UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

RAMON VENSON

CLUSTERS COMPUTACIONAIS NO PROCESSAMENTO DE TAREFAS DE APLICAÇÕES WEB

CRICIÚMA

2014

RAMON VENSON

CLUSTERS COMPUTACIONAIS NO PROCESSAMENTO DE TAREFAS DE APLICAÇÕES WEB

Trabalho de Conclusão de Curso, apresentado para obtenção de grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. MSc. Paulo João Martins

CRICIÚMA

2014

RAMON VENSON

CLUSTERS COMPUTACIONAIS NO PROCESSAMENTO DE TAREFAS DE APLICAÇÕES WEB

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Computação Distribuída.

Criciúma, 30 de junho de 2014.

BANCA EXAMINADORA

Prof. Paulo João Martins – Mestre – (UNESC) – Orientador

Prof. Rogério Antonio Casagrande – Mestre – (UNESC)

Prof. Fabrício Giordani – Especialista – (UNESC)

Dedico este trabalho às pessoas que me ensinaram os valores que pretendo levar para a vida toda.

AGRADECIMENTOS

Não há como olhar pra trás sem agradecer aos meus pais pela dedicação, carinho e educação que me deram, sempre me motivando a buscar o conhecimento e a preservar a humildade e honestidade. Sem eles, nada disso seria possível.

Agradeço também às minhas irmãs, Angela e Milene, que além de me aturararem durante tantos anos, me inspiram e me enchem de orgulho pelas professoras dedicadas e competentes que são.

Pelos momentos de descontração e inspiração, agradeço aos meus amigos. Até mesmo aqueles cujo a distância não impediu de contribuírem com palavras de força e camaradagem. Seria injusto citar somente alguns nomes, por isso, obrigado a todos vocês.

Não posso esquecer também de agradecer ao grupo que tem sido, até então, minha segunda família. Aos momentos “performáticos”, meu obrigado a todas as amizades do Grupo Folclórico Valsugana.

Agradeço ao meu orientador Paulo Martins, que apoiou minhas ideias e foi essencial para o crescimento das mesmas ao longo do trabalho. Além disso, meu agradecimento à professora Priscyla Simões, pelas dicas valiosas e pelo tempo que me concedeu para realização desta pesquisa.

E para que eu seja isento de qualquer injustiça que venha a cometer por deixar de citar pessoas importantes, deixo aqui o meu agradecimento geral a todos que contribuíram, ainda que de forma singela, para a realização deste trabalho. Obrigado, mesmo.

“There is no death. There is the Force” –

The Jedi Code, Star Wars

RESUMO

Um dos principais desafios enfrentados no desenvolvimento de aplicações com alta demanda de recursos computacionais pode ser resolvido por meio da utilização de sistemas computacionais distribuídos. Nestes sistemas, é possível dividir o esforço computacional entre diversas máquinas interconectadas e, dessa forma, realizar o processamento de algoritmos que não teriam viabilidade quando executados em máquinas individuais. Seguindo este mesmo paradigma, tem-se atualmente no desenvolvimento de aplicações para o ambiente web os mesmos desafios, visto que, além da complexidade das tarefas executadas por uma aplicação, o acesso à mesma costuma a ser realizado por diversos usuários simultaneamente, gerando uma carga de trabalho ainda maior para um único servidor. Sendo assim, esta pesquisa se dispôs a analisar dois modelos distintos de implementação de um cluster para o processamento de um aplicativo web: um deles utilizando um modelo tradicional, através da simples clusterização de um servidor de aplicação Glassfish; e o outro utilizando dos recursos de um framework para a construção de aplicações distribuídas denominado Java Parallel Processing Framework (JPPF). Ambos os modelos foram analisados com base em métricas de performance e fatores gerenciais e experimentados em diversas configurações distintas, levando-se em consideração o número de usuários, número de nós no cluster e carga de trabalho executada. Com base nos resultados atingidos, observou-se as principais vantagens e desvantagens dos modelos propostos, constatando-se a aplicabilidade dos mesmos e conferindo a importância da utilização de métodos de implementação paralela e distribuída no desenvolvimento de aplicações web.

Palavras-chave: Sistemas Distribuídos. Clusters. Aplicações Web. JPPF. Avaliação de Performance.

ABSTRACT

One of the main challenges faced in application development with high demand of computational resources can be solved through the use of distributed computing systems. In these systems, it is possible to divide the computational effort by several interconnected machines and thereby perform the processing of algorithms that would not viable when played on individual machines. Following that same paradigm, currently in application development for the web environment we have the same challenges, since, besides the complexity of the tasks performed by an application, access to it usually being done by many users simultaneously, generating a workload yet larger for a single server. Therefore, this research set out to examine two distinct models of implementing a cluster for processing a web application: one using a traditional model, through simple clustering of a Glassfish application server; and the other using the resources of a framework for building distributed applications called Java Parallel Processing Framework (JPPF). Both models were analyzed based on performance metrics and management factors and tested in several different configurations, taking into account the number of users, number of nodes in the cluster and workload performed. Based on the achieved results, we observed the main advantages and disadvantages of the proposed models, evidencing the applicability of these and giving the importance of the use of parallel and distributed methods in developing web applications.

Keywords: Distributed Systems. Clusters. Web Applications. JPPF. Performance Evalution.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CDDL	Common Development and Distribution License
E/S	Entrada e Saída
GPL	GNUGeneral Public License
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JEE	Java Enterprise Edition
JPPF	Java Parallel Processing Framework
JSP	JavaServer Pages
LAN	Local Area Network
MIMD	Multiple-Instruction stream-Mingle-Data stream
MISD	Multiple-Instruction stream-Single-Data stream
RM-ODP	Reference Model of Open Distributed Processing
SIMD	Single-Instruction stream-Multiple-Data stream
SISD	Single-Instruction stream-Single-Data stream
SLA	Service Level Agreement
TCP	Transmission Control Protocol
UCP	Unidade Central de Processamento
UDP	User Datagram Protocol
WEB	World Wide Web
XHTML	eXtensible HyperText Markup Language

SUMÁRIO

1 INTRODUÇÃO

Devido às limitações físicas atingidas no desenvolvimento da capacidade de processamento e ao aumento da complexidade dos algoritmos, o processamento distribuído representa uma resposta para a continuidade no avanço da capacidade de processamento com eficiência e confiabilidade, possibilitando a fragmentação do processamento e do acesso a serviços entre diversos computadores dispersos geograficamente (DANTAS, 2005).

Dentre os vários modelos desenvolvidos dentro da computação distribuída, os chamados aglomerados computacionais representam um agrupamento de computadores ligados por meio de uma rede. Estes ambientes, também conhecidos como clusters ou aglomerados computacionais, podem ultrapassar em muito o poder de máquinas isoladas, sendo possível configurá-lo de forma a permitir uma escalabilidade incremental. Dessa forma, o usuário pode começar com um ambiente modesto e, conforme a necessidade, expandi-lo apenas adicionando novos dispositivos computacionais ao cluster (STALLINGS, 2010).

Dentre várias ferramentas disponíveis para a criação de ambientes distribuídos, encontra-se Java Parallel Processing Framework (JPPF). Uma ferramenta desenvolvida na linguagem Java e de código fonte aberto que permite aos programadores desenvolver e gerenciar aplicações com abordagem distribuída, provendo uma interface de desenvolvimento de aplicações acompanhada de um ambiente de distribuição de carga (COHEN, 2013).

De acordo com o criador e mantenedor do projeto Laurent Cohen (2013), o framework oferece três diferentes tipos de componentes que se comunicam para formar um ambiente de cluster. O **cliente** é o responsável pelo ponto de entrada no ambiente, permitindo aos desenvolvedores submeter determinadas tarefas da sua aplicação ao servidor por intermédio de uma interface de programação disponibilizada pelo JPPF. O **servidor** atua como o ponto central da rede, administrando e repassando as tarefas processadas de volta ao cliente e as não processadas aos **nós**, que por sua vez processam essas tarefas e retornam o resultado ao servidor.

Esse tipo de construção desempenha o papel de auxiliar no processamento de aplicações de alta complexidade computacional, tornando viável a implementação de algoritmos de alto desempenho que não teriam uma boa performance em uma abordagem não-distribuída ou mesmo utilizando supercomputadores multiprocessados. Além disso, a distribuição de carga de processamento entre sistemas computacionais colabora para o reaproveitamento de recursos físicos considerados obsoletos ou de baixa capacidade, tornando possível a execução dessas aplicações em hardware de baixo custo (AYDIN; BAY, 2009, tradução nossa).

Tão relevante quanto o conceito de utilizar recursos computacionais interconectados para prover processamento é a ideia da World Wide Web (Web). Originalmente desenvolvida para prover páginas estáticas, o progresso da tecnologia permite hoje a construção de aplicações de alta complexidade que podem ser acessadas mundialmente através da Internet por meio de navegadores (FORD, 2004).

Entretanto, a execução de aplicações Web complexas, assim como aplicações *desktop*, podem gerar um alto custo de processamento, como é o caso de algoritmos de encriptação complexa, renderização de gráficos ou manipulação de imagens. Estas tarefas necessitam de um grande esforço computacional, e podem causar problemas para sistemas que utilizam um servidor para processamento da aplicação que não atenda os requisitos de processamento da mesma (SOUDERS, 2009).

Entre as formas de contornar este problema, pode-se apontar a clusterização de servidores de aplicação Web, que compartilham do objetivo de sustentar as aplicações, oferecendo soluções para balanceamento da carga de trabalho, alta disponibilidade e alta performance. Todavia, não se desconsidera completamente a utilização de plataformas como o JPPF, onde a distribuição do processamento pode dar-se apenas em tarefas específicas da aplicação. Além disso, essa plataforma fornece uma maior flexibilidade na configuração de hardware dos dispositivos computacionais integrantes do cluster, encorajando um modelo de baixo custo financeiro.

Sendo assim, a proposta deste trabalho é explorar e analisar a utilização de modelos de plataformas de clusters computacionais no processamento de aplicações web. Para isso, pretende-se realizar a execução de uma aplicação com tarefas de alta carga de processamento em um servidor de aplicação configurado em cluster, comparando os resultados obtidos com outro modelo, baseado na plataforma JPPF para a execução apenas das tarefas de maior complexidade.

1.1 OBJETIVO GERAL

Analisar e comparar a execução de uma aplicação web em um cluster de um servidor de aplicação com a execução por meio da plataforma JPPF.

1.2 OBJETIVOS ESPECÍFICOS

Como objetivos específicos desta pesquisa, pretende-se:

a) estudar e apresentar os conceitos de computação paralela e distribuída;
b) estudar e aplicar os conceitos da utilização da plataforma JPPF e suas características de programação;
c) estudar e apresentar as características da web e suas aplicações;
d) desenvolver uma aplicação web com tarefas de alta complexidade;
e) analisar o funcionamento da aplicação por meio de um servidor de aplicação clusterizado;
f) analisar o funcionamento da aplicação em um cluster baseado na plataforma JPPF;
g) realizar uma comparação entre os dois modelos analisados.

1.3 JUSTIFICATIVA

Os sistemas distribuídos consistem em um conjunto de computadores independentes que se apresentam ao usuário de maneira única e coerente. Essa definição não abrange o tipo de arquitetura dos computadores nem tampouco o meio pelo qual são interligados (TANEBAUM; STEEN; 2008).

Segundo Tanebaum e Steen (2008), os sistemas de computação distribuídos e aglomerados computacionais fazem parte dessa definição abrangente, sendo a última utilizada para executar um determinado programa em paralelo em diversas máquinas. Para Dantas (2005), um cluster computacional pode ser desenvolvido com máquinas de configurações distintas e relativamente pequenas, sendo a escalabilidade um fator de grande importância para que o desempenho geral do cluster cresça à medida que mais recursos se fazem necessários.

Uma das tecnologias distribuídas de maior sucesso no mundo é a Web, por onde circulam uma infinidade de conteúdos por meio de serviços interconectados. A web representa a plataforma de implementação mais comum disponível para desenvolvedores, atualmente: encontra-se em cada *smarthphone*, *tablet*, laptop, *desktop* ou outros dispositivos do meio. Projeções de crescimento revelam que, até 2020, cerca de 20 bilhões de dispositivos estejam conectados à Web através de navegadores, demonstrando a importância desse seguimento no desenvolvimento de novas aplicações (GRIGORIK, 2013).

Inicialmente, os sistemas projetados para a web eram escritos nas linguagens C e Perl e apresentavam basicamente páginas conectadas por hipertexto e estáticas. Com o tempo, novas tecnologias foram desenvolvidas com o intuito de gerar conteúdo dinâmico para a Web e, entre elas, destaca-se a linguagem java. Desenvolvida a princípio para a construção de aplicações tradicionais (paradigma *desktop*) e *applets*[1], a linguagem rapidamente demonstrou potencial para o desenvolvimento Web e distribuído, primeiro através dos *servlets*[2] e posteriormente com Java Server Pages[3] (JSF) (FORD, 2004).

Neste contexto, surgem aplicações de maior complexidade que despendem de maiores recursos computacionais, fazendo-se necessário a composição de novos modelos de implementação para garantir a escalabilidade e performance do sistema. Um dos meios utilizados para contornar problemas de escalabilidade é a clusterização de um servidor de aplicação. Alguns servidores oferecem ferramentas para facilitar implementação de um cluster com o objetivo de melhorar o desempenho e a

1 Software executado no contexto de outra aplicação.
2 Classe Java destinada a responder requisições em um servidor de aplicação.
3 Tecnologia na linguagem java para a criação de páginas web dinâmicas.

disponibilidade das aplicações alojadas. Entretanto, ao utilizar este recurso, o gerenciamento da distribuição de tarefas passa a ser realizado diretamente pelo servidor, sem a interferência do programador (ORACLE CORPORATION, 2002; RED HAT, 2014).

Permitindo também a construção de redes de processamento distribuído, o projeto Java Parallel Processing Framework utiliza a máquina virtual Java para oferecer uma interface de desenvolvimento distribuído voltada para a construção de aplicações que requerem uma grande quantidade de processamento. O JPPF permite dividir uma aplicação em pequenas partes que podem ser executadas simultaneamente em diferentes máquinas, por meio de chamadas de procedimento remoto. Uma das características de maior destaque nesta plataforma é sua facilidade de implementação e flexibilidade para criação de clusters com hardware de baixo custo e não-dedicado (COHEN, 2013).

Dado que uma aplicação Web construída na plataforma Java também pode utilizar dos recursos da plataforma JPPF, esta foi escolhida por ser uma implementação de código fonte aberto e que propicia facilidade de implementação de tarefas específicas da aplicação, em oposição à clusterização total do sistema por meio do servidor de aplicação. Dessa forma, propõe-se uma comparação entre os dois modelos de execução distribuída de uma aplicação Web de alta performance, com a finalidade de investigar e demonstrar as vantagens e desvantagens da utilização de ambos os modelos.

