UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS CORNÉLIO PROCÓPIO
DEPARTAMENTO DE COMPUTAÇÃO
ENGENHARIA DE COMPUTAÇÃO

MICHAEL KEESE MARTINS

# ANÁLISE DE DESEMPENHO DE ARCABOUÇOS EM BIG DATA

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO
2016

MICHAEL KEESE MARTINS

# ANÁLISE DE DESEMPENHO DE ARCABOUÇOS EM BIG DATA

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina Trabalho de Conclusão de Curso 2, do curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Prof. Me. Francisco Pereira Junior

CORNÉLIO PROCÓPIO

2016

Ministério da Educação
**Universidade Tecnológica Federal do Paraná**
Câmpus Cornélio Procópio
Departamento de Computação
Engenharia de Computação

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

## TERMO DE APROVAÇÃO

## Análise de Desempenho de Arcabouços em Big Data

**por**

**Michael Keese Martins**

Este Trabalho de Conclusão de Curso de graduação foi julgado adequado para obtenção do Título de Licenciatura e Bacharelado em Engenheria de Computação e aprovado em sua forma final pelo Programa de Graduação em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Cornélio Procópio, 28/06/2016

_____

Prof. Me. Francisco Pereira Junior

_____

Prof. Dr. Eduardo Cotrin Teixeira

_____

Prof. Dr. Giovani Volnei Meinerz

"A Folha de Aprovação assinada encontra-se na Coordenação do Curso"

## AGRADECIMENTOS

Primeiramente a Deus, por ter me dado condições para que eu pudesse desenvolver este trabalho.

À minha família (pais e irmã) por me darem todo o suporte, toda a confiança e por nunca terem deixado faltar coisa alguma para mim ao longo de todos os cinco anos de graduação. Este trabalho é o resultado de todo o esforço que despendi para fazer valer a pena o investimento e fé que eles depositaram em mim.

À minha namorada, que acompanhou todas as dificuldades, conquistas, nervosismos e coisas do gênero que surgiram ao longo desta caminhada. Além de toda torcida e apoio, que foram essenciais.

Aos amigos da minha turma na UTFPR, os quais jamais se recusaram a dar qualquer tipo de ajuda. Sem eles, a trajetória que tracei teria o dobro de obstáculos para que eu pudesse chegar ao desenvolvimento deste trabalho.

Ao meu orientador Francisco por todos os esclarecimentos, toda ajuda e confiança.

Aos professores que lecionaram as disciplinas que cursei e se destacaram por dedicação, seriedade e principalmente respeito à profissão que exercem: José Antônio Gonçalves, André Endo, Marilu Martens Oliveira, Sandra Mara Domiciano, Renata Mascari, Cláudia Brunosi, Joselene Marques, Roberto Molina de Souza, José Aparecido Lopes Junior, Pedro Henrique Bugatti, Claiton de Oliveira, Carlos Nascimento Silla Junior, Katia Romero Felizardo e Fábio Renan Durand.

## RESUMO

MARTINS, Michael Keese. **Análise de Desempenho de Arcabouços em Big Data**. 2016. 91 f. Proposta de Trabalho de Conclusão de Curso (Graduação) –Engenharia de Computação. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2016.

Devido a imensidão de dados gerados ao longo dos anos que compõem a era digital, algumas técnicas foram desenvolvidas para auxiliar em seu processamento. As técnicas tradicionais de processamento de informação se tornaram inviáveis na medida em que a quantidade de dados começou a crescer exponencialmente a cada ano. Porém, muitos casos inseridos neste contexto, podem ser modelados ao usar estruturas de grafos. Este trabalho propõe o processamento de alguns algoritmos modelados em grafos em vários cenários, no que diz respeito ao seu processamento. Para a definição de todos os cenários que foram estudados, utilizou-se um protocolo onde se encontravam elencadas todas as características de interesse, como por exemplo tempo de processamento, uso de memória, etc.

**Palavras-chave:** Big Data. Alto Desempenho. Processamento Paralelo.Grafos. Cluster.

## ABSTRACT

MARTINS, Michael Keese. **Frameworks Performance Analysis in Big Data**.2016. 91 f. Final Paper Proposal (Graduation) –Computer Engineering. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2016.

Due to the data immensity generated over the years that make up the digital age, some techniques have been developed to assist in its processing. The traditional techniques for processing information have become unviable from the moment when the amount of data started to grow exponentially each year. However, many cases that are inserted in this context, can be modeled using the graph structure. This work proposes the processing of some algorithms modeled in graphs in various scenarios, regarding to their processing. To set all scenarios studied, a protocol was developed where all features of interest were listed, such as processing time, memory usage, etc.



**Keywords**:Big Data. High Performance. Parallel Processing. Graphs. Cluster.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE SIGLAS

| | |
|---|---|
| BSP | Bulk Synchronous Parallel |
| CPU | Central Processing Unit |
| GPS | Graph Processing System |
| GPU | Graphics Processing Unit |
| HDFS | Hadoop Distributed File System |
| IBM | International Business Machines |
| IDE | Integrated Development Environment |
| JAR | Java Archive |
| pBD | Approximate Betweennessbased Divisive Algorithm |
| pLA | Greedy Local Aggregation Algorithm |
| pMA | Modularitymaximizing Agglomerative Clustering Algorithm |
| RAM | Random Access Memory |
| RNA | Rede Neural Artificial |
| RPM | Rotações Por Minuto |
| SAR | System Activity Reporter |
| SCC | Strongly Connected Component |
| SNAP | Smallworld Network Analysis and Partitioning |
| SNMP | Simple Network Management Protocol |
| SO | Sistema Operacional |
| WCC | Weakly Connected Component |
| XDR | External Data Representation |
| XML | Extensible Markup Language |

**SUMÁRIO**

# 1 INTRODUÇÃO

Em um mundo onde a quantidade de dados digitais dobra sua totalidade em um intervalo de pouco mais de três anos (Hilbert, 2007), tornou-se necessário que a tecnologia acompanhasse tamanho desenvolvimento para que houvessem estruturas capazes de armazenar a evolução da coleta de dados digitais. E isto de fato aconteceu, pois ao longo dos últimos 50 anos, o custo do armazenamento digital caiu pela metade a cada dois anos, aproximadamente, enquanto a densidade de armazenamento aumentou 50 milhões de vezes (MAYERSCHÖNBERGER; CUKIER, 2012). Então, deu-se início a era dos dados. A sociedade agora detém um potencial enorme no que diz respeito aos dados digitais, basta somente possuir a sabedoria de como extrair informações a partir deles.

Uma analogia muito adequada para este cenário é a de uma mina de ouro. O mundo desenvolveu maneiras de coletar dados e os armazenar de forma a criar uma imensa mina de ouro. Porém, para que uma mina seja aproveitada é preciso ter o conhecimento necessário para extrair o ouro. Ou seja, um banco de dados digital precisa ser garimpado para oferecer informações. Surge então um novo conceito para tratar este novo contexto na sociedade: o Big Data.

O grande objetivo do Big Data é encontrar respostas ao minerar uma grande quantia de dados. É importante ressaltar que a causalidade não é uma preocupação do Big Data, ou seja, entender o motivo de algo ter acontecido não é interessante aqui. O foco está em descobrir o que realmente aconteceu. Para isso, a ideia é encontrar correlações em um montante de dados e então estudá-las para poder extrair o que está acontecendo. Ao fazer isso, o Big Data tenta transformar uma imensa quantia de dados em informações valiosas. A manipulação destes dados demanda ferramentas altamente poderosas.

Em posse de supercomputadores, os pesquisadores possuem as ferramentas exigidas para trabalhar com enormes quantidades de dados. Este fato pode fazer com que alguns métodos muito tradicionais, utilizados pelospesquisadores ao longo da história, entrem em extinção. Como por exemplo, o

uso de amostragem para estimativas. A ideia por trás da amostragem é reduzir o tamanho da população (totalidade daquilo que se está estudando), de modo que o processo exija menor poder computacional para que se atinja um resultado aproximado (estimativa) aceitável de uma informação. Mas, a evolução da tecnologia permite que a preocupação com o poder de processamento seja descartada e, desta forma, é possível chegar no resultado exato de uma informação ao usar todos os dados coletados ao invés de se limitar a uma amostra.

Com o surgimento de todos estes conceitos e ideias, novas ferramentas foram desenvolvidas para auxiliar na manipulação massiva de dados digitais. Entre os exemplos, alguns destaques são os arcabouços: Hadoop, Mahout, Lucene, GoogleFS (GFS), Solr, Pegasus, etc.

## 1.1 PROBLEMA

O surgimento da internet trouxe consigo uma nova fonte de informações. Segundo a União Internacional das Telecomunicações (orgão vinculado à Organização das Nações Unidas), 3,2 bilhões de pessoas possuíam acesso à internet em maio de 2015, fato este, que pode nos oferecer uma avalanche de informação um tanto quanto interessante. A fim de organizar todo este conteúdo gerado através da internet, leva-se em conta os três pilares do Big Data: volume, velocidade e variedade (MAYERSCHÖNBERGER; CUKIER, 2012). Uma grande parcela de todo conjunto de dados oriundo da internet se encaixa perfeitamente na estrutura proposta por grafos. Tratam-se de grafos gigantescos (apresentados com maiores detalhes no tópico 2.1.2) que se tornam problemas computacionais de importância relevante. Atualmente existem arcabouços muito promissores para processamento de grafos, porém, não se sabe qual deles é a opção mais adequada, levando em consideração aspectos como uso de memória, tempo de execução,

tráfego de rede, entre outros. É provável que haja divergência entre a melhor opçãode arcabouço para diferentes problemas, ou seja, as chances de um arcabouço possuir o melhor desempenho para a solução de qualquer problema relacionado a grafos são praticamente nulas. Logo, qual arcabouço usar? Quando usá-lo? Por que usá-lo?

## 1.2 OBJETIVOS

### 1.2.1 Objetivo Geral

Avaliar o desempenho de arcabouços de Big Data para algoritmos que manipulamestruturas de grafos em sua implementação.

### 1.2.2 Objetivos Específicos

Para alcançar o objetivo geral foram seguidas as seguintes etapas:

- Estudo e instalação de arcabouços de processamento de grafos.
- Estudo de ferramentas de monitoramento de desempenho como: SAR, DSTAT, Munin e Ganglia.
- Estudo de alguns algoritmos de implementação em grafos que manipulam grande quantidade de dados.
- Escolha de parâmetros de desempenho adequados ao contexto deste trabalho, como por exemplo: tempo de execução, pico de uso de memória, tráfego de rede, entre outros.
- Criação de um protocolo de execução que estabeleça todas as informações de interesse.
- Execução dos algoritmos em grandes grafos escolhidos nos arcabouços estudados, respeitando o protocolo estabelecido.
- Utilização das ferramentas de monitoramento de desempenho na execução

dos algoritmos nos arcabouços, seguindo o protocolo estabelecido.

Comparação dos resultados obtidos nas ferramentas de monitoramento de desempenho para cada um dos cenários especificados e procurar determinar qual deles é o melhor, seguindo os critérios estabelecidos.

## 1.3 JUSTIFICATIVA

A modelagem em grafos no contexto atual de Big Data a partir da utilização de técnicas tradicionais não é uma boa alternativa. Isto se dá devido a dimensão que estes grafos tomam para tratar todos os dados necessários de um determinado caso em Big Data. A solução para este problema é usar de processamento paralelo, através do uso de ***cluster,*** e/ou arcabouços que implementam processamento paralelo de grafos. Um ***cluster*** é um conjunto de computadores que operam tarefas de maneira distribuída. Há vários tipos de ***cluster***, que são classificados conforme o propósito para o qual foram configurados. Pode-se utilizá-los para buscar alto desempenho, alta disponibilidade ou balanceamento de carga. Para o contexto deste trabalho, cita-se ***cluster*** com o propósito de alto desempenho.

Não há estudos que definam qual abordagem para o problema supracitado é a melhor. Na seção de Trabalhos Relacionados (tópico 2.4) há alguns exemplos de comparações estudadas, mas foram considerados poucos parâmetros para a realização da análise, fato este que permite que existam eventos que possivelmente tornam o resultado destes estudos incompletos. Este trabalho busca encontrar resultados confiáveis através da análise de parâmetros importantes de desempenho que serão detalhados nos próximos capítulos.

## 1.4 ORGANIZAÇÃO DO TEXTO

O trabalho foi organizado para que os próximos capítulos respeitem a seguinte estrutura: a apresentação dos conceitos que serviram de base para o desenvolvimento do trabalho e os principais trabalhos relacionados se encontram no capítulo 2; a metodologia de pesquisa, os materiais e os métodos que foram utilizados para atingir os objetivos do trabalho serão expostos no capítulo 3; as etapas planejadas para as pesquisas que envolvem a ambientação aos conceitos necessários para o desenvolvimento do trabalho serão apresentadas no capítulo 4; os resultados obtidos serão exibidos no capítulo 5; as considerações finais se encontram no capítulo 6, que incluem as dificuldades encontradas ao longo do trabalho e sugestões de trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

## 2.1 GRAFOS

Um grafo consiste em um conjunto de nós (ou vértices) e em um conjunto de arcos (ou arestas). Cada arco num grafo é especificado por um par de nós. Se os pares de nós que formam os arcos forem pares ordenados, diz-se que o grafo é um grafo orientado (ou dígrafo). As setas entre os nós representam arcos. O início de cada seta representa o segundo nó no par ordenado de nós que forma um arco, e o final de cada seta representa o primeiro nó no par. Um número pode ser associado a cada arco de um grafo. Um grafo desse tipo é chamado grafo ponderado ou rede. O número associado a um arco é chamado peso (TENENBAUM, 1995). A Figura 1a ilustra um grafo, enquanto as figuras 1b, 1c e 1d ilustram exemplos de dígrafos.

A aplicação de grafos como alternativa de modelagem de problemas é bastante difundida e possui alguns exemplos muito conhecidos, como o problema do caixeiro viajante. Tal problema consiste em um vendedor que precisa realizar a entrega dos produtos em várias cidades diferentes. A ideia é encontrar o trajeto que resulta no menor caminho a ser percorrido, para que se evite desperdício de tempo, combustível e eventuais outros gastos com transporte. Neste cenário, cada cidade é representada por um nó, enquanto as estradas que ligam uma cidade a outra são representadas por arcos. Desta forma, modela-se o problema em uma estrutura matematicamente conhecida, onde pode-se fazer uso de suas propriedades para encontrar informações que antes não eram facilmente acessíveis.

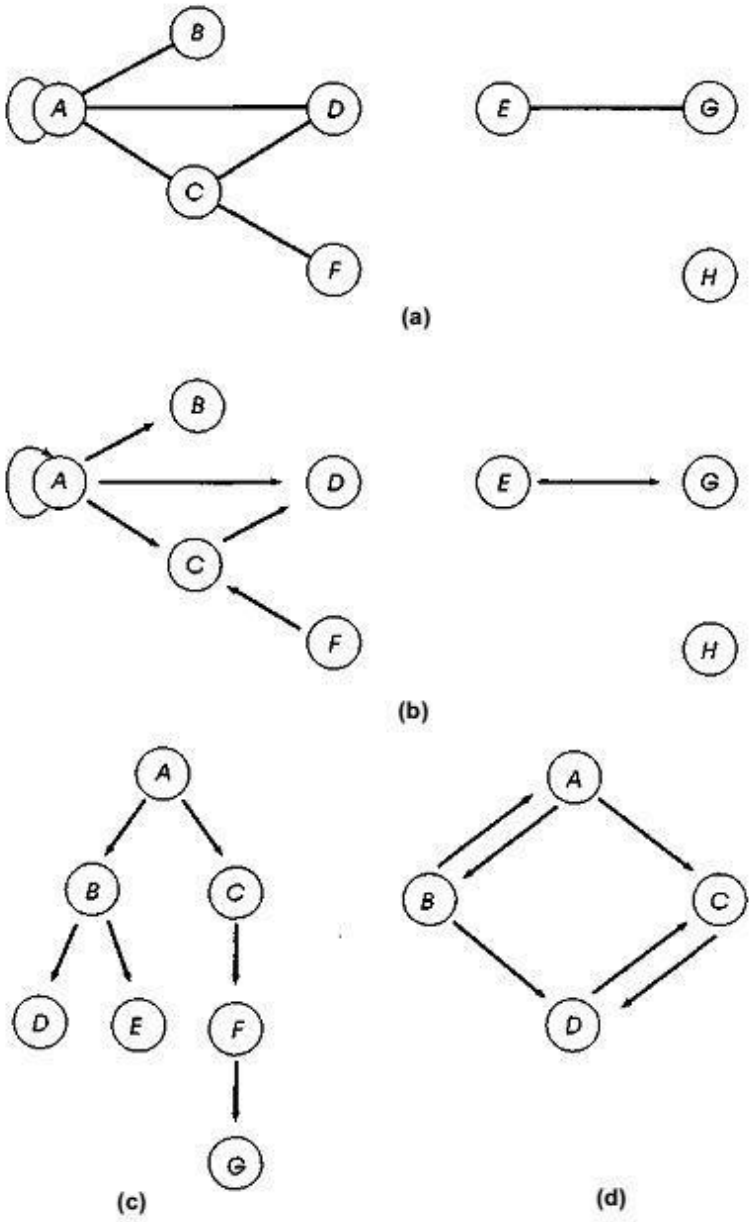


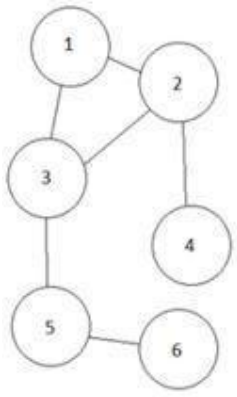

**Figura 1 Exemplos de grafos.**
**Fonte: (TENENBAUM, 1995).**

### 2.1.1 Representação de Grafos

Existem duas maneiras de representar um grafo. Uma delas é a matriz de adjacência. A segunda forma de representação de grafos é através da lista de adjacência.

a) Matriz de Adjacência

Trata-se de uma matriz ***n x n***, onde ***n*** representa o número total de nós do grafo. Cada elemento da matriz representa a relação dos vértices. Por exemplo: para uma matriz ***M[i,j]***na posição ***M[0,0]***temos a relação entre o vértice 0 com ele mesmo, enquanto que na posição ***M[0,1]***temos a relação entre o vértice 0 ($v_0$ ) com o vértice 1 ($v_1$ ). Caso haja adjacência na relação entre os vértices em questão, esta posição da matriz recebe 1, caso contrário a posição recebe 0. Se as arestas do grafo representado pela matriz de adjacência possuírem pesos, no lugar de 1 para representar uma eventual adjacência, coloca-se o valor do peso da aresta. A Figura 2 ilustra um exemplo de grafo e a matriz de adjacência gerada por ele.




| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 | 1 | 0 |

**Figura 2 Exemplo de grafo com sua respectiva matriz de adjacência.**
**Fonte: Autoria Própria.**

b) Lista de Adjacência

Nesta forma de representação de grafos, todas as arestas (que representam as relações entre os vértices) do grafo serão representadas de maneira muito intuitiva em uma lista. Cada linha da matriz de adjacência, ou seja, cada vértice do grafo possuirá uma lista encadeada (estrutura de dados muito utilizada em desenvolvimento de software). Os nós em cada lista encadeada gerada representam os vértices adjacentes ao vértice responsável pela geração da lista. Desta forma, obtêm-se uma visão individual das relações de cada vértice. A Figura 3 ilustra o mesmo exemplo de grafo da Figura 2 e a lista de adjacência gerada por ele (NONATO, 2000).

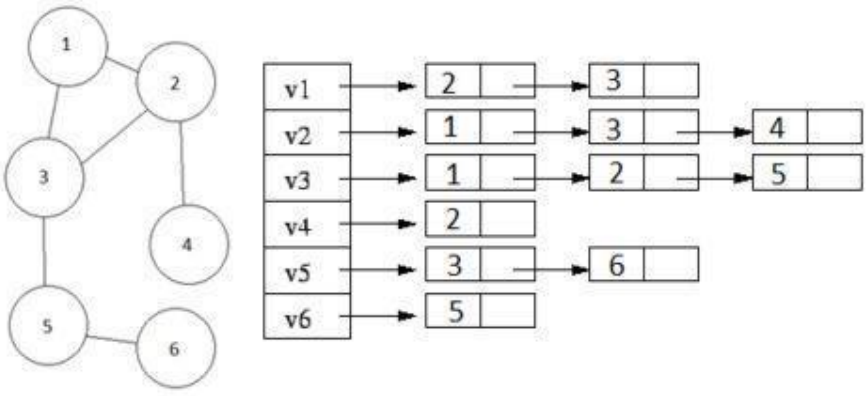



**Figura 3 Exemplo de grafo com sua respectiva lista de adjacência.**
**Fonte: Autoria Própria.**

2.1.2 Grafos em Big Data

Bons exemplos recentes de aplicação de grafos podem ser encontrados nas redes sociais. Dentre as redes sociais, destacam-se (MAYERSCHÖNBERGER; CUKIER, 2012):

- Facebook: mais de 10 milhões de fotos por hora. Aproximadamente 3 bilhões de curtidas ou comentários por dia.
- Youtube: 800 milhões de usuários mensais enviam mais de uma hora de vídeo por segundo.
- Twitter: 400 milhões de tweets por dia em 2012.

Um dos exemplos de aplicação de grafos nestes casos é a modelagem dos usuários como vértices e as relações que eles possuem com outros usuários como arestas. Um outro exemplo é modelar as curtidas de uma foto postada por uma pessoa, ou um vídeo, ou até mesmo um texto. Tais aplicações geraminformações, como por exemplo, descobrir qual tipo de público gostou de um determinado produto a partir das curtidas de uma foto postada e eventualmente alterar a forma como o produto é divulgado para agradar ainda mais. É possível notar, pelas grandezas citadas acima, que tratam-se de números assustadores que jamais poderiam ser processados por um único computador, devido a seus recursos escassos de memória e processamento. Sendo assim, para trabalhar todos esses dados, foram desenvolvidos novos algoritmos e arcabouços que fossem capazes de os implementar.

O tópico 2.4 deste trabalho traz mais exemplos de aplicação de grafos em Big Data.

