UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE GRADUAÇÃO EM TECNOLOGIA EM SISTEMAS PARA INTERNET

TASSIA KYOKO WATANABE TANAKA

DESENVOLVIMENTO DE UM SERVIÇO EMBUTIDO EM MÁQUINA VIRTUAL PARA APRESENTAÇÃO DE DADOS CIENTÍFICOS EM PLATAFORMA DE NUVEM

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO - PR

2014

TASSIA KYOKO WATANABE TANAKA

DESENVOLVIMENTO DE UM SERVIÇO EMBUTIDO EM MÁQUINA VIRTUAL PARA APRESENTAÇÃO DE DADOS CIENTÍFICOS EM PLATAFORMA DE NUVEM

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de Tecnólogo .

Orientador: Prof. Me. Rodrigo Hübner

Co-orientador: Prof. Me. Alessandro Kraemer

CAMPO MOURÃO - PR

2014

Dedico este trabalho de conclusão de curso aos meus avôs, Yoshimi Watanabe e Takeo Tanaka, in memorian.

AGRADECIMENTOS

Gostaria primeiramente de agradecer a Deus e a Meishu Sama por sempre iluminarem o meu caminho.

Agradeço a minha família pelo carinho e compreensão nos momentos em que a dedicação aos estudos foi exclusiva e a todos os que contribuiram direta ou indiretamente para que este trabalho fosse realizado.

Agradeço ao meu orientador Prof. Me. Rodrigo Hübner e co-orientador Prof. Me. Alessandro Kraemer pelos ensinamentos e dedicação dispensados no auxilio à concretização desse trabalho.

Agradeço ao Eduardo Albertini pela parceria e grande ajuda no desenvolvimento do referencial teórico.

Aos meus amigos, em especial à M. Sakakima e M. Guimarães, por todo apoio e dedicação.

Aos professores do curso por todo ensinamento proporcionado durante o curso e pela dedicação disponibilizada nas aulas, que contribuiu muito para a realização deste trabalho e para minha formação profissional. Deixo aqui registrado meu profundo agradecimento.

ATA DA DEFESA DO TABALHO DE CONCLUSÃO DE CURSO

Às **vinte e uma e vinte horas** do dia **vinte e seis de agosto de dois mil e quatorze** foi realizada no Mini-auditório do EAD da UTFPR-CM a sessão pública da defesa do Trabalho de Conclusão do Curso Superior de Tecnologia em Sistemas para Internet do acadêmico **Tassia Kyoko Watanabe Tanaka** com o título **Desenvolvimento de um serviço embutido em máquina virtual para apresentação de dados científicos em plataforma de nuvem**. Estavam presentes, além do acadêmico, os membros da banca examinadora composta pelo professor **Me. Rodrigo Hübner** (Orientador-Presidente), pelo professor **Dra. Aretha Barbosa Alencar** e pelo professor **Me. Juliano Folleis**. Inicialmente, o aluno fez a apresentação do seu trabalho, sendo, em seguida, arguido pela banca examinadora. Após as arguições, sem a presença do acadêmico, a banca examinadora o considerou **APROVADO** na disciplina de Trabalho de Conclusão de Curso e atribuiu, em consenso, a nota ____ (_____). Este resultado foi comunicado ao acadêmico e aos presentes na sessão pública. A banca examinadora também comunicou ao acadêmico que este resultado fica condicionado à entrega da versão final dentro dos padrões e da documentação exigida pela UTFPR ao professor Responsável do TCC no prazo de **onze dias**. Em seguida foi encerrada a sessão e, para constar, foi lavrada a presente Ata que segue assinada pelos membros da banca examinadora, após lida e considerada conforme.

Observações:

_____

_____

_____

_____

_____

Campo Mourão, 26 de agosto de 2014.

Prof. Dra. Aretha Barbosa Alencar Membro	Prof. Me. Juliano Folleis Membro	Prof. Me. Rodrigo Hübner Orientador

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

RESUMO

TANAKA, Tassia Kyoko Watanabe. DESENVOLVIMENTO DE UM SERVIÇO EMBUTIDO EM MÁQUINA VIRTUAL PARA APRESENTAÇÃO DE DADOS CIENTÍFICOS EM PLATAFORMA DE NUVEM. 50 f. Trabalho de Conclusão de Curso – Curso de Graduação em Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Campo Mourão - PR, 2014.

A computação em nuvem é um importante recurso para pesquisadores, oportunizando uma vasta quantidade de recursos computacionais para executar experimentos, que são normalmente realizados usando algoritmos codificados em linguagem de programação. Os problemas resolvidos por tais algoritmos costumam ser pontuais, tais como a descoberta de sequencias genéticas, caminhos mais curtos entre uma origem e um destino, entre outros. Os resultados produzidos por esses algoritmos são na maioria das vezes demasiadamente complexos de serem analisados. Este trabalho de conclusão de curso tem como resultado o desenvolvimento de um serviço web para facilitar a geração de gráficos para análise sobre resultados de experimentos científicos em nuvem. O sistema desenvolvido facilita a produção de gráficos com base nos resultados gerados pelos algoritmos. Neste sistema é possível produzir gráficos com média, desvio padrão e intervalo de confiança de modo bastante automatizado, o que contribui significativamente com a análise de experimentos científicos. Neste texto é apresentado as tecnologias utilizadas para desenvolver tal sistema, onde ele foi implantado e como foi avaliado.

Palavras-chave: Computação em Nuvem, Estatística, Gráficos, Máquina Virtual, OpenNebula, Serviço Web.

ABSTRACT

TANAKA, Tassia Kyoko Watanabe. SERVICE DEVELOPMENT EMBEDDED VIRTUAL MACHINE FOR DATA SUBMISSION SCIENTIFIC CLOUD PLATFORM. 50 f. Trabalho de Conclusão de Curso – Curso de Graduação em Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Campo Mourão - PR, 2014.

Cloud computing is an important resource for researchers, providing opportunities a vast amount of computing resources to perform experiments, which are usually performed using algorithms coded in a programming language. The problems solved by these algorithms tend to be specific, such as the discovery of genetic sequences, shortest paths between a source and a destination, among others. The results produced by these algorithms are most often too complex to be analyzed. This work of completion of course has resulted in the development of a web service to facilitate the generation of graphics for analysis on results of scientific experiments in cloud. The developed system facilitates the production of graphs based on the results generated by the algorithms. In this system, it is possible to produce graphs with mean, standard deviation and confidence interval fairly automated, which contributes significantly to the analysis of scientific experiments. This text is presented the technologies used to develop the system indicated, where it was deployed and evaluated.

Keywords: Cloud Computing, Statistcs, Graphics, Virtual Machine, OpenNebula, WebService.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A Computação em Nuvem vem sendo amplamente utilizada, tanto para armazenamento de dados, como para execução de aplicações científicas, por exemplo, para armazenar e processar gráfos em larga escala. Com essas aplicações surge também a necessidade de uso de sistemas computacionais mais potentes, tais como *clusters* virtuais usando MPI ou Hadoop, bem como serviços que automatizam processos para facilitar a utilização desses recursos, criando interfaces visuais mais amigáveis para seus utilizadores, que muitas vezes não possuem um perfil tecnicista. Embora muitos usuários não saibam o significado do termo "Computação em Nuvem", vários utilizam o serviço, pois um arquivo compartilhado via DropBox ou um arquivo em GoogleDocs funcionam basicamente dentro do princípio de Computação em Nuvem, já que os dados não estão armazenados localmente, mas sim na "nuvem", potencialmente em máquinas virtuais distribuídas em um ou em vários servidores que estão localizados em algum lugar do mundo e podem ser acessados via redes de computadores.

Atualmente existem diversos *softwares* que implantam plataformas de nuvem em contextos públicos e privados. Também existem combinações desses dois modelos, conhecidos como híbridos e de comunidade. Neste trabalho de conclusão de curso é apresentada a plataforma `OpenNebula`, que é um *software* livre para uma infraestrutura como serviço para seus usuários (virtualização de computadores e redes de comunicação) e é comumente utilizada para implantação de nuvem privada. A UTFPR de Campo Mourão (UTFPR-CM) dispõe de uma nuvem com esse tipo de implantação e essa é uma boa razão para abordar o `OpenNebula` neste trabalho.

A nuvem `OpenNebula` da UTFPR-CM tem o objetivo de atender projetos científicos de diversas áreas do conhecimento, assim como outras atividades acadêmicas que envolvam o ensino de programação paralela e distribuída. O contexto de experimentos científicos envolvem a solicitação de instâncias de *cluster* virtual e a execução de algoritmos que exigem alta capacidade de processamento. Esse tipo de serviço já é oferecido para alguns projetos da UTFPR-CM, entretanto, os resultados dos algoritmos executados, o que normalmente implicam em medir o tempo de computação e o de comunicação, não são processados estatísticamente pela nuvem.

O objetivo deste trabalho é automatizar o processo de aplicação estatística nos resultados dos algoritmos científicos e apresentar gráficos. Essa automatização é o foco deste trabalho e será incorporada dentro das máquinas virtuais, tratada como um serviço para aplicações científicas na nuvem da UTFPR-CM.

A geração de gráficos para apresentar média, desvio padrão, entre outros, sem processos de automatização é uma tarefa que demanda tempo e paciência do desenvolvedor. Para que os gráficos sejam gerados é necessário primeiramente que os resultados dos algoritmos sejam capturados e inseridos, por exemplo, em uma planilha. A filtragem separa dados sobre computação e comunicação que ocorreram durante a execução do algoritmo na nuvem. Após a inserção na planilha são aplicadas fórmulas estatísticas. Os resultados estatísticos são inseridos em outro ambiente, por exemplo, o `GnuPlot`, que gera uma diversidade de gráficos utilizando a média, o desvio padrão, o intervalo de confiança, entre outras possibilidades. Os dados que são os resultados da filtragem também poderiam ser analisados diretamente pelo `GnuPlot`, mas isso exige muito ou algum esforço de codificação. Este trabalho visou automatizar esse processo por meio de um sistema especialista. Para isso, basta que os resultados sejam armazenados em um arquivo legível pelo sistema, que irá processá-los gerando ao final do processo um gráfico para o usuário, sem que esse necessite realizar todo o procedimento manualmente. Com base nesse contexto de pesquisa, surgiram alguns questionamentos: Como automatizar a geração de gráficos com base em resultados de algoritmos? Como implantar essa automatização dentro da nuvem?

Antes de implantar um sistema dentro da nuvem é necessário primeiramente compreender como a nuvem é formada, compreendendo seus principais mecanismos, para em seguida identificar as tecnologias necessárias para criar uma interface para o usuário, bem como as tecnologias necessárias para capturar os resultados dos algoritmos e gerar gráficos.

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta a metodologia e as tecnologias que serão utilizadas para desenvolver o trabalho, de acordo com cada objetivo específico. O Capítulo 3 descreve os conceitos de nuvem, tipos de nuvem, tipos de serviços e como a nuvem é composta. O Capítulo 4 apresenta como o serviço foi implementado, suas funcionalidades, utilizações e testes realizados. O Capítulo 5 apresenta as conclusões e os trabalhos futuros. Por fim, as referências e o apêndice.

2 METODOLOGIA E TECNOLOGIAS UTILIZADAS

Ao longo deste trabalho são apresentados conceitos sobre a Computação em Nuvem com o objetivo de deixar claro o significado desse termo, onde ele é utilizado e os principais componentes arquiteturais de uma plataforma. Com base nesse estudo é possível compreender melhor como a nuvem funciona, e dessa forma, desenvolver um serviço embutido em máquina virtual que possa ser instanciado dentro dela. Essa etapa de pesquisa sobre a Computação em Nuvem foi realizada por meio de publicações científicas e sites da Internet.

Para descobrir as tecnologias que dão suporte à este trabalho, abordamos trabalhos que utilizam a plataforma de nuvem da UTFPR-CM, apresentada na Figura 1. Tais trabalhos envolvem identificação do formato da máquina virtual aceito pela plataforma e características de seus sistemas embutidos, assim como testes de aplicações científicas e interface com o usuário. O *software* gerenciador da nuvem é o `OpenNebula`.

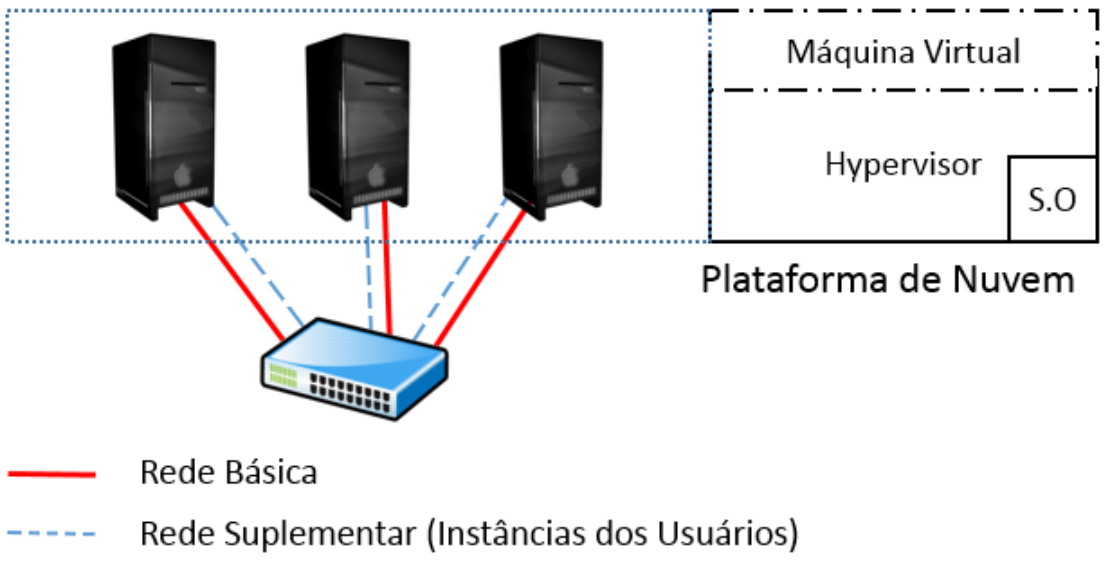


Figura 1: Infraestrutura de Nuvem da UTFPR de Campo Mourão.
Fonte: Adaptação de Cloud4University (2013).