1.4 ESTRUTURA DO TRABALHO

Este trabalho foi dividido em 7 capítulos que apresentação a fundamentação teórica, o desenvolvimento e os resultados da pesquisa realizada.

O capítulo 1 apresenta uma breve introdução sobre a problemática discutida e as justificativas para a realização do projeto, além da definição do objetivo geral e específicos deste trabalho.

O capítulo 2 expõe a história, características, importância, e principais classificações de sistemas computacionais distribuídos.

No capítulo 3, apresenta-se o *framework* para desenvolvimento de aplicações distribuídas JPPF. Este capítulo provê uma visão geral de ferramenta, revelando suas principais características técnicas e seus modelos de implementação de clusters computacionais.

A história e importância de sistemas Web são discutidos no capítulo 4, que apresenta, ainda, características referentes a servidores de aplicação na plataforma Java.

O capítulo 5 apresenta trabalhos relacionados aos temas abordados neste trabalho: sistemas distribuídos, JPPF e aplicações Web.

A metodologia utilizada para a construção dos modelos de cluster, a aplicação web desenvolvida e o roteiro de testes utilizado é revelado por meio do capítulo 6. Ainda neste capítulo, são apresentados os dados e informações coletadas no desenvolvimento das experimentações realizadas.

Por fim, o capítulo 7 apresenta a conclusão desta produção, onde busca-se demonstrar as vantagens e desvantagens da utilização dos modelos computacionais implementados, além de fornecer novas propostas para novos trabalhos relacionados a esta pesquisa.

2 SISTEMAS DISTRIBUÍDOS

Um sistema distribuído pode ser caracterizado por um conjunto de computadores independentes e interconectados que se apresentam ao usuário como se fossem um único sistema. Um computador independente (ou completo) pode ser entendido como máquinas que possuem a capacidade de funcionarem independentemente umas das outras. Dessa forma, um sistema distribuído deve possibilitar o emprego de conceitos como escalabilidade e alta disponibilidade (STALLINGS, 2010; TANENBAUM; STEEN, 2007).

Essa definição não abrange o tipo de conexão e nem apresenta uma configuração específica dos computadores que podem compor um sistema distribuído, que pode ser adaptado à medida que seus componentes se desenvolvem.

2.1 HISTÓRIA

As primeiras ideias relacionadas ao aproveitamento de recursos computacionais subutilizados por meio de redes de computadores surgiram no início da década de setenta. Por meio destas redes, seus idealizadores pretendiam utilizar os ciclos ociosos das máquinas no processamento de aplicações que exigiam um alto poder computacional (DANTAS, 2005).

A partir dos anos 80, o desenvolvimento das redes de computadores de alta velocidade, que permitiu que um grande volume de dados fossem movimentadas entre centenas de máquinas, e os avanços tecnológicos no desenvolvimento dos microprocessadores, alavancaram a utilização de sistemas computacionais compostos por uma grande quantidade de computadores trabalhando em conjunto para resolver tarefas de alta complexidade. Estes sistemas, geralmente, exigem um tempo menor de processamento quando comparados com sistemas centralizados (TANENBAUM; STEEN, 2007).

Durante os anos 90, houve um intenso crescimento da utilização de recursos computacionais distribuídos geograficamente. Com a popularização da Internet e o crescimento da engenharia de software no desenvolvimento de aplicações

distribuídas, diversos projetos relacionados e de grande escala como o SETI@HOME4 e o distributed.net5, onde os recursos computacionais não utilizados dos usuários são compartilhados com projetos de pesquisa, além de sistemas governamentais e de bancos, passaram a virar realidade no Brasil e no mundo (DANTAS, 2005).

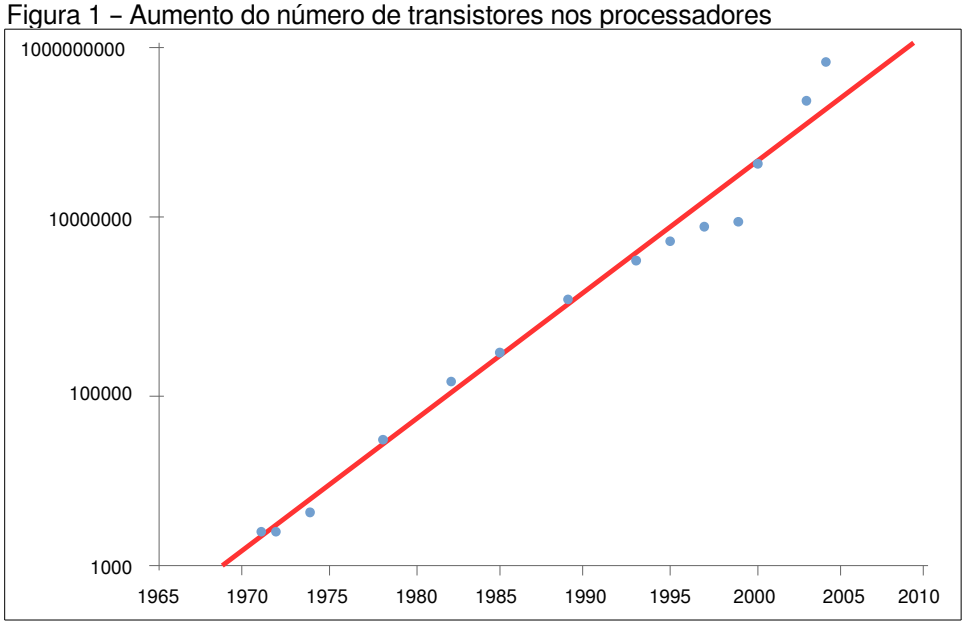
Todavia, no início da primeira década do século XXI houve uma queda acentuada no crescimento apresentado até então na oferta de processadores com clocks de maior frequência e no aumento do tamanho das memórias, impondo restrições ao desenvolvimento de softwares de maior complexidade computacional, colocando em risco a validade da conhecida Lei de Moore (DANTAS, 2005).

Segundo Moore (2006, tradução nossa), o aumento no número de componentes de um microprocessador tende a propiciar um aumento proporcional de desempenho. Dado a tendência de crescimento do número de componentes, é possível estimar, também, o ritmo de crescimento de processamento.

Em seu trabalho, Moore também considerou que a tendência do crescimento da capacidade de processamento seria dobrada a cada a cada 18 meses. Assim, a previsão realizada pelo engenheiro ficou conhecida como Lei de Moore, cuja a taxa de crescimento prevista sempre foi acompanhada e, em diversos casos, até superada. (DANTAS, 2005).

A figura 1 ilustra o crescimento do número de transistores dos processadores (no eixo vertical) pelo tempo (eixo horizontal), comparados à previsão realizada por Gordon Moore (linha vermelha):

Figura 1 – Aumento do número de transistores nos processadores



Fonte: ROBISON (2012)

Ainda que se observe alguns saltos entre os anos de 2000 a 2005, a queda no ritmo de crescimento dos transistores dos processadores durante a década de 90 pode ser explicada, dentre outros fatores, pelo consumo excessivo de energia. Para manter a tendência de desempenho à medida que o número de componentes por processador aumenta, é necessário recorrer à utilização de técnicas de projeto de maior complexidade e altas frequências de *clock*. O grande problema relacionado está no fato de que o consumo de energia cresce exponencialmente no ritmo do aumento da densidade e frequência de *clock* do processador. Estes argumentos contribuem para o aumento da importância de uma organização de processamento paralelo, no caso, *multicore*[4] e arquiteturas de sistemas distribuídos (STALLINGS, 2010).

2.2 TAXONOMIA DE FLYNN

A classificação de arquitetura de hardware de maior aceitação na área de arquitetura de computadores é conhecida como Taxonomia de Flynn. Ela leva em consideração o número de instruções executadas em relação ao conjunto de dados

4 Múltiplos núcleos de processamento.

para as quais as instruções são submetidas. Todo sistema computacional pode então, do ponto de vista do hardware, ser classificado da seguinte maneira (DANTAS, 2005):

a) single-Instruction stream-Single-Data stream (SISD): Computadores que executam uma instrução de um programa por vez. Modelo Tradicional de processador único. Segundo FLYNN (1972, tradução nossa), esta categoria representa a maioria dos computadores convencionais;
b) single-Instruction stream-Multiple-Data stream (SIMD): Computadores que executam uma única instrução sobre diversos itens de dados, aumentando o desempenho da configuração por conta da computação concorrente. Segundo FLYNN (1972, tradução nossa), inclui a maioria dos processadores baseados em *array*;
c) multiple-Instruction stream-Single-Data stream (MISD): Não se tem conhecimento de nenhum tipo de arquitetura de máquinas trabalhando com essa classificação. FLYNN (1972, tradução nossa) define esta categoria como arquiteturas que utilizam múltiplas instruções em apenas um item de dados;
d) multiple-Instruction stream-Mingle-Data stream (MIMD): Categoria ao qual se encontram os multiprocessadores, que trabalham com múltiplas instruções independentes sobre múltiplos itens de dados.

Este último (MIMD), incorpora as definições propostas pelos sistemas paralelos de computação. Estes sistemas podem executar várias instruções em paralelo por meio da utilização de vários núcleos de processamento. Dessa forma, os sistemas distribuídos também encontram-se nesta classificação.

2.3 IMPORTÂNCIA

Dentre as principais vantagens na utilização de sistemas distribuídos encontra-se a redução de custos. Ao implementar um sistema distribuído por meio de várias máquinas de baixo custo, é possível construir um sistema de grande poder computacional sem a utilização de equipamentos complexos e de preço elevado (DEITEL; DEITEL; CHOFFNES, 2005).

Outra característica de grande importância proporcionada por esse tipo de sistema é a alta disponibilidade. Uma arquitetura distribuída implementa a possibilidade de manter-se funcionando mesmo com a queda de boa parte dos seus recursos computacionais. Assim, com o devido tratamento de software, o sistema pode recuperar sua execução e continuar trabalhando normalmente, ignorando o hardware com problema. Essa característica também é conhecida como tolerância a falhas (STALLINGS, 2010).

Além do baixo custo e disponibilidade, é comum que um sistema distribuído tenha como finalidade o aumento de performance para a realização de uma determinada tarefa. Por meio da utilização destas estruturas computacionais é possível melhorar o desempenho de tarefas, sobrepondo-se às dificuldades apresentadas no desenvolvimento de hardwares de maior eficiência (DANTAS, 2005).

Entretanto, o fato de ser possível montar sistemas distribuídos não garante, necessariamente, que sua implementação é viável ou efetiva em todos os casos. Para garantir o funcionamento correto e a qualidade em relação a sistemas não distribuídos é preciso que algumas características importantes sejam apropriadamente implementadas, como a garantia que o sistema pode ser expandido e que seus recursos sejam acessados de forma razoavelmente fácil e transparente (TANENBAUM; STEEN, 2007).

2.4 CARACTERÍSTICAS

Existem diversas características que podem ser avaliadas em um ambiente distribuído. Propriedades relacionadas à segurança podem colaborar para que ambientes que exijam um alto grau de confiabilidade nos dados processados, enquanto a escalabilidade reduz a perda de desempenho decorrente da saturação do poder computacional de um sistema.

2.4.1 Segurança

Um problema de segurança refere-se a uma possível intrusão de um agente externo a um sistema distribuído. Um agente intruso pode interceptar as comunicações e até mesmo alterar as mensagens trocadas. Estas ações podem provocar consequências sérias a um sistema como, por exemplo, a alteração de uma transação bancária ou a alteração no resultado de um cálculo crítico (HADDAD et al, 2011, tradução nossa).

A complexidade da segurança pode ser analisada a partir de diferentes pontos do sistema. Os nós de processamento, transmissão e clientes são pontos clássicos que requerem atenção e proteção. O crescimento da heterogeneidade em diferentes camadas da infraestrutura do sistema também reflete no aumento da dificuldade em manter o sistema relativamente seguro (BELAPURKAR et al, 2009, tradução nossa).

2.4.2 Transparência

A capacidade de ocultar a localização física dos processos e recursos é uma característica importante de um sistema distribuído. Um sistema que é capaz de apresentar-se aos usuários (sejam eles pessoas ou aplicações) como se fosse um único sistema de computador é denominado transparente (TANENBAUM; STEEN, 2007).

O Reference Model of Open Distributed Processing (RM-ODP) especifica e descreve oito tipos de transparências de um sistema distribuído (ISO/IEC, 1996):

a) transparência de acesso: oculta as diferenças entre as representações de dados e invocação de mecanismos para acesso de um recurso;
b) transparência de falha: oculta a falha e a possível recuperação de um recurso;
c) transparência de localização: oculta a localização geográfica de um objeto, permitindo o acesso sem a informação da localidade;

d) transparência de migração: oculta a habilidade do sistema de modificar a localização de um recurso;
e) transparência de persistência: oculta a ativação e reativação de um objeto;
f) transparência de realocação: oculta a realocação dos recursos enquanto estão sendo utilizados;
g) transparência de replicação: oculta um recurso replicado, bem como suas restrições de disponibilidade e desempenho;
h) transparência de transação: oculta os processos de consistência de um objeto.

Existem casos, entretanto, onde o grau de transparência deve ser limitado para que o usuário não se sinta confuso em relação ao sistema (TANENBAUM; STEEN, 2007).

2.4.3 Alto Desempenho

Dentro do contexto atual, onde as dificuldades de ordem física desaceleram o ritmo de crescimento do desempenho de hardwares como memórias e processadores, o processamento de alto desempenho por meio de sistemas distribuídos representa uma resposta diferencial por meio da utilização de agregados de recursos computacionais dispersos localmente. Um sistema que tem como objetivo a redução do tempo computacional total de uma determinada tarefa pode ser identificado como um sistema de alto desempenho (DANTAS, 2005).

Da perspectiva de hardware, os fatores que mais interessam no desempenho são as velocidades da Unidade Central de Processamento (UCP), Entrada e Saída (E/S), e o desempenho da rede de interconexão. Dado que as velocidades de CPU e E/S são, a princípio, as mesmas de um computador com um único processador, os parâmetros fundamentais de interesse de um sistema paralelo são a latência e a largura de banda (TANENBAUM, 2007).

Normalmente, poucos programas conseguem um aproveitamento total desses recursos durante a execução paralela, dado que quase todos os programas

têm algum componente sequencial, geralmente a fase de inicialização, leitura de dados e coleta de resultados. Mas ainda assim, existem muitas aplicações em que, para que a velocidade fosse dobrada, houvesse a necessidade de aumentar o número de computadores do sistema distribuído em quatro vezes, ainda se trataria de uma solução relativamente barata para muitas aplicações e empresas (TANENBAUM, 2007).