## 2.2 ARCABOUÇOS PARA PROCESSAMENTO DE GRAFOS EM BIG DATA

Conforme supracitado, o processamento de grandes grafos tornou-se algo muito comum com o surgimento das redes sociais, ao representar os relacionamentos entre usuários e algumas outras aplicações. Uma ferramenta bastante conhecida e difundida para trabalhar em casos que envolvem números muito grandes de dados é o Hadoop. Mas, a maioria dos algoritmos que se utilizam

da estrutura de grafos, possuem características que não se adaptam bem ao modelo MapReduce (modelo de programação destinado a processar volumes grandes de dados em paralelo utilizado pelo Hadoop). Frente a esses problemas, a própria Google desenvolveu um ambiente de processamento específico para esse contexto, o Pregel (MALEWICZ et al., 2010). Além do Pregel, outros arcabouços que se utilizam da infraestrutura do Hadoop também surgiram como alternativas. Neste trabalho foram analisados alguns outros arcabouços de processamento de grandes quantidades de dados além do Hadoop. A relação mostrada pela Tabela 1 traz alguns exemplos de arcabouços. Os tópicos seguintes possuem maiores detalhes dos arcabouços listados.

**Tabela 1 Arcabouços de processamento de grandes quantidades de dados e seus respectivos desenvolvedores.**

| Arcabouço | Desenvolvedor |
|---|---|
| Hadoop | Apache |
| Hama | Apache |
| Pegasus | Carnegie Mellon University |
| Giraph | Apache |
| GBase | IBM |

### 2.2.1 Hadoop

O projeto Apache Hadoop desenvolve software de código aberto para computação distribuída. O Apache Hadoop é um arcabouço que permite o processamento distribuído de grandes conjuntos de dados em ***clusters*** de computadores que usam modelos de programação simples. Ele é projetado para escalar a partir de um único servidor para milhares de máquinas, cada uma oferecendo processamento e armazenamento local. Em vez de confiar em hardware para proporcionar alta disponibilidade, o arcabouço em si é designado a detectar e tratar falhas na camada de aplicação, de modo a fornecer um serviço altamente

disponível para um ***cluster*** de computadores, onde cada um dos quais pode ser propenso a falhas (The Apache Software Foundation, 2014).

Hadoop é utilizado no processamento de grandes quantidades de dados. Para isto, divide os dados em quantidades menores e os distribui por outras máquinas. Ele espera que o hardware falhe, por isso cria redundâncias e presume que os dados não sejam claros e ordenados. Não é tão preciso, logo, não pode ser utilizado para lançar um foguete no espaço ou para dar detalhes de conta bancária, mas em tarefas menos críticas ele é muito mais rápido. Com uso do Hadoop a Visa foi capaz de reduzir o tempo de processamento de dois anos de registros de um mês para 13 minutos (MAYERSCHÖNBERGER; CUKIER, 2012).

Neste trabalho, os métodos principais dos algoritmos utilizados para os testes e as análises no Hadoop são o *map()* e o *reduce()* que podem ser vistos no Código 1 e no Código 2, respectivamente, que são trechos do código de PageRank (algoritmo apresentado no tópico 2.3.1) implementado em java por terceiros (este código foi utilizado nas execuções deste trabalho). O Código 1 é responsável por coletar todos os nós distintos do grafo, que serão utilizados para calcular o valor inicial de PageRank no método *reduce()*, para tal, faz-se uma condição que desconsidera linhas de comentários no arquivo (linha 4) de representação do grafo e inicia-se uma busca pelos nós de origem (linha 7) e pelos nós de destino (linha 8), com a verificação de tabulações através do símbolo "\t" que são utilizadas como separadores. Os nós são inseridos em uma lista ("NODES"), tanto nós de origem como nós de destino, concluindo assim o papel deste método *map()*. O Código 2 faz uma varredura através de um dado nó ("key"), com o intuito de construir uma lista de valores iniciais de PageRank para cada nó seguindo o seguinte padrão:

<title> <pagerank> <link1>,<link2>,<link3>,<link4>,...,<linkN>

onde "title" é o nó chave, "pagerank" é o valor inicial de PageRank e "linkN" são os nós destino ligados ao nó chave.

```java
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        if (value.charAt(0) != '#') {

            int tabIndex = value.find("\t");
            String nodeA = Text.decode(value.getBytes(), 0, tabIndex);
            String nodeB = Text.decode(value.getBytes(), tabIndex + 1, value.getLength() - (tabIndex + 1));
            context.write(new Text(nodeA), new Text(nodeB));

            PageRank.NODES.add(nodeA);
            PageRank.NODES.add(nodeB);

        }

    }

}
```

**Código 1  Método map() do código de PageRank no Hadoop.**
**Fonte: https://github.com/danielepantaleone/hadoop-pagerank.**

```java
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

        boolean first = true;
        String links = (PageRank.DAMPING / PageRank.NODES.size()) + "\t";

        for (Text value : values) {
            if (!first)
                links += ",";
            links += value.toString();
            first = false;
        }

        context.write(key, new Text(links));
    }

}
```

**Código 2  Método reduce() do código de PageRank no Hadoop.**
**Fonte: https://github.com/danielepantaleone/hadoop-pagerank.**

### 2.2.2 Hama

Hama é um arcabouço de computação distribuída baseado em técnicas BSP (***Bulk Synchronous Parallel***) para processar matrizes, grafos, redes e outros

tipos de modelagens massivas de dados. O uso de técnicas computacionais BSP se justifica através do benefício obtido em velocidade de execução de ***loops***, mais especificamente em processos iterativos, como a busca do menor caminho em um grafo (YOON, E. J., 2011). Apache Hama (***Hadoop Matrix Algebra***) é uma estrutura para análise de Big Data, criada em 2012 como um projeto de nível superior da Apache Software Foundation.

Bem como os arcabouços Pregel e DistBelief (arcabouço do Google utilizado em ***clusters*** de computadores que contam com mais de 1000 máquinas destinadas a implementar grandes modelagens), também faz uso de modelos de programação baseados em vértice e neurônio (como exemplo: RNA, rede neural artificial) (The Apache Software Foundation, 2014). A Figura 4 mostra o logotipo do projeto Apache Hama.

**Figura 4 Logotipo do projeto Apache Hama.**
**Fonte: https://hama.apache.org.**

O método principal dos algoritmos utilizados nas execuções realizadas no Hama é o *compute()*,que pode ser visto no Código 3 que é um trecho do código de PageRank utilizado que esta incluido no pacote de exemplos nativos do arcabouço. Neste código pode ser notado o cálculo do PageRank nas linhas de 3 a 11, com destaque para o fator de amortecimento (“DAMPING_FACTOR”) que será explicado com maiores detalhes no tópico 2.3.1 deste trabalho. O resultado obtido nos cálculos é enviado na linha 20 para os nós vizinhos a cada “super-passo”, que segundo a própria Apache explica em seu site, é um procedimento de três etapas (computação local, processos de comunicação e sincronização) característico de programas BSP. Estes “super-passos” funcionam como iterações. Enquanto os valores de PageRank são calculados, os valores são atualizados e de certa forma melhorados a cada “super-passo”, pois a cada iteração recebe-se um novo valor de PageRank dos vizinhos. A função chamada na linha 16 é utilizada para parar o método *compute()*.

```
1     @Override
2     public void compute(Iterable<DoubleWritable> messages) throws IOException {
3       if (this.getSuperstepCount() == 0) {
4         this.setValue(new DoubleWritable(1.0 / this.getNumVertices()));
5       } else if (this.getSuperstepCount() >= 1) {
6         double sum = 0;
7         for (DoubleWritable msg : messages) {
8           sum += msg.get();
9         }
10        double alpha = (1.0d - DAMPING_FACTOR) / this.getNumVertices();
11        this.setValue(new DoubleWritable(alpha + (sum * DAMPING_FACTOR)));
12      }
13
14      DoubleWritable globalError = getLastAggregatedValue(0);
15      if (globalError != null && this.getSuperstepCount() >= ITERATION ) {
16        voteToHalt();
17        return;
18      }
19
20      sendMessageToNeighbors(new DoubleWritable(this.getValue().get()
21          / this.getEdges().size()));
22    }
23
24  }
```

**Código 3 Método compute() do código de PageRank no Hama.**
**Fonte: https://github.com/apache/hama.**

### 2.2.3 Giraph

Apache Giraph é um sistema de processamento de grafo interativo, construído para alta escalabilidade. É usado atualmente pelo Facebook para analisar o grafo social formado por usuários e suas conexões. Giraph surgiu como a alternativa de código aberto ao Pregel, a arquitetura de processamento gráfico desenvolvida pelo Google (MALEWICZ et al., 2010). Ambos os sistemas são inspirados pelo modelo BSP. Giraph adiciona vários recursos além dos existentes no modelo Pregel, incluindo ***master computation***, ***sharded aggregators***, ***edgeorientedinput***,***outofcore computation***, e mais alguns outros. Com um ciclo

dedesenvolvimento estável e uma crescente comunidade de usuários no mundo todo, Giraph é uma escolha natural para aproveitar o potencial dos conjuntos de dados estruturados em uma escala maciça (Ching et al., 2015).

Bem como o Hama, o método principal das aplicações desenvolvidas para este arcabouço é o *compute()*. Tal método pode ser visto no Código 4 que é um trecho do algoritmo SimplePageRankComputation (algoritmo que foi utilizado nas execuções deste trabalho), disponível no pacote JAR de exemplos nativos que acompanham a instalação do arcabouço. Nota-se no código que as linhas de 3 a 7 implementam a mesma lógica das linhas de 5 a 9 do Código 3 (PageRank no Hama), e as linhas de 8 a 16 realizam o cálculo do valor de PageRank, mais uma vez com destaque para o fator de amortecimento da fórmula do algoritmo que é definida como 0.85 na linha 9. Os resultados obtidos nos cálculos são enviados para as outras arestas na linha 21. E, assim como no Hama, o Giraph também trabalha com “super-passos”. O valor de PageRank é aprimorado a cada “super-passo” através das atualizações dos valores calculados. A função chamada na linha 24 é utilizada para parar o método *compute()*. A fórmula do cálculo deste algoritmo é explicada com maiores detalhes no tópico 2.3.1.

```java
@Override
public void compute(Iterable<DoubleWritable> messages) {
  if (getSuperstep() >= 1) {
    double sum = 0;
    for (DoubleWritable message : messages) {
      sum += message.get();
    }
    DoubleWritable vertexValue =
        new DoubleWritable((0.15f / getTotalNumVertices()) + 0.85f * sum);
    setValue(vertexValue);
    aggregate(MAX_AGG, vertexValue);
    aggregate(MIN_AGG, vertexValue);
    aggregate(SUM_AGG, new LongWritable(1));
    LOG.info(getId() + ": PageRank=" + vertexValue +
        " max=" + getAggregatedValue(MAX_AGG) +
        " min=" + getAggregatedValue(MIN_AGG));
  }

  if (getSuperstep() < MAX_SUPERSTEPS) {
    long edges = getNumEdges();
    sendMessageToAllEdges(
        new DoubleWritable(getValue().get() / edges));
  } else {
    voteToHalt();
  }
}
```

**Código 4 Método compute() do código de PageRank no Giraph.**
**Fonte: https://github.com/apache/giraph.**

## 2.3 ALGORITMOS EM GRAFOS

### 2.3.1 PageRank

Em meio a tantas páginas na web com conteúdos que possuem um alto nível de variedade, como definir qual delas é a mais relevante? Muito provavelmente, os parâmetros que definem a página mais importante encontrarão inúmeras candidatas com conteúdos muito similares. Para resolver este problema, alguns critérios auxiliam no processo de elencar todas as páginas que satisfazem os parâmetros desejados. Ao eleger o conteúdo que mais se aproxima do interesse das

pessoas, pode-se extrair benefícios, como velocidade de resultado de busca, objetividade, entre outros.

Basicamente, PageRank é um sistema para dar notas para páginas na web (Google, 2011).

No coração de um algoritmo de PageRank há uma fórmula matemática que parece assustadora, mas que é relativamente simples de se compreender (ROGERS, 2002).

Assume-se que uma determinada página ***A*** possui páginas $\boldsymbol{T_1 \ldots T_n}$ que apontam para ela (que a citam). O parâmetro ***d*** é um fator de balanceamento que pode ser definido entre 0 e 1. Geralmente define-se ***d*** como 0.85. O número de links que saem da página ***A*** são definidos por ***C(A)***. Finalmente, o PageRank da página ***A*** é dado por (ROGERS, 2002) como:

$$PR(A) = (1 - d) + d\left(\frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_n)}{C(T_n)}\right)$$

Logo, o PageRank de uma página depende diretamente do PageRank das páginas que apontam para ela. Há a possibilidade de que as páginas que apontam para a primeira formem um ciclo, o que parece tornar o cálculo mais complexo. Mas, é possível continuar o cálculo sem saber o valor final de PageRank das outras páginas. Um pouco estranho, porém, isso se dá devido ao fato de se obter valores cada vez mais aproximados dos valores finais na medida em que se realiza os cálculos para os valores intermediários. Então, o que deve ser feito é guardar em memória cada valor calculado e repetir os cálculos exaustivamente até encontrar uma convergência para algum valor.

### 2.3.2 Algoritmo de BellmanFord

Este algoritmo busca encontrar o menor caminho em um dígrafo ponderado. Um diferencial interessante é que pode haver presença de pesos

negativos neste grafo. Pode parecer estranha a idéia de pesos negativos ao imaginar distâncias em um mapa por exemplo, mas em vários outros casos esta aplicação é de grande ajuda. É importante ressaltar que se existe um ciclo negativo no grafo (a soma dos pesos no ciclo é negativa) que é alcançável por sua origem, então esta não pode ser uma solução para o menor caminho. Isso ocorre porque cada vez que se passa por este ciclo obtêm-se um caminho mais curto, logo, o caminho mais curto ficaria eternamente realizando voltas neste ciclo (ACAR, 2013). Para tratar problemas como este, o algoritmo deve reconhecer que ciclos deste tipo existem para poder produzir indicadores a fim de encerrá-los.

Basicamente, o algoritmo retorna “falso” caso encontre um ciclo negativo ou retorna “verdadeiro” quando encontra o caminho de menor peso e o reproduz. O algortimo usa a técnica do relaxamento (encontrar um caminho mais curto do que o que foi encontrado por último), diminuindo progressivamente a estimativa do peso de um caminho mais curto (CORMEN et al., 2002). O algoritmo executa em tempo$\boldsymbol{O}(\boldsymbol{v} \times \boldsymbol{a})$, onde $\boldsymbol{v}$ corresponde ao número de vértices enquanto $\boldsymbol{a}$ ao número de arestas do grafo.

### 2.3.3 Algoritmo de Dijkstra

Bem como o algoritmo de BellmanFord, o algoritmo de Dijkstra tem como objetivo encontrar o menor caminho entre vértices de um grafo, porém, o grande diferencial entre eles é a garantia de exatidão de solução caso haja arestas negativas no grafo. Dijkstra não garante exatidão para este tipo específico de grafo. O algoritmo em si inicia com uma estimativa de custo mínimo de distância entre o vértice de origem e o vértice de destino. Este valor estimado é ajustado a cada etapa. Dado um grafo orientado $\boldsymbol{G}(\boldsymbol{V}, \boldsymbol{A})$, onde $\boldsymbol{V}$ corresponde aos vértices, $\boldsymbol{A}$ às arestas e $\boldsymbol{x}$ como um vértice qualquer de $\boldsymbol{G}$, as etapas que compõem o algoritmo de Dijkstra são:

1. Atribuir valor 0 à estimativa do custo mínimo do vértice ***x*** (raiz da busca) e infinito para as estimativas restantes;

2. Atribuir um valor qualquer aos precedentes (o precedente de um vértice ***t*** é o vértice que precede ***t*** no caminho de custo mínimo de ***x*** para ***t***);

3. Enquanto houver vértices que ainda não foram analisados:

    a. Seja ***y*** um vértice ainda não analisado cuja estimativa seja a menor dentre todos os vértices não analisados;

    b. Analisar o vértice ***y*** ;

    c. Para todo vértice ***j*** ainda aberto que seja sucessor de ***y*** faça:

        i. Some a estimativa do vértice ***y*** com o custo do arco que une ***y*** a ***j***;

        ii. Caso esta soma seja melhor que a estimativa anterior para o vértice ***j***, substitua-a e anote ***y*** como precedente de ***j*** .

Assim que todos os vértices tenham sido analisados, os valores calculados são os menores caminhos entre o vértice considerado como raiz da busca até os demais vértices do grafo.

### 2.3.4 ShortestPaths

Algoritmo que calcula as menores distâncias para cada par de vértices no grafo. Os algoritmos apresentados em 2.3.2 e 2.3.3 são algoritmos clássicos que buscam resolver o problema do caminho mais curto com implementações diferentes, mas com a mesma essência, assim como o algoritmo de ShortestPaths. Os arcabouços utilizados neste trabalho (Hadoop, Hama e Giraph) já possuem implementações prontas no JAR de exemplos de aplicações, para o algoritmo de

ShortestPaths. Estas implementações foram utilizadas sem alterações em seus códigos para as realizações de testes e execuções deste trabalho.

O trecho apresentado no Código 5 mostra o método *compute()* do algoritmo SimpleShortestPathsComputation do arcabouço Giraph (que foi utilizado nas execuções analisadas). Neste código, cria-se uma variável na linha 7 (“minDist”) para que seja implementada toda a lógica para descobrir a menor distância entre os nós. As linhas 8 e 9 definem a menor distância e atribuem o valor para a variável criada. As linhas 11, 12, 13 e 14 são resposáveis por gerar toda a informação que conterá no log da execução do algoritmo (registro de eventos ocorridos durante a execução). O restante das linhas, até a linha 25, preparam os valores obtidos até então para a continuidade do algoritmo em sua próxima iteração. Isto é feito ao enviar os cálculos para o próximo nó, definido como menor distância pela iteração atual, para que a próxima iteração carregue estes cálculos e consiga trabalhar com o valor de distância agregado. A linha 26 finaliza o método *compute()*.

```
1   public void compute(
2       Vertex<LongWritable, DoubleWritable, FloatWritable> vertex,
3       Iterable<DoubleWritable> messages) throws IOException {
4     if (getSuperstep() == 0) {
5       vertex.setValue(new DoubleWritable(Double.MAX_VALUE));
6     }
7     double minDist = isSource(vertex) ? 0d : Double.MAX_VALUE;
8     for (DoubleWritable message : messages) {
9       minDist = Math.min(minDist, message.get());
10    }
11    if (LOG.isDebugEnabled()) {
12      LOG.debug("Vertex " + vertex.getId() + " got minDist = " + minDist +
13          " vertex value = " + vertex.getValue());
14    }
15    if (minDist < vertex.getValue().get()) {
16      vertex.setValue(new DoubleWritable(minDist));
17      for (Edge<LongWritable, FloatWritable> edge : vertex.getEdges()) {
18        double distance = minDist + edge.getValue().get();
19        if (LOG.isDebugEnabled()) {
20          LOG.debug("Vertex " + vertex.getId() + " sent to " +
21              edge.getTargetVertexId() + " = " + distance);
22        }
23        sendMessage(edge.getTargetVertexId(), new DoubleWritable(distance));
24      }
25    }
26    vertex.voteToHalt();
27  }
```

**Código 5 Método compute() do código de ShortestPaths no Giraph.**
**Fonte: https://apache.googlesource.com/giraph/.**

## 2.4 TRABALHOS RELACIONADOS

Em (BADER; MADDURI, 2008) são apresentados três algoritmos executados de forma paralela. Os algoritmos são analisados conforme seus designs, implementações e performance. Os três são: pBD (***Approximate BetweennessbasedDivisive Algorithm***), pMA (***Modularitymaximizing Agglomerative Clustering Algorithm***)e pLA (***Greedy Local Aggregation Algorithm***). Para avaliá-los, a ferramenta SNAP (***Smallworld Network Analysis***

***and Partitioning***) foi utilizada. Como resultado para os algoritmos estudados, notou-se que o SNAP é mais rápido que seus concorrentes em até duas ordens de magnitude, isto permitiu analisar redes que previamente foram definidas como grandes demais para serem tratadas.

Em (DEBBARMA et al., 2014) apresenta-se uma comparação de performance do uso do Hadoop MapReduce com mais dois arcabouços: iMapReduce e HaLoop. Para realizar tal comparação, os arcabouços são submetidos a dois algoritmos iterativos baseados em grafos: PageRank e Descendant Query. As análises foram realizadas utilizando um ***cluster*** de quatro máquinas, onde cada uma possuía um processador Intel(R) i7 com CPU de 3.40 GHz, 8GB de memória RAM, 250GB de disco rígido, e utiliza o sistema operacional Linux Ubuntu 12.10 64bit. Foi visto a partir dos resultados experimentais que o HaLoop oferece melhor desempenho em comparação ao Hadoop e o iMapReduce devido ao carregamento de dados estáticos que é realizado apenas uma vez e armazenado em cache.

Em (LI et al., 2014) o arcabouço GRapid é apresentado como alternativa para geração de implementações de aplicações genéricas de grafos para CPUs multicore, GPUs NVIDIA e Intel Xeon Phi. Utilizou-se este arcabouço para gerar rapidamente implementações eficientes de aplicações de grafos em diferentes processadores. Ao fim do trabalho, apresenta-se na conclusão que o GRapid atendeu a necessidade de velocidade devido a quantidade de testes proposta, uma vez que todos os testes foram realizados em tempo hábil, coisa que não seria possível com outras ferramentas disponíveis atualmente.

Em (BÚRIGO, 2015) uma análise comparativa é apresentada utilizando vários arcabouços que implementam o algoritmo de PageRank e o algoritmo do Caminho mais Curto. As análises eram feitas a fim de coletar dados referentes ao uso de memória e tempo de execução em diferentes configurações de máquinas, incluindo um ***cluster***. A partir dos resultados obtidos pelas análises, foi determinado que para os algoritmos propostos, o arcabouço mais veloz dentre os utilizados foi o Giraph. O segundo mais veloz foi o Hama, porém um diferencial entre os dois foi o pico de memória utilizada, onde o Hama se saiu melhor ao apresentar um pico menor.

Em (SILVA, 2014) realiza-se um estudo em diferentes arcabouços que oferecem suporte à computação iterativa, como: Pregel, Giraph, Hama, Spark, GPS,

HaLoop, Twister, CIEL e GoldenOrb. Uma análise de desempenho foi desenvolvida a fim de determinar qual arcabouço é o melhor para implementar o algoritmo de PageRank, levando em consideração o tempo de execução. O resultado obtido aponta os arcabouços mantidos pela Apache e o Twister como confiáveis, passando por todas as análises previstas. Bem como o resultado de (BÚRIGO, 2015) aqui o Giraph também obteve o melhor resultado para processamento de grafos de grandes bases de dados. Segundo os testes apresentados no trabalho, para grandes bases de dados, Twister e Hama apresentaram uma queda de desempenho considerável.