Para relatar um caso não automatizado de execução de algoritmo na nuvem apresenta-

mos a Figura 2. Nessa figura há dois tipos de linhas, uma que representa o tempo computacional (vermelha) e outra que representa o tempo de comunicação (azul). O eixo 'x' representa a quantidade de máquinas virtuais utilizadas e o eixo 'y' o tempo, em segundos, gasto no processamento. Esses gráficos foram gerados manualmente, por meio do `GnuPlot`, sendo necessário que o usuário saiba como utilizar a ferramenta para poder gerar os gráficos. Para gerar esses gráficos, os dados utilizados foram filtrados um a um pelo usuário. Um sistema que automatiza esse processo é proposto neste trabalho.

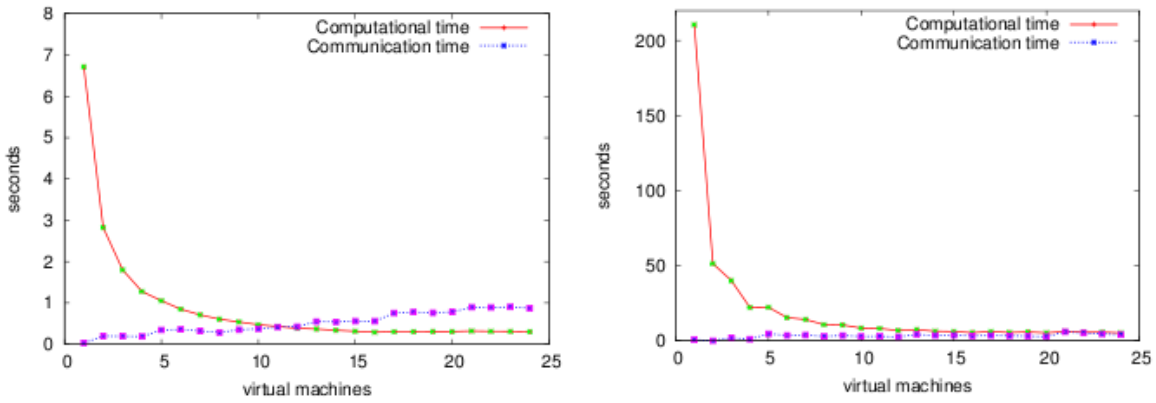


Figura 2: Exemplos de Gráficos.
Fonte: KRAEMER et al. (2013).

Para o desenvolvimento do serviço proposto identificamos primeiramente os parâmetros que os usuários necessitam, como a indicação do arquivo de saída do algoritmo executado, tipo de estatística (média, desvio padrão e intervalo de confiança), e tipo de gráfico (linha, coluna, etc). Essas necessidades foram identificadas por meio de entrevistas com professores que possuem publicações científicas usando gráficos sobre desempenho de algoritmos. O sistema proposto também deixa em aberto a possibilidade de adições de novas operações estatísticas ou tipos de gráficos. Após essa identificação, elaboramos um diagrama de fluxo do sistema. A implementação do sistema ocorreu em `script shell` e `Python-Tornado`. A geração dos gráficos é realizada pelo sistema `GnuPlot`. Por fim, testes foram realizados utilizando algoritmos de partição, ordenação paralela e método de Euler. Como resultado desse processo, os gráficos produzidos pelo serviço podem indicar, entre outras coisas, se os algoritmos são ou não escalares em nuvem. Esses gráficos normalmente apresentam o tempo de computação e o tempo de comunicação em diferentes cenários de *cluster*. Por exemplo, o tempo de processamento gasto com um processador, com dois processadores, e assim sucessivamente. A Figura 2 apresenta um exemplo de gráfico, conforme trabalho de KRAEMER et al. (2013).

O tipo da interface para o usuário depende dos recursos implantados na máquina virtual. Para este trabalho adaptamos a máquina virtual que foi utilizada por KRAEMER et al. (2013) durante a implantação de um *cluster* dinâmico na UTFPR-CM. Um detalhe importante desse contexto é que a interface com o usuário é via linha de comando (*shell bourne-again*, ou simplesmente `bash`). Existem muitas restrições de segurança que impedem alterações na configuração da rede de computadores da plataforma de nuvem e do acesso à UTFPR-CM. Por essa razão, o acesso às máquinas virtuais da nuvem é realizado via `SSH`. Com isso, implantamos um serviço web na máquina virtual e acessamos via `http` a partir de outra máquina, a fim de visualizar o resultado final em forma de gráfico, que é gerado em PDF.

Na seção seguinte são apresentadas as tecnologias que foram integradas para formar o novo sistema de serviço web, intitulado de serviço de apresentação de gráficos automatizado (SAGA). No capítulo de Referencial Teórico são apresentados detalhes sobre plataformas de nuvem.

2.1 TECNOLOGIAS QUE FORAM INTEGRADAS PARA FORMAR O SAGA

Um serviço web é capaz de integrar outros serviços, realizando comunicação entre diferentes aplicações. Para o desenvolvimento do serviço proposto foi utilizado a biblioteca para serviços web `Python-Tornado`. Esse serviço realiza a comunicação entre o servidor e o cliente. No cliente foi utilizada a linguagem de marcação `HTML`, enquanto que no servidor foi utilizado o `Python-Tornado`, atrelado a linguagem `Python` e alguns comandos de `GnuPlot` para gerar os gráficos, assim como comandos `shell` para obter dados do servidor. Dentro do `HTML` foi implementado o `WebSocket` e o `JavaScript`.

2.1.1 LINGUAGEM DE PROGRAMAÇÃO `PYTHON`

A linguagem de programação `Python` foi criada em 1990 por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda (CWI) e tinha como foco original usuários como físicos e engenheiros. O `Python` foi concebido a partir de outra linguagem existente na época, chamada ABC, que é uma linguagem orientada a objetos, interpretada e interativa.

Essa linguagem possui uma sintaxe clara e concisa, que favorece a legibilidade do código fonte, tornando o desenvolvimento mais produtivo. A linguagem inclui diversas estruturas de alto nível (listas, tuplas, dicionários, data/hora, números complexos e outras) e uma vasta coleção de módulos prontos para uso, além de *frameworks* de terceiros que podem ser adicio-

nados. Possui um grande conjunto de funcionalidades pré-interpretadas e portáveis, conhecidas como bibliotecas padrão que suportam diversas tarefas de programação em nível de aplicativo, desde processamento de texto até redes de computadores. Além disso, pode ser estendido com bibliotecas desenvolvidas pelo programador, assim como suporte a bibliotecas de outros fornecedores (LUTZ; ASCHER, 2007).

2.1.2 BIBLIOTECA PYTHON-TORNADO

O Python-Tornado é uma biblioteca que consegue lidar com grandes quantidades de conexões abertas e ao mesmo tempo ser leve para gerenciar grandes aplicações e utilitários. Ele possui ferramentas que lidam com a segurança e autenticação do usuário e é um servidor web orientado a evento, com fluxo de execução exclusiva, ou seja, cada fluxo tem o potencial para coordenar diversas conexões abertas. As ferramentas dessa biblioteca que foram utilizadas neste trabalho foram o tornado.websocket, tornado.web, tornado.httpserver e tornado.ioloop.

2.1.3 LINGUAGEM DE PROGRAMAÇÃO JAVASCRIPT

O JavaScript foi criado pela Netscape em parceria com a Sun Microsystems com a finalidade de fornecer um meio para adicionar interatividade a uma página web. A primeira versão, denominada JavaScript 1.0, foi lançada em 1995 e implementada em março de 1996 no navegador Netscape Navigator 2.0.

A linguagem de programação JavaScript foi criada para executar do lado do cliente, ou seja, a interpretação e o funcionamento da linguagem dependem de funcionalidades hospedadas no navegador do usuário. Em tese precisamos apenas que o navegador possua um interpretador JavaScript hospedado para que possa executar *scripts* desenvolvidos nessa linguagem.

Uma das funcionalidades do JavaScript é descrever marcações HTML dinamicamente e inserí-las na marcação de um documento existente, por exemplo mensagens de boas-vindas ou ainda, inserção de conteúdos diferenciados e escolhidos de acordo com o navegador do usuário. Outra funcionalidade é manipular o navegador, podendo controlar o comportamento deste em diversos aspectos, como por exemplo, criar janelas *pop-up*. Além disso é capaz de interagir com outras linguagens para cumprir tarefas complementares relacionadas a um fluxo de programação (SILVA, 2010).

2.1.4 WEBSOCKET

Segundo Ubl e Kitamura (2010) WebSocket é um protocolo que permite a comunicação bidirecional por canais *full-duplex* sobre um único soquete, que utiliza o TCP (protocolo de controle de transmissão) que é projetado para ser executado em navegadores e servidores web. A especificação WebSocket define uma API que estabelece conexões de soquete, entre um navegador da web e um servidor. Em outras palavras, existe uma conexão persistente entre o cliente e o servidor e em ambas as partes podem começar a enviar dados a qualquer momento. O WebSocket é muito utilizado em conexões em tempo real e de baixa latência entre o cliente e o servidor.

2.1.5 GNUPLOT

GnuPlot é um utilitário de linha de comando para gerar gráficos nas plataformas Linux, OS, Windows, entre outras. Ele foi criado em 1986 e está em desenvolvimento desde então. O GnuPlot serve para permitir que cientistas e estudantes visualizem funções matemáticas e dados de forma interativa, por exemplo, gráficos e outras não tão interativas assim, como *scripts* web. Ele também é usado como um mecanismo de plotagem por aplicativos de terceiros (WILLIAMS; KELLEY, 2004).

2.1.6 SHELL SCRIPT

O shell é um *prompt* da linha de comando que age como um interpretador, recebendo os comandos digitados pelo usuário e os executando. O Shell Script é uma linguagem de *script* muito utilizada no Linux que, por ser uma linguagem interpretada, não necessita de um compilador e sim de apenas um programa como o bash para interpretar os comandos dentro do *script*. Já os *scripts* são arquivos onde ficam armazenados os comandos de shell e que podem ser executados a qualquer momento (JUNGTHON; GOULART, 2011).

3 REFERENCIAL TEÓRICO

Neste capítulo será apresentado os conceitos e aplicações de computação em nuvem, os tipos de serviços da nuvem e a caracterização de uma plataforma para computação em nuvem.

3.1 CONCEITOS E APLICAÇÕES DE COMPUTAÇÃO EM NUVEM

A origem da expressão computação em nuvem (ou *cloud computing*) é desconhecida. A Britannica (2013) diz que a expressão descende da prática de utilizar desenhos de nuvens para representar a infraestrutura dos sistemas de computação e telecomunicação. Segundo a (NIST) Computação em nuvem é um modelo para permitir acesso onipresente, conveniente e sob demanda à rede para um conjunto compartilhado de recursos computacionais configuráveis (redes, servidores, armazenamento, aplicações e serviços) que podem rapidamente ser provisionados e liberados com um esforço mínimo de gerenciamento ou interação com o provedor de serviços. Sousa et al. (2010) definem a nuvem como uma abstração que oculta a complexidade da infraestrutura. Cada parte dessa infraestrutura possui um serviço a ser oferecido e esses geralmente são colocados em *Data Centers*, utilizando o compartilhamento de *hardware* para processamento e armazenamento.

Uma infraestrutura de nuvem é composta de várias máquinas físicas conectadas por meio de uma rede. Cada máquina possui configurações de *software* compatíveis com as demais máquinas que formam o cenário da nuvem. Também podem haver diferentes especificações de *hardware*, por exemplo, máquinas que possuem diferentes quantidades e tipos de processadores, quantidade de memória e até mesmo quantidade de armazenamento em discos diferentes. Apesar das diferenças de *hardware*, dentro da nuvem podem haver várias máquinas virtuais (MVs) em execução. A plataforma de nuvem, composta por vários subsistemas, permite a virtualização sob diferentes *hardwares*. A quantidade de MVs em execução depende da capacidade do *hardware* da máquina real (SOUSA et al., 2010). Complementarmente, cada plataforma de nuvem pode gerenciar a rede e os recursos de virtualização de formas diferentes.

Sousa et al. (2009) dizem que para acessar os serviços disponíveis na nuvem os usuá-

rios necessitam ter apenas alguns *softwares*, que por sua vez interagem com a nuvem. Esses *softwares* podem ser sistemas operacionais, navegadores web ou APIs de integração. Todos os recursos estão acessíveis na nuvem por meio da Internet e os usuários não necessitam de uma máquina com altos recursos, o que leva à diminuição do custo de aquisição/manutenção de *hardware* pelos usuários.

A computação em nuvem foi desenvolvida com a meta de fornecer serviços de baixo custo, fácil acesso e com garantias de disponibilidade e escalabilidade. É uma evolução dos serviços e tecnologias sob demanda, também conhecida como *Utility Computing*. O objetivo dela é fornecer recursos computacionais como armazenamento e processamento através de provedores e cobrar um determinado valor por isso. Uma vantagem deste tipo de serviço é que o seu valor é cobrado de acordo com a quantidade de uso (SOUSA et al., 2009).

Computação em nuvem é uma maneira bastante eficiente de maximizar a utilização dos recursos computacionais, por meio da virtualização de *hardware*. O *software* conhecido como *gerenciador de recursos virtuais* consegue criar várias instâncias de *hardware* simulando um computador. Com isso, vários computadores podem ser virtualizados em uma plataforma de nuvem. No ambiente de plataformas de computação em nuvem podem ser aplicadas técnicas de tolerância a falhas que permitem a continuidade do funcionamento de alguns recursos caso um ou mais desses apresente defeito (TAURION, 2009).

A partir de meados de 2008, a computação em nuvem tornou-se conhecida pelo mundo todo e foi amplamente adotada para facilitar a utilização de alguns serviços, como os repositórios de dados e serviços web. Alguns desses exemplos são: iCloud, Google Drive, Dropbox, Yahoo! Cloud, entre outros. Os serviços web implantados em nuvem são utilizados para facilitar e ajudar o desenvolvedor a criar e gerenciar suas aplicações. Um exemplo desse tipo de serviço é o Amazon Web Service (AWS).

Além da computação em nuvem ser utilizada como repositório de dados e como provedor de serviços web, ela pode ser utilizada também em meios acadêmicos. A Universidade da Califórnia (UC) em Berkeley, com o apoio da Amazon Web Services, utilizou a computação em nuvem em um de seus cursos para focar no desenvolvimento de aplicações SaaS (*Software* como Serviço). Outro exemplo é da Faculdade de Medicina de Wisconsin Biotecnologia e Centro de Bioengenharia em Milwaukee que utilizou a computação em nuvem para fazer pesquisas de proteínas e torná-las mais acessíveis para cientistas do mundo todo. Há também o caso da Faculdade de Engenharia Elétrica e Ciência da Computação (EECS) da Universidade Estadual de Washington, que utiliza a nuvem para a economia na aquisição de *hardware*, devido a cortes orçamentais (SULTAN, 2010).