2.4.4 Escalabilidade

Um grave problema apresentado por sistemas está na comunicação. À medida que o número de usuários e aplicações crescem, a alta taxa de comunicação com o sistema tende a transformá-lo em um gargalo no desempenho, ainda que seus recursos de processamento e armazenagem sejam ilimitados (TANENBAUM; STEEN, 2007).

O modo mais direto de aprimorar o desempenho de um sistema é por meio da adição de mais unidades de processamento. Um sistema que permite adicionar mais unidades evitando a criação de gargalos e aumentando a capacidade de computação é denominada escalável (TANENBAUM, 2007).

Além de controlar a perda de desempenho por conta da saturação dos canais de comunicação, o sistema escalável também deve impedir que os recursos de software se esgotem e possibilitar o controle de custo dos recursos, evitando o gasto financeiro demasiado para a ampliação do poder computacional (COULOURIS; DOLLIMORE; KINDBERG, 2007).

2.5 COMUNICAÇÃO

Um dos principais fatores de sucesso na implementação de um ambiente de computação distribuída é a escolha das tecnologias para a construção da infraestrutura de rede. Dentre as características desejadas, encontra-se a disponibilidade de largura de banda, taxas de transmissão compatíveis com o meio, possibilidade de anexar novas estruturas (escalabilidade), um retardo de comunicação

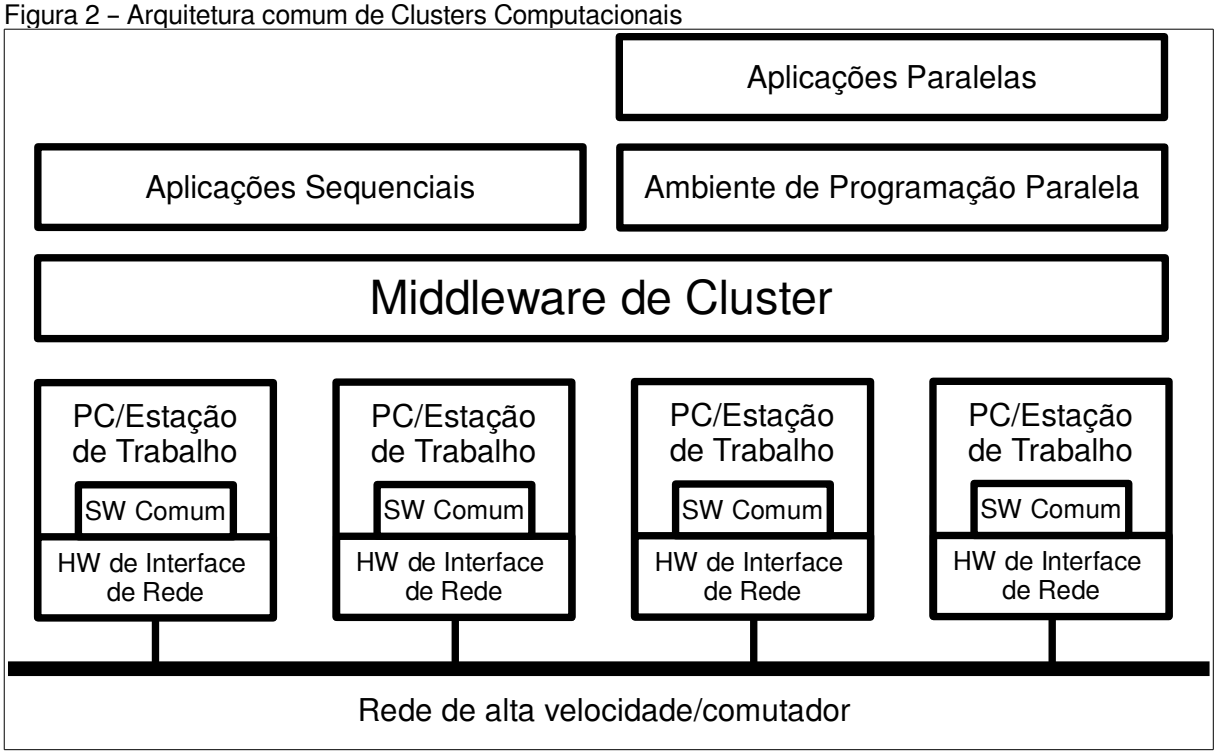
com o mínimo de influência possível e a utilização de protocolos de comunicação adequados. Dessa forma, é possível construir soluções distribuídas de software sem que o projeto de infraestrutura torne-se um limitador no aperfeiçoamento do desempenho (DANTAS, 2005).

Em programas que movimentam uma grande quantidade de dados (algo comum na área de ciências naturais, por exemplo), a largura de banda tende a ser um fator crítico para o desempenho do sistema. Dessa forma, para conseguir baixar a latência, pode-se, por exemplo, utilizar pacotes menores, visto que pacotes grandes bloqueiam a banda por um tempo maior (TANENBAUM, 2007).

2.6 MIDLEWARE PARA SISTEMAS DISTRIBUÍDOS

Uma arquitetura comum de um aglomerado computacional pode ser visualizada na figura 2, que contém os principais componentes deste tipo de estrutura:

Figura 2 – Arquitetura comum de Clusters Computacionais



Fonte: Adaptado de BUYYA (1999)

Os computadores individuais são interconectados por meio de uma Local Area Network (LAN), também chamada de rede local, de alta velocidade e possuem uma camada intermediária que possibilita a operação do cluster denominada *middleware*. Ele é o responsável por fornecer ao usuário a sensação de operar apenas uma única máquina, conhecida como imagem de sistema único. Além disso, esta aplicação ainda é responsável pelo balanceamento de carga e resposta das falhas de componentes, garantindo a alta disponibilidade do sistema (STALLINGS, 2010).

Hwang et al (1999, tradução nossa) lista os serviços e funções desejáveis em um *middleware* de cluster:

a) ponto de entrada único: o sistema deve ser acessado pelo usuário por meio de um único ponto do cluster em vez de cada computador individual;
b) hierarquia única de arquivos: o usuário vê uma hierarquia de diretórios unificada, abaixo do mesmo diretório raiz;
c) ponto de controle único: o gerenciamento e controle deve ser permitido por meio de uma única interface (ou estação de trabalho);
d) rede virtual única: qualquer nó pode acessar qualquer outro ponto, mesmo que estejam separados entre múltiplas redes;
e) espaço único de memória: o compartilhamento de memória distribuída deve ser permitido para que os programas compartilhem variáveis;
f) sistema único de gerenciamento de trabalhos: as tarefas devem ser submetidas ao cluster sem necessidade de especificar qual dos nós realizará a execução do mesmo;
g) interface de usuário única: uma interface gráfica comum deve ser suportada para todos os usuários, independente da estação de trabalho da qual é acessada;
h) espaço de entrada e saída único: qualquer nó pode acessar qualquer periférico remotamente sem conhecer sua localização física;
i) espaço único de processos: um processo em um nó pode criar ou se comunicar com outro processo em um nó remoto;

j) pontos de verificação: os estados dos processos e resultados computacionais intermediários devem ser salvos periodicamente para permitir a recuperação em caso de falhas;

k) migração de processos: permite o balanceamento de carga entre os nós do cluster.

Para Stallings (2010), os quatro últimos itens dessa lista focam em aprimorar a disponibilidade do cluster, enquanto o restante se preocupa com a visualização de imagem única do sistema.

2.7 TIPOS DE SISTEMAS DISTRIBUÍDOS

Dentre os modelos computacionais para computação distribuída, pode-se citar dois de maior abrangência: os clusters computacionais, que possuem um objetivo específico e cuja a administração dos nós geralmente é feita por uma única organização; e os grids, também conhecidos como malhas computacionais, que implementam diversos serviços interconectados, possivelmente entre diversas organizações.

2.7.1 Clusters

Um cluster pode ser definido como um conjunto de computadores completos que estejam interconectados e compartilhando trabalho, como um recurso computacional unificado, criando-se a possível ilusão de que existe apenas uma única máquina. O termo computador completo significa que cada um dos nós (como normalmente são chamados os computadores de um cluster) pode funcionar independente dos outros (STALLINGS, 2010).

Ele pode ser dividido em duas espécies distintas: o centralizado e o descentralizado (TANENBAUM, 2007):

a) centralizado: consiste em computadores montados em um único local. As máquinas do cluster são geralmente homogêneas e compactados

com a finalidade de reduzir o comprimento de conexões e otimizar o espaço físico ocupado;

b) descentralizado: é caracterizado por máquinas heterogêneas conectadas por meio de uma rede local, espalhados por um edifício ou campus, e possuem um conjunto de periféricos completo, como mouse, teclado e monitor. Grande parte desses computadores possuem usuários, mas ficam ociosos durante muitas horas do dia.

Quanto aos objetivos que podem ser explorados e alcançados por um cluster computacional (STALLINGS, 2010):

a) escalabilidade absoluta: pode ter até milhares de máquinas, cada uma sendo um multiprocessador. Dessa forma, é possível criar clusters que ultrapassam facilmente o poder computacional de máquinas de grande porte que trabalham sozinhas;
b) escalabilidade incremental: é possível adicionar novos computadores a um ambiente sem necessidade de grandes atualizações. O usuário pode iniciar com um ambiente modesto e expandi-lo conforme necessidade;
c) alta disponibilidade: a falha de um nó não tende a causar perda total do serviço, já que cada máquina pode trabalhar de maneira independente. A tolerância a falhas é geralmente tratada automaticamente em software;
d) preço/desempenho superior: o preço de um cluster computacional é normalmente menor que uma máquina de grande porte de mesmo desempenho.

2.7.2 Grids

Um grid computacional pode ser entendido como uma plataforma de computadores geograficamente dispersos que compartilham uma grande quantidade de serviços e recursos, gerenciados por diferentes organizações, e que podem ser acessados através de um meio transparente e único (DANTAS, 2005).

Pode-se dizer que a propriedade chave para um sistema computacional em grid é principalmente a coordenação da computação entre diferentes domínios. Dessa

forma, ela tende a se distinguir de sistemas convencionais de alta performance, como os clusters, no sentido de que tendem a ser mais flexíveis, heterogêneos e com um alto grau de dispersão geográfica (SERGE, 2011).

Ambos os sistemas utilizam-se de *middlewares* para a integração entre redes e computadores. Dentre eles um, que é capaz de prover uma plataforma fácil gerenciamento tanto para sistemas de grid quando de clusters é o JPPF, descrito no próximo capítulo.

3 JAVA PARALEL PROCESSING FRAMEWORK

Java Parallel Processing Framework[5] é um *framework* de código fonte aberto desenvolvido na linguagem java, possibilitando sua execução a partir de qualquer sistema operacional capaz de executar uma máquina virtual Java. Ele pode ser utilizado para construir sistemas computacionais distribuídos de maneira simples, permitindo a criação de aplicações com alto poder de processamento, que rodam em diversos computadores, reduzindo drasticamente o tempo de processamento (XIONG; WANG; XU, 2010, tradução nossa).

3.1 VISÃO GERAL

Um ambiente distribuído pode ser construído por meio do JPPF utilizando-se de três diferentes tipos de componentes que se comunicam entre si (COHEN, 2013, tradução nossa):

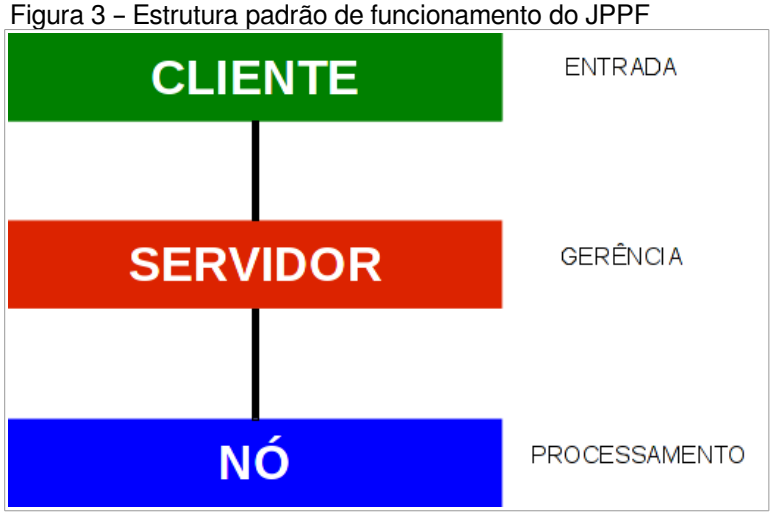
a) cliente: é o ponto de entrada do cluster/grid. É por meio do cliente do JPPF que se faz possível ao desenvolvedor construir sua própria aplicação de forma a submeter tarefas a serem executadas no ambiente. Toda comunicação do cliente desenvolvido com o restante dos componentes é feita por meio de uma Application Programming Interface (API), parte integrante do *framework* JPPF;
b) servidor: o servidor (também chamado de *driver*) é o componente que recebe e gerencia as tarefas submetidas pelos clientes, repassando aos nós do sistema para que sejam processados. Após o processamento, o servidor recebe a resposta dos nós e as repassa ao cliente;
c) nós: realizam o processamento efetivo das tarefas programadas pelo desenvolvedor.

A figura 3 mostra como é organizada a estrutura mais comum de funcionamento de um cluster JPPF:

---

5 Site oficial do projeto pode ser acessado em http://jppf.org – Nele é possível acessar a documentação técnica e funcional do JPPF, além do download de versões anteriores. Todo desenvolvimento é mantido por meio de uma página no site Source Forge (http://sourceforge.net/projects/jppf-project/)

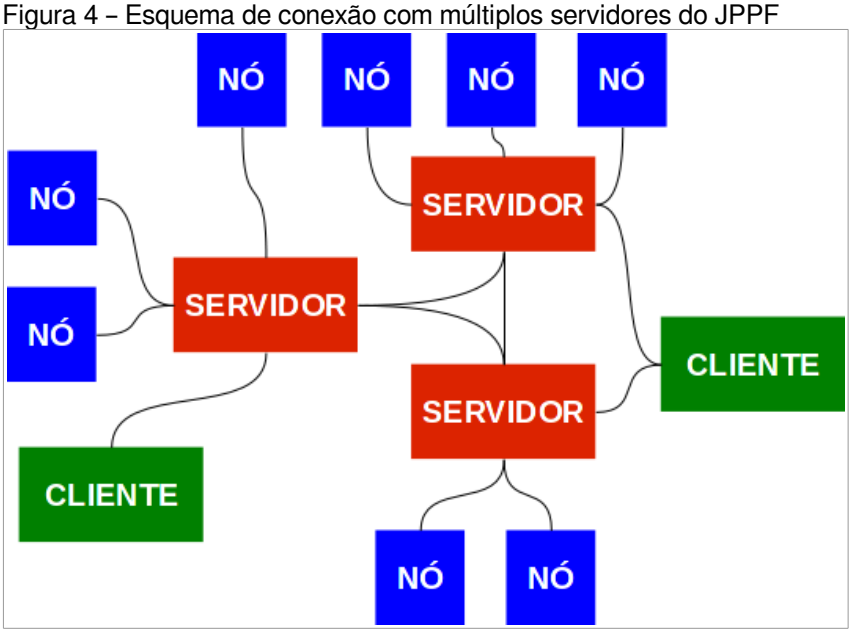
Figura 3 – Estrutura padrão de funcionamento do JPPF



Fonte: Adaptado de JPPF (2013).