# 3 FERRAMENTAS

Neste capítulo são apresentadas cada uma das ferramentas que foram usadas neste trabalho. Tratam-se de ferramentas de monitoramento de desempenho e uma ferramenta de suporte para preparação do ambiente de execuções e análises. Além destas ferramentas, o restante dos materiais que foram essenciais para o andamento do trabalho são os arcabouços, que estão apresentados no capítulo 2.

## 3.1. Ganglia

Ganglia é um sistema de monitoramento distribuído escalável para sistemas de computação de alto desempenho, como ***clusters*** e Grids. É baseado em um design hierárquico. Usa de tecnologias como XML para representação de dados, XDR para compactação, portabilidade no transporte de dados e RRDtool para armazenamento de dados e visualização. Além disto, utiliza de estruturas e algoritmos que evitam alguns problemas como: desperdícios gerais que eventualmente ocorrem nas iterações em cada nó e a alta concorrência. Sua implementação é robusta, foi portada para um amplo conjunto de sistemas operacionais e arquiteturas de processador, e está atualmente em uso em milhares de ***clusters*** ao redor do mundo. Tem sido usada para conectar ***clusters*** em universidades do mundo todo[1].

Para este trabalho, o Ganglia foi utilizado para monitorar uma máquina autônoma. Esta ferramenta de monitoramento, bem como o Munin, proporciona uma interface gráfica que facilita a extração de dados específicos de desempenho. A Figura 5 traz o gráfico de uso de CPU gerado pelo Ganglia.

[1] Fonte: http://ganglia.sourceforge.net/

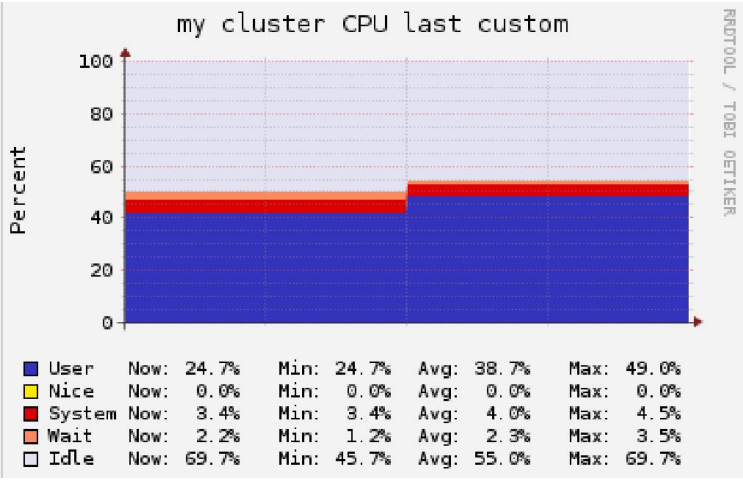



**Figura 5 Gráfico de uso de CPU.**
**Fonte: Ganglia.**

A legenda da Figura 5 possui os seguintes itens:

- User: consumo de CPU pelo usuário ativo no sistema;
- Nice: consumo de CPU por processos do usuário classificados como “nice” (classificação de prioridade de processos);
- System: consumo de CPU pelo sistema;
- Wait: consumo de CPU enquanto o processador espera instruções;
- Idle: CPU ocioso;

## 3.2 Munin

Munin é uma ferramenta de monitoramento de recursos que pode ajudar a analisar as tendências de recursos computacionais, bem como apontar problemas que ocorreram com eles. Foi escrito em Perl, porém, aceita plugins escritos em qualquer outra linguagem. As principais características desta ferramenta são[2]:

- Monitora todos os computadores e apresenta todas as informações colhidas em gráficos através de uma interface web.
- Foi desenvolvido com o objetivo de ser uma ferramenta de fácil utilização.

[2] Fonte: http://munin--monitoring.org/

- Conta com um vasto arsenal de plugins que facilitam a sua operação.
- Bem como o Ganglia, utiliza RRDTool.

O Munin possui interface gráfica para verificação de gráficos e filtragem de data e horário para que seja possível extrair com precisão os dados de desempenho em um momento específico, o que torna o processo de análise de desempenho mais fácil.

O Munin (versão 2.0.19-3.), comparado as outras ferramentas de monitoramento utilizadas neste trabalho (DSTAT e Ganglia) se destaca por trazer uma riqueza de detalhes em suas coletas de dados de desempenho. A Figura 6 mostra um pouco desta riqueza através da legenda do gráfico de uso de CPU gerado pelo Munin, que apresenta tudo que faz uso do CPU em um determinado momento.

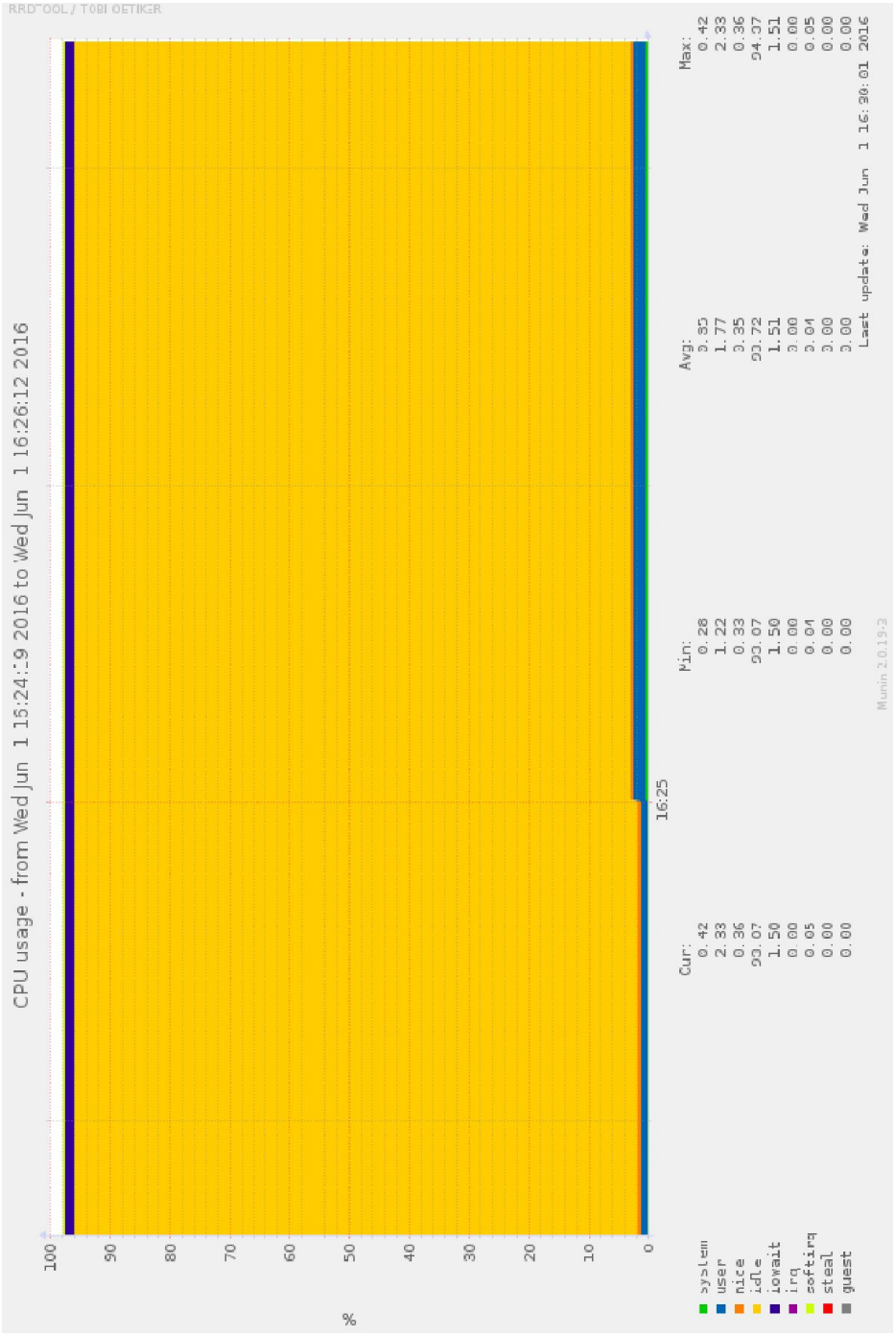



**Figura 6 Gráfico de uso de CPU.**
**Fonte: Munin.**

A legenda da Figura 6 possui os seguintes itens:

- system: consumo de CPU pelo sistema;
- user: consumo de CPU pelo usuário ativo no sistema;
- nice: consumo de CPU por processos do usuário classificados como “nice” (classificação de prioridade de processos);
- idle: CPU ocioso;

- iowait: consumo de CPU enquanto o processador espera instruções;
- irq: consumo de CPU em instruções de interrupção;
- softirq: consumo de CPU em instruções de interrupção de software;
- steal: consumo de CPU causado por esperas de CPU virtual para serem atendidos pelo CPU real;
- guest: consumo de CPU por um usuário visitante.

## 3.3 SAR

SAR é uma ferramenta de monitoramento nativa do sistema Linux, sendo um dos componentes do pacote sysstat (coleção de ferramentas de monitoramento de performance). Dentre as possibilidades de parâmetros de desempenho analisados pelo SAR, destacam-se[3]:

- Uso coletivo de processador.
- Uso individual de processador.
- Memória usada e memória disponível.
- Atividades ativas de input/output do sistema.
- Estatísticas de rede.

A versão instalada foi a 4.2.0-27-generic. Porém, a ferramenta não foi utilizada por operar de forma muito similar ao DSTAT. Optou-se então pelo uso do DSTAT, que será apresentado no próximo tópico, devido a maior disponibilidade de material encontrado nas pesquisas realizadas para embasamento.

[3] Fonte: http://www.thegeekstuff.com/2011/03/sarexamples/

## 3.4 DSTAT

DSTAT é uma ferramenta de estatística de recursos versátil. Basicamente, sua capacidade é dada pela combinação dos seguintes comandos nativos do linux:

- IOSTAT - analisa uma série de informações sobre o sistema no que diz respeito a entrada e saída. Sendo assim, é possível identificar qual disco esta sendo mais utilizado em aspectos de leitura ou gravação. Além disso, tambem fornece estatísticas do processador.

- VMSTAT - mostra uma tabela com informações sobre processos, uso de memória e paginação, entrada e saída de dados e atividades do processador.
- NETSTAT - fornece informações sobre as conexões de rede (entrada e saída), tabelas de roteamento e diversas estatísticas sobre o uso da interface de rede.

- IFSTAT - fornece informações sobre atividades de tráfego nas interfaces de rede. Além disto, pode ser utilizado para pesquisar ***hosts*** remotos através de SNMP.

Sua maior contribuição é monitorar os recursos de um servidor em temporeal. Porém, uma desvantagem em relação as outras duas ferramentas de monitoramento utilizadas no desenvolvimento deste trabalho (Ganglia e Munin) é a falta de uma interface gráfica. Os dados coletados pelo DSTAT são exibidos em tempo real no terminal do Linux. Para que seja possível realizar uma análise de desempenho nestes dados é preciso redirecionar a saída da execução do DSTAT e tratá-los.

Devido ao fato da coleta de dados ser em tempo real, o DSTAT acaba por ter mais precisão do que Munin e Ganglia, que por sua vez, fazem coletas em intervalos de tempo específicos e possivelmente maiores.

Este trabalho foi desenvolvido com o auxílio desta ferramenta e maiores detalhes a respeito de seu uso podem ser vistos no tópico 4.1.

## 3.5 ORACLE VM VIRTUAL BOX

Virtual Box é um emulador de sistemas operacionais em máquinas virtuais. Este software é bastante adequado para o contexto proposto por este trabalho, por oferecer recursos que auxiliam em seu desenvolvimento. Com ele é possível simular um ***cluster*** a partir de um único computador. Para tal, deve-se criar máquinas virtuais conforme a quantia de máquinas desejadas no ***cluster***, e fazer as devidas configurações de rede e afins de forma idêntica a configuração de um ***cluster*** real, para que estas funcionem como um ***cluster*** virtual. Além disto, o processo de instalação das ferramentas a serem utilizadas no trabalho torna-se mais seguro com o Virtual Box, pois a cada instalação bem-sucedida pode-se gerar um clone desta máquina virtual para que se caso alguma outra instalação não obtenha sucesso, o clone funcione como um ponto de restauração do sistema. A versão instalada foi a 5.0.16.

# 4 DESENVOLVIMENTO DA PESQUISA

Inicialmente, para construir um certo embasamento teórico, foram realizadas leituras e pesquisas a respeito de conceitos que envolvem os tópicos deste trabalho. Após estabelecer uma base de conhecimento, estudos sobre os algoritmos de grafos de interesse foram o próximo passo. Estes estudos forneceram as informações necessárias para a compreensão dos algoritmos utilizados no trabalho, que inclusive já foram citados no capítulo 2.

Paralelamente as duas primeiras etapas, as ferramentas de monitoramento estavam sendo instaladas e testadas para possibilitar a definição dos parâmetros de desempenho a serem analisados. As ferramentas de monitoramento em questão já foram citadas no capítulo 3 deste trabalho. Uma vez que os parâmetros de desempenho foram estabelecidos, partiu-se para a etapa de determinação de toda arquitetura computacional que foi responsável por executar os testes deste trabalho. Em seguida, foi desenvolvido um protocolo (apresentado no tópico 4.3, Figura 16) para ditar o procedimento a ser adotado para a realização das análises das execuções, de forma a padronizá-los. Então, os arcabouços foram instalados, permitindo o início da implementação dos algoritmos.

Finalmente, foi realizado um teste, apresentado no tópico 4.1 para garantir que todas as instalações e configurações estavam corretas. Com o sucesso do teste, pôde-se então executar os algoritmos. Enquanto os algoritmos eram executados, as ferramentas de monitoramento faziam suas coletas, compondo a etapa de coleta de dados. Os resultados coletados foram estruturados, para que posteriormente fossem comparados com os resultados de outras análises. A partir das informações obtidas pela comparação dos resultados foi possível encontrar as respostas para as perguntas levantadas no tópico 1.1 deste trabalho.

## 4.1 TESTE DE EXECUÇÃO DE ALGORITMO NO HADOOP

Para este teste, foram usados os seguintes recursos: Ubuntu 14.04, Hadoop 2.6, DSTAT, Gnuplot, Eclipse 3.8.1-5.1 e a implementação do algoritmo de PageRank.

Após a instalação dos arcabouços Hadoop, Hama e Giraph, além de ter as ferramentas de monitoramento prontas, foi realizado um teste para verificar se todos os softwares foram configurados da maneira correta. Para tal, um grafo foi gerado, tendo 15 nós, conforme mostra a Figura 7:

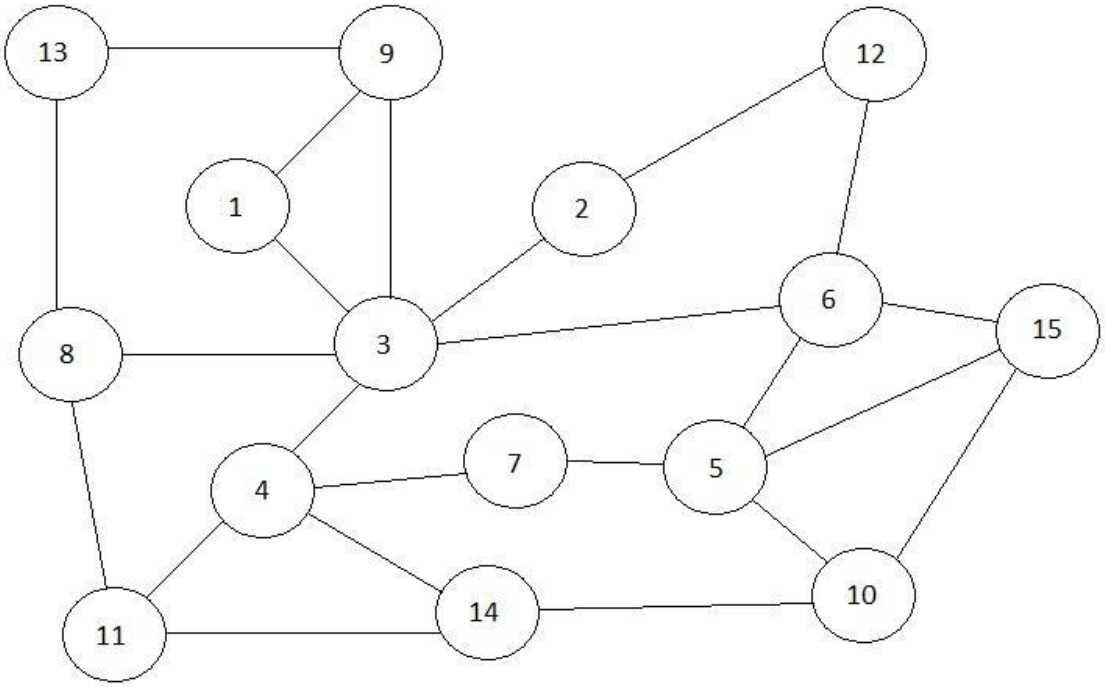



**Figura 7 Grafo gerado para teste.**
**Fonte: Autoria própria.**

Com o grafo pronto gerou-se um arquivo texto para representá-lo como uma lista de adjacência, com o intuito de utilizá-lo como entrada para o algoritmo de PageRank. Em seguida foi criada uma pasta no HDFS para armazenar o arquivo de entrada criado, com o seguinte comando no terminal do Ubuntu:

```
hadoop dfs -mkdir -p /usr/local/hadoop/input
```

e então, o mesmo foi movido para a nova pasta:

```
hadoop dfs -copyFromLocal /home/hduser/teste/grafoTeste.txt /usr/local/hadoop/input
```

com o algoritmo de PageRank já implementado, foi gerado um JAR do código na IDE Eclipse (“pagerank.jar”) para que ele fosse executado no arcabouço Hadoop. Partiu-se então para a execução do PageRank no Hadoop, com o seguinte comando:

```
hadoop jar /usr/local/jars/pagerank.jar --input /usr/local/hadoop/input --output /usr/local/hadoop/outputteste2 & nohup dstat -tcmn 1 > dstat.dat &
```

nota-se no comando utilizado, que o DSTAT é iniciado simultaneamente com a execução do PageRank, porém, em ***background,*** devido ao parâmetro "nohup" utilizado. A saída gerada pelo DSTAT é redirecionada para um novo arquivo "dstat.dat". Quando a execução do código termina, deve-se encerrar a execução do DSTAT com o comando:

```
kill -9 <numero_do_processo_do_DSTAT>
```

Após estes passos, foram gerados três ***shell scripts*** para manipular a saída contida no arquivo "dstat.dat", gerando três gráficos com o Gnuplot. O primeiro (Código 6) tratava das estatísticas referentes ao monitoramento da CPU, o segundo (Código 7) das estatísticas referentes a memória e o terceiro (Código 8) das estatísticas referentes ao uso da rede. Todos estes ***scripts*** configuram os títulos dos gráficos (“set title”) e os títulos dos eixos dos gráficos (“set xlabel” e “set ylabel”) a serem gerados. As linhas dos códigos iniciadas com “plot” configuram as colunas onde se encontram os valores de interesse, ou seja, para o gráfico de uso de CPU por exemplo, a linha iniciada com “plot” do ***script*** correspondente configura a coluna onde estão relacionados os valores coletados para uso de CPU.

```
#!/usr/bin/gnuplot
set terminal png
set output 'cpu.png'
set title 'Uso de CPU'
set xlabel 'Tempo'
set ylabel 'Porcentagem'
set xdata time
set timefmt '%d-%m %H:%M:%S'
set format x '%H:%M'
plot 'dstat.dat' using 1:3 title 'Sistema' with lines, \
'dstat.dat' using 1:2 title 'Usuário' with lines, \
'dstat.dat' using 1:4 title 'Ocioso' with lines
```

**Código 6 Script para gerar gráfico de uso de CPU com Gnuplot.**
**Fonte: Autoria própria.**

```
#!/usr/bin/gnuplot
set terminal png
set output 'memory.png'
set title 'Uso de Memória'
set xlabel 'Tempo'
set ylabel 'Tamanho(MB)'
set xdata time
set timefmt '%d-%m %H:%M:%S'
set format x '%H:%M'
plot 'dstat.dat' using 1:8 title 'Usada' with lines, \
'dstat.dat' using 1:9 title 'Buffer' with lines, \
'dstat.dat' using 1:10 title 'Cache' with lines, \
'dstat.dat' using 1:11 title 'Livre' with lines
```

**Código 7 Script para gerar gráfico de uso de memória com Gnuplot.**
**Fonte: Autoria própria.**

```
#!/usr/bin/gnuplot
set terminal png
set output 'network.png'
set title 'Tráfego de Rede'
set xlabel 'Tempo'
set ylabel 'Tamanho(k)'
set xdata time
set timefmt '%d-%m %H:%M:%S'
set format x '%H:%M'
plot 'dstat.dat' using 1:11 title 'Enviado' with lines, \
'dstat.dat' using 1:12 title 'Recebido' with lines
```

**Código 8 Script para gerar gráfico de uso de memória com Gnuplot.**
**Fonte: Autoria própria.**

Todos os scripts foram executados, e os gráficos gerados podem ser vistos na Figura 8, na Figura 9 e na Figura 10:

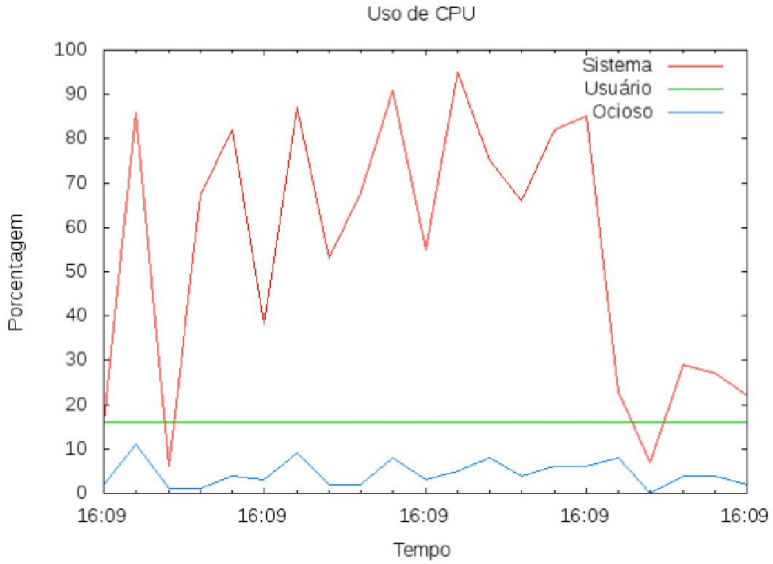



**Figura 8 Gráfico de tempo x porcentagem de uso da CPU no teste do algoritmo PageRank.**
**Fonte: DSTAT.**

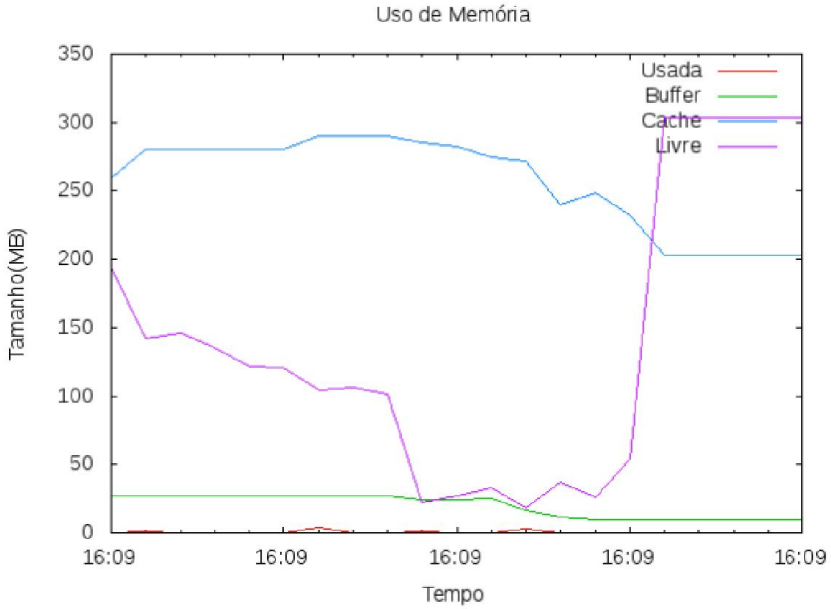



**Figura 9 Gráfico de tempo x quantidade de memória RAM utilizada no teste do algoritmo PageRank.**
**Fonte: DSTAT.**

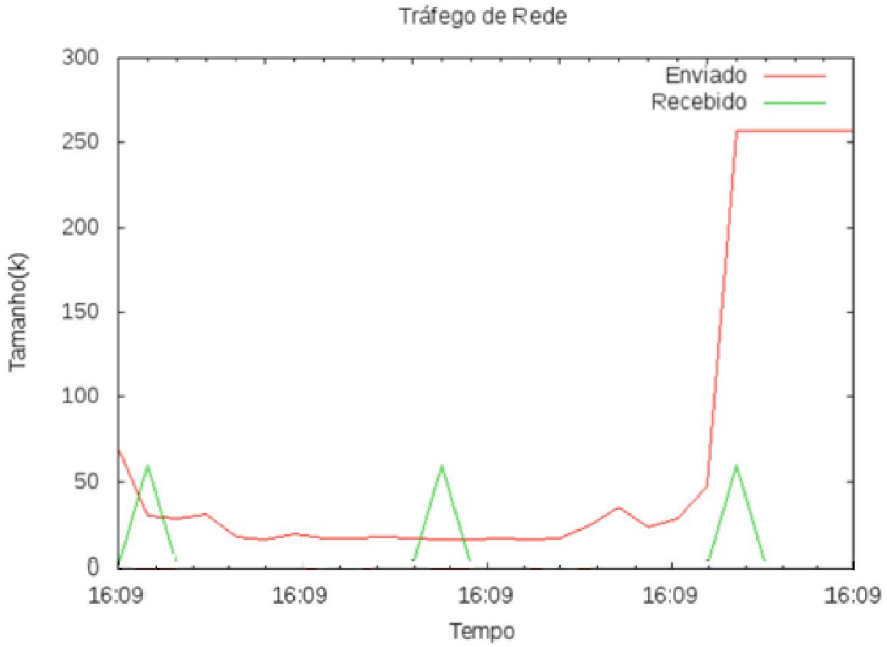



**Figura 10 Gráfico de tempo x quantidade de tráfego na rede no teste do algoritmo PageRank. Fonte: DSTAT.**

Por fim, para verificar a saída gerada pelo JAR executado no Hadoop, utilizou-se o comando mostrado na Figura 11:

```
hduser@michael-tcc:~/dstat/testes$ hdfs dfs -cat /usr/local/hadoop/outputteste2/
result/part-r-00000
16/04/26 20:45:19 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
0.24189738929271698     7
0.2444562464952469      1
0.2645689845085144      13
0.26513761281967163     2
0.27206650376319885     12
0.2960543930530548      15
0.3052578270435333      10
0.30554214119911194     14
0.3066793978214264      11
0.3184416592121124      8
0.3374171257019043      9
0.3653854727745056      6
0.37003982067108154     4
0.377537339925766       5
0.5061430931091309      3
```

**Figura 11 Exibição da saída gerada pelo algoritmo PageRank. Fonte: Autoria própria.**

Pode-se verificar que a saída é composta por duas colunas, onde a primeira representa o resultado calculado do PageRank e a segunda representa o nó em questão. O resultado é exibido de forma ordenada, levando em consideração

os valores crescentes de PageRank.

Com o sucesso deste teste, foi possível prosseguir com a execução do algoritmo de PageRank para as entradas das bases de dados propostas para este trabalho.

## 4.2 BASES DE DADOS

Para a execução dos algoritmos nos arcabouços, utilizou-se dois grafos direcionados retirados da internet. Os grafos escolhidos foram o "web-NotreDame" e o "web-Stanford", este último foi utilizado também em (BÚRIGO, 2015). O primeiro trata-se de um grafo criado pela University of NotreDame onde os vértices representam páginas web da universidade e as arestas representam hiperlinks entre as páginas. Da mesma forma, o segundo grafo também representa páginas da internet, mas desta vez são páginas da universidade de Stanford, da Califórnia, e suas arestas representam os hiperlinks existentes entre elas. O primeiro grafo foi gerado em 1999, enquanto que o segundo foi em 2002 e algumas de suas estatísticas principais estão expostas na Tabela 2 (web-NotreDame) e na Tabela 3 (web-Stanford).

**Tabela 2 Estatísticas do grafo web-NotreDame.**
**Fonte: http://snap.stanford.edu/data/web-NotreDame.html**

| Estatísticas web-NotreDame | |
|---|---|
| **Vértices** | 325729 |
| **Arestas** | 1497134 |
| **Vértices no maior WCC** | 325729 (100%) |
| **Arestas no maior WCC** | 1497134 (100%) |
| **Vértices no maior SCC** | 53968 (16.6%) |
| **Arestas no maior SCC** | 304685 (20.4%) |
| **Coeficiente de agrupamento** | 23.46% |
| **Número de triângulos** | 8910005 |
| **Fração de triângulos fechados** | 3.1% |
| **90-percentile effective diameter** | 9.4 |

**Tabela 3 Estatísticas do grafo web-Stanford.**
**Fonte: http://snap.stanford.edu/data/web-Stanford.html**

| Estatísticas web-Stanford | |
|---|---|
| **Vértices** | 281903 |
| **Arestas** | 2312497 |
| **Vértices no maior WCC** | 255265 (90.6%) |
| **Arestas no maior WCC** | 2234572 (96.6%) |
| **Vértices no maior SCC** | 150532 (53.4%) |
| **Arestas no maior SCC** | 1576314 (68.2%) |
| **Coeficiente de agrupamento** | 59.76% |
| **Número de triângulos** | 11329473 |
| **Fração de triângulos fechados** | 0.29% |
| **90-percentile effective diameter** | 9.7 |

A seguir, uma breve descrição de cada uma das características informadas pelas tabelas apresentadas:

- Vértices: quantidade total de vértices, ou nós, que o grafo possui;
- Arestas: quantidade total de arestas, ou arcos, que o grafo possui;
- Vértices no maior WCC: quantidade de vértices no maior componente

fracamente ligado;

- Arestas no maior WCC: quantidade de arestas no maior componente fracamente ligado;
- Vértices no maior SCC: quantidade de vértices no maior componente fortemente ligado;
- Arestas no maior SCC: quantidade de arestas no maior componente fortemente ligado;
- Coeficiente de agrupamento: estatística que representa a tendência de agrupamento dos vértices do grafo com demais vértices;
- Número de triângulos: quantidade de triângulos formados através das arestas que unem os vértices do grafo;
- Fração de triângulos fechados: proporção da quantidade de triângulos que foram fechados com o total de possibilidades de triângulos que poderiam ser fechados;
- 90-percentile effective diameter: quantidade de arestas necessárias para atingir 90% dos vértices do grafo.

Apesar da verificação de cada uma das características, não houve aprofundamento na relaçãodesses indicadores com os resultados obtidos neste trabalho, exceto os dois primeiros, quantidade de vértices e arestas.

Os dois grafos estavam representados com a mesma estrutura em arquivos de texto, com algo parecido com uma lista de adjacência, forma de representação de grafos que foi explicada no capítulo 2 deste trabalho. A Figura 12 ilustra o trecho inicial do arquivo "web-Stanford.txt".

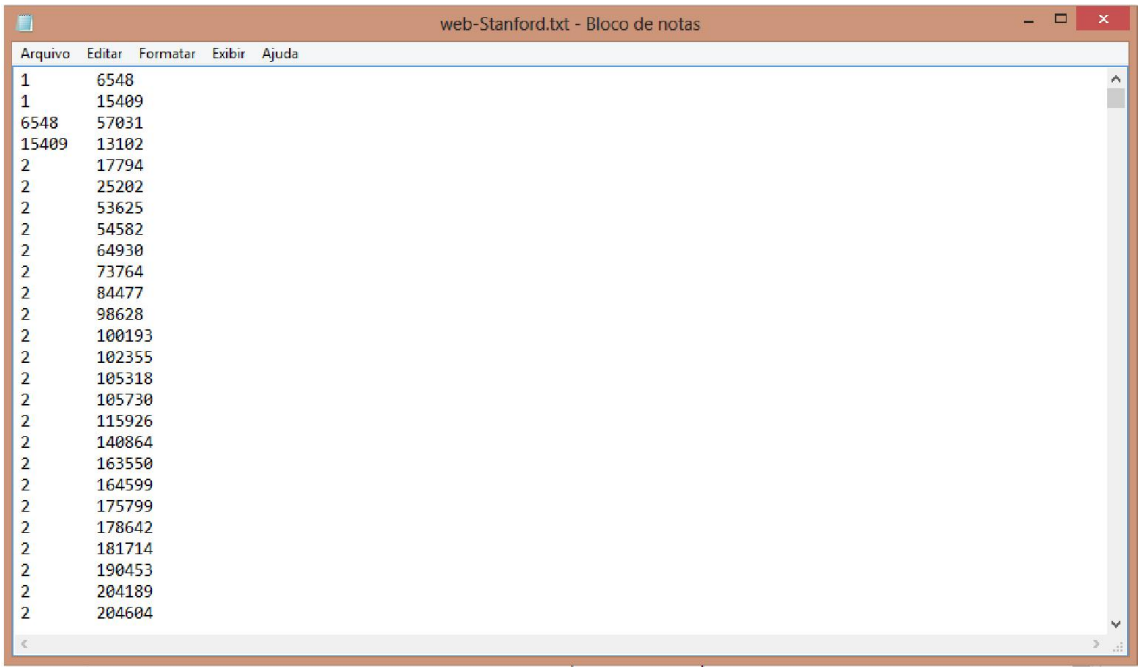



**Figura 12 Trecho do arquivo web-Stanford.txt.**
**Fonte:http://snap.stanford.edu/data/web-Stanford.html.**

É possível notar, através da figura, que o grafo é representado por duas colunas de números. A primeira coluna faz referência ao vértice origem de uma aresta, enquanto que a segunda mostra o vértice destino desta aresta. Ou seja, o arquivo está preenchido com os registros das arestas que o grafo possui. O grafo "web-Stanford" está representado com a mesma estrutura da Figura 12.

Com estes dois arquivos de texto foi possível executar o algoritmo de PageRank no arcabouço Hadoop, porém, os outros arcabouços executam este mesmo algoritmo utilizando uma entrada de dados com outra formatação. Além disso, outros algoritmos exigem entradas de dados diferentes, como por exemplo o algoritmo ShortestPaths, que espera pesos de arestas como um dos parâmetros de entrada. Como podemos ver na Figura 12, os arquivos originais de representação dos grafos não informam os pesos das arestas. Devido a esta particularidade dos arcabouços e dos algoritmos, foi necessário desenvolver códigos em linguagem C para ler e formatar os arquivos dos dois grafos e, desta forma, gerar arquivos novos com formatação compatível com a entrada de dados de cada arcabouço. Um destes códigos pode ser visto em Código 9, que foi utilizado para gerar o arquivo web-NotreDame-tipo3.txt. Neste código, o arquivo original da base de dados “web-NotreDame.txt” é aberto para leitura (parâmetro “r” na função fopen) na linha 7 e um

arquivo é criado e aberto na linha 8 para que nele seja inserido a formatação compatível com o arcabouço Giraph. A linha 13 atribui a duas váriaveis, os dois primeiros números do arquivo original. Estes dois valores são inseridos, conforme a exigência do arcabouço, no novo arquivo através da linha 16. Nas linhas de 19 a 27 são inseridos os números que correspondem aos vértices de destino do primeiro número inserido na linha, que é o vértice de origem. Ao finalizar os vértices de destino deste determinado vértice de origem, parte-se para as inserções das próximas linhas, ao seguir a mesma lógica explanada nas linhas de 33 a 56.

```c
#include <stdio.h>
#include <stdlib.h>

int main (void){
    FILE *file;
    FILE *fileO;
    file = fopen("web-NotreDame.txt", "r");
    fileO = fopen("web-NotreDame-tipo3.txt", "w");

    int x1, x2, y1, y2, i=0, random;
    char *ch;

        fscanf(file, "%i %i\n", &x1, &y1);

        random = rand() % 100 + 1;
        fprintf(fileO, "[%i,0,[[%i,0],[%i,%i]", x1, x1, y1, random);

        fscanf(file, "%i", &x2);
        if(x2 == x1){
            while(x2 == x1){
                fscanf(file, "%i\n", &y2);

                random = rand() % 100 + 1;
                fprintf(fileO, ",[%i,%i]", y2, random);
                fscanf(file, "%i", &x2);
            }
        }

        fprintf(fileO, "]]\n");
        fscanf(file, "\n");
        x1 = x2;

        while(x1!= 325728 && i!=3){
            if(x1 == 325728)
                i++;

            fscanf(file, "%i\n", &y1);

            random = rand() % 100 + 1;
            fprintf(fileO, "[%i,0,[[%i,0],[%i,%i]", x1, x1, y1, random);

            fscanf(file, "%i", &x2);
            if(x2 == x1){
                while(x2 == x1){
                    fscanf(file, "%i\n", &y2);

                    random = rand() % 100 + 1;
                    fprintf(fileO, ",[%i,%i]", y2, random);
                    fscanf(file, "%i", &x2);
                }
            }

            fprintf(fileO, "]]\n");
            fscanf(file, "\n");
            x1 = x2;
        }

    fclose(file);
    fclose(fileO);

    system("pause");
    return 0;
}
```

**Código 9 Código implementado para gerar o arquivo web-NotreDame-tipo3.txt.**
**Fonte: Autoria própria.**

Três estruturas de representação de grafo diferentes foram necessárias,

portanto, seis códigos foram desenvolvidos para realizar esta tarefa (três para cada base de dados). Ao fim, foram gerados seis arquivos, três tipos (formatos) para cada grafo: o Tipo 1 apresentado pela Figura 13, o Tipo 2 apresentado pela Figura 14 e o Tipo 3 apresentado pela Figura 15.

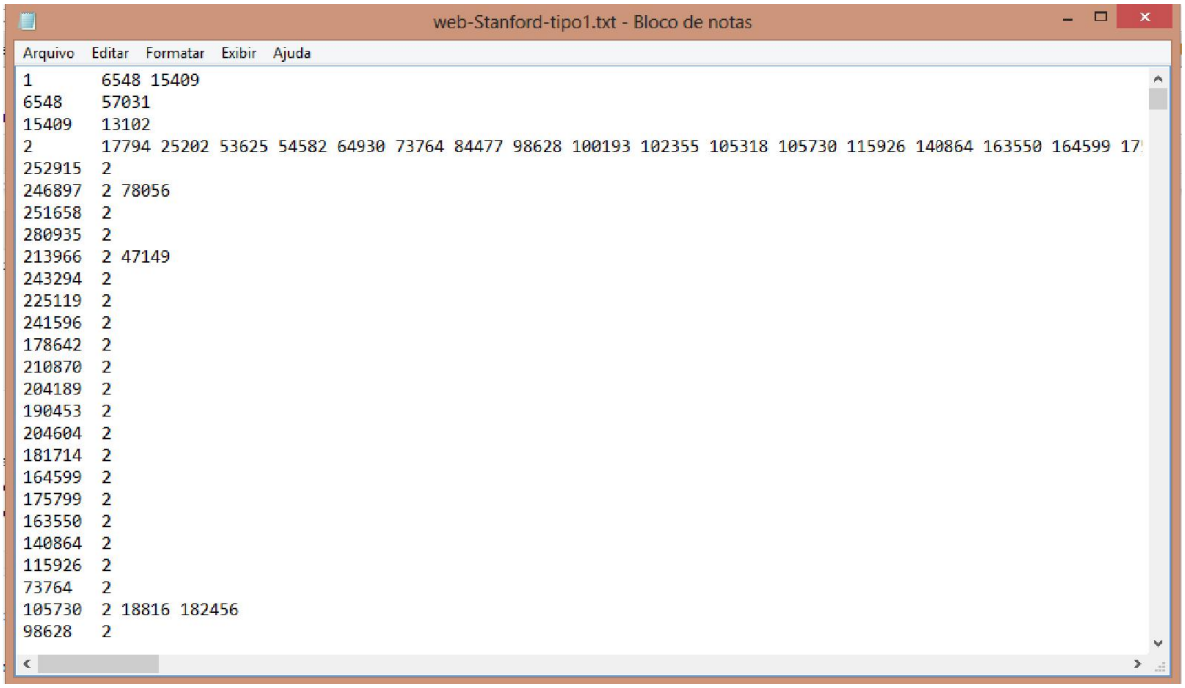



**Figura 13 Trecho do arquivo web-Stanford-tipo1.txt.**
**Fonte:Autoria própria.**

Neste formato do Tipo 1, a primeira coluna de números representa os vértices de origem. As colunas seguintes representam os vértices de destino do vértice da primeira coluna. Ou seja, cada linha deste arquivo representam todas as arestas de um determinado vértice, que é especificado na primeira coluna das linhas.

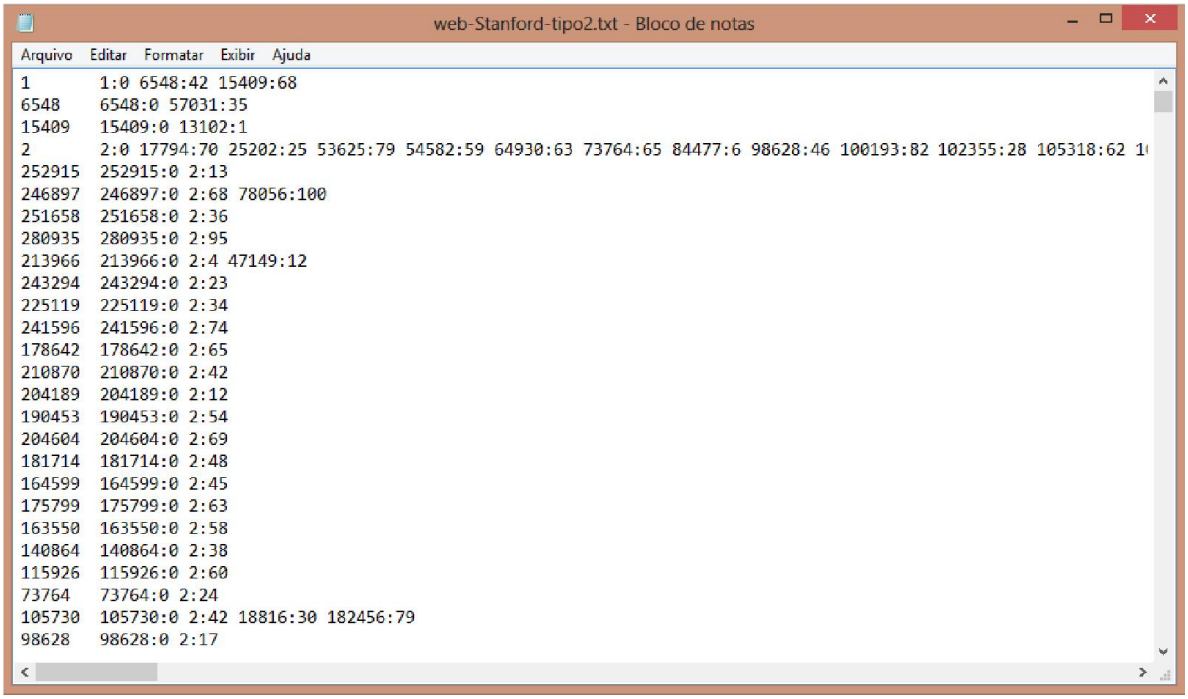



**Figura 14 Trecho do arquivo web-Stanford-tipo2.txt.**
**Fonte:Autoria própria.**

Neste formato do Tipo 2 foram inseridos pesos para as arestas do grafo. Os pesos foram gerados de forma aleatória para cada aresta, respeitando um intervalo de 1 a 100. A primeira coluna de números representa os vértices de origem. As colunas seguintes representam os vértices de destino do vértice da primeira coluna, seguidos do peso da aresta que liga estes dois vértices. Logo, as linhas deste arquivo seguem o seguinte padrão:

Vo Vd1:Pa1 Vd2:Pa2 Vd3:Pa3 Vdn:Pan

onde:

$V_o$ = Vértice de origem;

$V_{dn}$ = Vértice de destino *n*;

$P_{an}$ = Peso da aresta *n* que faz a ligação entre $V_{dn}$ e $V_o$.