3.1.1 TIPOS DE NUVEM

De acordo com a finalidade das plataformas de nuvem, Sousa et al. (2009) classificam em três tipos: nuvem privada, nuvem pública e nuvem híbrida. Alguns autores incluem o tipo de nuvem comunitária. Essas divisões foram realizadas de acordo com os tipos de acesso, serviços e restrições de quem acessa a nuvem. A Figura 3 apresenta uma arquitetura geral sobre esses tipos.

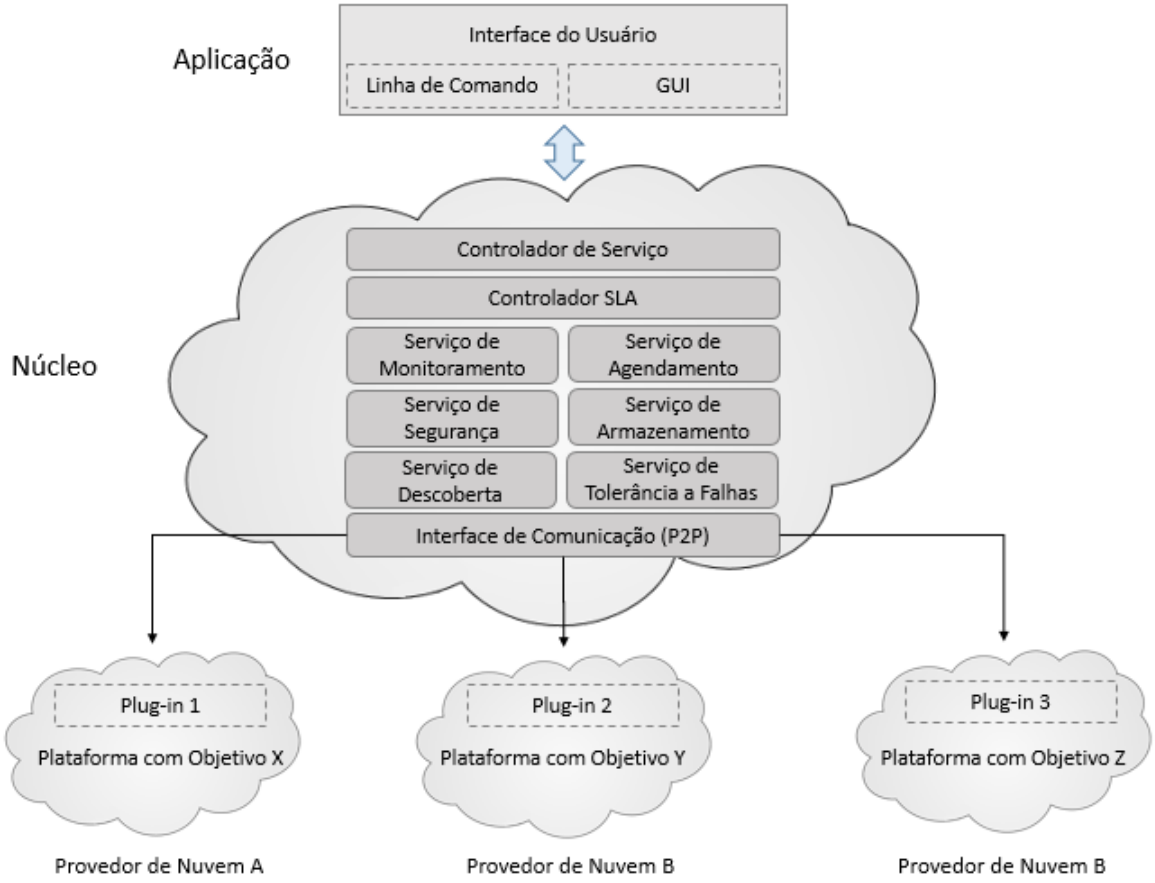


Figura 3: Arquitetura genérica com nuvens integradas.
Fonte: Adaptação de Saldanha et al. (2012, p. 5)

A Interface do Usuário é o ponto de partida para que a nuvem instancie os recursos desejados. Os usuários podem utilizar diversos tipos de interface, por exemplo, por meio de linha de comando, formulários web, etc. O núcleo da nuvem, normalmente um computador dito como mestre, recebe as solicitações de tarefas dos usuários que verifica o contrato do cliente e aciona demais serviços para fazer a instanciação dos recursos. Esses serviços dentro da nuvem são apresentados na Seção 3.2. Na mesma Figura 3, podemos observar que vários provedores de nuvem podem estar integrados, formando assim uma grande rede de comunicação e serviços

que inclui diferentes tipos de plataformas.

No tipo de nuvem privada, a infraestrutura da nuvem é utilizada somente por uma organização, sendo que esta pode ser uma nuvem local ou remota. A administração é feita pela própria empresa ou por empresas terceirizadas (MELL; GRACE, 2013). A Figura 4 apresenta um pacote de serviços chamado de C-Cloud, que é formado por várias nuvens privadas, caracterizando-se assim como uma grande Nuvem. Nesse exemplo há uma sequência de interações entre usuários, *hardware* e serviços disponíveis.

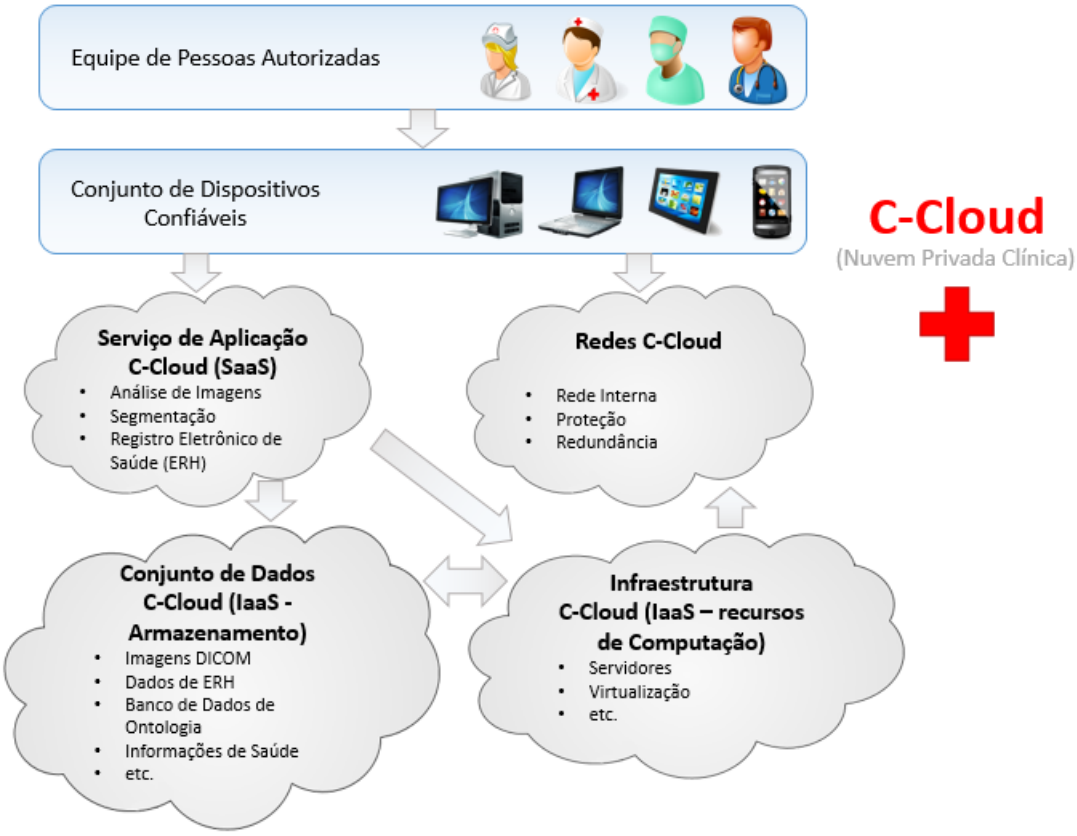


Figura 4: Arquitetura de uma nuvem privada.

Fonte: Schoenhagen et al. (2013, p. 10)

Na Figura 4 é possível observar a divisão da nuvem para que cada segmento possa prover um serviço diferente para os usuários, por exemplo, uma parte responsável por disponibilizar serviços de rede, outra parte disponibilizando serviços de processamento e outra disponibilizando o armazenamento de resultado de exames. Tudo isso é solicitado através de alguns dispositivos que fazem interface com o usuário. Esse tipo de nuvem pode ser utilizado por uma rede de hospitais, na qual eles podem compartilhar os dados de exames, independentes da localização real de cada hospital.

Na nuvem pública a infraestrutura é disponibilizada para o público em geral. Nesse

tipo de nuvem o usuário pode ser cobrado pelas solicitações (exemplo, Amazon) ou não (exemplo, Google Docs). Independente do porte das empresas, todas podem usar a nuvem para resolverem seus problemas, como servidor de aplicações, repositório de dados, etc. Estudantes e usuários domésticos também podem utilizar a nuvem pública, contudo, esse tipo de nuvem é de uso geral e para uma ampla diversidade de público.

Na nuvem híbrida há duas ou mais nuvens que podem ser públicas, privadas ou comunitárias conectadas através de uma rede. Essas nuvens integradas são vistas como uma grande nuvem, onde cada nuvem é responsável por um serviço diferente das outras e assim se completam para prover todos os serviços necessários. Um exemplo de infraestrutura de nuvem híbrida foi apresentado na Figura 3, onde há várias nuvens que são ligadas por um *plugin* de comunicação (SOUSA et al., 2009).

Por fim, na nuvem comunitária há o compartilhamento da nuvem por diversas instituições que possuem um interesse em comum, como a missão, requisitos de segurança, entre outros (MELL; GRACE, 2013).

3.2 TIPOS DE SERVIÇO

Atualmente os serviços de computação em nuvem podem ser divididos em vários modelos, entre eles Infraestrutura como Serviço (IaaS), *Software* como Serviço (SaaS), Plataforma como Serviço (PaaS), Desenvolvimento como Serviço (DevaaS), Comunicação como Serviço (CaaS), Tudo como Serviço (EaaS), Internet como Serviço (NaaS), Banco de Dados como Serviço (DBaaS), dentre outros, que são solicitados de acordo com a oferta de serviços. Mas diversos autores abordam apenas três como sendo os principais modelos (SaaS, PaaS e IaaS). A Figura 5 apresenta alguns setores que tem relacionamento com esses modelos.

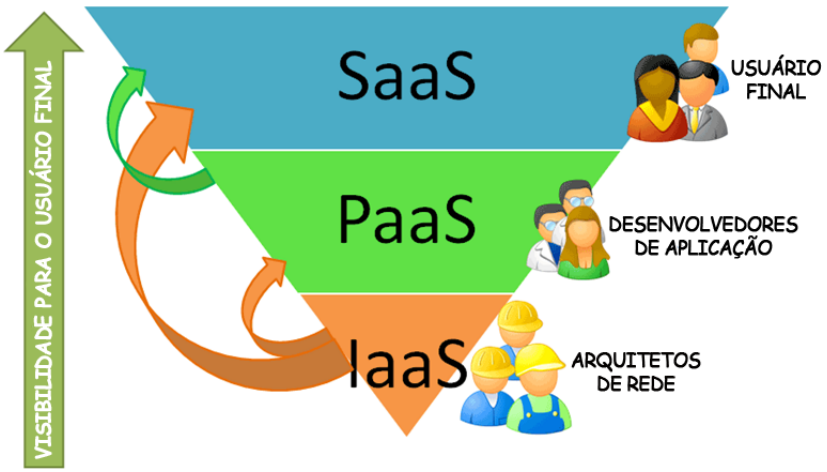


Figura 5: Relacionanento entre diversas áreas de atuação e os tipos de serviço da nuvem.
Fonte: Meriat (2011)

Na Figura 5, temos alguns exemplos de usuários que podem utilizar os serviços de computação em nuvem. Os usuários finais utilizam o SaaS, pois não é necessário nenhum tipo de programação, como exemplo, temos o Skype que é um *software* de comunicação, onde o usuário utiliza o serviço e tem a possibilidade de utilizá-lo gratuitamente ou pagar por algum serviço em específico. Desenvolvedores de aplicação utilizam o PaaS, pois este oferece uma boa infraestrutura para implementação de aplicações na nuvem, assim os desenvolvedores não ficam restritos apenas as configurações de *hardware* e *software* de sua própria máquina. Arquitetos de rede utilizam o IaaS, pois este serviço oferece acesso aos recursos de *hardware* como armazenamento, rede e outros.

3.2.1 *SOFTWARE* COMO SERVIÇO (SAAS)

No modelo SaaS as aplicações são entregues para os usuários por meio da Internet, proporcionando sistemas de *software* com propósitos específicos. Não é necessário instalação e manutenção de *software*, basta acessá-lo por meio de um navegador web. Com isso, o usuário livra-se de um *software* muito complexo e do gerenciamento de *hardware* (SULTAN, 2010). O usuário não administra e nem controla a infraestrutura, apenas solicita configurações de *hardwares* e *softwares* que devem ser instanciados. Algumas vezes, a própria plataforma oferece soluções prontas. Assim, os desenvolvedores podem se concentrar apenas na inovação e não na infraestrutura, o que proporciona o desenvolvimento rápido de sistemas de *software*.

Por ser um serviço web, poderá ser acessado a qualquer momento e de qualquer lugar, necessitando apenas de uma conexão com a Internet e oferecendo uma maior integração entre os serviços de *software*. Alguns exemplos de SaaS são os serviços de *Customer Relationship*

Management (CRM) da *Salesforce* e o Google Docs (SOUSA et al., 2010).

3.2.2 PLATAFORMA COMO SERVIÇO (PAAS)

O modelo PaaS oferece uma infraestrutura de alto nível de integração para implementar e testar aplicações na nuvem (SOUSA et al., 2010). Para entender esse modelo é necessário lembrar do modelo de computação tradicional, onde cada aplicativo gerenciado localmente requer *hardware*, um sistema operacional, banco de dados, servidores web e outros serviços (SULTAN, 2010). No PaaS o usuário não administra ou controla a infraestrutura, mas tem controle sobre as aplicações implantadas e possivelmente as configurações das aplicações hospedadas nessa infraestrutura (SOUSA et al., 2010). Também podemos tratar o PaaS como uma plataforma de desenvolvimento e implantação de um conjunto de APIs, bibliotecas, linguagens de programação e ferramentas associadas utilizadas para a criação de aplicativos (VORAS et al., 2011).