Visualiza-se que o servidor realiza um papel central na arquitetura do ambiente, gerenciando todas as requisições e respostas do cluster. Isso também significa que o servidor se torna um ponto único de falha, ou seja, a falha deste componente provoca a paralisação de todo o ambiente. Para minimizar este risco, o JPPF disponibiliza a habilidade de conectar múltiplos servidores, adicionando opções de conectividade para clientes e nós (figura 4) (COHEN, 2013, tradução nossa).

Figura 4 – Esquema de conexão com múltiplos servidores do JPPF



Fonte: Adaptado de JPPF (2013).

A conexão entre dois servidores é direcional: se um servidor A conecta-se ao servidor B, A verá B como um cliente e B verá A como um nó. Dessa forma, o servidor pode enviar tarefas ao servidor B, que por sua vez distribui para outros nós. Essa relação se torna bidirecional se realizarmos a conexão do servidor B com o A. Assim, podemos dizer que o ponto único de falha pode ser definido como a combinação entre a redundância e a configuração dinâmica da topologia utilizada (COHEN, 2013, tradução nossa).

3.2 DISTRIBUIÇÃO DE CARGA

O JPPF trabalha com duas unidades de trabalho como forma de facilitar o desenvolvimento e organizar a carga de trabalho inserida nos clientes.

A primeira e menor unidade que pode ser manipulada por um ambiente JPPF é a unidade *task* (tarefa). Cada tarefa carrega os processos que o desenvolvedor deseja que sejam processados pelos nós, podendo ser, por exemplo, um algoritmo de multiplicação de matrizes (COHEN, 2013, tradução nossa).

Em seguida, a unidade de *job* tem como função agrupar unidades de tarefa de um mesmo contexto para auxiliar o desenvolvedor na distribuição dos processos em um grid ou cluster JPPF. Cada trabalho pode definir um Service Level Agreement (SLA) exclusivo, que especifica diversas características comportamentais, como o número de nós que podem processar determinada unidade de tarefa ou a prioridade de processamento da mesma. De maneira geral, as seguintes características podem ser alteradas pelo SLA: (COHEN, 2013, tradução nossa):

a) filtro de nós: especifica que tipo de nós podem receber um determinado trabalho. Pode ser útil em ambientes de clusters heterogêneos, onde não se pretende ocupar nós com grande poder de processamento em atividades de baixa necessidade computacional;
b) número de nós: especifica a quantidade de nós que podem receber um trabalho, evitando a quantidade de paralelismo para situações onde o ganho com o mesmo é limitado e/ou disponibilizando nós para trabalhos de maior prioridade;

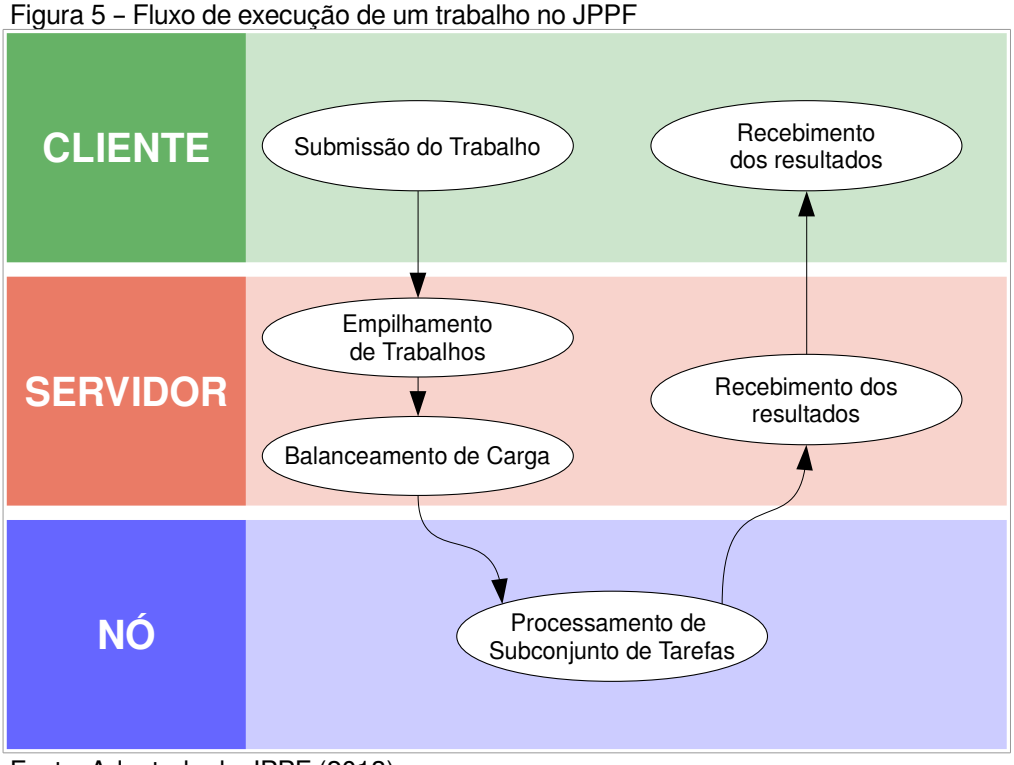
c) prioridade: determina a prioridade de execução de uma unidade de trabalho. Dessa forma, alguns trabalhos com tarefas de alta prioridade podem avançar para os nós antes de trabalhos com tarefas de pouca importância para o cliente;

d) agendamento: pode-se determinar um tempo para início e expiração de um determinado trabalho. Assim, é possível que tarefas aguardem na fila do nó e sejam executadas somente quando o tempo estipulado for atingido. Da mesma forma, tarefas que não são mais necessárias por questões de tempo podem ser descartadas para evitar processamento desnecessário;

e) balanceamento personalizado: É possível incluir metadados definidos pelo desenvolvedor que possam ser lidos pelo balanceador de cargas no servidor JPPF com a finalidade de modificar a forma de distribuição dos trabalhos.

O SLA acompanha o trabalho o fluxo de execução de um trabalho, que pode ser observado na figura 5:

Figura 5 – Fluxo de execução de um trabalho no JPPF



Fonte: Adaptado de JPPF (2013).

Após o cliente submeter um trabalho, o servidor recebe o mesmo e o coloca em uma fila de espera, onde aguarda a aplicação do SLA para determinar quando e como o trabalho deve ser submetido, passando então pelo balanceador de carga, que avalia os nós conectados ao servidor para determinar quais as melhores opções para o envio das tarefas (COHEN, 2013, tradução nossa).

Cada nó recebe então um subconjunto com tarefas de uma unidade de trabalho, onde a quantidade depende do algoritmo de balanceamento utilizado e da disponibilidade de *threads* de processamento configuradas no componente. O processamento das tarefas é então realizado e os resultados são enviados de volta o servidor, que apenas repassa ao cliente de origem do trabalho. O tratamento dos resultados fica por conta da aplicação cliente (COHEN, 2013, tradução nossa).

3.3 COMUNICAÇÃO

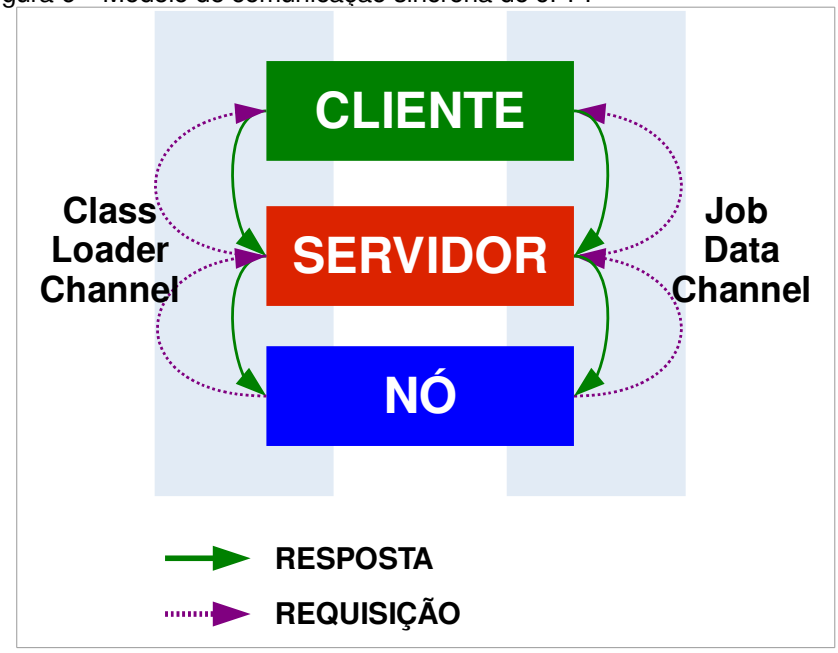
Em um sistema distribuído é de extrema importância a qualidade da comunicação para o funcionamento e desempenho. O JPPF funciona por meio de um modelo de comunicação baseado no paradigma de requisição e resposta, além de implementar um protocolo próprio na camada de aplicação para a troca de mensagens.

3.3.1 Conexão

Cada conexão entre um servidor e outro componente qualquer é composta de dois canais de rede. Um dos canais é utilizado exclusivamente para o transporte dos dados de um trabalho (Job data channel), enquanto o outro é utilizado para carregar as classes java que são necessárias para a execução da tarefa (Class loader channel). Essas classes são carregadas sob demanda e de forma transparente ao desenvolvedor do cliente (COHEN, 2013, tradução nossa).

Toda comunicação no JPPF é realizada de maneira síncrona, como ilustrado na figura 6:

Figura 6 – Modelo de comunicação síncrona do JPPF



Fonte: Adaptado de JPPF (2013).

Esse modelo permite atribuir quem faz a requisição e quem responde de acordo com os componentes que estão se comunicando e seus respectivos canais. Existem ainda, outras implicações a cerca do modelo de comunicação:

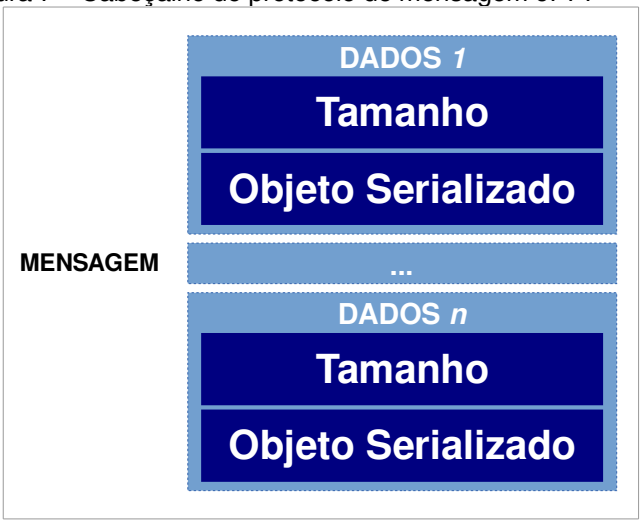
a) os nós podem executar apenas um trabalho por vez, mas podem executar várias tasks em paralelo;
b) cada conexão entre cliente e servidor pode processar um único trabalho por vez, mas cada cliente pode ter inúmeras conexões com o mesmo servidor ou com vários servidores;
c) um servidor conectado a outro pode executar apenas um nó por vez, tal como se fosse um nó do segundo.

3.3.2 Protocolo

Como descrito na seção anterior, toda comunicação realizada em um grid/cluster JPPF é realizada por meio da troca de mensagens em um modelo requisição/resposta (COHEN, 2013, tradução nossa).

O JPPF implementa um protocolo próprio para troca de mensagens de requisição e resposta na camada do ambiente, e todas possuem basicamente a mesma estrutura: blocos de bytes precedidas pelo tamanho do bloco, como é representado na figura 7 (COHEN, 2013, tradução nossa):

Figura 7 – Cabeçalho do protocolo de mensagem JPPF

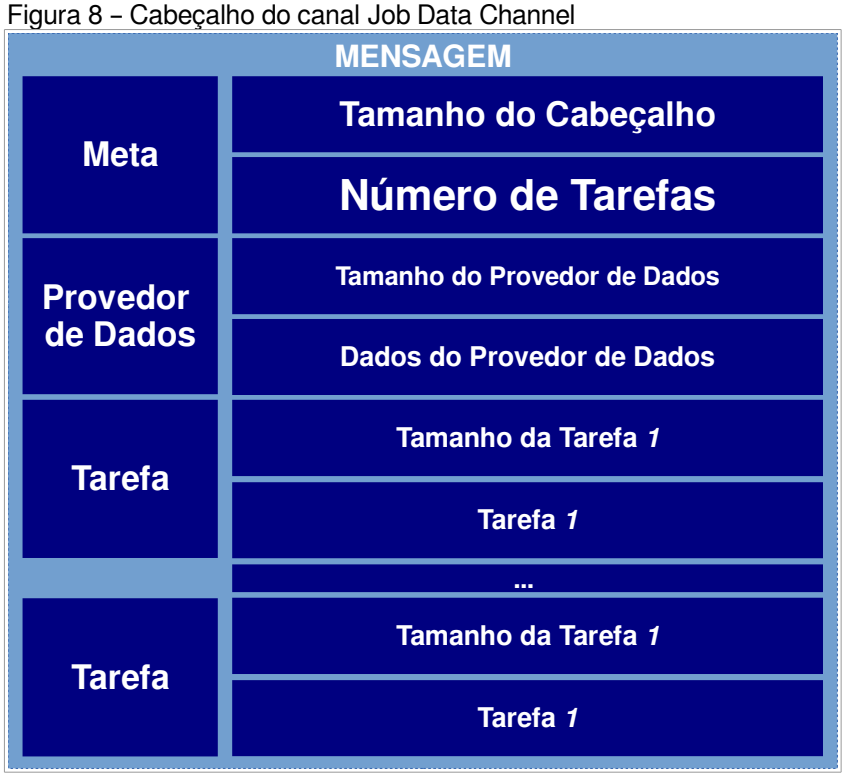


Fonte: Adaptado de JPPF (2013).