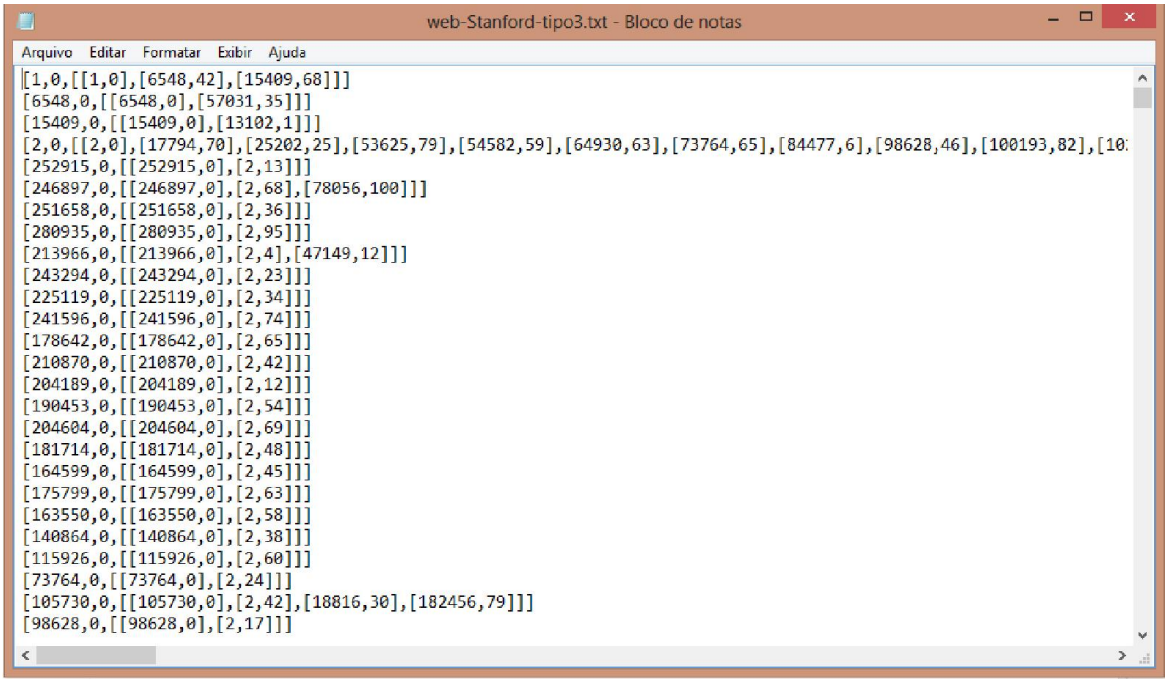



**Figura 15 Trecho do arquivo web-Stanford-tipo3.txt.**
**Fonte:Autoria própria.**

Neste formato do Tipo 3 os pesos do formato do Tipo 2 foram mantidos. O padrão seguido neste arquivo é:

[Vo,Pvo,[[Vd1,Pa1],[Vd2,Pa2],[Vdn,Pan]]]

onde:

$V_o$ = Vértice de origem;

$P_{vo}$ = Peso do vértice de origem;

$V_{dn}$ = Vértice de destino *n*;

$P_{an}$ = Peso da aresta *n* que faz a ligação entre $V_{dn}$ e $V_o$.

## 4.3 PROTOCOLO DE EXECUÇÃO DOS ALGORITMOS

Para realizar as execuções dos algoritmos de forma a não influenciar nos resultados que seriam obtidos, foi desenvolvido um protocolo que dita exatamente os passos que devem ser adotados para tal procedimento. Assim, todas as

execuções serão realizadas respeitando um procedimento padrão. A Figura 16 apresenta um fluxograma que ilustra a ordem dos passos das execuções dos algoritmos nos arcabouços.

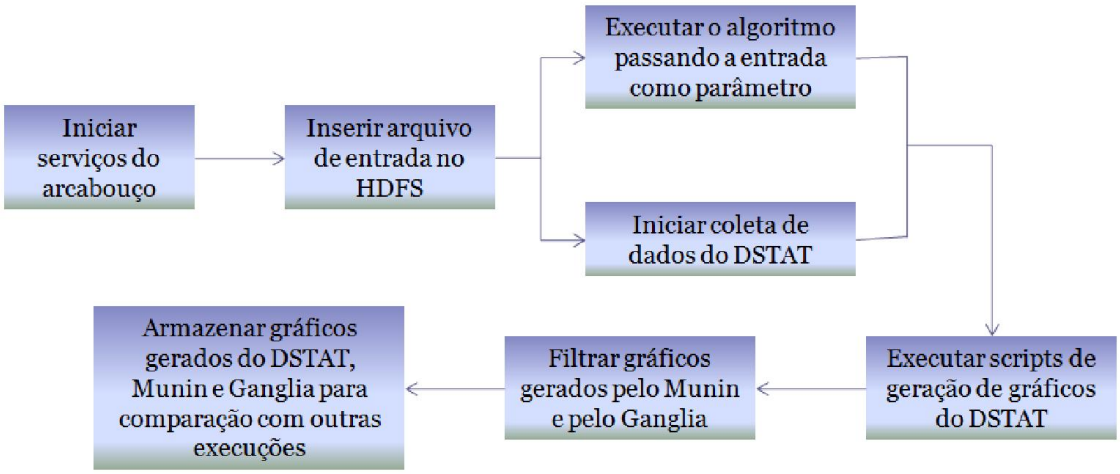



**Figura 16 Fluxograma do protocolo de execução.**
**Fonte:Autoria própria.**

Observações sobre o fluxograma da Figura 16:

- As execuções nos arcabouços Hadoop, Hama e Giraph foram realizadas com o uso do HDFS, que é o sistema de arquivos distribuído do Hadoop;
- O DSTAT trabalha de forma diferente dos outros dois softwares (Munin e Ganglia), que possuem interface web para apresentar seus gráficos e oferecer ferramentas de filtragem para estes gráficos. Portanto, para facilitar a coleta de dados com o DSTAT e não haver necessidade de filtragem, o mesmo é iniciado em sincronismo com a execução do algoritmo;
- A etapa de filtragem dos gráficos gerados pelo Munin e pelo Ganglia é feita a partir do horário de início e fim da execução do algoritmo, registrados pelo DSTAT.

## 4.4 EXECUÇÕES DOS ALGORITMOS

Para aumentar a confiabilidade dos resultados obtidos nas execuções dos

algoritmos, decidiu-se por executar cada combinação (arcabouço / algoritmo / base de dados) cinco vezes cada. Com os cinco resultados prontos, analisou-se os tempos de execução individuais a fim de descartar os resultados obtidos com o maior e menor tempo. Os próximos tópicos apresentam tabelas (Tabela 4, Tabela 5, Tabela 6 e Tabela 7) que mostram estes números em segundos e destacam as execuções que foram descartadas.

A ideia por trás deste procedimento é descartar execuções que por algum motivo foram influenciadas e tiveram seu desempenho comprometido, seja por algum processo executado pelo SO, ou por algum processo executado pelo navegador, que se encontrava aberto durante as execuções para o acompanhamento do sistema de arquivos HDFS, ou por qualquer outro motivo que não esteja relacionado diretamente à execução do algoritmo.

Ao fim da etapa de descarte de execuções, sobraram três execuções. Os dados coletados destas execuções que restaram foram utilizados para que se extraísse uma média de valores de interesse (uso de memória, uso de CPU e tráfego de rede), incluindo o próprio tempo de execução (Tabela 8). Desta forma, a média obtida representa um valor mais confiável quando comparado ao valor de uma única execução de um determinado algoritmo. Estes resultados podem ser vistos no Capítulo 5 deste trabalho.

### 4.4.1 PageRank

- web-Stanford

**Tabela 4 Tempo de execução em segundos do PageRank na base web-Stanford nos respectivos arcabouços.**
**Fonte: Autoria própria**

| | EXECUÇÃO | | | | |
|---|---|---|---|---|---|
| **ARCABOUÇO** | **1** | **2** | **3** | **4** | **5** |
| **Giraph** | 254 | 193 | 198 | 209 | 194 |
| **Hadoop** | 65 | 79 | 88 | 70 | 71 |
| **Hama** | 67 | 73 | 70 | 66 | 64 |

- web-NotreDame

**Tabela 5 Tempo de execução em segundos do PageRank na base web-NotreDame nos respectivos arcabouços.**
**Fonte: Autoria própria**

| | EXECUÇÃO | | | | |
|---|---|---|---|---|---|
| **ARCABOUÇO** | **1** | **2** | **3** | **4** | **5** |
| **Giraph** | 229 | 226 | 210 | 214 | 231 |
| **Hadoop** | 59 | 54 | 56 | 67 | 55 |
| **Hama** | 49 | 73 | 68 | 75 | 64 |

### 4.4.2 ShortestPaths

- web-Stanford

**Tabela 6 Tempo de execução em segundos do ShortestPaths na base web-Stanford nos respectivos arcabouços.**
**Fonte: Autoria própria**

| | EXECUÇÃO | | | | |
|---|---|---|---|---|---|
| **ARCABOUÇO** | **1** | **2** | **3** | **4** | **5** |
| **Giraph** | 61 | 80 | 89 | 61 | 56 |
| **Hadoop** | 168 | 187 | 175 | 172 | 181 |
| **Hama** | 171 | 175 | 164 | 159 | 159 |

- web-NotreDame

**Tabela 7 Tempo de execução em segundos do ShortestPaths na base web-NotreDame nos respectivos arcabouços.**
**Fonte: Autoria própria**

| | EXECUÇÃO | | | | |
|---|---|---|---|---|---|
| **ARCABOUÇO** | **1** | **2** | **3** | **4** | **5** |
| **Giraph** | 67 | 69 | 64 | 65 | 69 |
| **Hadoop** | 109 | 113 | 122 | 102 | 115 |
| **Hama** | 114 | 105 | 105 | 99 | 96 |

### 4.4.3 Tempo de Execução Médio dos Algoritmos

A Tabela 8 apresenta os valores médios de tempo de execução extraídos das três execuções restantes após o procedimento de descarte. Os valores apresentados foram adotados como valor final para a execução de cada combinação de arcabouço (Hadoop, Hama e Giraph), algoritmo (PageRank e ShortestPaths) e base de dados (web-Stanford e web-NotreDame).

**Tabela 8 Tempo de execução médio, em segundos, dos algoritmos nas bases de dados nos respectivos arcabouços.**
**Fonte: Autoria própria**

| ALGORITMOS | Bases de Dados | ARCABOUÇOS | | |
|---|---|---|---|---|
| | | Giraph | Hadoop | Hama |
| PageRank | web-Stanford | 200 | 73 | 68 |
| | web-NotreDame | 223 | 57 | 68 |
| ShortestPaths | web-Stanford | 67 | 176 | 165 |
| | web-NotreDame | 67 | 112 | 103 |

# 5 RESULTADOS OBTIDOS

Os arcabouços foram instalados em máquinas virtuais a partir do uso do VirtualBox. Hadoop foi instalado em duas versões: versão 2.6 para realizar suas execuções e para as execuções em conjunto com o Hama, e versão 0.20.203 para as execuções em conjunto com o Giraph. O Hama foi instalado na versão 0.7.1. O Giraph foi instalado na versão 1.2.0. Hama e Giraph fazem uso de alguns recursos que estão implementados no Hadoop, por este motivo há a necessidade de sua instalação para que seja possível realizar execuções em ambos. Foram utilizadas versões diferentes de Hadoop para Hama e Giraph devido a problemas de compatibilidade. O Giraph não funcionou com a versão 2.6 do Hadoop, então, instalou-se a versão 0.20.203 por ser a versão indicada no guia de instalação disponível no site oficial do arcabouço em questão (Giraph).

Durante a preparação das execuções, houve a preocupação em iniciar somente os serviços necessários para realizar a aplicação.

Todas as execuções foram monitoradas por Munin, Ganglia e DSTAT. Todos os gráficos de uso de memória, uso de CPU e tráfego de rede foram gerados e filtrados. Os dados coletados de tráfego de rede foram desconsiderados devido a não implementação de um ***cluster*** para realizar as execuções. Notou-se que o DSTAT se mostrou mais preciso em suas coletas ao apresentar dados em um intervalo menor de tempo (um segundo). Além disto, segundo o próprio site dos desenvolvedores, Munin e Ganglia são indicados para monitoramento de ***clusters***.

Os resultados apresentados a seguir foram obtidos a partir de análises de dados coletados de execuções realizadas em uma única máquina. Esta máquina possui a seguinte configuração:

Memória - pente único de 8 GB Corsair 1600 MHz DDR3.

Processador - Intel ® Core (™) i5-4570 3.2 GHz 6 MB cache L3.

Placa de vídeo - Gigabyte GeForce GTX660 Ti 3072MB DDR5.

Disco rígido - Seagate SATA 3 500 GB 7200 RPM 6 GB/s.

Placa-mãe - ASUS H81M-C/BR.

Nesta máquina foram instaladas as duas máquinas virtuais, ambas com 2 GB de memória e 60 GB de disco rígido.

É importante destacar e ressaltar que todas as execuções foram

realizadas nas configurações padrões, definidas nos códigos nativos de cada arcabouço.

## 5.1 PAGERANK - WEB-STANFORD

As análises deste tópico são baseadas nos gráficos da Figura 17 (uso de CPU) e nos gráficos da Figura 18 (uso de memória).

Para a execução do algoritmo de PageRank na base web-Stanford nos arcabouços, notou-se primeiramente que o arcabouço Hama obteve o melhor resultado relacionado a tempo de execução, levando uma média de 68 segundos para concluir suas iterações.

Além disto, o Hama também foi o arcabouço que menos fez uso de CPU, seguido de perto pelo Hadoop, que teve um desempenho similar quando comparado ao tempo de execução do algoritmo e uso de CPU. Ambos (Hama e Hadoop) tiveram um desempenho muito satisfatório se comparados ao terceiro arcabouço, o Giraph. Este último, por sua vez, executou o algoritmo na base web-Stanford com uma média de 200 segundos, o que representa quase três vezes o tempo que os dois primeiros precisaram para fazer o mesmo procedimento. Ao analisar o uso de CPU percebe-se que o desempenho do Giraph para esta execução foi de fato abaixo dos outros arcabouços analisados, pois na maior parte do tempo de execução o gráfico mostra que o processador estava entre 85% e 95% de uso.

No que diz respeito ao uso de memória, os três arcabouços apresentaram desempenho um tanto quanto parecido. Todos eles mantiveram o uso de memória abaixo de 1.000 MB na maior parte da execução, com exceção do Hadoop que manteve o uso abaixo de 500 MB durante toda a execução. A diferença entre o desempenho do Hadoop e os outros arcabouços analisados é dada por alguns picos de uso de memória que ocorreram em certos momentos da execução realizada pelo Hama e pelo Giraph.

Em resumo, a análise desta execução trouxe os seguintes resultados:

● Hadoop - 73 segundos de tempo médio de execução. O uso de CPU variou bastante, e esteve entre 20% e 40% em uma boa parte da execução, porém, na maior parte da execução esteve entre 90% e 100%. O pico de uso de CPU atingiu aproximadamente 98%. O uso de memória esteve abaixo de 500 MB a execução toda. O pico de uso de memória atingiu aproximadamente 430 MB em cache;

● Giraph - 200 segundos de tempo médio de execução. O uso de CPU esteve entre 85% e 95% na maior parte da execução. O pico de uso de CPU atingiu aproximadamente 98%. O uso de memória esteve abaixo de 500 MB na maior parte da execução, porém se manteve acima de 2.000 MB com uso de memória em buffer em boa parte da execução. O pico de uso de memória atingiu aproximandamente 9.700 MB em cache, outro pico importante foi em buffer, de aproximadamente 8.000 MB;

● Hama - 68 segundos de tempo médio de execução. O uso de CPU esteve entre 60% e 70% na maior parte da execução. O pico de uso de CPU atingiu aproximadamente 97%. O uso de memória esteve abaixo de 500 MB na maior parte da execução, porém se manteve acima de 2.000 MB com uso de memória em buffer em boa parte da execução. O pico de uso de memória atingiu aproximandamente 9.700 MB em buffer;

Conclui-se então, que para esta execução, a partir da análise realizada o arcabouço Hama obteve o melhor desempenho. O pior desempenho foi analisado nos resultados obtidos pelo arcabouço Giraph, enquanto que o Hadoop obteve um resultado muito próximo do arcabouço Hama.

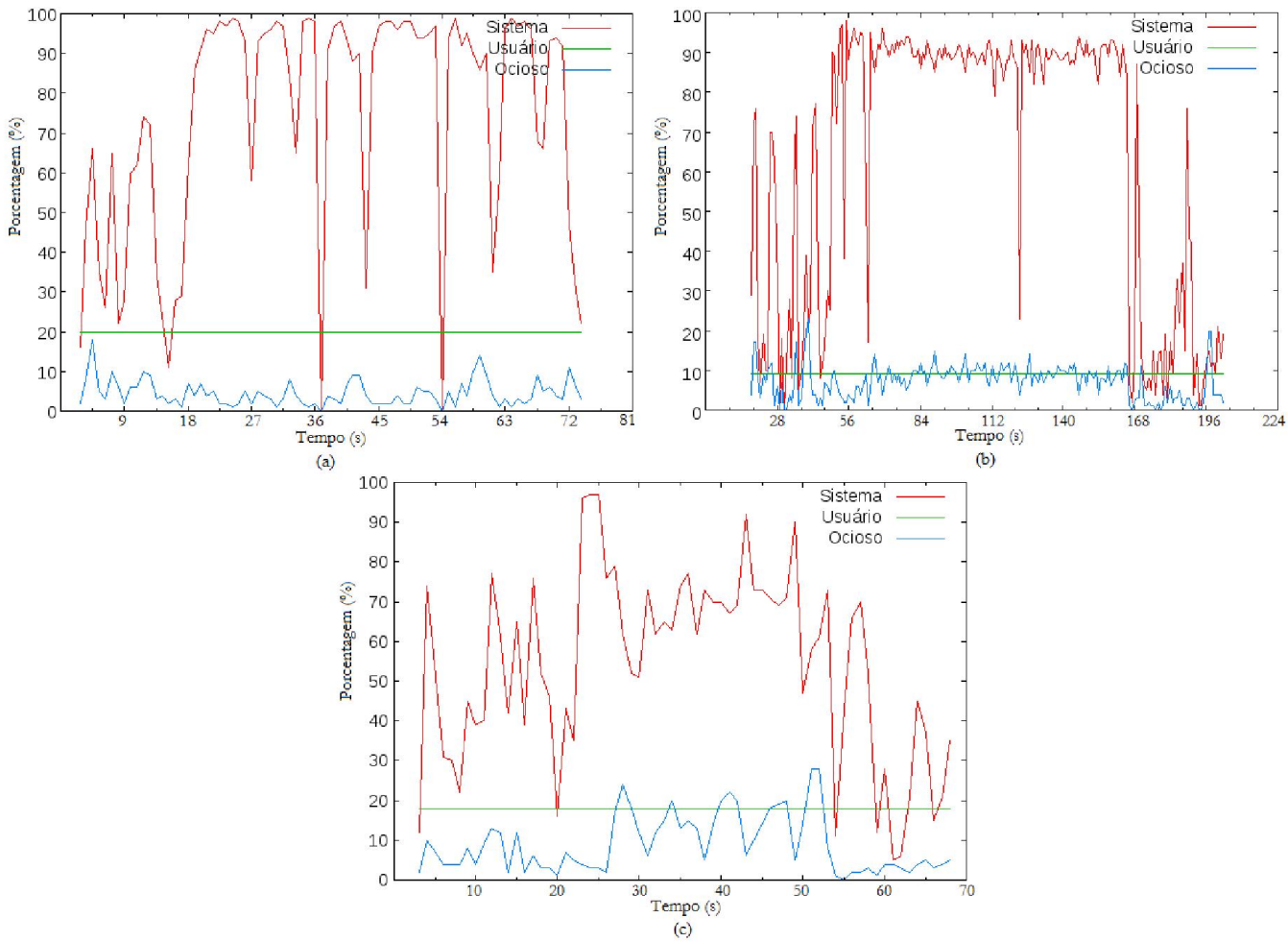



**Figura 17 Gráficos de uso de CPU (%) por tempo de execução (segundos). Onde: (a) Hadoop, (b) Giraph e (c) Hama.**
**Fonte:DSTAT.**

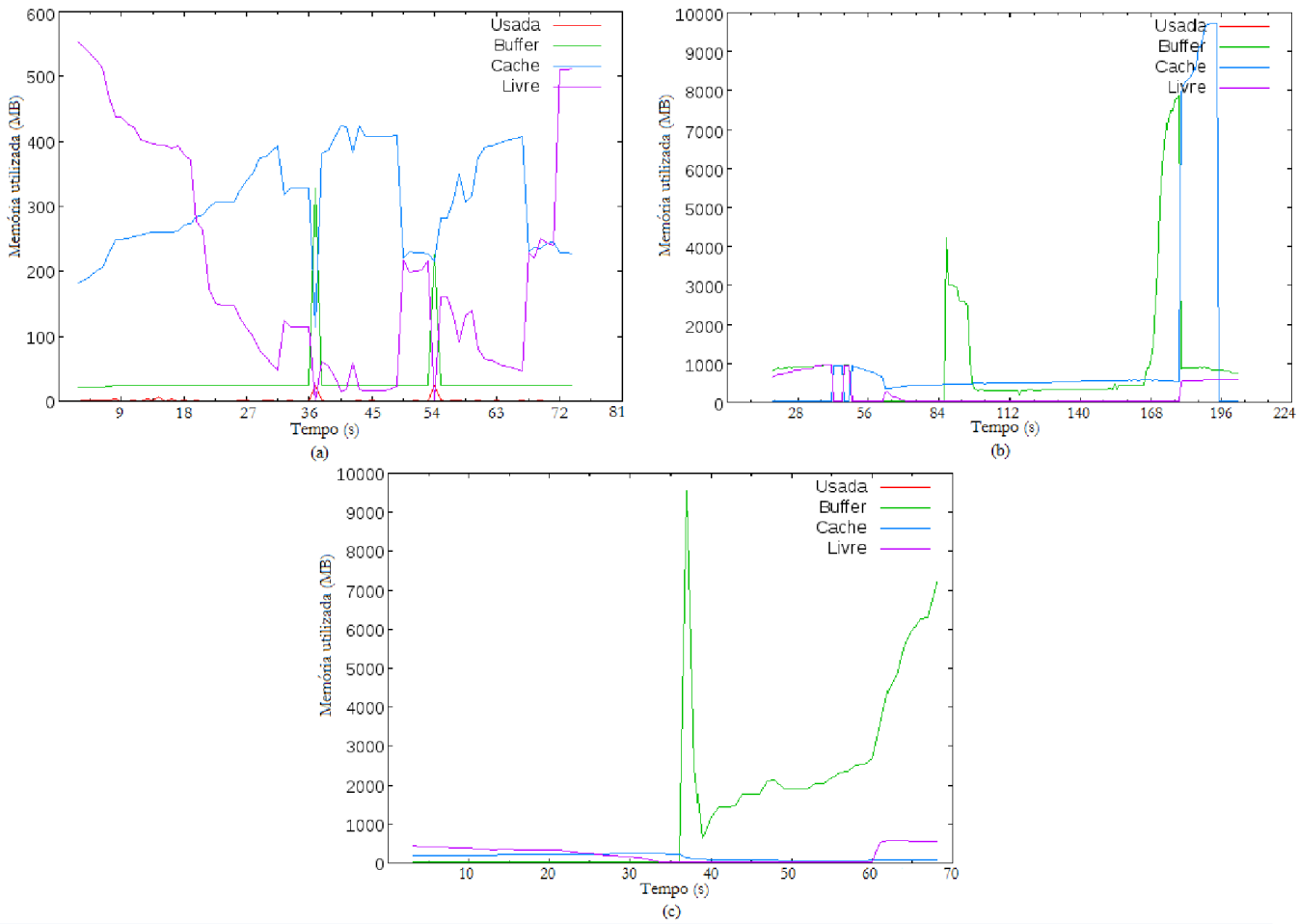



**Figura 18 Gráficos de uso de memória (MB) por tempo de execução (segundos). Onde: (a) Hadoop, (b) Giraph e (c) Hama.**
**Fonte:DSTAT.**

## 5.2 PAGERANK - WEB-NOTREDAME

As análises deste tópico são baseadas nos gráficos da Figura 19 (uso de CPU) e nos gráficos da Figura 20 (uso de memória).