No PaaS os desenvolvedores dispõem de ambientes escaláveis, porém, há algumas limitações que o ambiente impõe desde a concepção das aplicações até a utilização de sistemas de gerenciamento de banco de dados (SGBDs). O PaaS permite que usuários se inscrevam para solicitações de serviços de TI ou para resoluções de problemas pela web. *PaaS Google App Engine* e Aneka são exemplos dessa plataforma (SOUSA et al., 2010).

3.2.3 INFRAESTRUTURA COMO SERVIÇO (IAAS)

O IaaS é útil como camada base para outros modelos de serviço, como o PaaS e o SaaS (DAWOUD et al., 2010). O IaaS é responsável por fornecer toda a infraestrutura necessária para o PaaS e o SaaS, tendo como principal objetivo tornar mais fácil e acessível o fornecimento de recursos, tais como servidores, rede, armazenamento e também recursos para construir ambientes sob demanda, que podem incluir sistemas operacionais e aplicativos dos quais o PaaS e o SaaS não se preocupam em gerenciar. Algumas características relevantes do IaaS são a interface única para administração da infraestrutura, uma API para interação com os *hosts*, *switches*, balanceadores, roteadores e o suporte para a adição de novos equipamentos de forma simples e transparente. No IaaS os usuários possuem controle sobre os sistemas operacionais, armazenamento e aplicativos implantados (SOUSA et al., 2010).

A infraestrutura pode se modificar dinamicamente, aumentando ou diminuindo os recursos de acordo com as necessidades das aplicações. Isso é possível porque a infraestrutura é baseada em técnicas de virtualização de recursos de computação. Podemos mencionar como

exemplos de IaaS o *Amazon Elastic Cloud Computing* (EC2) e o *Elastic Utility Computing Architecture Linking Your Programs To Useful Systems* (*Eucalyptus*) (SOUSA et al., 2010).

3.3 CARACTERIZAÇÃO DE UMA PLATAFORMA PARA COMPUTAÇÃO EM NUVEM

A computação em nuvem possui algumas plataformas de código livre, entre elas podemos citar o *Eucalyptus*, `OpenNebula`, *Nimbus* e *OpenStack*. Neste trabalho utilizamos especificamente o `OpenNebula`. Existem também plataformas privadas, como o *Elastic Compute Cloud* (EC2) e *Simple Storage Service* (S3), ambas da Amazon. Como não temos acesso às plataformas privadas proprietárias, podemos apenas destacar os componentes que as plataformas de código livre tem em comum, entre eles, o *cluster* de máquinas físicas, o virtualizador dos recursos computacionais e o gerenciador de recursos da nuvem, explanados a seguir.

Por muito tempo os supercomputadores foram líderes no campo da computação, porém, devido a necessidade de processar *softwares* que exigem maior desempenho e escalabilidade pelas áreas de ciência, engenharia e negócios. No entanto, ainda que existam supercomputadores os *clusters* surgiram como uma alternativa de baixo custo. Um *cluster* é um conjunto de computadores paralelos ou distribuídos interligados entre si através de uma rede de alta velocidade, dividindo a carga de trabalho computacional. Eles executam tarefas computacionais em conjunto que não seriam possíveis de executar em um computador comum. Do ponto de vista do usuário, esses computadores interligados podem ser vistos como um único computador virtual onde as solicitações do usuário são recebidas e distribuídas entre todos os computadores que formam o *cluster* (SADASHIV; KUMAR, 2011). Assim, pode-se processar grande quantidade de dados sem sobrecarregar um único computador. Esses dados são distribuídos por um *host* ”mestre” para os *hosts* ”escravos”. Um exemplo de uso de *cluster* consiste em executar algoritmos paralelos que computam simultaneamente grande volume de dados. Sendo o objetivo nesse caso, a obtenção de maior desempenho.

Dentro da plataforma de nuvem, cada *host* do *cluster* pode virtualizar recursos computacionais como *hardware*, disco rígido e rede. Entretanto para que isso ocorra é necessário que um gerenciador de recursos de virtualização esteja instalado, que é comumente conhecido como *Hypervisor* e a Figura 6 apresenta seus principais recursos.

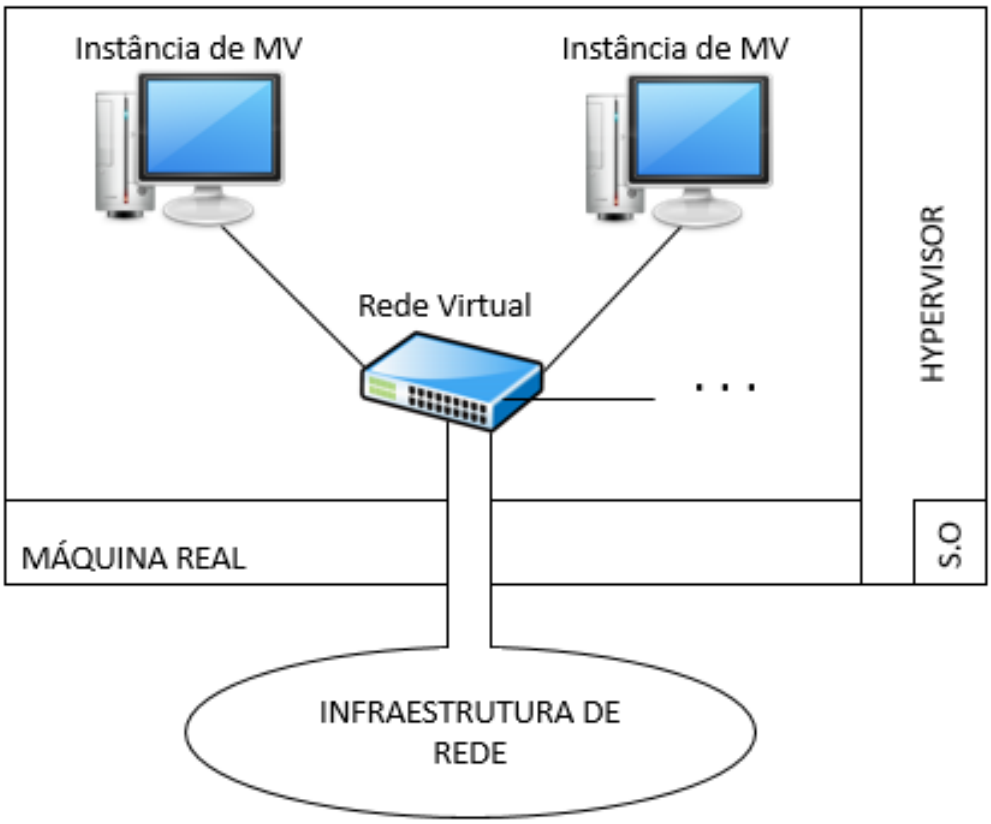


Figura 6: Principais recursos de gerenciamento do *Hypervisor*

Fonte: Adaptação de KRAEMER et al. (2013, p. 4)

O *Hypervisor* possui um monitor de recursos que tem o papel de fazer a comunicação entre a máquina virtual e a máquina real, ou seja, ela recebe as instruções enviadas pela máquina virtual, trata-as e as repassa para a máquina real, e vice-versa (SMITH; NAIR, 2005). Existem diversos *Hypervisor* disponíveis no mercado, alguns exemplos são: Xen, KVM (*Kernel-based Virtual Machine*), VMWare, Qemu, Microsoft Hyper-V, Virtual Box, entre outros. Dentre esses, destacamos o KVM, o Xen e o VMWare. O KVM é o *Hypervisor* padrão da plataforma `OpenNebula` que é a plataforma utilizada em nossos experimentos. O Xen é utilizado principalmente pela Amazon. O VMWare é uma plataforma proprietária que pode ser adquirida para formar todos os tipos de nuvem. Dessa forma abordamos os principais *Hypervisors* de *software* livre (KVM e Xen) e *software* proprietário (VMWare). A Tabela 1 apresenta as principais características desses virtualizadores.

Hypervisor	**VMWare vSphere 5.1**	**Red Hat RHEV 3.2**[1]	**Citrix XenServer 6.2**
Última Revisão	Setembro 2012	Junho 2013	Junho 2013
Gerenciamento Baseado em Navegador	Sim	Sim	Não
Gerenciamento de Operações Avançadas	Sim	Não	Não
Segurança	ESXi Firewall, vShield Zones e vShield Endpoint	SELinux, iptables,VLANs e Port Mirroring	iptables
Migração Viva de MVs	vMotion, Metro vMotion	Live Migration	XenMotion
Migração Viva Automatizada	CPU, Mem, Armazenamento, vCD	CPU	não
Migração de Armazenamento	Live Storage vMotion	Fully suported	Storage XenMotion
Tamanho do *cluster*	32 hosts/ 4000 MVs	200 hosts/*cluster*	16 hosts
Taxa Máx. de Consolidação	512 MVs, 2048 vCPU	Ilimitado	500 MVs(Windows) | 650 MVs(Linux) por Host
Máx. CPU por Host	160	160	160
Máx. Cores por CPU	Ilimitado	Ilimitado	Ilimitado
Máx. Memória por Host	2 TB	2 TB	1 TB
Máx. vCPU por MV	64	160	16(Windows) | 32(Linux)
Máx. RAM por MV	1 TB	2 TB	128 GB
Overcommit Dinâmico	Memory Ballooning	Virtio	DMC
Compartilhamento de Paginação	Transparent Page Sharing	KSM	não

Hypervisor	**VMWare vSphere 5.1**	**Red Hat RHEV 3.2**[1]	**Citrix XenServer 6.2**
Suporte a SO Convidado	Ótimo	Limitado	Bom
API de Scripting	CIM/SMASH API, SDK, Perl, Power CLI	REST API,Pyton CLI, Hooks, SDK	SDK, API, PowerShell
API de Nuvem	vCloud API	REST API, Deltacloud API	CloudStack API, AWS API
Armazenamento Suportado	DAS, NFS,FS, iSCSI, SDD for Swap, FCoE, FC HBA	DAS, iSCSI, NFS,FC, FCoE, SAS, POSIX	DAS, SAS, iSCSI, NAS, FC, FCoE
Formato do Disco Virtual	vmdk, raw disk (RDM)	RAW, Qcow2	vhd, raw disk(LUN)
Máx. Tamanho do Disco	2 TB (vmdk) / 64 TB (RDM)	8 TB	2 TB
QoS de Armazenamento	SIOC, NFS	não	básico
Switch de Rede Avançado	vDS	Não	Open vSwitch
VLAN	Sim	Sim	Limitado
QoS de Rede	netIOC	Não	Sim

Tabela 1: **Comparação de diversas características entre os principais *Hypervisors*. Fonte: Virtualization Matrix**

O *Hypervisor* é utilizado pelo Gerenciador de Recursos da Nuvem para que as instâncias sejam criadas. O usuário solicita instâncias, que por sua vez são processadas pelo Gerenciador de Recursos que é o responsável por distribuí-las nos demais *hosts* da nuvem (máquinas do *cluster* físico) e essa distribuição de recursos pode ser balanceada e monitorada. O balanceamento é uma forma de não sobrecarregar um *host* da nuvem e o monitoramento é uma forma de acompanhar o uso dos recursos computacionais, como CPU, Memória RAM e rede. Assim, a cada novo pedido de instância haverá uma alocação de forma balanceada, que é resultado

[1]Apesar do Red Hat RHEV 3.2 ser um *software* proprietário, em sua implemetação ele utiliza o *Hypervisor* KVM, por isso a sua presença nessa comparação, pois os dados apresentados são referentes ao KVM.

de uma otimização da plataforma. Na plataforma `OpenNebula` esse recurso não faz parte da instalação padrão.

O Gerenciador de Recursos da Nuvem, ou simplesmente Gerenciador da Nuvem, é responsável por designar as tarefas que serão executadas dentro da nuvem, desde a adição de máquinas físicas ao *cluster* até a instância de novas MVs. Para as MVs realizarem atividades de redimensionamento de recursos conforme a demanda são utilizados dois módulos chamados de *Policy Decision Module* (PDM) e *Policy Enforcer Module* (PEM). O PDM é responsável por receber os pedidos das MVs e analisá-los e quando o pedido é aceito, ele repassa as ações a serem executadas para o PEM. Caso o pedido seja negado, uma notificação de rejeição é enviada ao usuário. O PEM se comunica com a nuvem e é o responsável pela criação, implantação e migração de MVs. Além disso, ele é responsável por executar os pedidos repassados pelo PDM como por exemplo, listar as MVs em execução, os *hosts* disponíveis, entre outros (APOSTOL et al., 2011).

A Tabela 2 mostra um comparativo entre diversos gerenciadores de nuvem e nessa comparação estão os *Hypervisors* que podem ser utilizados por cada um deles.

Gerenciador de Nuvem	**Open Source**	**Interface do Usuário**	***Hypervisor***
Amazon Web Service	Não	AWS Management Console, AWS APIs, Comand Line Interface (CLI)	XEN
Apache CloudStack	Sim	CloudStack API, AWS APIs, CLI, Web Interface	XEN, KVM, QEMU, VMWare e Hyper-V
Eucalyptus	Sim	AWS APIs, CLI, Web Interface	XEN, KVM, QEMU, VMWare e Hyper-V
`OpenNebula`	Sim	AWS APIs, Open Cloud Computing Interface (OCCI), CLI, Sunstone Web Interface	XEN, KVM, QEMU, VMWare e Hyper-V
OpenStack	Sim	AWS APIs, CLI, *OpenStack* Dashboard Web Interface	XEN, KVM, QEMU, PowerVM, Hyper-V, LXC e VMWare
VMWare vCloud	Não	vCloud API,vCloud Director e vCloud Connector	VMWare

Tabela 2: Comparação entre diversos Gerenciadores de Nuvem.
Fonte: Adaptação de KRAEMER et al. (2013, p. 2)

Em baixo nível arquitetural, uma plataforma de nuvem é formada pelo *cluster* físico que contêm o *Hypervisor* implantado em cada *host*. Para gerenciar todos os recursos computacionais disponibilizados por essa infraestrutura, a plataforma usa o Gerenciador de Nuvem. Entre diversas plataformas disponíveis utilizaremos o `OpenNebula` porque é *software* livre, já está implantado na UTFPR-CM, se integra com os principais *Hypervisors* e dispõe de várias formas de interfacear com o usuário, conforme a Tabela 2.