A estrutura é ligeiramente diferente entre os dois canais de comunicação.

No Job Data Channel, uma requisição é composta de um cabeçalho contendo informações como o número de tarefas em um trabalho, o SLA e metadados; um container somente de leitura denominado Data Provider, que permite o compartilhamento de informações entre tasks; e a task em si. A resposta neste canal é similar, mas não possui o Data Provider, como pode ser visualizado na figura 8 (COHEN, 2013, tradução nossa).

Figura 8 – Cabeçalho do canal Job Data Channel



Fonte: Adaptado de JPPF (2013).

No canal do Class Loader, todas as mensagens, sejam de requisição ou resposta, contém apenas um tamanho e um objeto serializado, como pode ser observado na figura 9 (COHEN, 2013, tradução nossa):

Figura 9 – Cabeçalho do canal Class Loader

Fonte: Adaptado de JPPF (2013).

3.4 PARALELISMO

A principal vantagem na utilização de sistemas distribuídos encontra-se na capacidade de executar diversas tarefas simultaneamente (TANENBAUM; STEEN, 2007).

Dessa forma, o JPPF propõe a sua utilização sobre diversos aspectos com a finalidade de aumentar o grau de paralelismo presente no ambiente de cluster/grid.

3.4.1 Paralelismo sob a perspectiva do Cliente

Representando a primeira instância, ou a instância de entrada de um cluster/grid JPPF, o cliente possibilita a criação de diversas formas de paralelismo (COHEN, 2013, tradução nossa):

a) único cliente, múltiplos trabalhos: um único cliente pode ter diversos trabalhos em paralelo. Entretanto, trabalhos submetidos por meio de uma única conexão serão enfileirados do lado do cliente e serão processados pelo servidor um a um, pois o servidor pode aceitar apenas um único trabalho por conexão. Para permitir o paralelismo, é necessário configurar o *pool of connections* do cliente. O tamanho do p*ool* determina o número de conexões e, consequentemente, o número de unidades de trabalho que podem ser processados paralelamente pelo servidor;
b) múltiplos clientes: diversas aplicações com funções distintas são uma forma de paralelismo natural no nível do cliente. Mesmo que haja apenas uma conexão por cliente, todas as conexões trabalharão concorrentemente;
c) execução remota e local: clientes possuem a habilidade de executar trabalhos localmente, no mesmo processo do cliente. Dessa forma, é possível misturar as duas abordagens para fornecer uma fonte adicional de paralelismo.

3.4.2 Paralelismo sob a perspectiva do Servidor

Do ponto de vista do servidor, é possível obter também três fontes de paralelismo distintas (COHEN, 2013, tradução nossa):

a) número de conexões: o número de conexões de clientes representa diretamente o número de trabalhos que podem ser processados pelo cluster/*grid* ao mesmo tempo;
b) número de nós: dado que cada nó pode executar apenas um único trabalho por vez, o número de nós conectados ao servidor também determina a quantidade máxima de trabalho que podem ser processados pelo cluster/grid ao mesmo tempo;
c) balanceamento de carga: o balanceamento de carga é responsável por dividir as unidades de trabalho em subconjuntos de tarefas e distribuir entre os nós disponíveis. Dessa forma, maximizar a utilização dos nós pode colaborar para que mais trabalhos sejam executados paralelamente.

3.4.3 Paralelismo sob a perspectiva do Nó

Da perspectiva do nó, as *threads* de processamento representam a única possibilidade de paralelismo. Cada nó pode especificar uma quantidade de *threads* para processamento de tarefas. Essa quantidade determina o número máximo de *tasks* que podem ser executadas em paralelo e concorrentemente por um único nó. Para diminuir a concorrência, é possível ajustar esse número para o atual número de processadores disponíveis no nó (COHEN, 2013, tradução nossa).

3.5 JPPF EM SERVIDORES WEB

Toda a estrutura do JPPF foi desenvolvida para ser executada em sistemas operacionais que possuam uma máquina virtual compatível com a plataforma Java. Servidores de aplicação para a Web também podem aproveitar os recursos disponíveis da interface de programação JPPF para a criação de aplicações web distribuídas e que utilizem modelos computacionais não dedicados. As características da plataforma Web serão apresentadas no capítulo a seguir.

4 SISTEMAS WEB

Desde a criação da World Wide Web, no início da década de 90, uma enorme quantidade de aplicações para esta plataforma foram desenvolvidas e são utilizadas diariamente. Um sistema Web é tipicamente composto de algoritmos e uma base de dados (back-end) e páginas web (o front-end), pelas quais o usuário pode interagir através da utilização de navegadores. Estas aplicações podem ser categorizadas em dois tipos – estáticas, quando o conteúdo da página não é modificado de acordo com as entradas do usuário; e dinâmicas, quado o conteúdo enviado ao usuário é personalizado de acordo com as ações, sequências e entradas do utilizador (LI; DAS; DOWE, 2014).

A história e as características da Web estão intimamente ligadas ao surgimento das redes de computadores, cujo a visão geral da história pode ser visualizada na figura 10.

4.1 HISTÓRIA DA INTERNET

Figura 10 – Surgimento da Internet e da Web

1960	→ Primeiras redes de computadores
1965	
1970	→ ArpaNET: Primeira rede baseada em pacotes → Primeiro protocolo para troca de e-mails
1975	→ Princípio das redes Ethernet → Protocolos TCP, UDP e IP
1980	→ Redes de comunicação independentes da ArpaNET
1985	→ Estabelecimento de padrões
1990	→ Interrupção definitiva da ArpaNET → Criação da Web
1995	→ Primeiros provedores de serviços da Internet → Progresso significativo no desenvolvimento das redes

Fonte: Adaptado de KUROSE (2003).

As primeiras redes de computadores foram implementadas no início da década de 60, quando as redes de telefonia eram mundialmente dominantes na área de comunicação. Dado a grande importância que os computadores adquiriram neste período, tornou-se natural considerar a conexão entre máquinas para oferecer recursos a usuários geograficamente distribuídos. Através do trabalho realizado ao longo da década por grupos de pesquisa na busca por técnicas de implementação de circuitos e troca de dados, chegou-se, em 1967, à primeira publicação de um modelo de rede de computadores baseada na troca de pacotes que ficou conhecida como ARPAnet, antecessor direto da Internet (KUROSE; ROSS, 2003).

Em 1972, a implementação da ARPAnet, realizada no território dos Estados Unidos, possuía aproximadamente 15 nós. Para comunicar-se com um determinado servidor era necessário apenas que um computador estivesse conectado a outro nó da rede. No mesmo ano, o primeiro protocolo para troca de e-mails foi implementado. O surgimento do princípio das redes baseadas na arquitetura Ethernet ampliou o crescimento de redes locais, cujo a conexão operava sobre pequenas distâncias, criando o conceito de redes para a conexão de outras redes de computadores. Ainda no fim da década de 70, surgiram os protocolos (da forma que conhecemos hoje) Transmission Control Protocol (TCP), User Datagram Protocol (UDP) e o Internet Protocol (IP) (KUROSE; ROSS, 2003).

No fim da década de 70, a rede ARPAnet possuía aproximadamente 200 hosts interconectados, número que chegaria a 100 mil através da rede Internet no final da década de 80. A maior parte desse crescimento foi resultado dos esforços na criação de redes que conectassem as universidades, que também alavancaram as pesquisas na área. Diversas redes de computadores paralelas surgiram sem necessariamente uma conexão com a ARPAnet. A conexão destas redes viria, posteriormente, estabelecer a definição de Internet. Ainda nesse tempo, uma grande quantidade de protocolos e modelos desenvolvidos estabeleceram os padrões que viriam a ser empregados nos anos seguintes (KUROSE; ROSS, 2003).

A década de 90 foi marcada pela intensa popularização e comercialização das redes de computadores. Inicialmente, a ARPAnet foi oficialmente descontinuada. Algumas outras redes passaram a fazer o papel de *backbone*, por onde grande parte

do tráfego de dados seria propagado até atingir redes menores. Em 1995, esse tráfego seria oficialmente realizado por provedores de serviço de Internet comerciais. Durante os anos 90, a pesquisa e desenvolvimento na área de redes fez significantes progressos na implementação de roteadores de alta velocidade, proporcionando um aumento significativo da transmissão de dados. No entanto, o principal evento desta década foi a criação da Web, que estimulou o compartilhamento de informações de forma distribuída e de fácil acesso (KUROSE; ROSS, 2003).

4.2 A WEB

A Web é um grande exemplo de como uma ideia simples pode ter resultados extraordinários. Projetada originalmente como uma forma de prover páginas estáticas conectadas, este meio cresceu rapidamente para englobar conteúdos dinâmicos e tornou-se parte da vida cotidiana do mundo (FORD, 2004).

Inventada no CERN por Tim Berners-Lee em 1989-1991, baseado nas ideias de hipertexto, originalmente expostas em 1940, por Bush, e 1960, por Ted Nelson. Berners-Lee e seus colegas desenvolveram as versões iniciais dos protocolos HyperText Markup Language (HTML), Hypertext Transfer Protocol (HTTP), *Web servers* e um browser – os quatro componentes chaves da Web (KUROSE; ROSS, 2003).

Em 1992, já havia cerca de 200 *Web servers* em operação, que interagiam com o navegador inicial desenvolvido pelo CERN. Inicialmente, este navegador possuía apenas uma interface de linha de comando. Com a difusão das interfaces gráficas, novos navegadores gráficos foram desenvolvidos e tornaram ainda mais acessível o acesso aos conteúdos disponibilizados pelos servidores na Web (KUROSE; ROSS, 2003).

4.3 APLICAÇÕES WEB

Os primeiros esforços na área foram escritos basicamente nas linguagens C e Perl. Com o tempo e o avanço da tecnologia, novas interfaces de programação

foram desenvolvidas para melhorar e sobrepujar as limitações das linguagens e ferramentas existentes. Nesse âmbito, a linguagem Java rapidamente demonstrou eficácia no mundo das aplicações Web distribuídas. Inicialmente, os primeiros passos dados pela linguagem foram através dos *servlets*, que permitiam aos desenvolvedores a criação de conteúdos dinâmicos de maneira fácil e rápida (FORD, 2004).

Todavia, a diferenciação entre o desenvolvimento das aplicações (servlets Java) e suas interfaces gráficas (desenvolvidas em HTML) gerou um novo desafio para os profissionais responsáveis pela construção destes sistemas: misturar o conteúdo gerado dinamicamente com as interfaces desenvolvidas pelos Web designers. Esta situação direcionou o desenvolvimento da tecnologia JavaServer Pages, que permite uma maior distinção dos componentes da aplicação (FORD, 2004).

Para a disponibilização das aplicações web desenvolvidas em Java são servidores de aplicação especializados, que realizam a função dos primeiros servidores web, distribuindo o conteúdo aos navegadores cliente, além de processar e gerenciar toda a aplicação.

4.4 SERVIDORES DE APLICAÇÃO JAVA

De um modo geral, um servidor de aplicação é responsável pelo fornecimento de páginas web e fornecem um modelo de *container* para as aplicações. Além disso, é possível que estes servidores também forneçam ferramentas de gestão e/ou desenvolvimento e distribuam as solicitações em servidores físicos, no caso da utilização de um modelo clusterizado. Os servidores de aplicação Java fornecem uma plataforma de execução que adere as especificações Java Enterprise Edition (JEE) e permitem a utilização de tecnologias de *servlets* e JSP. Dentre os servidores de código fonte aberto de maior popularidade, pode-se citar os softwares Glassfish, Apache Tomcat e Wildfly (OTTINGER, 2008).

4.4.1 GlassFish

O GlassFish é um projeto de código fonte aberto, que tem como objetivo desenvolver um servidor de aplicações para a plataforma JEE e tecnologias Web baseadas na plataforma Java. Atualmente, o projeto é liderado pela empresa Oracle e mantido pelo modelo de comunidade de desenvolvedores. Deste projeto, deriva-se um servidor denominado GlassFish Server Open Source Edition, disponibilizado através das licenças Common Development and Distribution License (CDDL) e GNU General Public License (GPL). Em sua quarta versão, o GlasshFish Server provê recursos como Web Container, painel de administração para configuração e gerenciamento e suporte para clusterização e balanceamento de carga (ORACLE, 2013).

4.4.2 Apache Tomcat

O Apache Tomcat é uma implementação de software de código fonte aberto que permite a utilização das tecnologias Java Servlet e JSP, desenvolvido e disponibilizado através da licença Apache versão 2. Este projeto também é liderado pela Apache Software Foundation e mantido por uma comunidade de desenvolvedores (APACHE, 2014).

4.4.3 WildFly

Liderado atualmente pela empresa Red Hat, o WildFly é uma continuação direta do projeto Jboss Application Server, cujo o nome foi modificado para diminuir a ambiguidade com a linha de softwares oferecida pela Red Hat. Entre as principais características destacadas pelo servidor, encontra-se um eficaz gerenciamento de memória e alta performance na inicialização das aplicações (RED HAT, 2014).

5 TRABALHOS CORRELATOS

Os trabalhos apresentados a seguir estão relacionados com os objetivos e/ou temas já apresentados neste trabalho: Sistemas Distribuídos, JPPF e Sistemas Web.

5.1 MOWS

O MOWS é um projeto de servidor cache e web escrito em na linguagem Java desenvolvido em 1997. Ele foi desenvolvido para ser um servidor cooperativo e portátil, com módulos que podem ser carregados localmente ou remotamente na rede. Dentre as funcionalidades desenvolvidas para o MOWS, estão os módulos de CGI, filtro de arquivos HTML, mapa de imagens, cache de memória, proxy, cache de disco e redirecionamento (YOSHIDA, 1997).

Seu funcionamento permite a execução de servidores de páginas web de modo descentralizado, ou a constituição de servidores de cache com a finalidade de diminuir a carga de trabalho sobre um servidor central. Atualmente o projeto encontra-se descontinuado (YOSHIDA, 1997).

5.2 BALANCEAMENTO DE REQUISIÇÕES EM CLUSTER DE SERVIDORES WEB: UMA EXTENSÃO PARA O MOD_PROXY_BALANCER DO APACHE

O trabalho descreve e discute questões arquiteturais e operacionais sobre o balanceamento de requisições, abordando técnicas e soluções para um balanceamento efetivo e funcional (SATO, 2007).