Para a execução do algoritmo de PageRank na base web-NotreDame nos arcabouços, notou-se primeiramente que o arcabouço Hadoop obteve o melhor resultado relacionado a tempo de execução, levando uma média de aproximadamente 57 segundos para concluir suas iterações.

Apesar disto, o Hama foi o arcabouço que menos fez uso de CPU, desta

vez com uma divergência um pouco maior de desempenho para o arcabouço Hadoop (comparação com a análise em 5.1), que ainda assim obteve um desempenho similar quando comparado ao tempo de execução do algoritmo e uso de CPU. Ambos (Hama e Hadoop), mais uma vez, tiveram um desempenho muito satisfatório se comparados ao terceiro arcabouço, o Giraph. Este último, por sua vez, executou o algoritmo na base web-Stanford com uma média de 223 segundos, o que representa quase quatro vezes o tempo que o Hadoop levou para fazer o mesmo procedimento. Ao analisar o uso de CPU percebe-se que o desempenho do Giraph para esta execução foi de fato abaixo dos outros arcabouços analisados, pois por mais de 100 dos 223 segundos de execução o gráfico mostra que o processador estava entre 85% e 95% de uso.

No que diz respeito ao uso de memória, os três arcabouços novamente apresentaram desempenho um tanto quanto parecido. Todos eles mantiveram o uso de memória abaixo de 1.000 MB na maior parte da execução, com algumas poucas variações de uso de memória em buffer.

Em resumo, a análise desta execução trouxe os seguintes resultados:

● Hadoop - 57 segundos de tempo médio de execução. O uso de CPU variou bastante e esteve poucas vezes abaixo de 50%, na maior parte da execução esteve entre 80% e 100%. O pico de uso de CPU atingiu aproximadamente 99%. O uso de memória esteve abaixo de 1.000 MB durante quase toda a execução, a exceção foi aproximadamente nos últimos dez segundos de execução, onde a memória foi usada em buffer com aproximadamente 7.700 MB. O pico de uso de memória atingiu aproximadamente 7.800 MB em buffer;

● Giraph - 223 segundos de tempo médio de execução. O uso de CPU esteve entre 85% e 100% na maior parte da execução. O pico de uso de CPU atingiu aproximadamente 97%. O uso de memória esteve abaixo de 1.000 MB na maior parte da execução, porém, em alguns momentos foi utilizada em buffer acima de 8.000 MB e em cache acima de 6.000 MB. O pico de uso de memória atingiu aproximandamente 8.100 MB em buffer, outro pico importante foi em cache, de aproximadamente 6.400 MB;

● Hama - Aproximadamente 68 segundos de tempo médio de execução (tempo igual ao da execução do PageRank com a base web-Stanford). O uso de CPU esteve entre 70% e 90% na maior parte da execução. O pico de uso de CPU atingiu

aproximadamente 98%. O uso de memória esteve abaixo de 1.000 MB na maior parte da execução, porém houveram variações em uso de memória em buffer, que não ultrapassaram os 2.700 MB, valor registrado como pico de uso de memória;

Conclui-se então, que para esta execução, a partir da análise realizada o arcabouço Hadoop obteve o melhor tempo de execução, porém, o arcabouço Hama apresentou resultados melhores de uso de CPU e de uso de memória mesmo com um tempo de execução 11 segundos mais lento. O pior desempenho foi novamente analisado nos resultados obtidos pelo arcabouço Giraph.

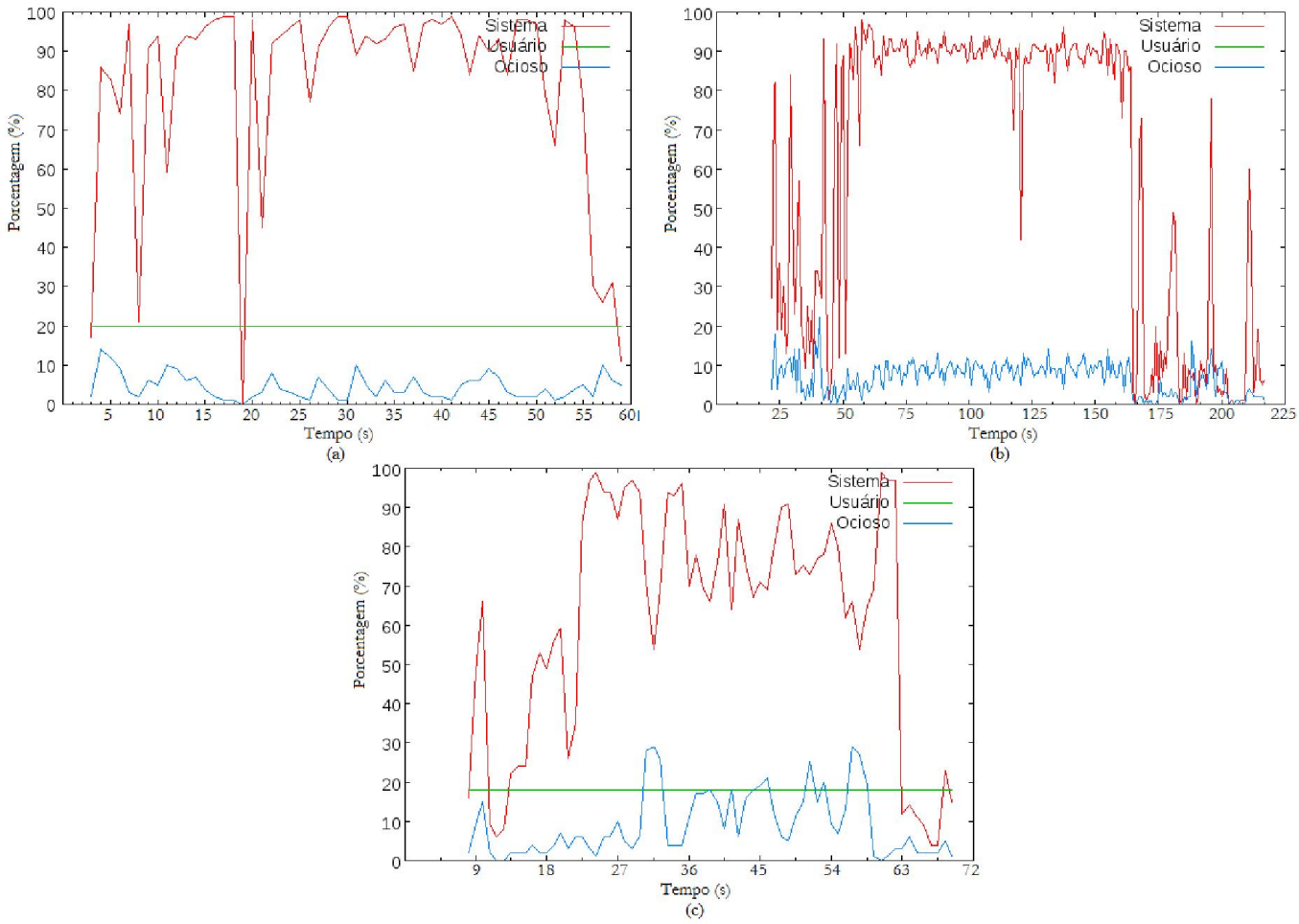



**Figura 19 Gráficos de uso de CPU (%) por tempo de execução (segundos). Onde: (a) Hadoop, (b) Giraph e (c) Hama.**
**Fonte:DSTAT.**

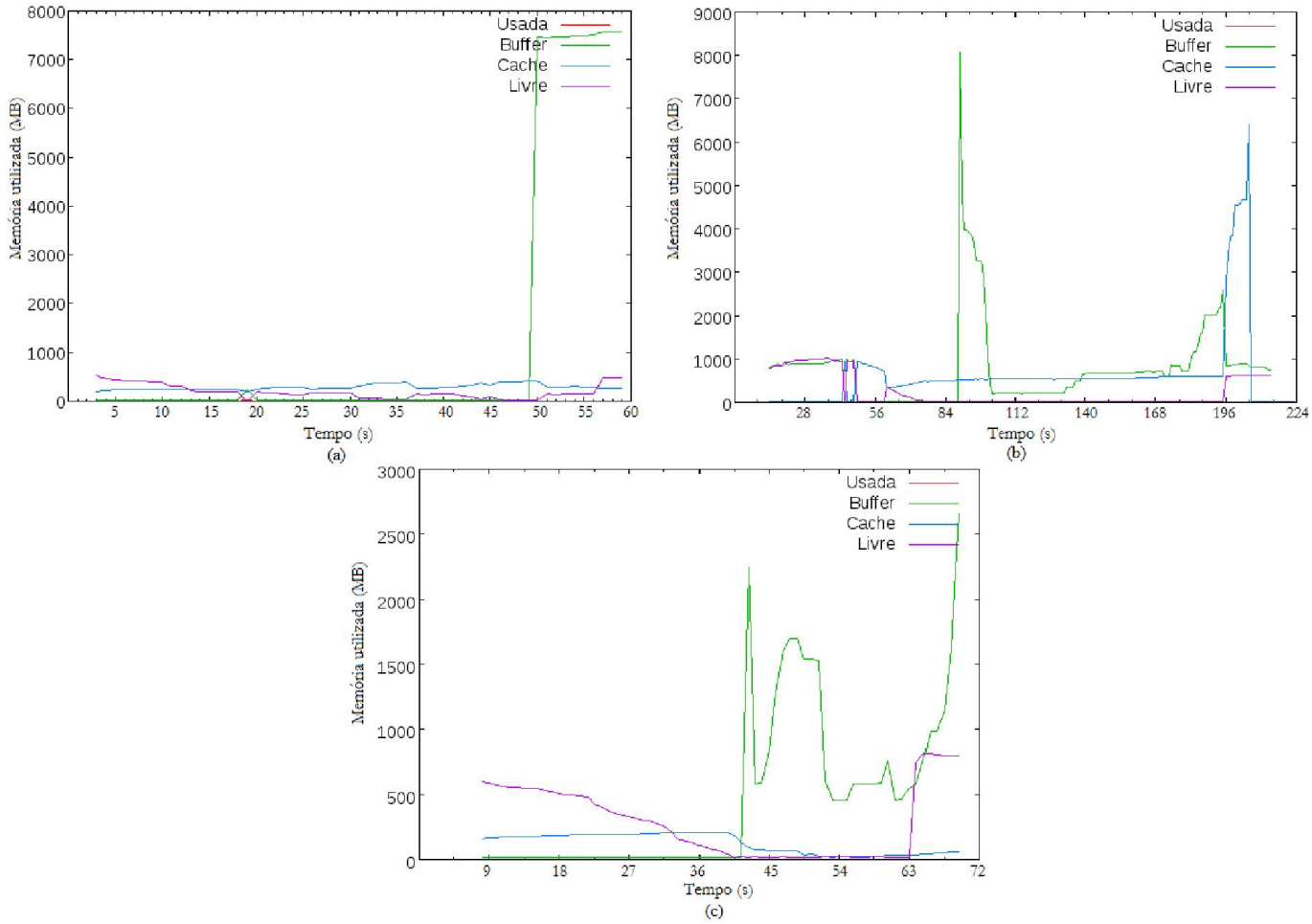



**Figura 20 Gráficos de uso de memória (MB) por tempo de execução (segundos). Onde: (a) Hadoop, (b) Giraph e (c) Hama.**
**Fonte:DSTAT.**

## 5.3 PAGERANK - WEB-STANFORD X WEB-NOTREDAME

Para verificar o desempenho dos arcabouços na execução do PageRank e comparar de maneira mais visual as diferenças entre as execuções de ambas as bases de dados, a Figura 21 mostra todos os gráficos de uso de CPU e a Figura 22 mostra todos os gráficos de uso de memória.

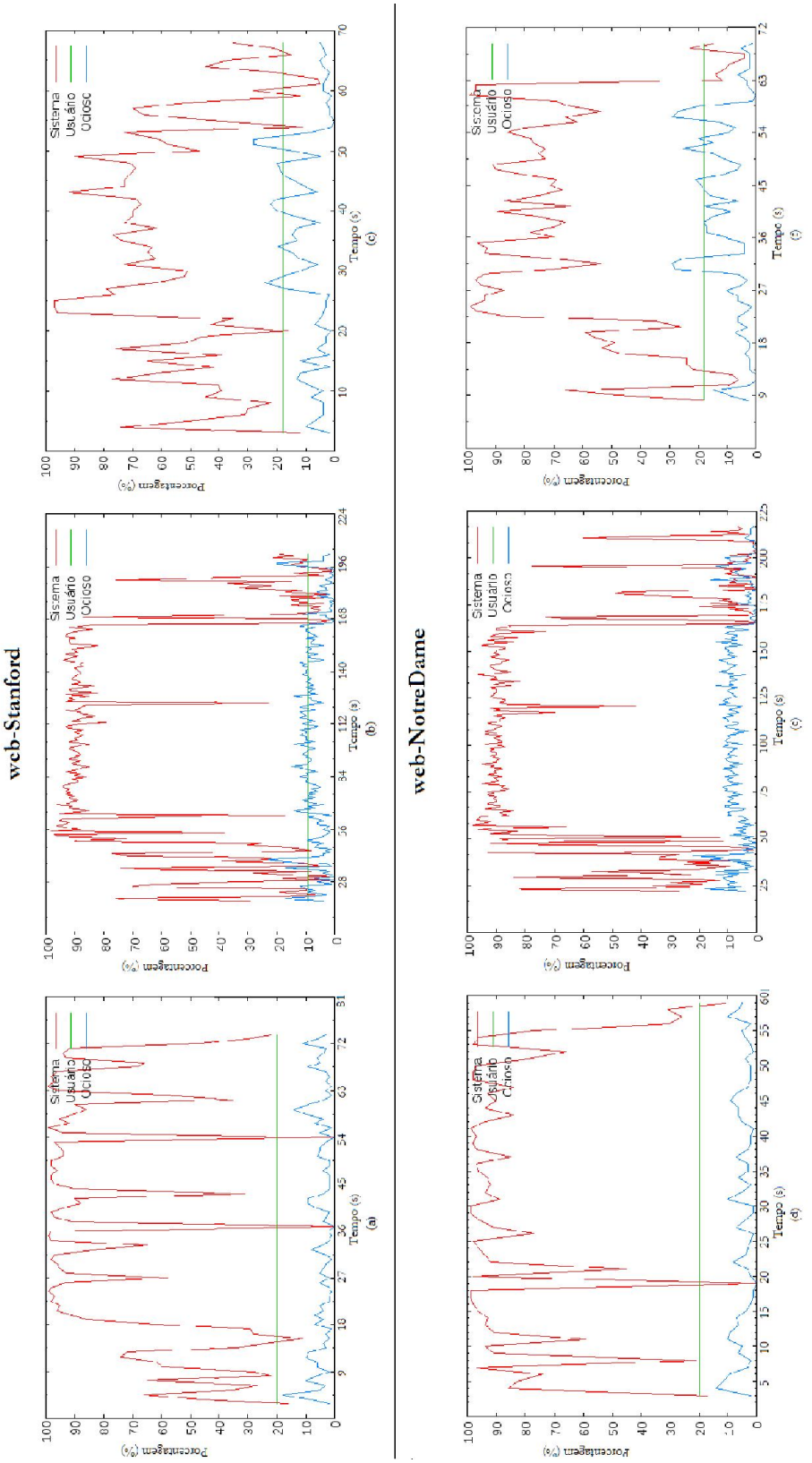



**Figura 21 Gráficos de uso de CPU (%) por tempo de execução (segundos). Onde: (a) e (d) Hadoop, (b) e (e) Giraph e (c) e (f) Hama.**
**Fonte:DSTAT.**

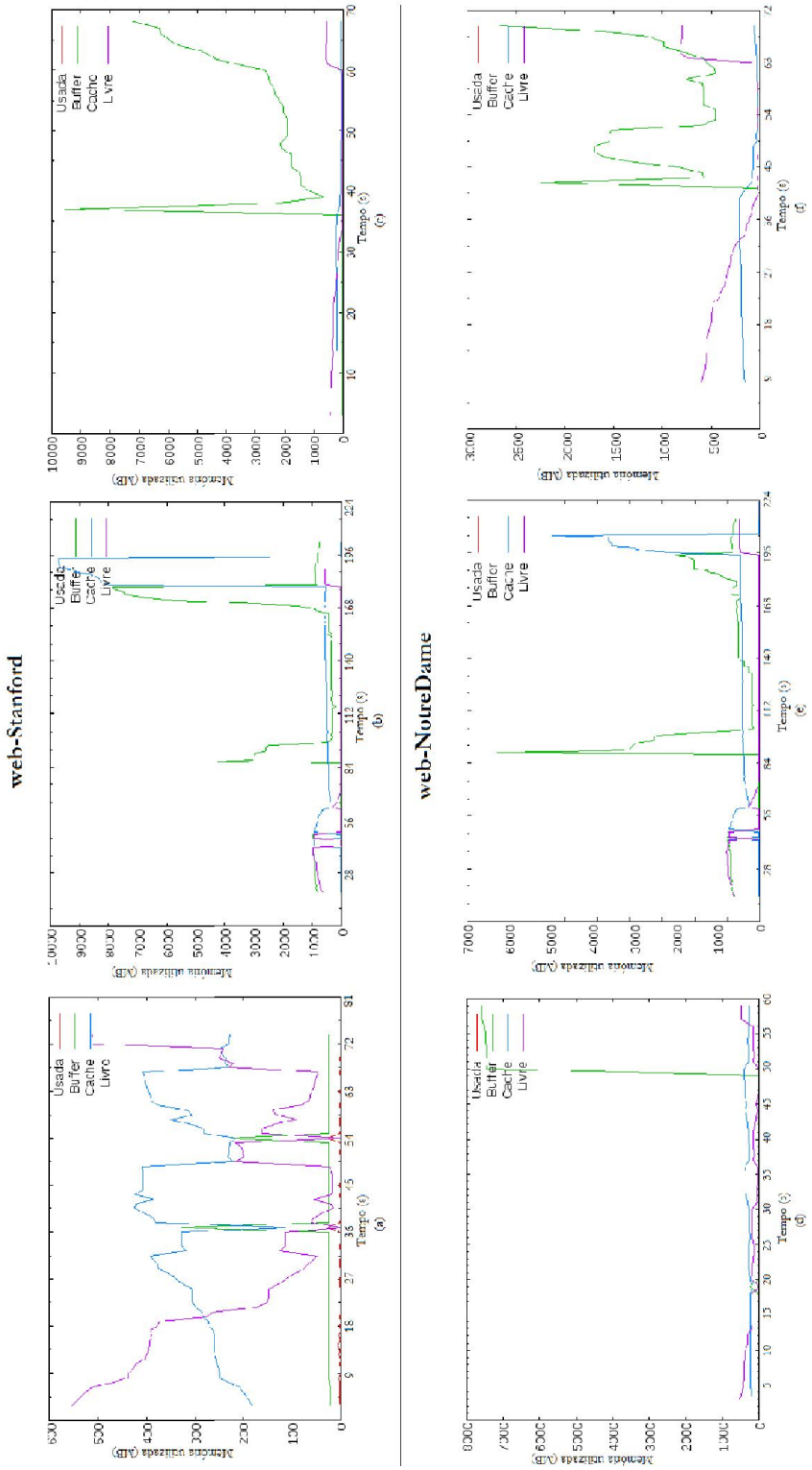



**Figura 22 Gráficos de uso de memória (MB) por tempo de execução (segundos). Onde: (a) e (d) Hadoop, (b) e (e) Giraph e (c) e (f) Hama. Fonte:DSTAT.**

## 5.4 SHORTESTPATHS - WEB-STANFORD

As análises deste tópico são baseadas nos gráficos da Figura 23 (uso de CPU) e nos gráficos da Figura 24 (uso de memória).

Para a execução do algoritmo de ShortestPaths na base web-Stanford nos arcabouços, notou-se primeiramente que o arcabouço Giraph obteve o melhor resultado relacionado a tempo de execução, levando uma média de aproximadamente 67 segundos para concluir suas iterações.