O `OpenNebula` possui um conjunto de ferramentas de *software* livre de computação em nuvem utilizado para gerenciar a infraestrutura de *Data Centers* complexos e heterogêneos, podendo oferecer formas flexíveis que ajustam dinamicamente os recursos necessários e oportuniza uma melhor interoperabilidade para construir nuvens privadas, públicas ou híbridas, do tipo IaaS (Infraestrutura como Serviço) (WEN et al., 2012). Em 2008 o `OpenNebula` foi publicado por Ignácio M. Llorente e Rubén S. Montera e desde então opera como um projeto de *software* livre, sendo o carro-chefe da União Européia no domínio dos projetos de investigação

em relação à virtualização e computação em nuvem (WIND, 2011). Algumas comunidades que utilizam ativamente o `OpenNebula` são: o Centro de Astronomia Espacial Européia (*European Space Astronomy Centre*) e a Organização Européia para Pesquisa Nuclear (*European Organization for Nuclear Research* (CERN)) (CORDEIRO et al., 2010). Ele foi projetado para ser modular de modo a permitir a sua integração com o maior número de *Hypervisors* e ambientes possível (CORDEIRO et al., 2010). KVM, XEN, Hyper-V e VMWare (conforme Tabela 2) são alguns exemplos de *Hypervisors* compatíveis com o `OpenNebula`. A infraestrutura física do `OpenNebula` adota uma arquitetura de *cluster-like* clássica com um *front-end*, e um conjunto de nós onde MVs serão executadas. Há pelo menos uma rede física juntando todos os nós do *cluster* com o *front-end* que executa os principais processos do `OpenNebula` enquanto os nós do *cluster* são *hosts* habilitados chamados de "escravos" e que tem o papel de iniciar as MVs utilizando algum *Hypervisor* e que fornecem os recursos necessários para as MVs (CORDEIRO et al., 2010).

Arquiteturalmente a plataforma `OpenNebula` é composta por três camadas, conforme Figura 7.

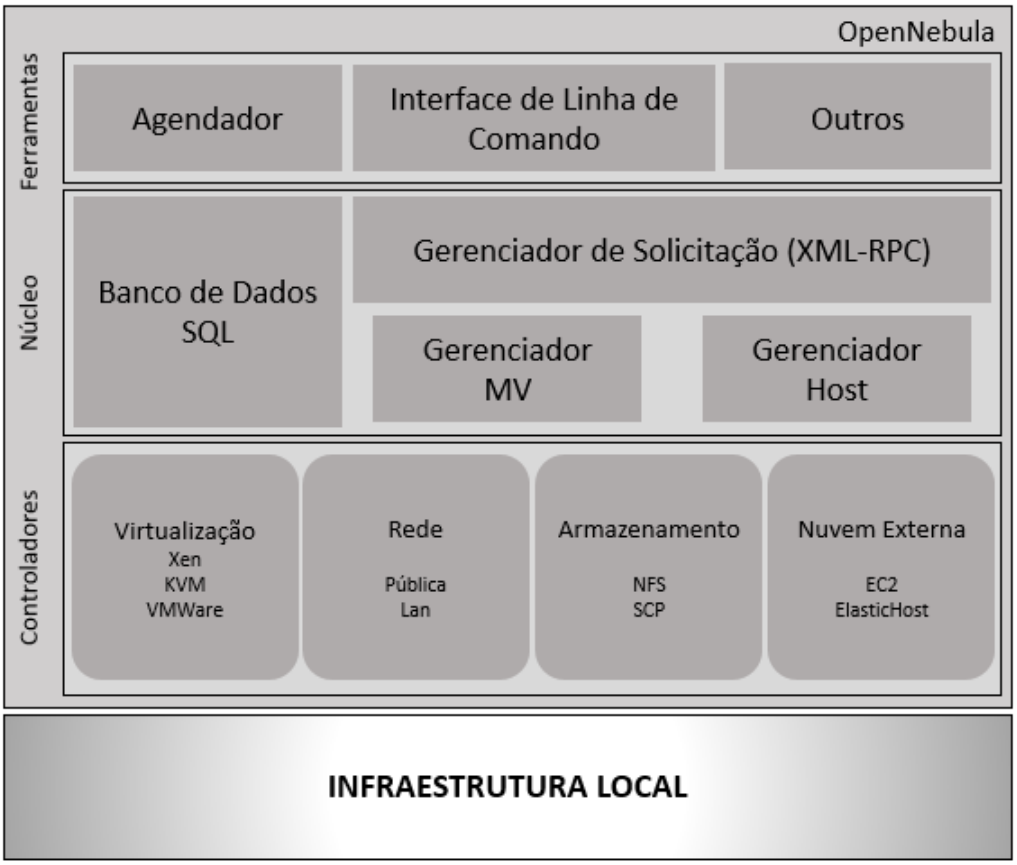


Figura 7: Arquitetura do Gerenciador de Nuvem `OpenNebula`

Fonte: Adaptação de Cordeiro et al. (2010), p.4.

A primeira camada (Ferramentas) contém funções para os administradores e os usuários da nuvem (WIND, 2011) e um dos componentes dessa camada é a Interface de Linha de Comando (*Command Line Interface* ou CLI), que pode ser utilizado pelos administradores para manipular a infraestrutura através de comandos, e o módulo Agendador é responsável pela alocação de MVs (CORDEIRO et al., 2010). A segunda camada (Núcleo) possui componentes usados para processar solicitações dos usuários e controle de recursos (WIND, 2011), sendo seu principal componente o Gerenciador de Solicitações (CORDEIRO et al., 2010). Por fim, a terceira camada (Controladores) dá suporte a diferentes tipos de plataformas subjacentes onde, nesse nível existem controladores que são executados em processos separados que se comunicam com o módulo central através de um protocolo de mensagem simples. Possui também controladores que regulam a transferência de dados e controlam as máquinas virtuais que estão em cada *host*, independente dos *Hypervisors* (WIND, 2011). Ainda nessa camada há controladores que lidam com a transferência de arquivos, que são implementados por protocolos de rede como NFS e SSH. Possuem os controladores para gerenciar máquinas virtuais que são dependentes do *Hypervisor* instalado no *host* da nuvem. E os que solicitam serviços de nuvens externas como o Amazon EC2 (CORDEIRO et al., 2010).

4 IMPLEMENTAÇÃO DO SERVIÇO

Neste capítulo é apresentado como o serviço deste trabalho foi implementado, suas funcionalidades, suas utilizações e as discussões dos gráficos elaborados pelos testes realizados com este serviço.

4.1 SERVIÇO PROPOSTO NESTE TRABALHO

Na Figura 8 é apresentado o que ocorre quando o usuário utiliza o serviço proposto neste trabalho. É apresentado para o usuário uma página web com a qual ele interage e essa interface está localizada no nó mestre da nuvem, e dentro do `HTML` está o `WebSocket` que se comunica com o `WebService`, enviando requisições a todo momento. O `WebService` está localizado na MV mestre, no qual espera por uma requisição para processar e enviar uma determinada resposta. Ainda dentro do `WebService`, ocorre uma comunicação do serviço com o prompt da linha de comando e com uma biblioteca do `GnuPlot`.

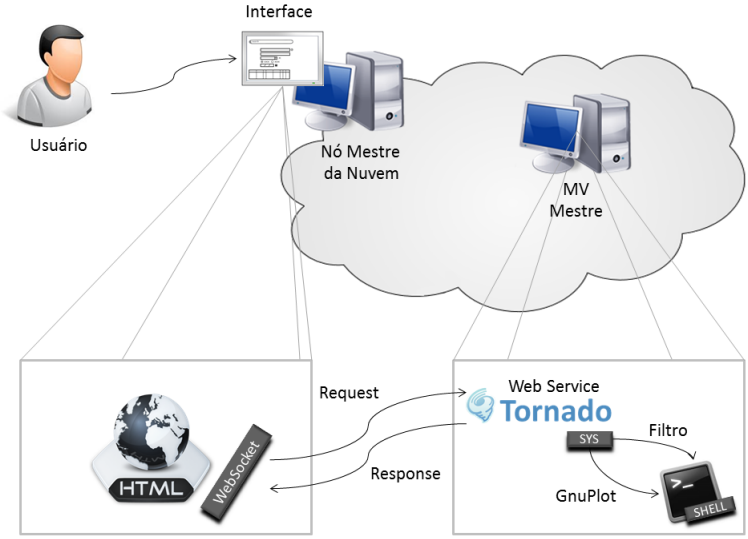


Figura 8: Contexto do `WebService` na nuvem

4.2 REQUISITOS PARA A INTERFACE DO SISTEMA

Para a utilização do serviço proposto, serão listados alguns requisitos necessários que deverão ser seguidos para que a ferramenta funcione corretamente:

- O usuário deve possuir os arquivos de saída dos algoritmos já processados na nuvem, um exemplo de saída encontra-se no apêndice B;
- É necessário que os arquivos de saída estejam devidamente formatados e que os tempos a serem analisados estejam todos na mesma coluna a fim de evitar possíveis erros;
- É necessário verificar em qual posição da coluna está o tempo de comunicação/computação a ser analisado;
- Os dados de saída dos algoritmos devem estar com a extensão '.txt' ou '.csv';
- O padrão de nome dos arquivos deve ser o seguinte: "`nome_arquivo_x1`", onde `x1` é a quantidade de processadores que foram utilizados para processar o algoritmo;
- É necessário que o usuário saiba o caminho para a pasta de onde estão armazenados os arquivos de saída;
- É necessário conhecer o(s) nome(s) do(s) arquivo(s) de saída(s) no qual será(ão) utilizado(s) para gerar o(s) gráfico(s);
- Verificar se os arquivos de saída possuem o tempo de processamento com ponto ou vírgula para adicionar como filtro quando solicitado pelo sistema;

4.3 IMPLANTAÇÃO DO SISTEMA NO MODELO DE MV NA NUVEM DA UTFPR-CM

O `WebService` desenvolvido neste trabalho foi implantado como um serviço na MV mestre da nuvem e só será utilizado quando a MV estiver instanciada. Assim, o *socket* sempre estará acessível para os usuários e não haverá necessidade de implantar o serviço nas demais MV's, pois as saídas são armazenadas em uma pasta compartilhada pela própria MV mestre.

Em cada MV foi utilizado o sistema operacional Debian e nele existe um `skeleton` para adição de novos serviços administrados durante seus modos de operação (boot, desligamento, etc). O `skeleton` é um arquivo com a estrutura básica das funções de *start* e *stop*, possibilitando ao desenvolvedor condições de realizar apenas uma cópia desse arquivo e incluir os comandos para iniciar o `WebService`. O `WebService` proposto neste trabalho atua na MV

mestre como um desses serviços e o acesso é realizado por meio do IP do *socket* da mesma MV. Nos capítulos seguintes, serão apresentados os testes realizados com o serviço desenvolvido.

4.4 TESTES COM SAÍDAS REAIS

As saídas foram criadas à partir de algoritmos que fazem parte de uma biblioteca MPI (Message Passing Interface), chamada CGMLIB dos pesquisadores Chan e Dehne (1999) e que foi incorporada ao modelo de MVs da nuvem da UTFPR-CM. O MPI realiza uma comunicação de dados padrão em computação paralela e diante das várias modalidades que existem nessa computação paralela, o MPI oferece uma infraestrutura para a solução de problemas que necessitam passar as informações por vários processadores ou nós de um *cluster*.

4.4.1 TESTE COM SAÍDAS DO ALGORITMO MÉTODO DE EULER

No gráfico da Figura 9 foram analisados os tempos de comunicação e de computação das saídas do algoritmo Método de Euler, onde a linha vermelha representa o tempo computacional e a linha azul o tempo de comunicação. Os pontos em azul representam a média de tempo, os pontos em verde o desvio padrão, conforme Figura 9(a) e os pontos em rosa, o intervalo de confiança, conforme Figura 9(b).

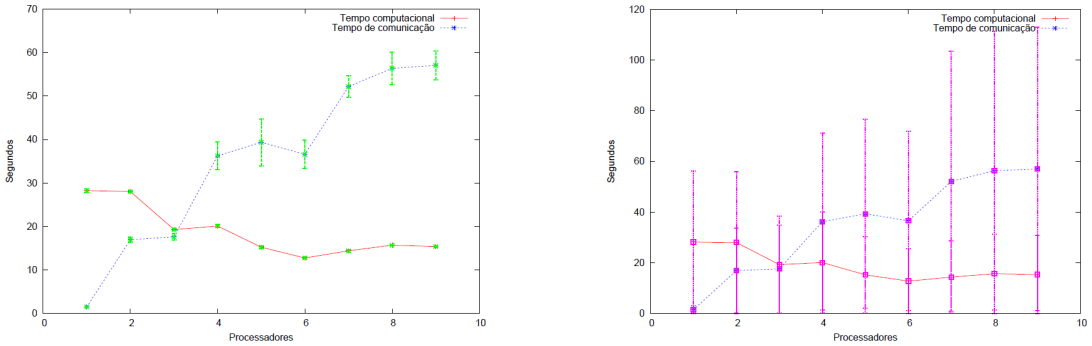
Podemos observar que conforme a quantidade de processadores aumenta, o tempo computacional diminui, enquanto o inverso ocorre com o tempo de comunicação.

Foi verificado também que a partir de sete processadores, o tempo computacional teve um aumento não significativo. Assim o usuário pode observar que a partir desse momento, já não é mais viável aumentar a quantidade de máquinas virtuais, pois o tempo de processamento pode não diminuir.

4.4.2 TESTE COM SAÍDAS DE ALGORITMO DE ORDENAÇÃO PARALELA

No gráfico da Figura 10 foram analisados os tempos de comunicação e de computação das saídas de um algoritmo de Ordenação Paralela, onde a linha vermelha representa o tempo computacional e a linha azul o tempo de comunicação. Os pontos azuis representam a média de tempo (Figura 10(a) e 10(b)), os pontos verdes, o desvio padrão (Figura 10(a)) e os pontos rosas representam o intervalo de confiança (Figura 10(b)).