Experimentos reais foram realizados sobre um cluster de servidores Apache provido com o módulo de balanceamento de carga, usando sobrecargas sintéticas intensivas e monitoradas em diferentes configurações (SATO, 2007).

A partir da análise e avaliação do comportamento e dos resultados destes experimentos, um novo método de distribuição de requisições foi proposto, experimentado e avaliado. O método tira proveito das situações em que a sobrecarga de rede não é detectada pelo balanceador pelo fato de ser gerada por agentes

externos ao serviço Web. Nestas situações de sobrecarga, os resultados mostraram que o novo método é mais eficiente quando comparado com os métodos já existentes sendo, recomendado para clusters não dedicados exclusivamente ao serviço Web (SATO, 2007).

5.3 ANALAZING WEB SERVER PERFORMANCE UNDER DYNAMIC USER WORKLOADS

Este artigo analisa o efeito do uso de uma carga de trabalho dinâmica mais realista sobre as métricas de desempenho da web. Através da análise de um cenário típico *e-commerce* e a comparação dos resultados obtidos utilizando-se diferentes níveis de carga de trabalho dinâmica em vez de cargas de trabalho tradicionais (PEÑA-ORTIZ et al, 2013).

Os resultados experimentais mostram que, quando uma carga de trabalho mais dinâmico e interativo é levado em conta, os índices de desempenho podem amplamente diferentes e afetam sensivelmente a fronteira estresse no servidor. Por exemplo, o uso do processador pode aumentar 30%, devido ao dinamismo, afetando negativamente o tempo de resposta médio percebido pelos usuários, o que também pode se transformar em efeitos indesejáveis nas políticas de marketing e fidelidade. (PEÑA-ORTIZ et al, 2013).

5.4 ANÁLISE DO BALANCEAMENTO DE REQUISIÇÕES EM CLUSTERS WEB NÃO DEDICADOS

Este trabalho descreve e discute questões arquiteturais e operacionais sobre o balanceamento de requisições, principalmente o dinâmico, abordando conceitos, técnicas e soluções (SATO; MARTINI; GONÇALVES, 2011).

Experimentos reais foram realizados em diferentes configurações sobre um cluster de servidores Apache não-dedicado exclusivamente ao serviço web. O módulo de balanceamento de carga mod_proxy_balancer foi usado com sobrecargas

sintéticas intensivas e um novo método de balanceamento denominado byalltraffic foi proposto e avaliado (SATO; MARTINI; GONÇALVES, 2011).

Os resultados mostram que o balanceamento neste tipo de sistema será mais eficiente se a carga de rede externa ao serviço web for detectada e usada no fator de balanceamento (SATO; MARTINI; GONÇALVES, 2011).

6 AVALIAÇÃO DE CLUSTER COMPUTACIONAL PARA APLICAÇÕES WEB

Mediante a utilização dos conceitos de computação distribuída, sistemas web e o framework JPPF apresentados neste trabalho, realizou-se uma análise comparativa de performance e características técnicas da utilização de duas abordagens distintas de clusters para processamento de tarefas de alta performance em aplicações web. A partir desta análise, foi possível observar particularidades na utilização de ambos os modelos, apresentando dificuldades e sugerindo diferentes situações para a implementação.

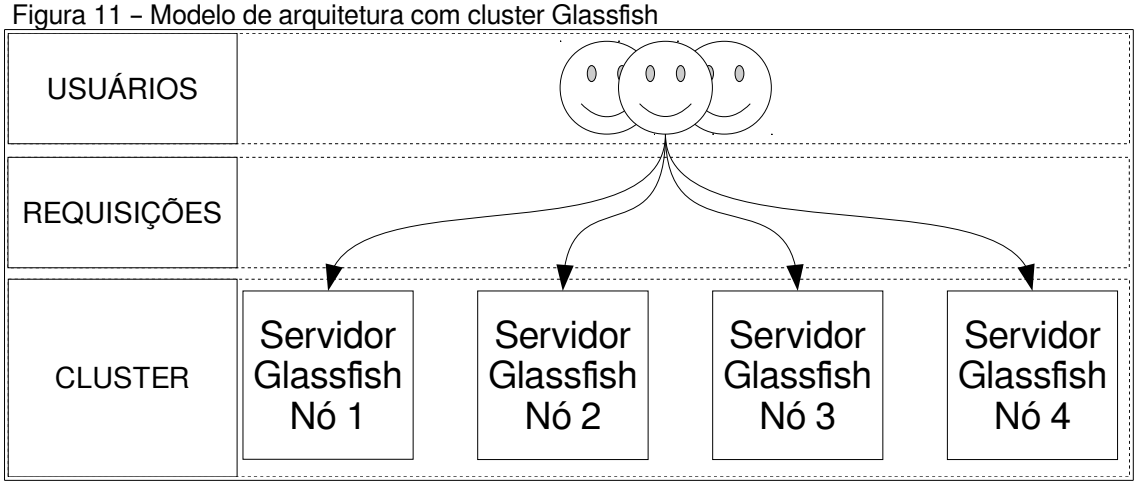
6.1 METODOLOGIA

A metodologia de desenvolvimento do trabalho consistiu inicialmente na elaboração de dois modelos de cluster para a execução de aplicações web baseadas na plataforma JEE e no desenvolvimento de uma aplicação protótipo para a realização dos testes de performance. Para a realização dos testes, foi utilizado como ferramenta o software JMeter, pelo qual realizou-se a simulação de diferentes números de usuários e cargas de trabalho e a posterior coleta dos dados para análise.

6.1.1 Arquiteturas propostas para avaliação

Foram elaborados dois modelos de clusters para a execução de aplicações web de forma distribuída.

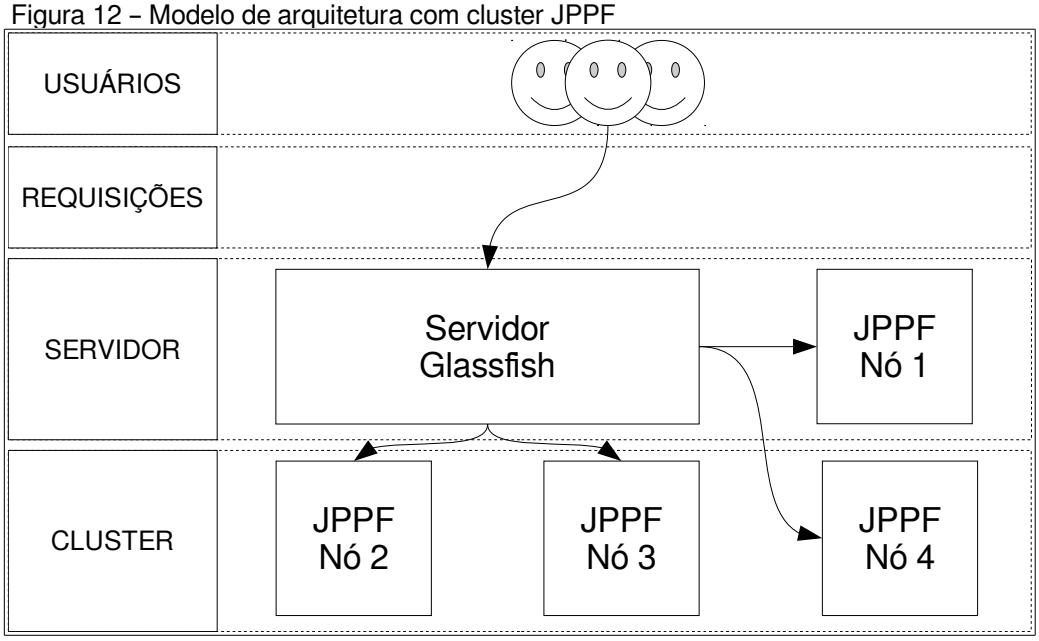
Figura 11 – Modelo de arquitetura com cluster Glassfish



Fonte: Do Autor.

No primeiro modelo, utilizando o Glassfish Server Open Source Edition, utilizou-se do suporte para a clusterização de servidores web disponível na ferramenta para a criação do ambiente de execução. Foram determinadas três cenários diferentes neste modelo: um único servidor, dois servidores clusterizados e quatro servidores clusterizados.

Figura 12 – Modelo de arquitetura com cluster JPPF



Fonte: Do autor.

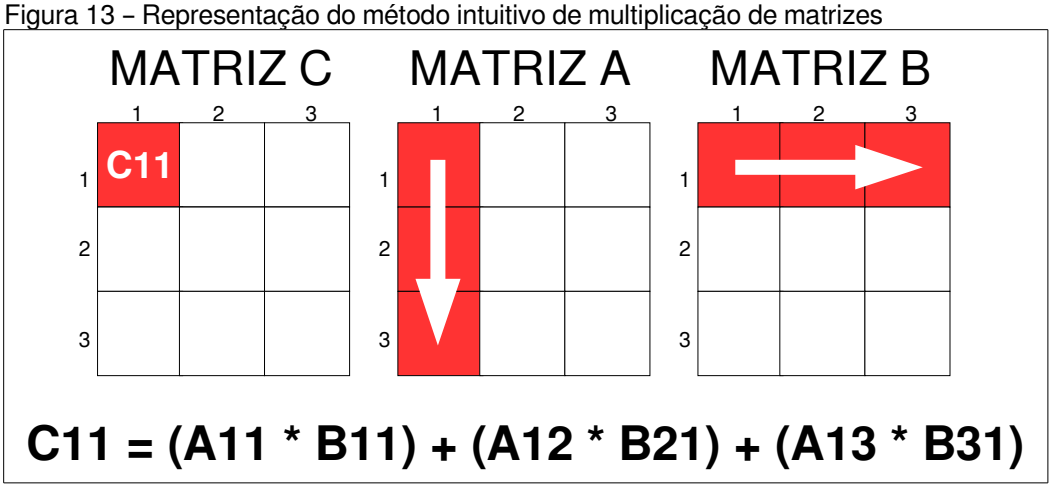
No segundo modelo, utilizou-se de um único servidor Glassfish para realizar a execução da aplicação. Entretanto, o processamento das tarefas implementadas na aplicação ficou ao encargo dos nós de um cluster de computadores construído por intermédio do *framework* JPPF. Da mesma forma que o primeiro modelo, foram determinadas três cenários diferentes neste modelo: um único nó, dois nós e quatro nós. Um dos nós do cluster foi executado na mesma máquina do servidor de aplicação Glassfish com o intuito de diminuir o tráfego de rede nas tarefas de menor carga de trabalho.

6.1.2 Desenvolvimento da Aplicação

A aplicação protótipo desenvolvida para a realização dos testes foi feita com base na especificação JEE e utilização do framework JSP, que auxiliou no desenvolvimento e organização da aplicação web.

A tarefa escolhida para implementação foi a de multiplicação de matrizes, em sua forma intuitiva, devido à sua alta complexidade ($O(n^3)$) e perspectiva de paralelização do método. A execução da multiplicação é realizada através da soma dos produtos da linha da matriz A pela coluna da matriz B, tal como mostra a figura 13.

Figura 13 – Representação do método intuitivo de multiplicação de matrizes



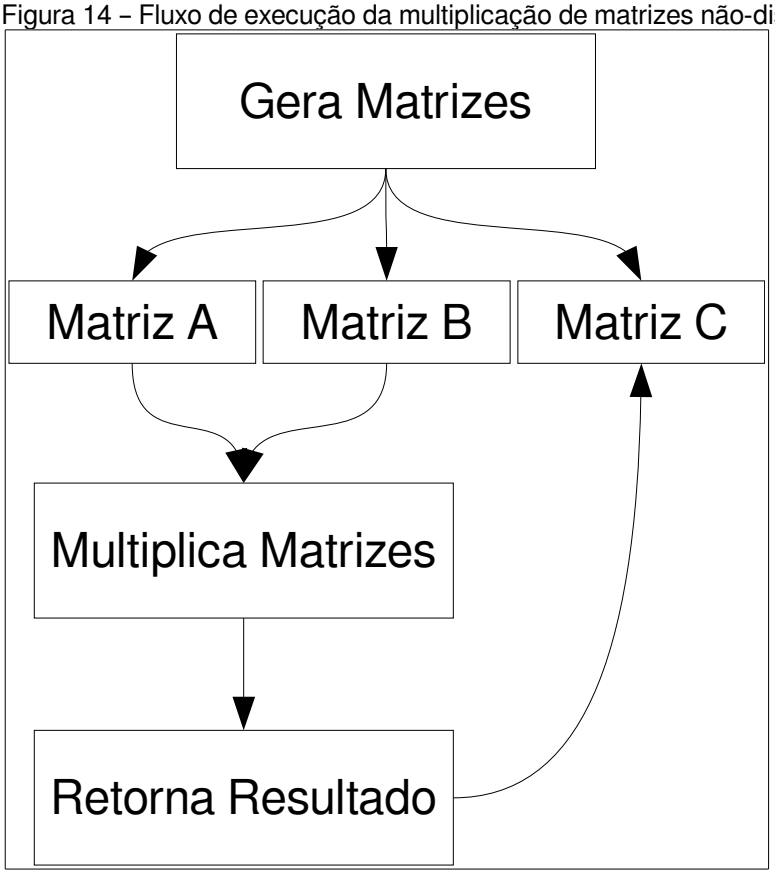
Fonte: Do autor.

Uma classe denominada Matrix foi implementada para a geração aleatória de três matrizes quadráticas de um tamanho previamente informado, sendo as duas

primeiras (matrixA e matrixB) os alvos da multiplicação, e a terceira (matrixC) o resultado dessa multiplicação. No entanto, foram implementados dois métodos distintos para a multiplicação das matrizes.

O primeiro método tinha como propósito a arquitetura de clusters somente com o servidor Glassfish. Não foram utilizadas neste método nenhuma técnica de programação paralela ou distribuída, de forma que a paralelização da tarefa ficasse por conta apenas do número de usuários realizando a requisição da tarefa.