Além disto, o Giraph foi o arcabouço que menos fez uso de CPU, onde em vários momentos da execução esteve entre 20% e 40%, mas também é possível ver que em boa parte do tempo apresentou resultados entre 85% e 100%. Hadoop e Hama mantiveram o uso de CPU entre 70% e 100% durante mais da metade do tempo de execução. Ambos (Hadoop e Hama), desta vez apresentaram um desempenho muito abaixo do terceiro arcabouço, o Giraph, se analisarmos tempo de execução e uso de CPU.

O uso de memória mostrou diferenças relevantes na comparação do Giraph com os outros dois arcabouços. Hadoop e Hama apresentaram uso de memória em buffer com valores altos, que chegaram a atingir 10.000 MB, enquanto o Giraph só atingiu um pico de aproximadamente 980 MB também em buffer.

Em resumo, a análise desta execução trouxe os seguintes resultados:

● Hadoop - 176 segundos de tempo médio de execução. O uso de CPU esteve entre 70% e 100% em mais da metade da execução. O pico de uso de CPU atingiu aproximadamente 99%. O uso de memória apresentou muitas variações de uso em buffer em boa parte da execução, atingindo um pico de 10.000 MB;

● Giraph - Aproximadamente 67 segundos de tempo médio de execução. O uso de CPU esteve entre 20% e 40% em grande parte da execução, mas também esteve entre 85% e 100% em alguns momentos. O pico de uso de CPU atingiu aproximadamente 98%. O uso de memória esteve abaixo de 1.000 MB durante toda a execução. O pico de uso de memória atingiu aproximandamente 980 MB em buffer;

● Hama - Aproximadamente 165 segundos de tempo médio de execução. O uso de CPU esteve entre 70% e 100% na maior parte da execução, bem como

analisado na execução do Hadoop. O pico de uso de CPU atingiu aproximadamente 99%. O uso de memória, assim como o Hadoop, também apresentou variações em mais da metade da execução em buffer, atingindo pico de 10.000 MB;

Conclui-se então, que para esta execução, a partir da análise realizada o arcabouço Giraph apresentou os melhores resultados, tanto de tempo de execução, quanto de uso de CPU e memória. A diferença entre o Giraph e os outros dois arcabouços (Hadoop e Hama) pode ser considerada a maior dentre todas as análises feitas neste trabalho, uma vez que o desempenho do Giraph foi muito satisfatório, obtendo resultados que se comparados aos dos outros dois arcabouços representam números até três vezes melhores. O uso de memória nesta análise apresentou diferenças muito importantes, pois o Giraph manteve o gráfico abaixo de 700 MB durante quase toda a execução, com exceção de um uníco pico de 980 MB. Hadoop e Hama apresentaram valores muito altos de uso de memória em buffer, o que torna a diferença de desempenho entre eles e o Giraph ainda maior.

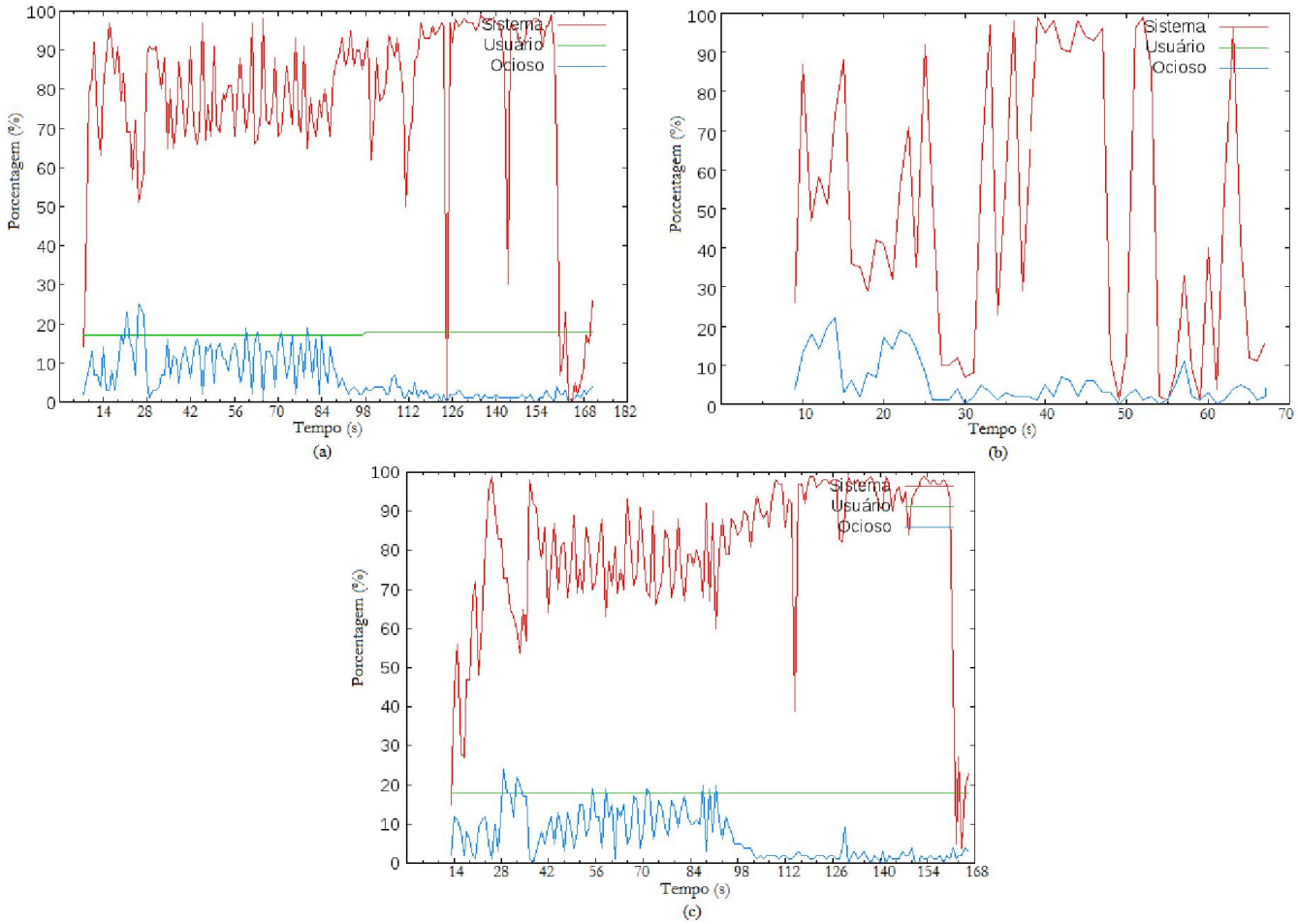



**Figura 23 Gráficos de uso de CPU (%) por tempo de execução (segundos). Onde: (a) Hadoop, (b) Giraph e (c) Hama.**
**Fonte:DSTAT.**

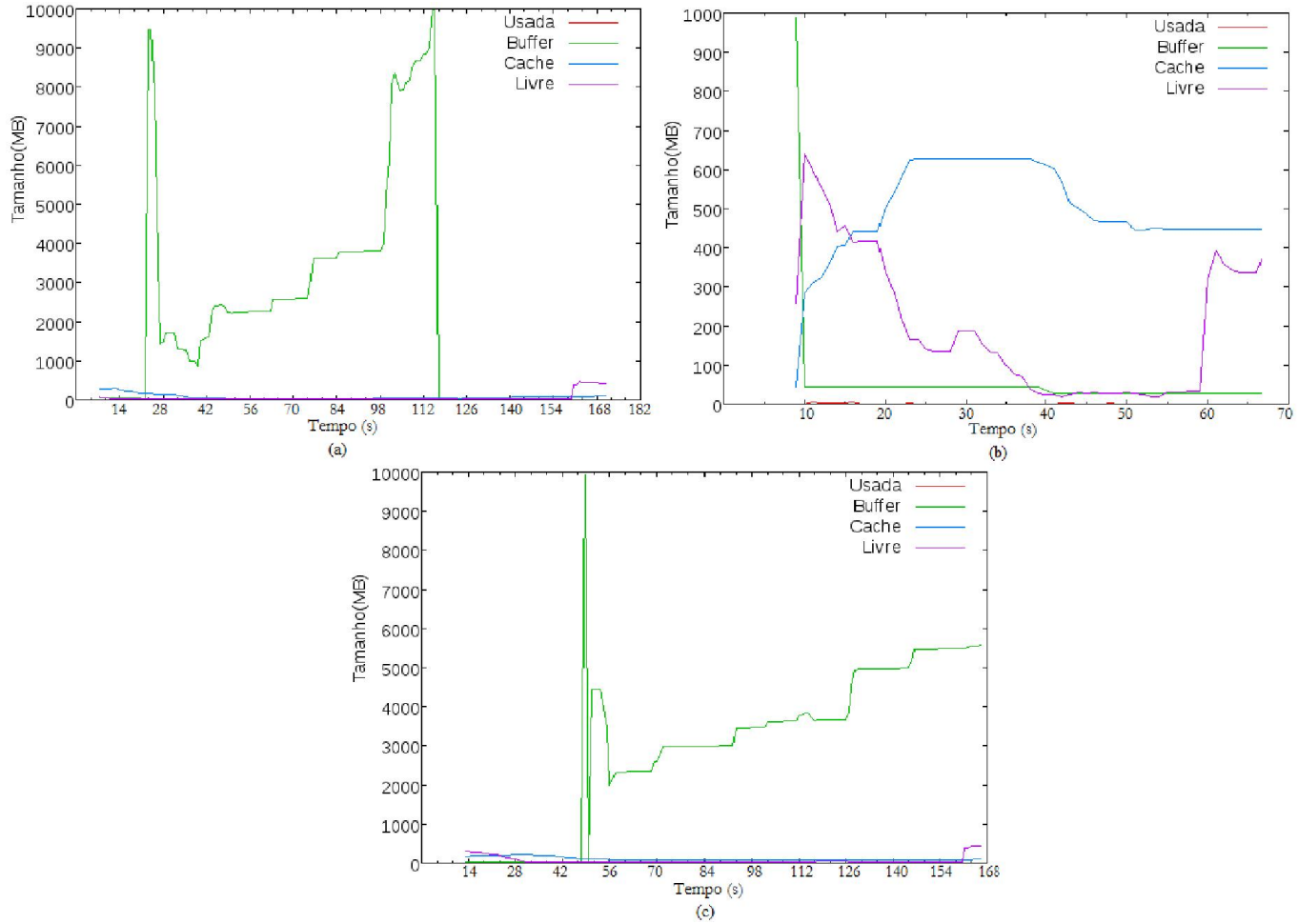



**Figura 24 Gráficos de uso de memória (MB) por tempo de execução (segundos). Onde: (a) Hadoop, (b) Giraph e (c) Hama.**
**Fonte:DSTAT.**

## 5.5 SHORTESTPATHS - WEB-NOTREDAME

As análises deste tópico são baseadas nos gráficos da Figura 25 (uso de CPU) e nos gráficos da Figura 26 (uso de memória).

Para a execução do algoritmo de ShortestPaths na base web-NotreDame nos arcabouços, notou-se primeiramente que o arcabouço Giraph obteve o melhor resultado relacionado a tempo de execução, levando uma média de aproximadamente 67 segundos para concluir suas iterações, exatamente o mesmo valor da execução na base web-Stanford.

O Giraph também foi o arcabouço que menos fez uso de CPU, o gráfico aponta que o arcabouço em questão manteve o uso abaixo de 80% na maior parte do tempo da execução, somente em alguns poucos momentos o gráfico passou da marca de 80%. Neste quesito, Hadoop e Hama demonstraram desempenho parecido ao manter o uso de CPU entre 70% e 90% na maior parte da execução. Bem como visto na análise do tópico 5.4, ambos (Hama e Hadoop), apresentaram um desempenho muito abaixo do terceiro arcabouço, o Giraph, quando analisarmos tempo de execução e uso de CPU.

Novamente, o uso de memória não mostrou diferenças relevantes entre os três arcabouços. Como nas outras análises, o que diferencia um arcabouço do outro na questão do uso de memória são alguns picos de uso em buffer, notados nos gráficos do Hadoop e do Hama. Todos eles mantiveram o uso de memória abaixo de 500 MB na maior parte da execução.

Em resumo, a análise desta execução trouxe os seguintes resultados:

● Hadoop - Aproximadamente 112 segundos de tempo médio de execução. O uso de CPU esteve entre 70% e 90% na maior parte da execução. O pico de uso de CPU atingiu aproximadamente 98%. O uso de memória esteve abaixo de 500 MB em boa parte da execução, mas houveram muitas variações de uso em buffer atingindo valores como 6.000 MB, 7.000 MB e acima de 9.000 em um trecho considerável da execução. O pico de uso de memória atingiu aproximadamente 9.900 MB em buffer;

● Giraph - Aproximadamente 67 segundos de tempo médio de execução. O uso de CPU esteve acima de 80% por menos de 20 segundos da execução, enquanto que no restante da execução os valores analisados foram muito satisfatórios se comparados com as outras execuções já analisadas. O pico de uso de CPU atingiu aproximadamente 97%. O uso de memória esteve abaixo de 1.000 MB durante toda a execução. O pico de uso de memória atingiu aproximandamente 960 MB em buffer;

● Hama - Aproximadamente 103 segundos de tempo médio de execução. O uso de CPU esteve entre 70% e 90% na maior parte da execução, assim como analisado na execução do Hadoop. O pico de uso de CPU atingiu aproximadamente 99%. O uso de memória esteve abaixo de 500 MB na maior parte da execução, porém houveram variações em uso de memória em buffer, que atingiram valores

como 1.000 MB, 2.200 MB e 2.700 MB aproximadamente, sendo o último valor o pico de uso de memória atingido;

Conclui-se então, que para esta execução, a partir da análise realizada o arcabouço Giraph apresentou os melhores resultados, tanto de tempo de execução, quanto de uso de CPU e memória. A diferença entre o Giraph e os outros dois arcabouços (Hadoop e Hama) foi considerável para esta execução. Pode-se destacar o uso de memória nesta análise, onde o Giraph não apresentou grandes variações como os outros arcabouços, se mostrando um tanto quanto estável. Ao analisar todos os desempenhos, pode-se considerar que o Hadoop foi o pior dos três arcabouços nesta execução, seguido de perto pelo Hama.

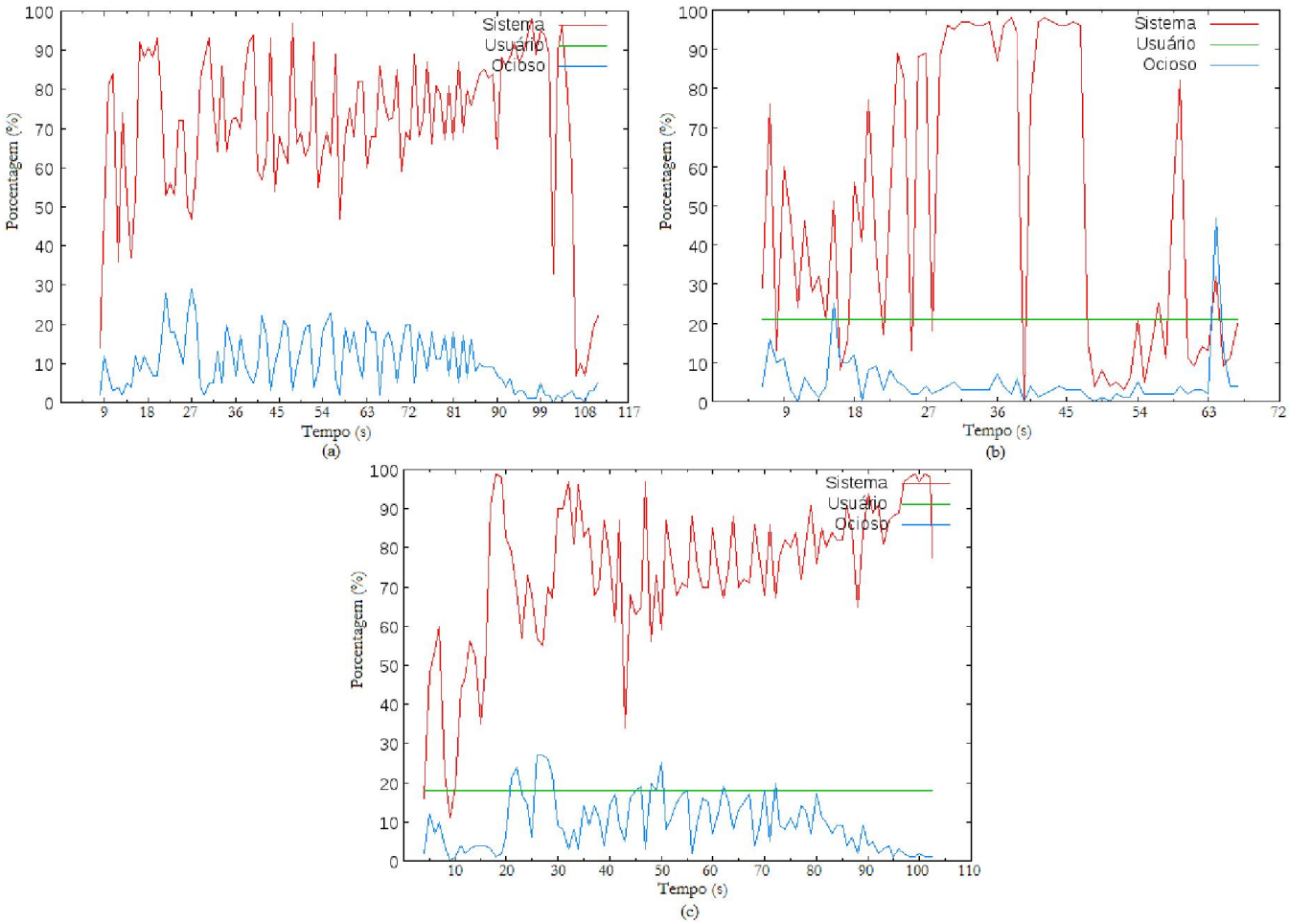



**Figura 25 Gráficos de uso de CPU (%) por tempo de execução (segundos). Onde: (a) Hadoop, (b) Giraph e (c) Hama.**
**Fonte:DSTAT.**

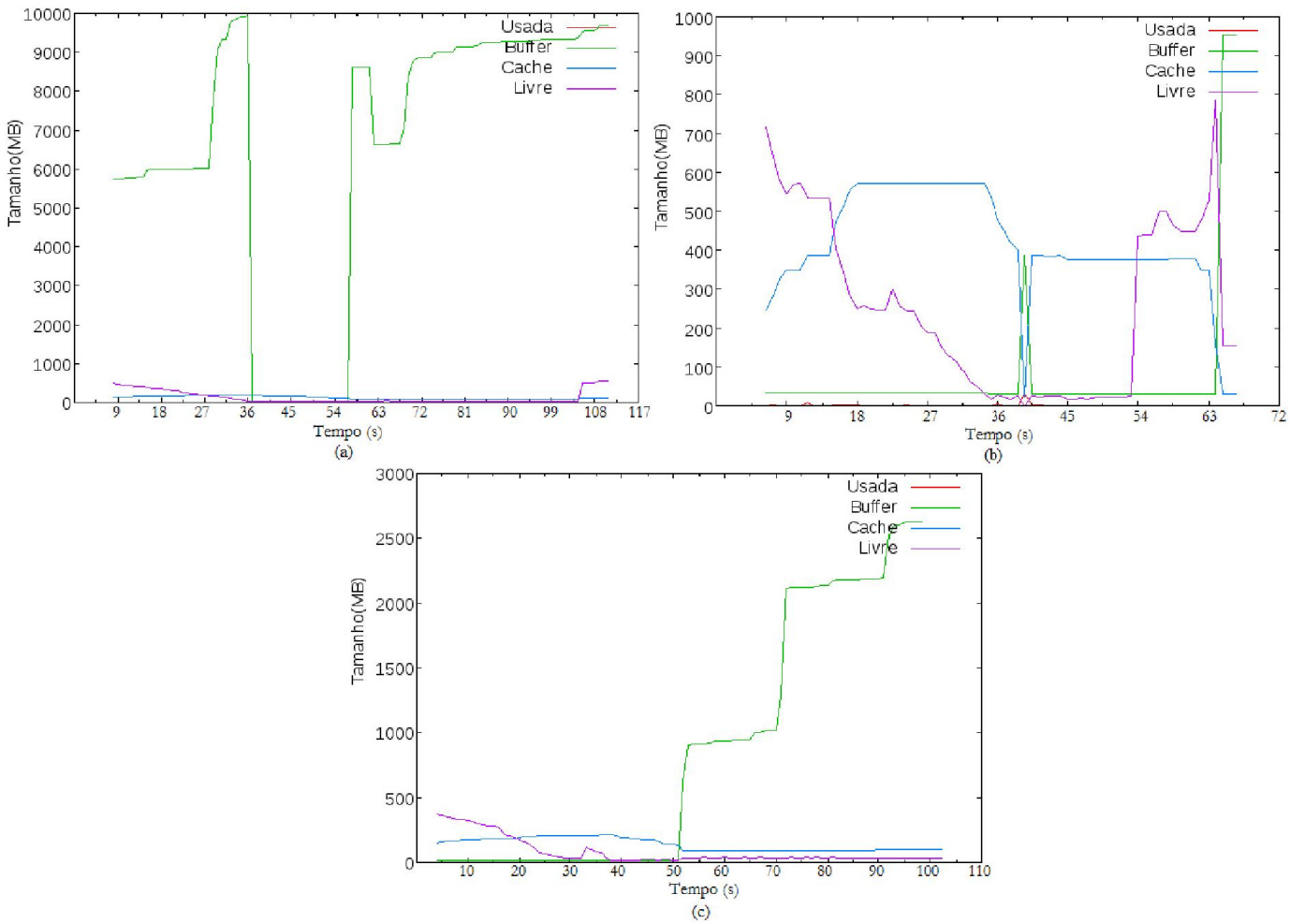



**Figura 26 Gráficos de uso de memória (MB) por tempo de execução (segundos). Onde: (a) Hadoop, (b) Giraph e (c) Hama.**
**Fonte:DSTAT.**

## 5.6 SHORTESTPATHS - WEB-STANFORD X WEB-NOTREDAME

Para verificar o desempenho dos arcabouços na execução do ShortestPaths e comparar de maneira mais visual as diferenças entre as execuções de ambas as bases de dados, a Figura 27 mostra todos os gráficos de uso de CPU e a Figura 28 mostra todos os gráficos de uso de memória.