Podemos observar que a Figura 10 segue um padrão parecido com o da Figura 9, pois conforme a quantidade de processadores aumenta o tempo computacional diminui e o tempo de



(a) Algoritmo do Método de Euler com média e desvio padrão

(b) Algoritmo do Método de Euler com média e intervalo de confiança

Figura 9: Gráfico das saídas do algoritmo do Método de Euler. Pontos em azul representam a média em ambos os gráficos, pontos em verde, o desvio padrão conforme Figura 9(a) e pontos em rosa, o intervalo de confiança, conforme Figura 9(b)

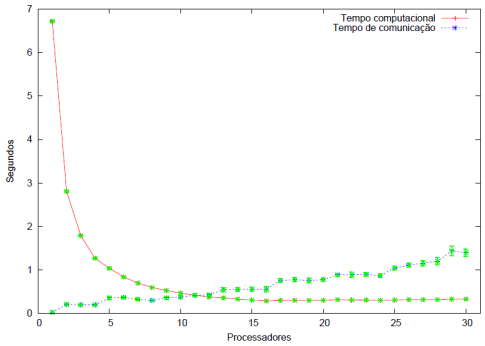
comunicação aumenta, mas diferente do Método de Euler é possível verificar uma diminuição significativa quando comparamos o procedimento realizado com apenas um processador com o de quinze processadores.

A partir de quinze processadores é possível observar que não há uma diminuição significativa com relação ao tempo de processamento, porém também é observado que quanto mais processadores são utilizados, maior é o tempo de comunicação entre as MVs escravas e a MV mestre, ou seja, é necessário dividir o algoritmo para mais máquinas e depois juntar todos os dados na MV mestre, o que demanda maior tempo de processamento e torna inviável o aumento de processadores.

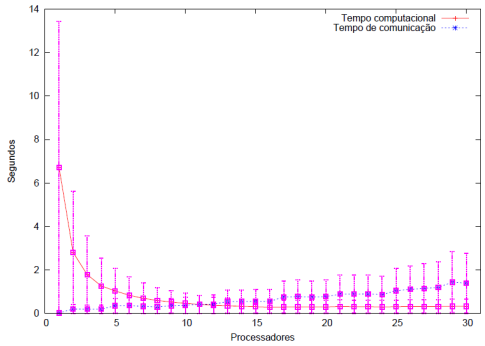
4.4.3 TESTE COM SAÍDAS DE ALGORITMO DE PARTIÇÃO

No gráfico da Figura 11 foram analisados os tempos de comunicação e de computação das saídas de um algoritmo de Partição, onde a linha vermelha representa o tempo computacional e a linha azul o tempo de comunicação. Na Figura 11(a) os pontos verdes representam o desvio padrão, na Figura 11(b) os pontos rosas, o intervalo de confiança e em ambas os pontos azuis são a média.

No algoritmo de Partição podemos observar que diferentemente dos outros dois algoritmos, o tempo de comunicação em nenhum momento fica maior que o tempo computacional, conforme a quantidade de processadores aumenta. A partir do vigésimo primeiro processador os valores se igualam e além disso, após o vigésimo quinto processador é visto que o aumento



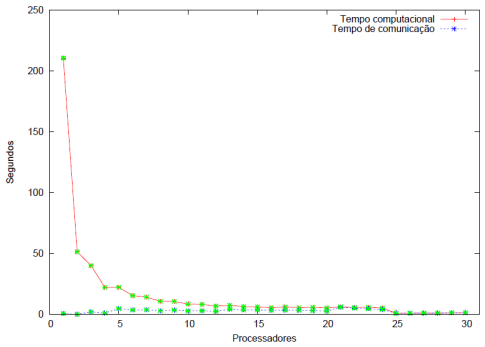
(a) Algoritmo de Ordenação Paralela com média e desvio padrão



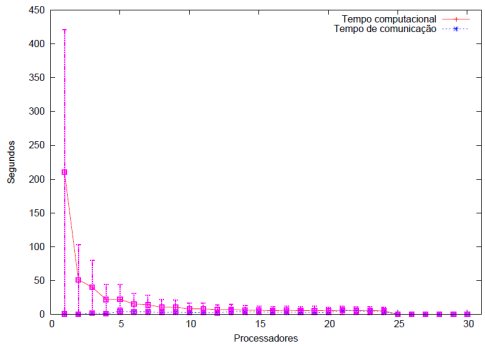
(b) Algoritmo de Ordenação Paralela com média e intervalo de confiança

Figura 10: Gráfico das saídas do algoritmo de ordenação Paralela. Pontos em azul representam a média em ambos os gráficos, pontos em verde, o desvio padrão conforme Figura 10(a) e pontos em rosa, o intervalo de confiança, conforme Figura 10(b)

na quantidade de processadores não interfere na diminuição do tempo.



(a) Algoritmo de Partição com média e desvio padrão



(b) Algoritmo de Partição com média e intervalo de confiança

Figura 11: Gráfico de saídas do algoritmo de Partição. Pontos em azul representam a média em ambos os gráficos, pontos em verde o desvio padrão conforme Figura 11(a) e pontos em rosa, o intervalo de confiança, conforme Figura 11(b)

4.5 ESTATÍSTICAS APLICADAS NA APRESENTAÇÃO DOS RESULTADOS

No SAGA, o usuário poderá escolher quais os cálculos estatísticos deseja utilizar, sendo que para efetuar essas análises, foram utilizadas as seguinte fórmulas:

Média:

$$\frac{\sum(ti)}{n} \tag{1}$$

onde,

n: Número total de elementos.

$\sum(ti)$: É o somatório de todos os tempos que serão analisados.

Variância:

$$\frac{\sum(ti - Media)^2}{(n-1)} \tag{2}$$

onde,

$\sum(ti - Media)^2$: É o somatório do quadrado do tempo menos a média.

(n-1): É o número total de elementos menos um.

Desvio Padrão:

$$\sqrt{V} \tag{3}$$

onde,

$\sqrt{V}$: Raiz quadrada da variância.

Intervalo de Confiança 95%:

$$M - 1.96 * (\frac{Dp}{\sqrt{n}}) \tag{4}$$

onde,

M: É a média.

Dp: É o desvio padrão.

$\sqrt{N}$: É a raiz quadrada do número total de elementos.

4.6 COMO ACESSAR O SERVIÇO

Na Tabela 3 é possível verificar os links de acesso para o `WebService`. Na ligação `ip_da_maquinavirtual:8888/config`, são encontrados dez parâmetros de entrada para enviar ao servidor e que servem para criar o gráfico. Os parâmetros de entrada são:

- **Diretório do Arquivo**: Local onde estão armazenados as saídas dos algoritmos que foram processados na nuvem e que serão analisados pelo serviço para gerar os gráficos. A entrada é o caminho relativo até a pasta que será analisada;
- **Nome do(s) arquivo(s)**: Nome das saídas que serão analisadas. Estes nomes podem ser

passados separados por (ponto e vírgula) (por exemplo: `arquivo_x1;arquivo_x2`) ou também é possível digitar " ** " e o programa interpreta que todos os arquivos dentro do diretório escolhido serão analisados;

- **Separador decimal**: É necessário escolher se o tempo que será analisado pelo algoritmo está formatado com (ponto) ou (vírgula) , pois no programa a vírgula é substituída pelo ponto;
- **Campo para filtragem**: Campo dentro do arquivo que será analisado pelo programa;
- **Delimitador**: Caractere que delimita as palavras da saída, para que o programa separe o arquivo em colunas;
- **Posição requisitada**: Coluna onde o tempo que será analisado se encontra dentro do arquivo;
- **Estatística aplicada**: Será aplicada no gráfico uma análise estatística que já tem a média por padrão. Podem ser escolhidas outras duas estatísticas, dentre elas, o desvio padrão ou intervalo de confiança ou então ambas;
- **Identificador único**: Nome que os arquivos que serão gerados irão receber. Serão gerados 3 arquivos, um arquivo contendo o *script* `GnuPlot`, um arquivo texto e um arquivo pdf;
- **Label X**: Nome que o eixo X terá no gráfico gerado;
- **Label Y**: Nome que o eixo Y terá no gráfico gerado.

Na ligação `ip_da_maquinavirtual:8888/merge`, é necessário passar como parâmetro dois tipos de entrada, são elas:

- **Nome dos identificadores**: Nome dos dois arquivos pdf de gráficos que foram gerados a partir da ligação de configuração. Estes nomes são separados por (ponto e vírgula) como exemplo `Arquivo_A;Arquivo_B`;
- **Identificador único do novo arquivo**: Nome do novo arquivo que será gerado. Serão gerados dois arquivos, um arquivo `GnuPlot` e um pdf.

Existem outras duas ligações (`ip_da_maquinavirtual:8888/gnuplot` e `ip_da_maquinavirtual:8888/pdf`), ambas as interfaces foram criadas para realizar o download do arquivo de script gnuplot e do arquivo pdf respectivamente e era necessário que

o usuário escolhesse o arquivo a ser baixado e o local em que este arquivo seria armazenado, clicando em baixar. Porém, após vários testes de implementação sem sucesso, foi descartado o seu uso.

Acesso	**Parâmetro**	**Tipo de Entrada**
.../config	Diretório do Arquivo	Caminho completo ou diretório em que se encontra os arquivos
	Nome do(s) arquivo(s)	** para todos os arquivos ou nome dos arquivos separados por vírgula
	Separador decimal	ponto (.) ou vírgula (,)
	Campo para filtragem	Campo que será analisado. Ex: computational
	Delimitador	Delimitador que separa as palavras dentro do arquivo. Ex: ‘ ’ ou ‘;’
	Posição requisitada	Posição em que se encontra o tempo que será analisado. Ex: 10
	Estatística aplicada	Média (fixo), desvio padrão (selecionável), intervalo de confiança (selecionável)
	Identificador único	Nome que os arquivos gerados irão receber. Ex: arquivo.gp; arquivo.pdf; arquivo.txt
	Label x	Nome da Label x
	Label y	Nome da Label y
.../merge	Nome dos Identificadores	Nome do dentificador único dos arquivos gerados na configuração, separados por ‘;’
	Identificador único do novo arquivo	Nome que os arquivos gerados irão receber. Ex: arquivo.gp; arquivo.pdf

Tabela 3: Links de Acesso para o `WebService`

A Figura 12 representa a interface da ligação `ip_da_maquinavirtual:8888/config`, nesta interface o usuário configura os dados que serão passados por parâmetro para o `WebService`, que gera três tipos de arquivo. Um arquivo de *script* do `GnuPlot` para gerar o gráfico que pode ser baixado pelo usuário e modificado da forma que necessitar para gerar outros tipos de gráficos e que encontra-se no apêndice C. Outro arquivo é um pdf que contém o gráfico gerado e que também pode ser baixado. O terceiro arquivo é um texto utilizado pelo *script* `GnuPlot` para gerar o gráfico pdf e que está anexado no apêndice D. Todos esses arquivos são armazenados em uma pasta local do servidor. Os parâmetros requisitados descritos na Tabela 3, foram escolhidos desta forma, para que o serviço fosse o mais flexível e genérico possível.

[0] As “...” representam o endereço e a porta da MV: IP_da_maquina:8888

Figura 12: Tela de Configuração para gerar o Gráfico.

A Figura 13 representa a interface da ligação `ip_da_maquinavirtual:8888/merge`, nesta interface é possível passar o caminho do arquivo de dois pdf de gráficos já criados para um outro gráfico que junta os mesmos e dessa forma apresentar diferentes gráficos em apenas um. Para compor esse segundo gráfico é necessário passar o identificador único dos dois gráficos que serão agrupados, sendo que este identificador foi pré-definido nas configurações do primeiro, e é necessário também nomear o novo arquivo que será gerado e ao enviar esses dados para o `WebService` um novo gráfico é formado e armazenado na pasta local do servidor.

Figura 13: Tela para juntar dois gráficos.

Foram criadas nove classes para o serviço proposto que são apresentadas na Tabela 4, são elas: `WebSocketHandler`, `WebSocketHandlerGrafico`, `IndexPageHandler`, `GerarPDF`, `GerarGnuplot`, `SocketGnu`, `Config`, `Graficos` e `Application`.

A classe `Application` gerencia a chamada das outras classes. Quando o usuário digita uma das ligações apresentadas na Tabela 3, é esta classe que fica responsável por realizar a

chamada das outras classes, identificando por meio da ligação, qual deve chamar.

As classes IndexPageHandler, GerarPDF, GerarGnuplot, Graficos e Config, são classes que chamam os arquivos HTML para abrir uma interface para o usuário. O IndexPageHandler é solicitado quando a ligação ip_da_maquinavirtual:8888/ é ativado, nesta página aparece apenas uma mensagem para o usuário com o nome desse projeto em questão. GerarPDF e GerarGnuplot são classes chamadas pelas ligações ip_da_maquinavirtual:8888/pdf e ip_da_maquinavirtual:8888/gnuplot respectivamente, e abrem uma tela para que o usuário digite o identificador único dos arquivos gerados para poder baixar o pdf ou o *script* GnuPlot, apesar da página HTML estar funcionando, não é possível baixar os arquivos, pois as classes que realizam tal função não está implementada corretamente. Graficos é ativado quando a ligação ip_da_maquinavirtual:8888/merge é solicitado pelo usuário que abre a página referente a Figura 13, já exemplificada. Config é ativada quando a ligação ip_da_maquinavirtual:8888/config é solicitado pelo usuário abrindo a página referente a Figura 12, também já exemplificada.

As classes WebSocketHandler, WebSocketHandlerGrafico e SocketGnu, são classes que realizam o processamento do sistema. O SocketGnu é ativado pela comunicação entre o WebSocket da ligação ip_da_maquinavirtual:8888/gnuplot e a classe, mas não está sendo utilizado pois a implementação não funcionou. Os parâmetros exemplificados na Tabela 3, são os parâmetros de entrada apresentados na Tabela 4 e que são passados para as classes WebSocketHandler (ativada na comunicação entre o WebSocket da página ip_da_maquinavirtual:8888/config e a classe) e WebSocketHandlerGrafico (ativado na comunicação entre o WebSocket da página ip_da_maquinavirtual:8888/merge e a classe), após o processamento uma mensagem é enviada para o usuário como parâmetro de saída.

As classes WebSocketHandler, WebSocketHandlerGrafico e SocketGnu, são classes que realizam o processamento do algoritmo. O SocketGnu é ativado pela comunicação entre o WebSocket da ligação ip_da_maquinavirtual:8888/gnuplot e a classe, porém não está sendo utilizado pois a implementação não funcionou. Já os parâmetros exemplificados na Tabela 3, são os parâmetros de entrada apresentados na Tabela 4 e que são passados para as classes WebSocketHandler (ativada na comunicação entre o WebSocket da página ip_da_maquinavirtual:8888/config e a classe) e WebSocketHandlerGrafico (ativado na comunicação entre o WebSocket da página ip_da_maquinavirtual:8888/merge e a classe) após o processamento é enviada uma mensagem para o usuário como parâmetro de saída.