Figura 14 – Fluxo de execução da multiplicação de matrizes não-distribuída

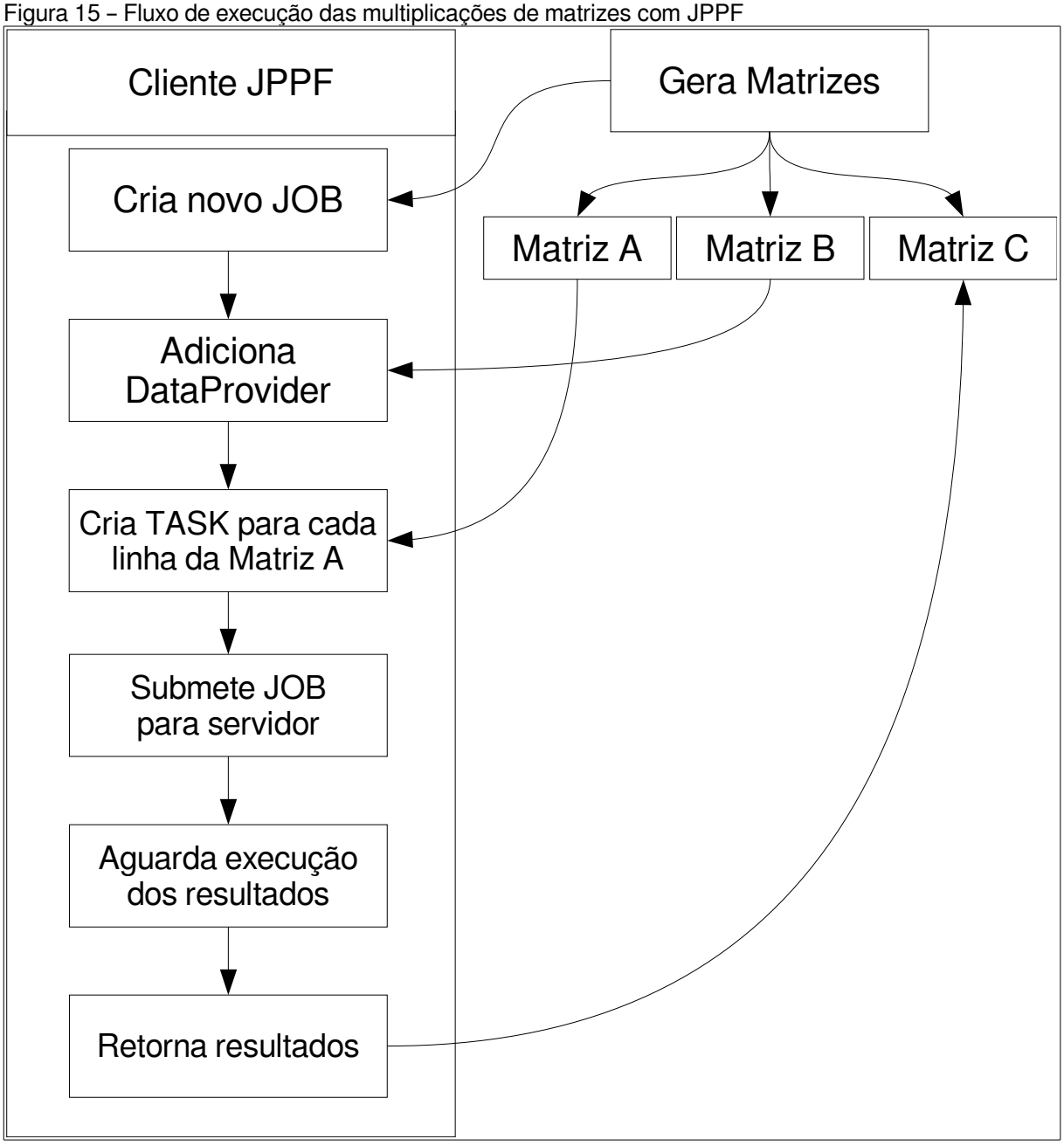


Fonte: Do autor.

O segundo método utilizou-se dos recursos da biblioteca JPPF para a construção de um modelo de processamento paralelizado e distribuído da tarefa de multiplicação de matrizes. Esse método foi implementado de forma que cada requisição efetuada por um usuário, gerasse um novo j*ob* composto por um número de *tasks* igual ao tamanho de cada matriz. Cada uma destas *tasks* recebia como parâmetro uma das linhas da matrixA, a ser multiplicada pelas colunas da matrixB,

que pela natureza do processo de multiplicação, foi disponibilizada a cada task através do recurso DataProvider do JPPF, onde cada uma das tarefas possuía acesso de leitura à matriz referente através da rede. O resultado da multiplicação de cada linha era retornado para ser posteriormente agrupado na matriz de resposta matrixC.

Figura 15 – Fluxo de execução das multiplicações de matrizes com JPPF



Fonte: Do autor.

Para gerenciar a conexão da aplicação com o servidor JPPF, foi implementada uma classe denominada JPPFController. Essa classe era responsável

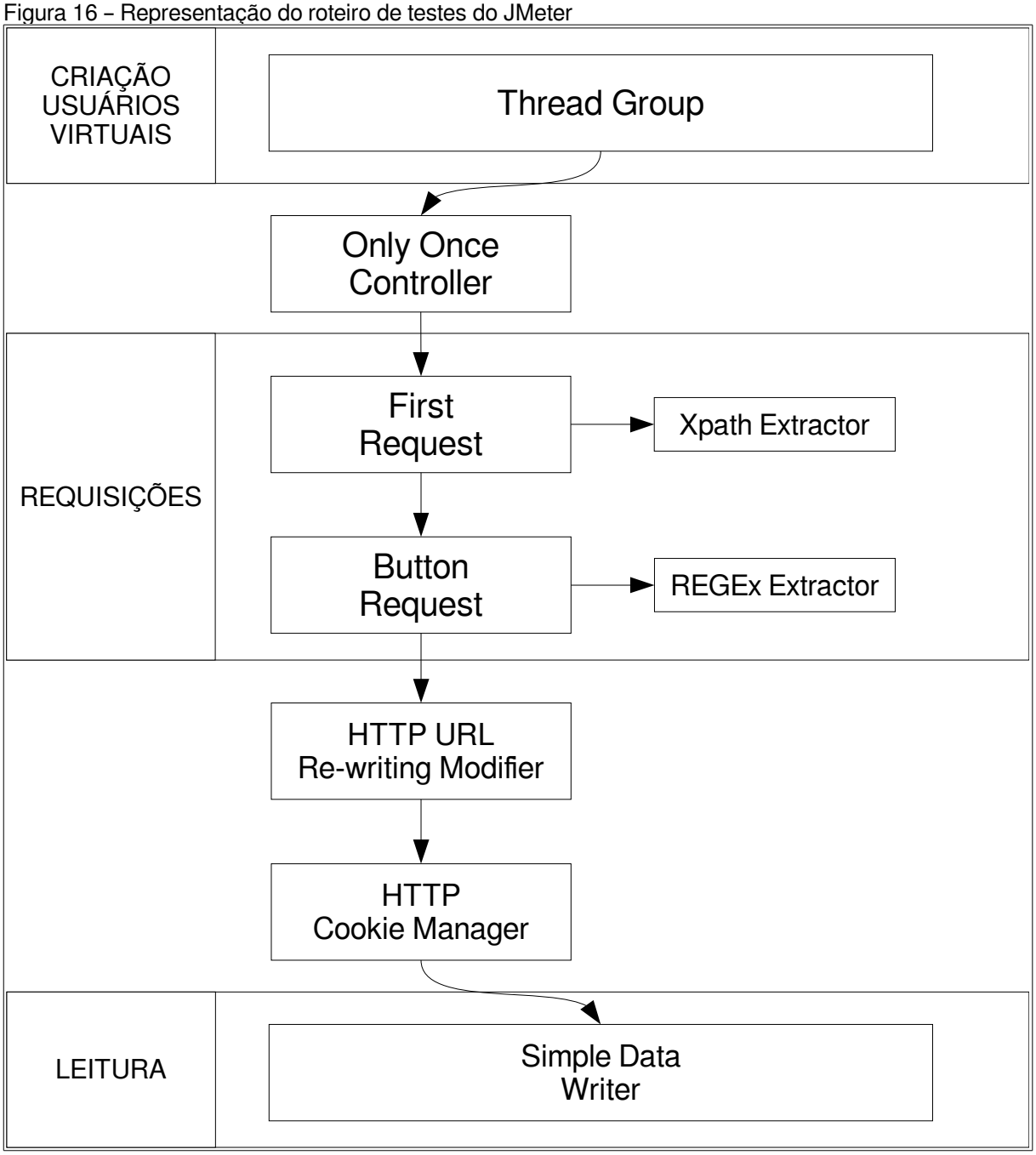
por realizar e estabelecer a conexão cliente/servidor apenas uma única vez, de modo a reduzir a carga de inicialização e execução das tarefas a cada requisição de um usuário à aplicação.

A interface gráfica da aplicação foi desenvolvida através de uma única página eXtensible HyperText Markup Language (XHML) com botões para acesso aos métodos citados durante a execução do script de testes, não sendo o escopo principal da implementação. As requisições realizadas através desta interface eram repassadas para a classe PrototipoMB, responsável pela execução principal da aplicação e pela coleta dos tempos de execução dos métodos chamados.

6.1.3 Roteiro de testes com JMeter

Para o desenvolvimento do roteiro dos testes, foi utilizado a ferramenta Apache JMeter, utilizada para testes de carga e performance com foco em aplicações web. O JMeter é um software *desktop* e de código fonte aberto e permite a criação de scripts (roteiros) com suporte *multithread*, onde é possível simular o acesso de um grande número de usuários virtuais realizando o acesso simultâneo a uma determinada aplicação (APACHE SOFTWARE FOUNDATION, 2014).

Figura 16 – Representação do roteiro de testes do JMeter



Fonte: Do autor.

O primeiro componente adicionado ao plano de testes foi o Thread Group. Através desse componente é possível definir um número de threads (análogo ao conceito de usuários virtuais) que realizarão a execução das funções definidas no script. Dentro deste componente, foi inserido o componente Only Once Controller e

dois componentes para realizar requisições HTTP: o First Request e o Button Request.

O First Request era responsável pelo acesso inicial à aplicação, onde, através do XPath Extractor, era possível guardar o atributo de ViewState de cada acesso, vinculado-o a cada usuário virtual. O Button Request era o componente efetivamente responsável pela requisição dos métodos implementados na aplicação por meio dos botões presentes na interface. A resposta do servidor à requisição deste componente continha o tempo de execução do método, sendo este extraído e incluído na tabela de resultados gerada pelo JMeter.

Por fim, os componentes HTTP URL Re-writing Modifier e HTTP Cookie Manager realizavam o armazenamento dos identificadores de sessão (JSESSIONID) e cookies.

Para a leitura dos dados, foi integrado ao script o componente Simple Data Writer, agente responsável por escrever em um arquivo os dados gerados pelo teste. Foi incluso neste arquivo a variável executionTime, extraída anteriormente pelo Button Request.

Para a execução dos testes no servidor clusterizado com Glassfish, criou-se quatro threadGroups distintos e o número total de usuários avaliado foi dividido igualmente entre todas as instâncias. Esse fato foi necessário dado que não houve a configuração de nenhuma ferramenta responsável pela distribuição de requisições entre os servidores Glassfish, circunstância não observada no cluster JPPF devido ao seu balanceamento de carga automatizado. Dessa forma, garantiu-se que todos os servidores receberiam exatamente o mesmo número de requisições de usuários.

6.1.4 Infraestrutura utilizada

Na realização dos testes, foram utilizados cinco computadores de arquitetura PC 64bits, equipados com processadores Intel I3 e quatro *gigabytes* de memória, localizados no Laboratório de Computação Distribuída do curso de Ciência da Computação da UNESC.

Os testes foram realizados em uma rede cabeada *ethernet*, de velocidade *gigabit* e por meio de um *switch* dedicado exclusivamente para o cluster.

6.1.5 Desenvolvimento dos testes

O roteiro executado pela ferramenta JMeter foi processado através de uma máquina adicional presente na mesma rede do ambiente de execução, sendo esta, responsável exclusivamente pela execução do roteiro e geração dos resultados.

O tamanho das matrizes foram determinados em três situações diferentes: matrizes 64x64, matrizes 128x128 e matrizes 256x256. Estes valores foram selecionados com o propósito de garantir consistência nos tempos retornados e diminuir a alta utilização de memória pela aplicação. O número de usuários seguiu uma lógica parecida, sendo determinado os valores de 1 a 256, com crescimento exponencial (1, 2, 4, 8, 16, 32, 64, 128, 256).

Figura 17 – Representação do plano de testes

Fonte: Do Autor.

O plano de testes foi executado levando-se em conta:

a) os seis cenários elaborados (três cenários para o cluster Glassfish e três para o cluster JPPF);
b) três cargas de trabalho (tamanho das matrizes) distintas (64, 128 e 256);
c) e nove grupos de usuários (1, 2, 4, 8, 16, 32, 64, 128, 256).

Assim sendo, foram realizados 162 testes de configurações distintas, onde, cada teste, foi repetido 3 vezes com a finalidade de aprimorar a consistência dos dados coletados.

Dentre os problemas analisados durante o andamento dos testes, pode-se relatar a grande quantidade de requisições não respondidas pelo servidor nos casos onde houve grande quantidade de usuários (como 256 e 128) e matrizes de grande porte (256).

Após a execução dos testes, os dados coletados foram organizados de forma a extrair três métricas para a avaliação. As métricas escolhidas foram:

a) latência: refere-se ao tempo total entre o envio da requisição e o recebimento da resposta pelo usuário. Neste caso, a latência do componente First Request foram somados à latência total;
b) tempo de execução: tempo retornado pela resposta componente Button Request, equivalente ao tempo total de execução do método de multiplicação de matriz;
c) número máximo de usuários ativos: número máximo de usuários (*threads*) processadas pelo servidor simultaneamente.

Todos os dados foram analisados e separados por tamanho da matriz utilizada e do número de nós do cluster JPPF e Glassfish.

6.2 RESULTADOS OBTIDOS

A análise dos dados obtidos foi realizada por meio da construção de gráficos comparativos das métricas observadas. Além disso, foram abordados também fatores técnicos referentes à instalação, arquitetura e funcionamento dos modelos propostos.

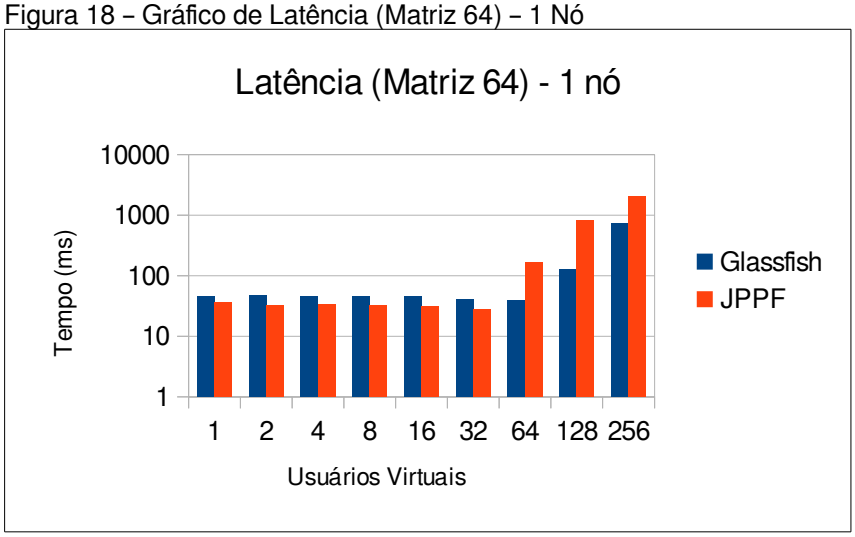
Todos os gráficos estão apresentados na escala logarítmica para uma melhor visualização, devido à grande discrepância entre os valores de tempo. A métrica do número máximo de usuários foi organizada em tabelas, também com a finalidade de uma melhor demonstração dos dados.