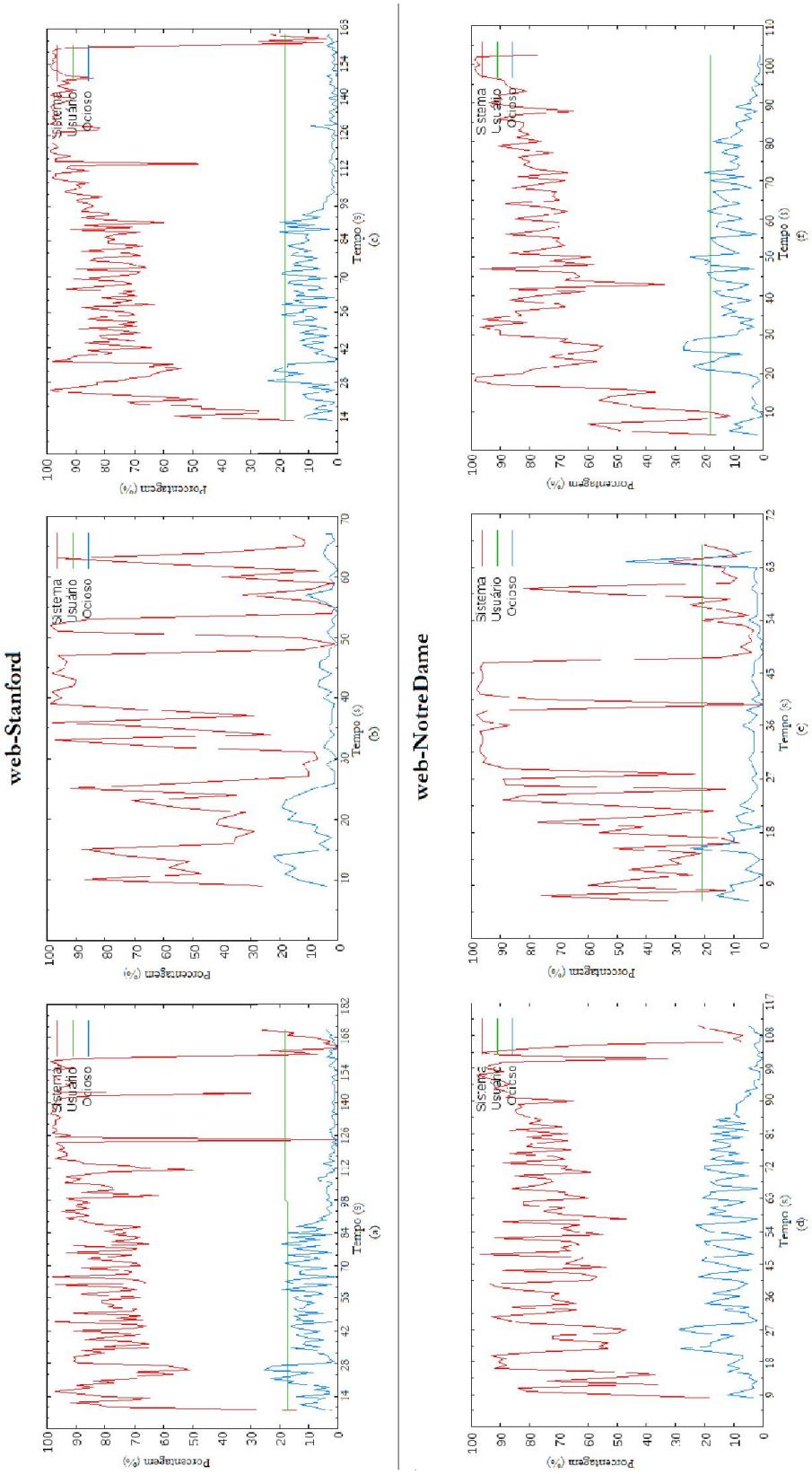



**Figura 27 Gráficos de uso de CPU (%) por tempo de execução (segundos). Onde: (a) e (d) Hadoop, (b) e (e) Giraph e (c) e (f) Hama.**
**Fonte:DSTAT.**

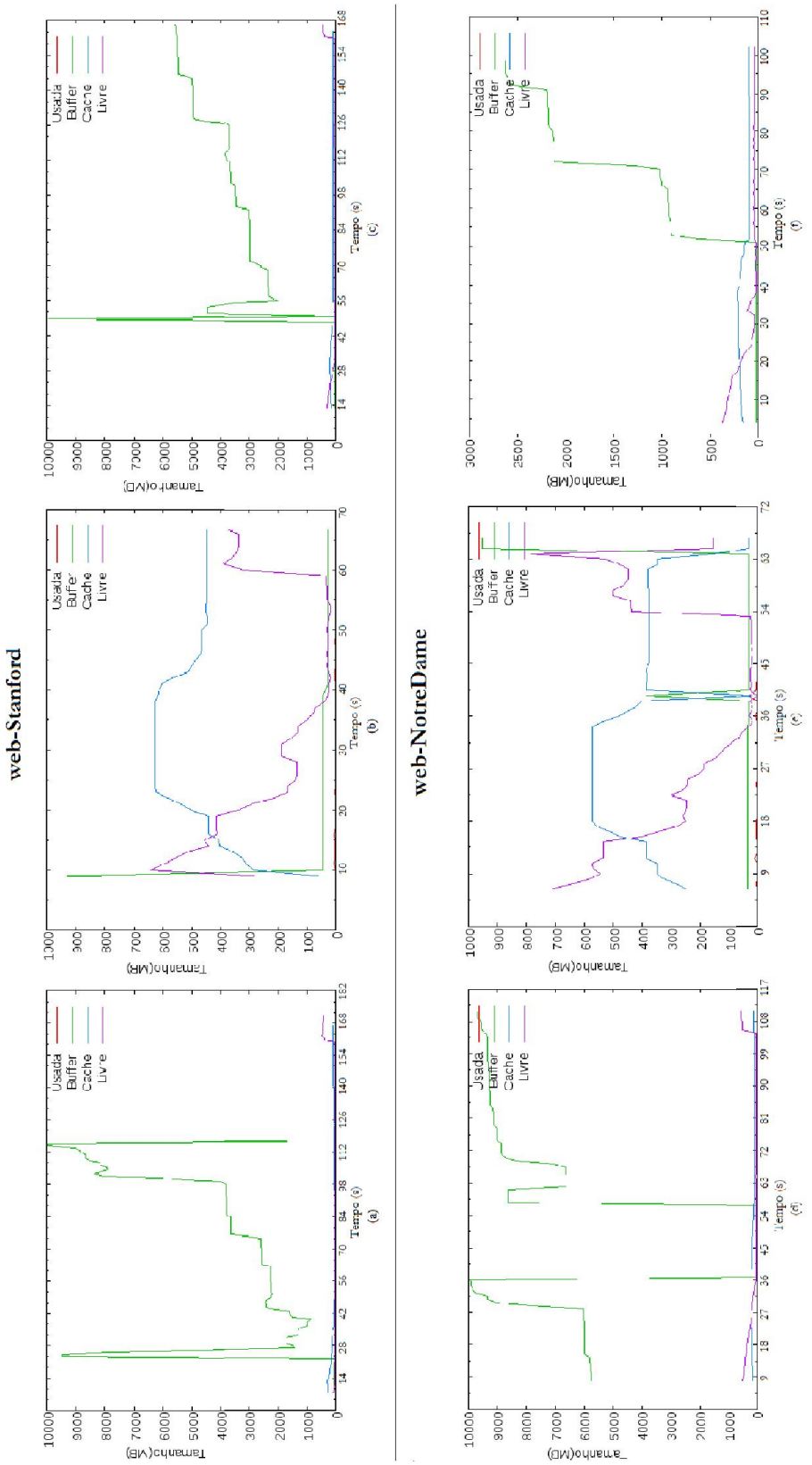



**Figura 28 Gráficos de uso de memória (MB) por tempo de execução (segundos). Onde: (a) e (d) Hadoop, (b) e (e) Giraph e (c) e (f) Hama. Fonte:DSTAT.**

## 5.7 PAGERANK X SHORTESTPATHS

Por fim, comparou-se o desempenho dos arcabouços entre as execuções dos dois algoritmos. Muitas diferenças podem ser destacadas. Nas execuções de PageRank o melhor desempenho foi apresentado pelo arcabouço Hama, nas duas bases de dados. Vale lembrar que o arcabouço Hadoop apresentou um desempenho que esteve bem próximo ao do Hama para estas execuções. Porém, nas análises das execuções no algoritmo ShortestPaths, os resultados dos desempenhos dos arcabouços se inverteram e o Giraph (que demonstrou desempenho ruim quando comparado ao Hadoop e ao Hama nas execuções de PageRank) foi muito superior aos outros arcabouços, tanto para execuções na base web-Stanford quanto para execuções na base web-NotreDame. Esta comparação finaliza a etapa de comparação e análise de resultados. O próximo tópico (5.8) lista as maiores contribuições que este trabalho traz, a partir dos resultados obtidos neste capítulo, para estudos e pesquisas que envolvem o processamento de grandes quantidades de dados modelados em grafos através de arcabouços.

## 5.8 MAIORES CONTRIBUIÇÕES

O processo de avaliação de desempenho de arcabouços pode ser realizado de várias formas. Há muitas abordagens e diversas variantes que envolvem todas as etapas de preparação, instalação e configuração dos arcabouços que podem interferir nos resultados que serão coletados. O hardware e a utilização de máquinas virtuais também apresentam complicadores para se comparar um trabalho/experimento com outro. Além disto, muitas outras variantes surgem se consideradas as bases de dados e os algoritmos utilizados para as execuções nos arcabouços propostos. Assim, este trabalhou, por meio de um protocolo de execução que definiu uma ordem de execução das tarefas, priorizou a organização de algunsdesses aspectos, de maneira que estas variantes causassem o menor

impacto possível na coleta de dados de desempenho dos arcabouços.

Além disso, os algoritmos utilizados (PageRank e ShortestPaths) foram implementados pelos desenvolvedores dos arcabouços, sem alteração alguma em seus códigos.

Todas estas preocupações, em conjunto com a utilização de ferramentas de propósito exclusivo de monitoramento de desempenho (DSTAT, Munin e Ganglia), permitiram que a maior contribuição deste trabalho seja a obtenção de resultados confiáveis. Desta forma, as etapas do desenvolvimento foram cuidadosamente definidas para que o resultado final da análise do desempenho dos arcabouços em um contexto de Big Data não seja prejudicada.

Existem outras contribuições importantes deste trabalho, como a quantidade e o tipo dos parâmetros analisados, que foram: tempo de execução, uso de memória RAM e uso de processador. Alguns trabalhos relacionados, citados no tópico 2.4, como (BÚRIGO, 2015) por exemplo, utilizouparâmetros coletados na própria implementação do algoritmo executado. Tais parâmetros, coletados desta forma, não são tão confiáveis (devido a várias interferências a que estão vulneráveis enquanto são coletados) quanto parâmetros coletados a partir de uma ferramenta utilizada exclusivamente para este propósito.

# 6 CONSIDERAÇÕES FINAIS

É possível concluir a partir de todas as análises realizadas, que para o contexto deste trabalho Hadoop e Hama apesar de realizarem as execuções dos algoritmos nas bases de forma diferente, demonstraram desempenho muito similar. Em todas as análises ambos se comportaram de maneira muito parecida em todos os quesitos analisados (tempo de execução, uso de CPU e uso de memória). Giraph se mostrou um ótimo arcabouço para a execução do algoritmo ShortestPaths, mas, seu desempenho nas execuções de PageRank não foi satisfatório quando comparado a Hadoop e Hama. Contudo, Hadoop e Hama não mostraram bom desempenho nas execuções de ShortestPaths.

A alternância de bases de dados nas execuções também trouxe uma informação interessante. Nas execuções de ShortestPaths os arcabouços demonstraram desempenho melhor na base web-NotreDame, conforme apresentado na Figura 27. A partir das informações disponibilizadas na Tabela 2 e na Tabela 3, sabe-se que a base web-NotreDame possui mais vértices (325.729) e menos arestas (1.497.134) do que a base web-Stanford (que possui 281.903 vértices e 2.312.497 arestas), logo, pode-se deduzir que para execuções do algoritmo de ShortestPaths, no contexto imposto, o número de arestas comprometem mais o desempenho dos arcabouços do que o número de vértices. Nas execuções de PageRank, a mudança de base de dados não apresentou diferenças de desempenho relevantes.

O uso de memória apresentado nas execuções apontou alguns picos de uso em buffer que extrapolaram a memória atribuída às máquinas virtuais, com valores que chegaram a 10.000 MB enquanto que as máquinas virtuais possuíam 2.000 MB de memória. As execuções que mais fizeram uso destes picos de memória foram as que apresentaram os maiores tempos de execução, isto se deve ao uso de memória virtual por falta de memória disponível. A velocidade de leitura e escrita em memória virtual é muito inferior quando comparada a velocidade em memória RAM.

Os resultados mostram, em uma análise geral, que o arcabouço Giraph foi o melhor dentre os três estudados, mesmo que com um desempenho ruim para executar PageRank. O mal desempenho no algoritmo de PageRank foi compensado pelo desempenho no algoritmo de ShortestPaths. De forma análoga, Hama e

Hadoop também compensam seus desempenhos ruins no algoritmo de ShortestPaths com seus desempenhos no algoritmo de PageRank, porém, não com a mesma eficiência do Giraph. Mas, os estudos e análises deste trabalho indicam o uso do Hama para execuções de PageRank ou Hadoop como segunda opção, e para execuções de ShortestPaths é indicado o uso do Giraph.

## 6.1 DIFICULDADES ENCONTRADAS

### 6.1.1 Instalação e Configuração

- Arcabouços

O maior problema encontrado no desenvolvimento deste trabalho relacionado aos arcabouços, foi fazê-los funcionar, devido a vários fatores. Inicialmente, os arcabouços Hadoop e Hama foram instalados, com o auxílio de vários tutoriais retirados da internet, e submetidos à testes básicos de execuções de algoritmos propostos pelo próprio site da Apache (desenvolvedora de ambos os arcabouços). Os testes foram concluídos satisfatoriamente, e então deu-se início a instalação do Giraph. Para este arcabouço, encontrou-se dificuldade na escassez de material de apoio disponível para configuração, instalação e informações em geral. Ainda assim, prosseguiu-se com sua instalação através do guia fornecido pelo site oficial do Giraph. Após alguns problemas de configuração do arcabouço e dificuldades ocasionados por não interpretar o guia da maneira correta, obteve-se sucesso na instalação e nas execuções dos testes propostos pelo guia. Sabe-se que o Giraph e o Hama utilizam alguns dos recursos do Hadoop, e este foi outro problema encontrado, uma vez que o Hadoop que havia sido instalado era a versão 2.6, enquanto que o guia de instalação do Giraph orientava a instalação do Hadoop 0.20.203. Por este motivo o Giraph foi instalado em outra máquina virtual.

Para o arcabouço Spark, logo no início, novamente, detectou-se o problema de escassez de material de apoio, porém, o Spark possui ainda menos conteúdo disponível na internet do que o Giraph. Ainda assim, um bom guia foi encontrado e seguido. A instalação e configuração do Spark prosseguiu com sucesso, mas este guia não fornecia informações sobre testes para verificar a funcionalidade do arcabouço. Não foi encontrado bons materiais na internet que instruam os passos de realizações de testes no Spark e por este motivo, decidiu-se continuar o trabalho somente com os outros três arcabouços instalados e testados (Hadoop, Hama e Giraph).

- Ferramentas de Monitoramento de Desempenho

A primeira ferramenta instalada foi o DSTAT. Tanto o DSTAT quanto o SAR são executados no terminal e exibem seus resultados também no terminal. Logo, a primeira dificuldade foi encontrar formas de iniciar estas ferramentas no momento exato em que os algoritmos eram iniciados nos arcabouços, para que não houvesse coletas de dados em momentos onde o algoritmo não estava sendo executado, o que influenciaria no resultado final da coleta de resultados. A solução foi simples, pois tanto a execução do algoritmo no arcabouço quanto a execução de ambas as ferramentas de monitoramento tratam-se de processos no terminal. Para iniciar processos ao mesmo tempo no terminal, basta usar o caractere & entre um comando de execução e outro.

O segundo problema encontrado para o DSTAT e o SAR foi o fato dos resultados aferidos por ambos serem exibidos no próprio terminal em tempo real. Para resolver este problema, as saídas das duas ferramentas foram redirecionadas para arquivos de texto. Mas então, surgiu outra dificuldade, que foi tratar os dados armazenados nos arquivos de texto gerados. Para tal, usou-se a ferramenta Gnuplot. Foi desenvolvido um shell script para manipular e filtrar os dados de interesse do arquivo de forma que estes dados extraídos estivessem prontos para serem utilizadas pelo Gnuplot. O script foi criado para gerar três arquivos de imagem que continham gráficos dos dados de tráfego de rede, uso de memória e uso de

processador, a partir dos dados fornecidos pelo DSTAT e SAR filtrados pelo próprio script. O Munin e o Ganglia possuem interfaces gráficas muito intuitivas e fáceis de manipular, fato este que permitiu o uso de ambas sem que dificuldades fossem encontradas.

### 6.1.2 Algoritmos

Foram feitas várias pesquisas para encontrar implementações de algoritmos nos arcabouços estudados, mas não há implementações tão específicas disponíveis na internet. Seria necessário manipular os códigos dos algoritmos para adaptá-los a cada arcabouço. O objetivo inicial relacionado aos algoritmos que seriam utilizados nas análises era implementar o maior número possível de algoritmos. Mas, os problemas com instalação e configuração apresentados no tópico 7.1.1 tomaram mais tempo do que era esperado. Por este motivo optou-se pelos algoritmos já implementados que são instalados em conjunto com os arcabouços no JAR de exemplos que cada um possui. Todos os três arcabouços utilizados possuíam implementações de PageRank e ShortestPaths prontas e por isso estes foram os algoritmos estudados neste trabalho.

## 6.2 TRABALHOS FUTUROS

O trabalho realizado pode ser aperfeiçoado com uma grande gama de alternativas para expandir os resultados obtidos. Algumas sugestões são:

- Implementar mais algoritmos;
- Instalar e realizar as execuções em ***cluster*** para avaliar a escalabilidade dos arcabouços quanto a recursos como processador e memória. E inserir o tráfego de rede como um parâmetro de desempenho a ser analisado;
- Realizar um número maior de execuções para aumentar a confiabilidade das médias obtidas;
- Avaliar um número maior de arcabouços;
- Variar quantidade de iterações dos algoritmos;
- Utilizar mais bases de dados, com números de vértices e arestas muito maiores;
- Analisar e alterar (caso necessário) as implementações dos algoritmos para que estes possuam valores de complexidade computacional próximos.

## REFERÊNCIAS


ACAR, U. Parallel and Sequential Data Structures and Algorithms. https://www.cs.cmu.edu/afs/cs/academic/class/15210s13/www/lectures/lecture13.pdf Fev. 2013. Acesso em 07 de out. 2015

BADER, D.; MADDURI, K. **SNAP, Smallworld Network Analysis andPartitioning:** an opensource parallel graph framework for the exploration oflargescale networks. College of Computing Georgia Institute of Technology, 2008.

BÚRIGO, Tayllan. ESTUDO COMPARATIVO DE ARCABOUÇOS PARA PROCESSAMENTO PARALELO DE GRAFOS. 59 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Departamento Acadêmico de Computação, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2015.

CHING, A.; EDUNOV, S.; KABILJO, M.; LOGOTHETIS, D.; MUTHUKRISHNAN, S. **One Trillion Edges:** Graph processing at FacebookScale. 1 Hacker Lane MenloPark, California, 2015.

CORMEN, T.H.;LEISERSON, C.E.; RIVEST, R.L.; STEIN, C.; Algoritmos, Teoria e Prática. Tradução da 2ª edição americana, 2002.

DEBBARMA, A.; B., Annappa; MUDE, R. Performance Analysis of Graph Based Iterative Algorithms on MapReduce Framework. Department of Computer Science and Engineering National Institute of Technology Karnataka, Surathkal, Karnataka, India, 2014.

Google. Por que usar o Google. http://www.google.com.br/why_use.html. 2011. Acesso em 15 de nov. 2015.

HILBERT, Martin (2007), **Digital Processes and Democratic Theory:** Dynamics, risk and opportunities that arise when democratic institutions meet digital information and communication technologies; peerreviewd online publication;

LI, D.; CHAKRADHAR, S.; BECCHI, M. **GRapid:**a Compilation and Runtime Framework for Rapid Prototyping of Graph Applications on Manycore Processors. University of MissouriColumbia, 2014.

MADDURI, K. A HighPerformance Framework For Analyzing Massive Complex Networks. A Thesis Presented to The Academic Faculty, 2008.

MALEWICZ, G., AUSTERN, M. H., BIK, A. J., DEHNERT, J. C., HORN, I., LEISER, N., and CZAJKOWSKI, G. **Pregel:** a system for largescale graph processing. In ***SIGMOD '10: Proceedings of the 2010 international conference on Management ofdata***(New York, NY, USA, 2010), ACM, pp. 135–146.

MAYERSCHÖNBERGER, Viktor; CUKIER, Kenneth. **Big Data –** A Revolution That Will Transform How We Live, Work, and Think. New York: First Mariner Books, 2012.

NONATO, L. G. **Listas de Adjacência**. jun 2000. http://www.lcad.icmc.usp.br/~nonato/ED/Grafos/node76.html. Acesso em 28 de set. 2015.

ROGERS, Ian. **The Google Pagerank Algorithm and How It Works**.2002. http://www.cs.princeton.edu/~chazelle/courses/BIB/pagerank.htm. Acesso em 15 de nov. 2015

SILVA, Davi Bernardo. ARCABOUÇO PARA PROCESSAMENTO DE GRAFOS: UMA ANÁLISE COMPARATIVA. 88 f. Trabalho de Diplomação – Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2014.

TENENBAUM, A. M. Estrutura de Dados Usando C. São Paulo: MAKRON Books, 1995.

What is Apache Hadoop? https://hadoop.apache.org/. The Apache Software Foundation, 2014. Acesso em 28 de set. 2015.

YOON, E. J. **Apache Hama (v0.2): User Guide**aBSPbased distributed computing framework. Publicado em 27 de fevereiro de 2011. http://pt.slideshare.net/udanax/apachehama02userguide. Acesso em 15 de nov. 2015.