Classe	**Parâmetros de Entrada**	**Parâmetros de Saída**
WebSocketHandler	Diretório do Arquivo, Nome do Arquivo, Separador Decimal, Campo para Filtragem, Delimitador, Posição Requisitada, Estatística Aplicada, Identificador Único, Label X, Label Y	Mensagem de Status
WebSocketHandlerGrafico	Nome dos Identificadores, Identificador Único	Mensagem de Status
IndexPageHandler	Sem parâmetros de entrada	Mensagem de Status
GerarPDF	Sem parâmetros de entrada	Página HTML
GerarGnuplot	Sem parâmetros de entrada	Página HTML
SocketGnu	Nome do Arquivo	Txt do Arquivo gerado
Config	Sem parâmetros de entrada	Página HTML
Graficos	Sem parâmetros de entrada	Página HTML
Application	Sem parâmetros de entrada	Chama a classe requisitada

Tabela 4: Classes do `WebService`

5 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho de conclusão de curso foi apresentado as atividades de pesquisa e desenvolvimento na elaboração de uma interface Humano-Computador para facilitar a interpretação da análise de estatísticas, referentes aos resultados de experimentos científicos executados em nuvem. A nuvem privada da UTFPR-CM permite realizar experimentos com algoritmos de diversas áreas de conhecimento. Entretanto, resultados de algoritmos não são muitas vezes fáceis de serem interpretados. A fim de construir um sistema de geração de gráficos para resolver esse problema, apresentando média, desvio padrão e intervalo de confiança, foi realizado algumas pesquisas sobre os fundamentos da computação em nuvem e como esses gráficos podem ser gerados de forma automatizada. Também foi apresentado um exemplo de geração de gráfico de forma não automatizada, onde identificamos que essa atividade demanda muito esforço técnico. Por fim, apresentamos o SAGA que pode ser incorporado em plataformas de nuvem. Esse sistema foi testado analisando resultados de algoritmos científicos e pode ser classificado como satisfatório.

A partir deste estudo, são propostos os seguintes trabalhos futuros:

- Realizar uma implementação para permitir que o usuário consiga adicionar outros métodos estatísticos em seus resultados;
- Realizar uma implementação para aumentar a quantidade e a possibilidade de gráficos que podem ser gerados (como um histograma, dispersão e etc.).

REFERÊNCIAS

APOSTOL, E.; BALUTA, I.; GORGOI, A.; CRISTEA, V. Efficient manager for virtualized resource provisioning in cloud systems. In: **Intelligent Computer Communication and Processing (ICCP), 2011 IEEE International Conference on**. Cluj-Napoca, Romania: IEEE Xplore, 2011. p. 511–517.

BRITANNICA, E. **Encyclopedia Britannica - Cloud Computing**. 2013. Acesso em 21 ago. 2013. Disponível em: <http://www.britannica.com/EBchecked/topic/1483678/cloud-computing>.

CHAN, A.; DEHNE, F. **cgmGRAPH / cgmLIB: A Library For Coarse-Grained Parallel Computing**. 1999. Acesso em 01 ago. 2014. Disponível em: <http://people.scs.carleton.ca/~dehne/projects/Cgmlib/>.

CLOUD4UNIVERSITY. **Cloud computing environment for academic activities and research projects.** 2013. Acesso em 20 set. 2013. Disponível em: http://cloud4university.org/.

CORDEIRO, T.; DAMALIO, D.; PEREIRA, N.; ENDO, P.; PALHARES, A.; GONçALVES, G.; SADOK, D.; KELNER, J.; MELANDER, B.; SOUZA, V.; MåNGS, J.-E. Open source cloud computing platforms. In: **Grid and Cooperative Computing (GCC), 2010 9th International Conference on**. Nanjing, China: IEEE CS Press, 2010. p. 366–371.

DAWOUD, W.; TAKOUNA, I.; MEINEL, C. Infrastructure as a service security: Challenges and solutions. In: **Informatics and Systems (INFOS), 2010 The 7th International Conference on**. Cairo, Egypt: IEEE Xplore, 2010. p. 1–8.

JUNGTHON, G.; GOULART, C. M. Artigo sobre shell script. 2011.

KRAEMER, A.; OLIVEIRA, J. C.; SANTOS, F. G.; MACIEL, A. C.; GOLDMAN, A.; CORDEIRO, D. A. Dynamic creation of bsp/cgm clusters on cloud computing platforms. **4TH International Conference on Emerging Intelligent Data and Web Technologies - Workshop: Third International Workshop on the Service for Large Scale Distributed Systems (SeDiS-2013)**, 2013.

LUTZ, M.; ASCHER, D. **Aprendendo Python**. Porto Alegre - RS: Bookman, 2007.

MELL, P.; GRACE, T. **The NIST Definition of Cloud Computing**. 2013. Acesso em 29 ago. 2013. Disponível em: <http://www.cloudbook.net/resources/stories/the-nist-definition-of-cloud-computing>.

MERIAT, V. **Modelos de Serviço na Nuvem: IaaS, PaaS e Saas**. 2011. Acesso em 10 jun. 2014. Disponível em: <http://vitormeriat.com.br/2011/07/08/modelos-de-servio-na-nuvem-iaas-paas-e-saas/>.

SADASHIV, N.; KUMAR, S. Cluster, grid and cloud computing: A detailed comparison. In: **Computer Science Education (ICCSE), 2011 6th International Conference on**. Singapore, Singapore: IEEE Xplore, 2011. p. 477–482.

SALDANHA, H.; RIBEIRO, E.; BORGES, C.; ARAúJO, A.; GALLON, R.; HOLANDA, M.; WALTER, M.; TOGAWA, R.; SETUBAL, J. Towards a hybrid federated cloud platform to efficiently execute bioinformatics workflows, bioinformatics, dr. horacio pérez-sánchez. **InTech**, 2012.

SCHOENHAGEN, P.; ZIMMERMANN, M.; FALKNER, J. Advanced 3-d analysis, client-server systems, and cloud computing — integration of cardiovascular imaging data into clinical workflows of transcatheter aortic valve replacement. **Cardiovascular Diagnosis and Therapy**, v. 3, n. 2, p. 80–92, 2013. ISSN 2223-3660. Disponível em: <http://www.thecdt.org/article/view/1583>.

SILVA, M. S. **JavaScript, Guia do Programador**. São Paulo - SP: Novatec, 2010.

SMITH, J.; NAIR, R. The architecture of virtual machines. **Computer**, v. 38, n. 5, p. 32–38, 2005.

SOUSA, F. R.; MOREIRA, L. O.; MACÊDO, J. A. F. de; MACHADO, J. C. Gerenciamento de dados em nuvem: Conceitos, sistemas e desafios. **Topicos em sistemas colaborativos, interativos, multimidia, web e bancos de dados, Sociedade Brasileira de Computacao**, p. 101–130, 2010.

SOUSA, F. R.; MOREIRA, L. O.; MACHADO, J. C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. **III Escola Regional de Computação Ceará-Maranhão-Piauí, ERCEMAPI**, v. 1, 2009.

SULTAN, N. Cloud computing for education: A new dawn? **International Journal of Information Management**, Elsevier, v. 30, n. 2, p. 109–116, 2010.

TAURION, C. **Computação em Nuvem - Transformando o Mundo da Tecnologia da Informação**. Rio de Janeiro - RJ: Brasport, 2009.

UBL, M.; KITAMURA, E. **Apresentando WebSockets: trazendo soquetes para a web**. 2010. Acesso em 19 jul. 2014. Disponível em: <http://www.html5rocks.com/pt/tutorials/websockets/basics/>.

VORAS, I.; MIHALJEVIC, B.; ORLIC, M.; PLETIKOSA, M.; ZAGAR, M.; PAVIC, T.; ZIMMER, K.; CAVRAK, I.; PAUNOVIC, V.; BOSNIC, I.; TOMIC, S. Evaluating open-source cloud computing solutions. In: **MIPRO, 2011 Proceedings of the 34th International Convention**. Opatija, Croatia: IEEE Xplore, 2011. p. 209–214.

WEN, X.; GU, G.; LI, Q.; GAO, Y.; ZHANG, X. Comparison of open-source cloud management platforms: Openstack and opennebula. In: **Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on**. Sichuan, China: IEEE Xplore, 2012. p. 2457–2461.

WILLIAMS, T.; KELLEY, C. **Gnuplot**. 2004. Acesso em 19 jul. 2014. Disponível em: <http://www.gnuplot.info/>.

WIND, S. Open source cloud computing management platforms: Introduction, comparison, and recommendations for implementation. In: **Open Systems (ICOS), 2011 IEEE Conference on**. Langkawi, Malaysia: IEEE Xplore, 2011. p. 175–179.

APÊNDICE A – CÓDIGO FONTE DO SERVIÇO CRIADO

```
import tornado.web
import tornado.websocket
import tornado.httpserver
import tornado.ioloop
import Gnuplot
from math import sqrt
from commands import getoutput as gop
from os import sys

values = []
medias = []
proc_media = {}
global enviar

class WebSocketHandler(tornado.websocket.WebSocketHandler):
  def open(self):
    pass

  def on_message(self, value):
    enviar = True
    gnuplot = False
    if value == "enviar":
      enviar = False
    if enviar:
      try:
        values.append(value)
        if len(values) > 9:
          nomes_arquivos = gop('ls␣%s␣-B' %values[0])
          nomes_arquivos = nomes_arquivos.split('\n')
          if values[1] == "**":
            for nome in nomes_arquivos:
              arquivo = open('%s/%s' %(values[0], nome))
              qtde_processador = nome.split('_x')
              lines = arquivo.readlines()
```

```
                soma = 0
                valores = []
                for line in lines:
                  if '%s' %values[3] in line:
                    valor = line.split('%s' %values[4])
                            if (int(values[5])-1) <= len(valor):
                              numero = valor[int(values[5])-1]
                      if 's.' in numero:
                        numero = numero.replace('s.','')
                      if '.' in numero or ',' in numero:
                        if '%s' %values[2] == ',':
                          numero = numero.replace(',','.')
                        valores.append(float(numero))
                        soma+= float(numero)
                if soma != 0:
                  resultado = soma/len(valores)
                  variancia = 0
                  for v in valores:
                    variancia+= (v-resultado)**2
                  variancia/= (len(valores)-1)
                  dp = sqrt(variancia)
                  ic95 = resultado - 1.96 * (dp/sqrt(len(valores)))
                          medias.append('%2.2f' %resultado)
                          proc_media[qtde_processador[1]] = '%2.2f \
%2.2f %2.2f' %(resultado,dp,ic95)
                          self.write_message("A media de tempo " + \
str('%2.2f' %resultado))
                arquivo.close()
            else:
              for nome in nomes_arquivos:
                arquivos = values[1].split(";")
                for arq in arquivos:
                  if nome == arq:
                            arquivo = open('%s/%s' %(values[0], arq))
                            qtde_processador = nome.split('_x')
                            lines = arquivo.readlines()
                            soma = 0
                            valores = []
                            for line in lines:
                              if '%s' %values[3] in line:
                                valor = line.split('%s' %values[4])
                                numero = valor[int(values[5])-1]
                                if 's.' in numero:
```

```
                                numero = numero.replace('s.','')
                            if '.' in numero or ',' in numero:
                                        if '%s' %values[2] == ',':
                                    numero = numero.replace(',','.')
                                valores.append(float(numero))
                                soma+= float(numero)
                                if soma != 0:
                            resultado = soma/len(valores)
                            variancia = 0
                            for v in valores:
                              variancia+= (v-resultado)**2
                            variancia/= (len(valores)-1)
                            dp = sqrt(variancia)
                            ic95 = resultado - 1.96 * \
(dp/sqrt(len(valores)))
                            medias.append('%2.2f' %resultado)
                            proc_media[qtde_processador[1]] = \
'%2.2f %2.2f %2.2f' %(resultado,dp,ic95)
                            self.write_message("A media de tempo \
" + str('%2.2f' %resultado))
                          arquivo.close()
                    if len(medias) > 0:
                      try:
                        i = 1
                                linhas = 0
                                arq_txt = open('%s.txt' %values[6], 'w')
                                arq_txt.write("# VM's %s media dp ic95 \
\n" %values[3])
                        while (linhas < len(proc_media)):
                                  if proc_media.has_key('%s' %i):
                                    arq_txt.write("%s %s \n" \
%(i,proc_media['%s' %i]))
                                linhas+=1
                          i+=1
                        ultimo_processador = i
                                arq_txt.close()
                                saida = open('%s.gp' %values[6], 'w')
                                g = Gnuplot.Gnuplot(persist=1)
                                g("set term postscript enhanced color")
                                saida.write("set term postscript \
enhanced color \n")
                        g("set output '| ps2pdf - %s.pdf'" %values[6])
                                saida.write("set output '| ps2pdf - %s.pdf' \
```

```
\n" %values[6])
                g("set␣xlabel␣'%s'" %values[8])
                        saida.write("set␣xlabel␣'%s'␣\n" %values[8])
                        g("set␣ylabel␣'%s'" %values[9])
                        saida.write("set␣ylabel␣'%s'␣\n" %values[9])
                        g("set␣style␣line␣1␣default")
                        saida.write("set␣style␣line␣1␣default␣\n")
                        g("set␣style␣line␣2␣lw␣3")
                        saida.write("set␣style␣line␣2␣lw␣3␣\n")
                        g("set␣style␣line␣3␣default")
                        saida.write("set␣style␣line␣3␣default␣\n")
                        g("set␣style␣line␣4␣lw␣3")
                        saida.write("set␣style␣line␣4␣lw␣3␣\n")
                        g("set␣xrange[0:%i]" %ultimo_processador)
                        saida.write("set␣xrange[0:%i]␣\n" \
%ultimo_processador)
                        plot = "␣'%s.txt'␣using␣1:2␣title␣'%s␣\
time'␣with␣linespoints␣ls␣1" %(values[6],values[3])
                        saida.write("plot␣'%s.txt'␣using␣1:2␣title␣\
'%s␣time'␣with␣linespoints␣ls␣1" %(values[6],values[3]))
                use = '1'
                        estatisticas = values[7].split(',')
                        for estatistica in estatisticas:
                          if estatistica == 'media':
                            use+= ':2'
                          if estatistica == 'dp':
                            use+= ':3'
                            plot+=",'%s.txt'␣using␣%s␣notitle␣\
with␣errorbars␣ls␣2" %(values[6],use)
                            saida.write(",'%s.txt'␣using␣%s␣notitle␣\
with␣errorbars␣ls␣2" %(values[6],use))
                  if estatistica == 'ic95':
                            use+= ':4'
                            plot+=",'%s.txt'␣using␣%s␣notitle␣with␣\
errorbars␣ls␣4" %(values[6],use)
                        saida.write(",'%s.txt'␣using␣%s␣notitle␣\
with␣errorbars␣ls␣4" %(values[6],use))
                g("plot␣%s" %plot)
                self.write_message("Pdf␣gerado␣com␣sucesso")
                saida.close()
              except:
                        self.write_message("Erro␣ao␣gerar␣o␣Pdf")
            del values[0:len(values)]
```

```
            del medias[0:len(medias)]
          except IndexError:
            self.write_message("IndexError")
            del values[0:len(values)]
            del medias[0:len(medias)]
          except ValueError:
            self.write_message("ValueError")
            del values[0:len(values)]
            del medias[0:len(values)]
          except:
            self.write_message("Erro inesperado: ", \
sys.exc_info()[0])
            del values[0:len(values)]
            del medias[0:len(medias)]
        if value == "pdf":
          GerarPdf()
        if value == "gnu":
          GerarGnuplot()

  def on_close(self):
    pass

class WebSocketHandlerGrafico(tornado.websocket.WebSocketHandler):
  def open(self):
    pass

  def on_message(self, value):
    enviar = True
    if value == "enviar":
      enviar = False
    if enviar:
      try:
        values.append(value)
        if len(values) > 1:
          print values
          nome_arquivo = values[0].split(';')
          arquivo1 = open('%s.gp' %nome_arquivo[0])
          arquivo2 = open('%s.gp' %nome_arquivo[1])
          saida = open('%s.gp' %values[1], 'w')
          g = Gnuplot.Gnuplot(persist=1)
          arquivo1_lines = arquivo1.readlines()
          arquivo2_lines = arquivo2.readlines()
          plot = ""
```

```
            for line in arquivo1_lines:
              if "set output" in line:
                g("set output '| ps2pdf - %s.pdf'" %values[1])
                saida.write("set output '| ps2pdf - %s.pdf' \
\n" %values[1])
              elif "plot" in line:
                plot+='%s' %line
              else:
                g('%s' %line)
                saida.write('%s' %line)
            for line in arquivo2_lines:
              if "plot" in line:
                line = line.replace('plot', ',')
                line = line.replace('ls 1', 'ls 3')
                plot+='%s' %line
                g('%s' %plot)
                saida.write('%s' %plot)
            self.write_message("Pdf gerado com sucesso")
            saida.close()
            del values[0:len(values)]
        except:
          self.write_message("Campos incorretos")
          del values[0:len(values)]

    def on_close(self):
      pass

class IndexPageHandler(tornado.web.RequestHandler):
    def get(self):
      self.write("Interface Humano Computador para promover a \
usabilidade de um ambiente de Computacao em Nuvem no \
contexto de Experimentos Cientificos")

class GerarPdf(tornado.web.RequestHandler):
    def get(self):
      self.render("pdf.html")

class GerarGnuplot(tornado.web.RequestHandler):
    def get(self):
      self.render("gnu.html")

class SocketGnu(tornado.websocket.WebSocketHandler):
    def open(self):
```

```
    pass

  def on_message(self, value):
    arquivo = '%s.gp' %value
    buf_size = 4096
    self.set_header('Content-Type','application/octet-stream')
    self.set_header('Content-Disposition','attachment; \
filename='+arquivo)
    with open(arquivo, 'rb') as f:
      while True:
          data = f.read(buf_size)
          if not data:
              break
          self.write(data)
      self.finish()

  def on_close(self):
    pass

class Configuracao(tornado.web.RequestHandler):
  def get(self):
    self.render("teste.html")

class Graficos(tornado.web.RequestHandler):
  def get(self):
    self.render("graficos.html")

class Application(tornado.web.Application):
  def __init__(self):
    handlers = [
     (r'/', IndexPageHandler),
     (r'/config', Configuracao),
     (r'/websocket', WebSocketHandler),
     (r'/merge', Graficos),
     (r'/websocketgrafico', WebSocketHandlerGrafico),
     (r'/pdf', GerarPdf),
     (r'/socketgnu', SocketGnu),
     (r'/gnuplot', GerarGnuplot)
    ]
    settings = {
        'template_path': 'templates'
    }
    tornado.web.Application.__init__(self, handlers, **settings)
```

```
if __name__ == '__main__':
    ws_teste = Application()
    server = tornado.httpserver.HTTPServer(ws_teste)
    server.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

APÊNDICE B – EXEMPLO DE ARQUIVO DE SAÍDA UTILIZADO NO SISTEMA

