proceedings of the 2013 international conference on Management of data*. [S.l.: s.n.], 2013.
- SHROFF, G. *Enterprise Cloud Computing: Technology, Architecture, Applications*. [S.l.]: Cambridge University Press, 2010. 273 p.
- TAGARAKIS, A. *et al.* Management zones delineation using fuzzy clustering techniques in grapevines. *Precision Agriculture*, v. 14, p. 18–39, 2013.
- TANENBAUM, A. S.; STEEN, M. V. *Distributed Systems: Principles and Paradigms*. [S.l.]: Pearson Prentice Hall, 2007.
- TANIAR, D. *et al.* *High Performance Parallel Database Processing and Grid Databases*. [S.l.]: John Wiley & Sons, 2008. 440 p.
- TAURION, C. *Cloud Computing - Computação em Nuvem*. BRASPORT, 2009. ISBN 9788574524238. Disponível em: <<http://books.google.com.br/books?id=mvir2X-A2mcC>>.
- TAURION, C. *Big Data*. [S.l.]: Brasport, 2013. 102 p.
- BORTHAKUR, D. (Ed.). *The Hadoop Distributed File System: Architecture and Design*. 2007.
- VIEIRA, M. A. *et al.* Object based image analysis and data mining applied to a remotely sensed landsat time-series to map sugarcane over large areas. *Remote Sensing of Environment*, v. 123, p. 553–562, 2012.
- WHITE, T. *Hadoop: The Definitive Guide*. Third edition. [S.l.]: O'Reilly, 2012.

WILKINSON, B.; ALLEN, C. M. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. 2nd. ed. [S.l.]: Pearson/Prentice Hall, 2005. 467 p.

WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd. ed. [S.l.]: Morgan Kaufmann series in data management systems, 2005. 525 p.

ZAHARIA, M. *et al.* Spark: Cluster computing with working sets. *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.

ZHOU, P.; LEI, J.; YE, W. Large-scale data sets clustering based on mapreduce and hadoop. *Journal of Computational Information Systems*, v. 7, p. 5956–5963, 2011.

APÊNDICE A – ARQUIVOS DE CONFIGURAÇÃO DO HADOOP

Código A.1: **core-site.xml**

```
1 <configuration>
2   <property>
3     <name>fs.default.name</name>
4     <value>hdfs://10.20.6.224:54310</value>
5   </property>
6   <property>
7     <name>hadoop.tmp.dir</name>
8     <value>/home/lays/temp</value>
9   </property>
10 </configuration>
```

Código A.2: **mapred-site.xml**

```
1 <configuration>
2   <property>
3     <name>mapred.job.tracker</name>
4     <value>10.20.6.224:54311</value>
5   </property>
6 </configuration>
```

Código A.3: **hdfs-site.xml**

```
1 <configuration>
2   <property>
3     <name>dfs.replication</name>
4     <value>3</value>
5   </property>
6 </configuration>
```

APÊNDICE B – IMAGENS EXTRAÍDAS DA INTERFACE *WEB*

Figura 16: Informações sobre os *jobs* MapReduce executados

Jobid	Started	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
job_201504191847_0001	Sun Apr 19 18:50:20 BRT 2015	NORMAL	root	KMeansParalelo	100,00%	32	32	100,00%	1	1
job_201504191847_0002	Sun Apr 19 18:50:55 BRT 2015	NORMAL	root	KMeansParalelo	100,00%	32	32	100,00%	1	1
job_201504191847_0003	Sun Apr 19 18:51:21 BRT 2015	NORMAL	root	KMeansParalelo	100,00%	32	32	100,00%	1	1
job_201504191847_0004	Sun Apr 19 18:51:49 BRT 2015	NORMAL	root	KMeansParalelo	100,00%	32	32	100,00%	1	1
job_201504191847_0005	Sun Apr 19 18:52:20 BRT 2015	NORMAL	root	KMeansParalelo	100,00%	32	32	100,00%	1	1
job_201504191847_0006	Sun Apr 19 18:52:48 BRT 2015	NORMAL	root	KMeansParalelo	100,00%	32	32	100,00%	1	1
job_201504191847_0007	Sun Apr 19 18:53:15 BRT 2015	NORMAL	root	KMeansParalelo	100,00%	32	32	100,00%	1	1

APÊNDICE C – PUBLICAÇÃO

Algoritmo K-Means Paralelo com base no MapReduce para Mineração de dados agrícolas, publicado nos Anais do 5º EATI - Encontro Anual de Tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação, Universidade Federal de Santa Maria, Frederico Westphalen, RS, Novembro, 2014.