```
Average computational time – Wall Clock: 2.82124s, CPU Ticks: 2.8125s.
Average communication time – Wall Clock: 0.203157s, CPU Ticks: 0.208s.
Average elapsed time – Wall Clock: 3.0245s, CPU Ticks: 3.021s.
Average computational time – Wall Clock: 2.82376s, CPU Ticks: 2.8065s.
Average communication time – Wall Clock: 0.185049s, CPU Ticks: 0.2015s.
Average elapsed time – Wall Clock: 3.00891s, CPU Ticks: 3.008s.
Average computational time – Wall Clock: 2.84092s, CPU Ticks: 2.834s.
Average communication time – Wall Clock: 0.198614s, CPU Ticks: 0.206s.
Average elapsed time – Wall Clock: 3.03964s, CPU Ticks: 3.04s.
Average computational time – Wall Clock: 2.80144s, CPU Ticks: 2.7915s.
Average communication time – Wall Clock: 0.196048s, CPU Ticks: 0.2035s.
Average elapsed time – Wall Clock: 2.99759s, CPU Ticks: 2.995s.
Average computational time – Wall Clock: 2.82171s, CPU Ticks: 2.8095s.
Average communication time – Wall Clock: 0.184931s, CPU Ticks: 0.1935s.
Average elapsed time – Wall Clock: 3.00674s, CPU Ticks: 3.003s.
Average computational time – Wall Clock: 2.86582s, CPU Ticks: 2.849s.
Average communication time – Wall Clock: 0.241892s, CPU Ticks: 0.2565s.
Average elapsed time – Wall Clock: 3.10781s, CPU Ticks: 3.1055s.
Average computational time – Wall Clock: 2.82475s, CPU Ticks: 2.8135s.
Average communication time – Wall Clock: 0.220675s, CPU Ticks: 0.231s.
Average elapsed time – Wall Clock: 3.04552s, CPU Ticks: 3.0445s.
Average computational time – Wall Clock: 2.87071s, CPU Ticks: 2.8605s.
Average communication time – Wall Clock: 0.229225s, CPU Ticks: 0.2375s.
Average elapsed time – Wall Clock: 3.10003s, CPU Ticks: 3.098s.
Average computational time – Wall Clock: 2.79745s, CPU Ticks: 2.783s.
Average communication time – Wall Clock: 0.182862s, CPU Ticks: 0.195s.
Average elapsed time – Wall Clock: 2.98041s, CPU Ticks: 2.9785s.
Average computational time – Wall Clock: 2.82643s, CPU Ticks: 2.8105s.
Average communication time – Wall Clock: 0.205438s, CPU Ticks: 0.218s.
Average elapsed time – Wall Clock: 3.03197s, CPU Ticks: 3.0285s.
Average computational time – Wall Clock: 2.82213s, CPU Ticks: 2.8125s.
Average communication time – Wall Clock: 0.201194s, CPU Ticks: 0.209s.
Average elapsed time – Wall Clock: 3.02343s, CPU Ticks: 3.0215s.
Average computational time – Wall Clock: 2.82954s, CPU Ticks: 2.824s.
```

Average communication time – Wall Clock: 0.168605s, CPU Ticks: 0.17s.
Average elapsed time – Wall Clock: 2.99824s, CPU Ticks: 2.994s.
Average computational time – Wall Clock: 2.82882s, CPU Ticks: 2.8215s.
Average communication time – Wall Clock: 0.190413s, CPU Ticks: 0.1975s.
Average elapsed time – Wall Clock: 3.01933s, CPU Ticks: 3.019s.
Average computational time – Wall Clock: 2.82706s, CPU Ticks: 2.8175s.
Average communication time – Wall Clock: 0.205407s, CPU Ticks: 0.212s.
Average elapsed time – Wall Clock: 3.03256s, CPU Ticks: 3.0295s.
Average computational time – Wall Clock: 2.82905s, CPU Ticks: 2.819s.
Average communication time – Wall Clock: 0.19392s, CPU Ticks: 0.203s.
Average elapsed time – Wall Clock: 3.02307s, CPU Ticks: 3.022s.
Average computational time – Wall Clock: 2.81418s, CPU Ticks: 2.794s.
Average communication time – Wall Clock: 0.202331s, CPU Ticks: 0.219s.
Average elapsed time – Wall Clock: 3.01661s, CPU Ticks: 3.013s.
Average computational time – Wall Clock: 2.82098s, CPU Ticks: 2.822s.
Average communication time – Wall Clock: 0.211627s, CPU Ticks: 0.212s.
Average elapsed time – Wall Clock: 3.03271s, CPU Ticks: 3.034s.
Average computational time – Wall Clock: 2.8257s, CPU Ticks: 2.813s.
Average communication time – Wall Clock: 0.16344s, CPU Ticks: 0.176s.
Average elapsed time – Wall Clock: 2.98924s, CPU Ticks: 2.989s.
Average computational time – Wall Clock: 2.82763s, CPU Ticks: 2.8275s.
Average communication time – Wall Clock: 0.188879s, CPU Ticks: 0.189s.
Average elapsed time – Wall Clock: 3.01661s, CPU Ticks: 3.0165s.
Average computational time – Wall Clock: 2.82166s, CPU Ticks: 2.8065s.
Average communication time – Wall Clock: 0.192296s, CPU Ticks: 0.206s.
Average elapsed time – Wall Clock: 3.01405s, CPU Ticks: 3.0125s.
Average computational time – Wall Clock: 2.82657s, CPU Ticks: 2.82s.
Average communication time – Wall Clock: 0.194999s, CPU Ticks: 0.2s.
Average elapsed time – Wall Clock: 3.02166s, CPU Ticks: 3.02s.
Average computational time – Wall Clock: 2.81689s, CPU Ticks: 2.8015s.
Average communication time – Wall Clock: 0.19532s, CPU Ticks: 0.21s.
Average elapsed time – Wall Clock: 3.01231s, CPU Ticks: 3.0115s.
Average computational time – Wall Clock: 2.81545s, CPU Ticks: 2.8125s.
Average communication time – Wall Clock: 0.238692s, CPU Ticks: 0.2405s.
Average elapsed time – Wall Clock: 3.05424s, CPU Ticks: 3.0535s.
Average computational time – Wall Clock: 2.82681s, CPU Ticks: 2.82s.
Average communication time – Wall Clock: 0.191263s, CPU Ticks: 0.1985s.
Average elapsed time – Wall Clock: 3.01818s, CPU Ticks: 3.0185s.
Average computational time – Wall Clock: 2.7983s, CPU Ticks: 2.7855s.
Average communication time – Wall Clock: 0.171039s, CPU Ticks: 0.1815s.
Average elapsed time – Wall Clock: 2.96943s, CPU Ticks: 2.967s.
Average computational time – Wall Clock: 2.81594s, CPU Ticks: 2.81s.
Average communication time – Wall Clock: 0.188626s, CPU Ticks: 0.1935s.

```
Average elapsed time – Wall Clock: 3.00466s, CPU Ticks: 3.004s.
Average computational time – Wall Clock: 2.8152s, CPU Ticks: 2.7965s.
Average communication time – Wall Clock: 0.200596s, CPU Ticks: 0.218s.
Average elapsed time – Wall Clock: 3.0159s, CPU Ticks: 3.0145s.
Average computational time – Wall Clock: 2.82572s, CPU Ticks: 2.8115s.
Average communication time – Wall Clock: 0.230946s, CPU Ticks: 0.245s.
Average elapsed time – Wall Clock: 3.05677s, CPU Ticks: 3.0565s.
Average computational time – Wall Clock: 2.82357s, CPU Ticks: 2.8165s.
Average communication time – Wall Clock: 0.206952s, CPU Ticks: 0.211s.
Average elapsed time – Wall Clock: 3.03062s, CPU Ticks: 3.0275s.
Average computational time – Wall Clock: 2.83026s, CPU Ticks: 2.824s.
Average communication time – Wall Clock: 0.173525s, CPU Ticks: 0.1775s.
Average elapsed time – Wall Clock: 3.00388s, CPU Ticks: 3.0015s.
```

APÊNDICE C – EXEMPLO DE ARQUIVO GNUPLOT GERADO PELO SISTEMA

```
set term postscript enhanced color
set output '| ps2pdf - ParallelSorting_333333333_DPIC.pdf'
set xlabel 'Processadores'
set ylabel 'Segundos'
set style line 1 default
set style line 2 lw 3
set style line 3 default
set style line 4 lw 3
set xrange[0:31]
plot 'ParallelSorting_333333333_DPIC.txt' using 1:2 title 'computational time'
with linespoints ls 1,'ParallelSorting_333333333_DPIC.txt' using 1:2:3 notitle
with errorbars ls 2,'ParallelSorting_333333333_DPIC.txt' using 1:2:3:4 notitle
with errorbars ls 4
```

APÊNDICE D – EXEMPLO DE ARQUIVO TEXTO GERADO PELO SISTEMA

# VM's	computational	media	dp	ic95
1	6.72	0.02	6.71	
2	2.81	0.02	2.81	
3	1.79	0.01	1.79	
4	1.27	0.01	1.27	
5	1.04	0.01	1.04	
6	0.84	0.00	0.84	
7	0.70	0.00	0.70	
8	0.60	0.00	0.60	
9	0.53	0.00	0.53	
10	0.47	0.00	0.47	
11	0.42	0.00	0.42	
12	0.38	0.00	0.38	
13	0.36	0.00	0.36	
14	0.33	0.00	0.33	
15	0.31	0.00	0.31	
16	0.29	0.00	0.29	
17	0.30	0.00	0.30	
18	0.30	0.00	0.30	
19	0.30	0.00	0.30	
20	0.30	0.00	0.30	
21	0.32	0.00	0.32	
22	0.31	0.00	0.31	
23	0.31	0.00	0.31	
24	0.30	0.00	0.30	
25	0.31	0.00	0.31	
26	0.32	0.00	0.32	
27	0.32	0.00	0.32	
28	0.32	0.00	0.32	
29	0.33	0.00	0.33	
30	0.33	0.00	0.33	