6.2.1 Avaliação do desempenho

O primeiro modelo analisado foi utilizando apenas um nó de processamento. No modelo Glassfish, houve apenas um único servidor de aplicação responsável por executar a aplicação desenvolvida. Já no modelo JPPF, o servidor de aplicação foi utilizado em conjunto com um nó do JPPF.

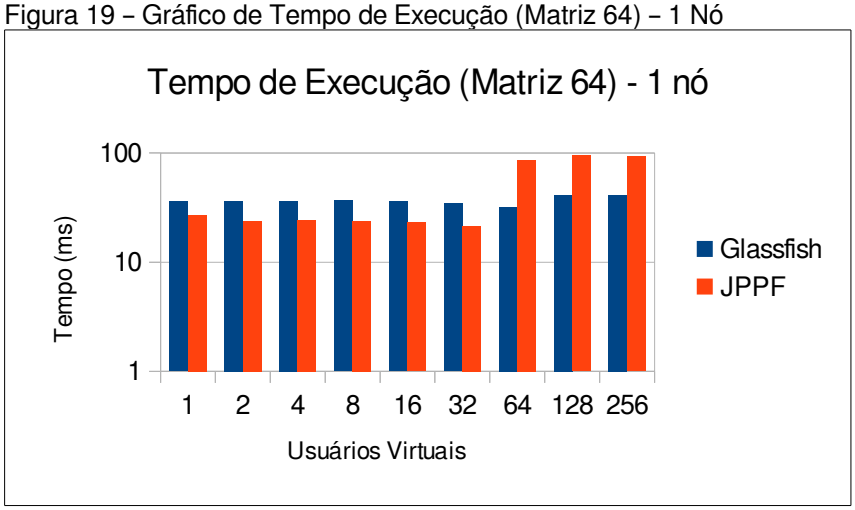
Os resultados obtidos com a execução da tarefa com matrizes de 64x64 foram os seguintes:

Figura 18 – Gráfico de Latência (Matriz 64) – 1 Nó



Fonte: Do Autor.

Figura 19 – Gráfico de Tempo de Execução (Matriz 64) – 1 Nó



Fonte: Do Autor.

Tabela 1 – Tabela de Número Máximo de Usuários (Matriz 64) – 1 Nó

Usuários	Glassfish	JPPF
1	1	1
2	1	1
4	1	1
8	1	1
16	1	1
32	2	2
64	4	17
128	25	80
256	154	208

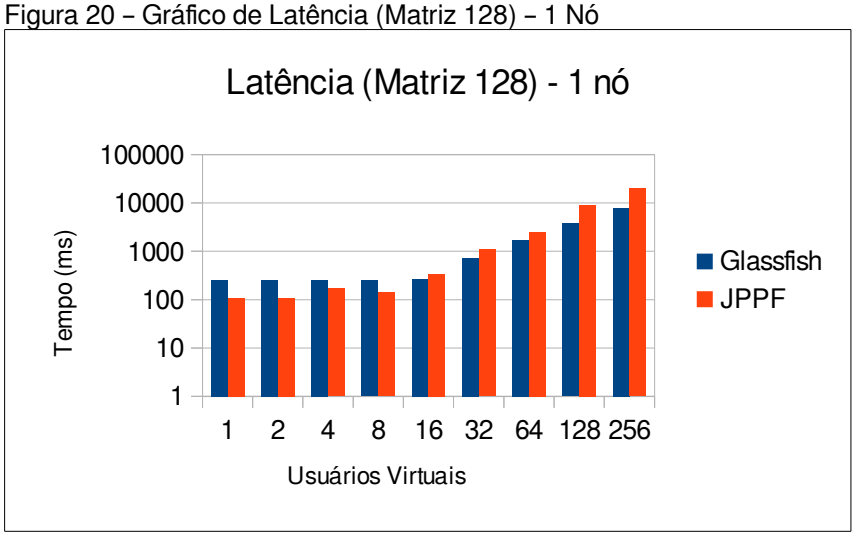
Fonte: Do Autor.

Em uma perspectiva inicial, observa-se que o modelo utilizando o JPPF foi ligeiramente superior nos casos com poucos usuários, mesmo tratando uma camada adicional para o processamento dos dados com relação ao modelo exclusivamente com Glassfish. Isso demonstra a importância do desenvolvimento das tarefas de forma paralela, técnica explorada pelo JPPF e que resultou numa maior aproveitamento do processador da máquina utilizada.

Outra particularidade observada foi o aumento do tempo do modelo JPPF nos casos com maiores usuários. A tabela 1 apresenta a dificuldade deste modelo em responder as requisições de muitos usuários com velocidade, quando comparado ao modelo Glassfish.

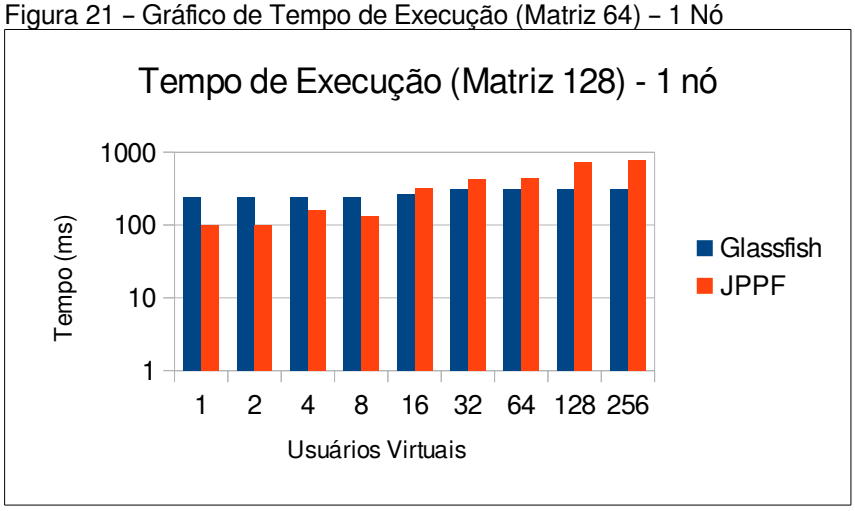
Os resultados obtidos com a execução de matrizes de tamanho 128x128 foram os seguintes:

Figura 20 – Gráfico de Latência (Matriz 128) – 1 Nó



Fonte: Do Autor.

Figura 21 – Gráfico de Tempo de Execução (Matriz 64) – 1 Nó



Fonte: Do Autor.

Tabela 2 – Tabela de Número Máximo de Usuários (Matriz 128) – 1 Nó

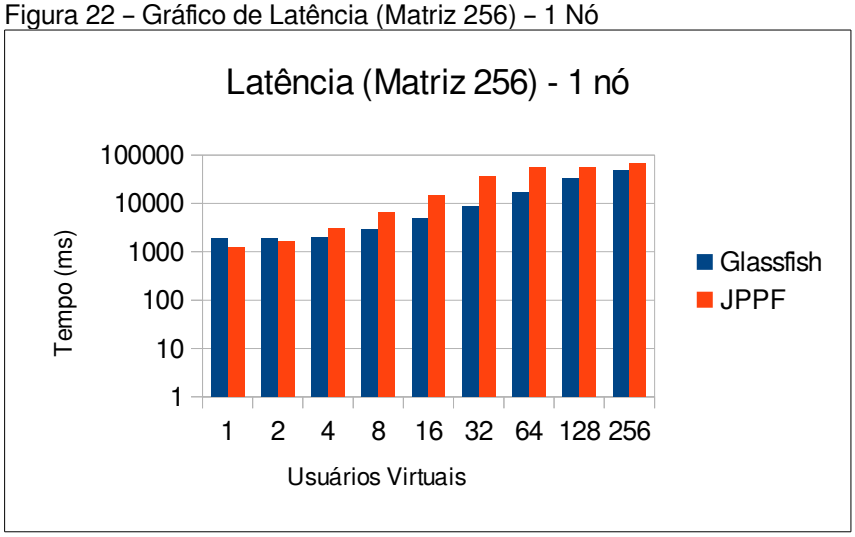
Usuários	Glassfish	JPPF
1	1	1
2	1	1
4	2	1
8	3	3
16	5	7
32	20	23
64	52	54
128	116	124
256	244	251

Fonte: Do Autor.

Com o aumento da carga de trabalho, o modelo JPPF teve um menor desempenho a partir da configuração com 16 usuários.

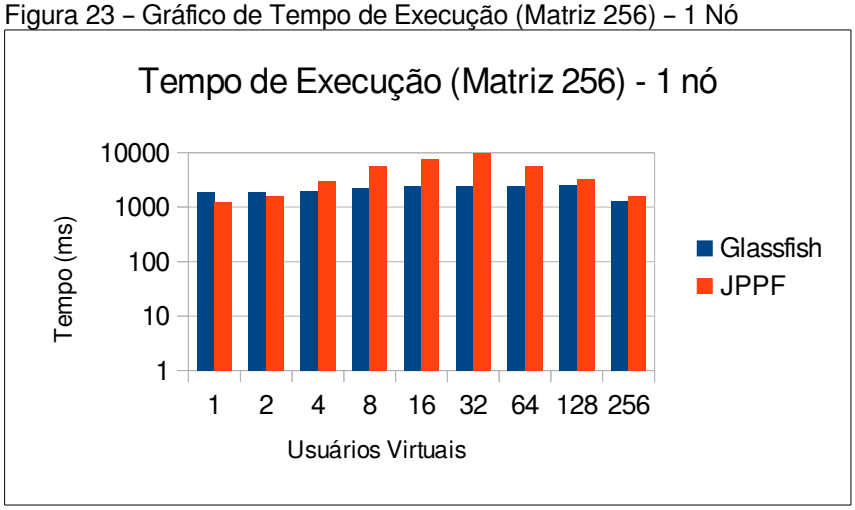
Com o modelo com um único nó, o último teste realizado foi utilizando matrizes de 256x256:

Figura 22 – Gráfico de Latência (Matriz 256) – 1 Nó



Fonte: Do Autor.

Figura 23 – Gráfico de Tempo de Execução (Matriz 256) – 1 Nó



Fonte: Do Autor.

Tabela 3 – Tabela de Número Máximo de Usuários (Matriz 256) – 1 Nó

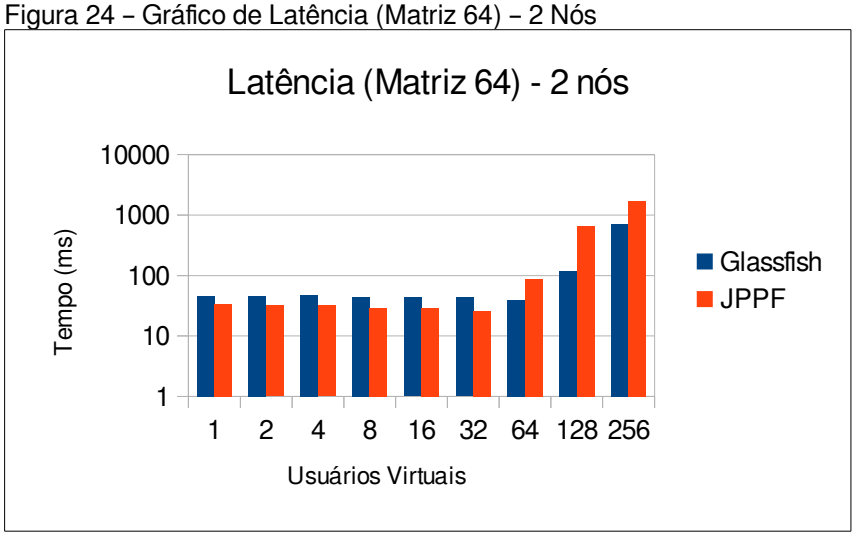
Usuários	Glassfish	JPPF
1	1	1
2	2	2
4	4	4
8	8	8
16	16	16
32	32	32
64	64	64
128	128	128
256	256	256

Fonte: Do Autor.

Mais uma vez o modelo JPPF regrediu quando comparado ao modelo Glassfish. Um aspecto importante a ser exposto durante essa etapa foi a de que um grande número de requisições efetuadas nas configurações com 64, 128 e 256 usuários não foram respondidas pelo servidor. Este fato foi observado com maior frequência no modelo JPPF.

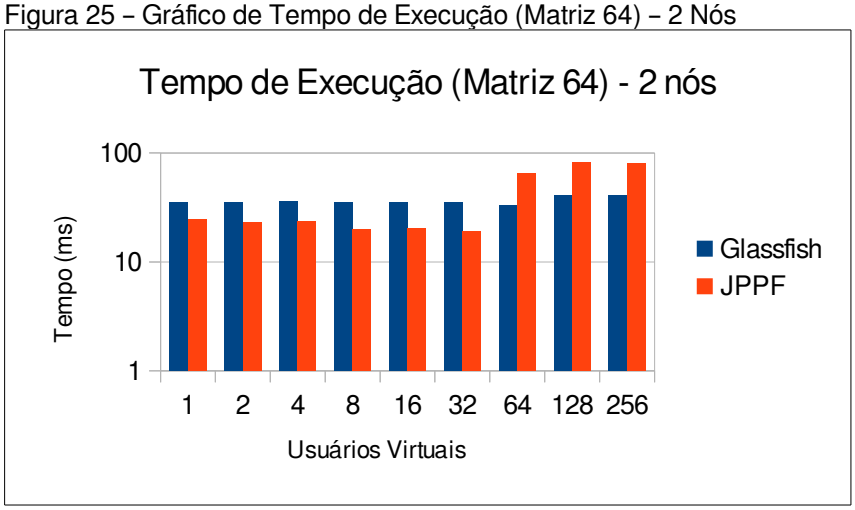
O segundo ambiente analisado foi utilizando-se de 2 nós de processamento para ambos os modelos. É importante ressaltar que no modelo JPPF, um dos nós foi implementado com o servidor de aplicação, com a finalidade de reduzir o custo de rede gerado pela transmissão de dados entre os nós e a aplicação. Foram considerados os testes nesse modelo com matrizes de 64x64:

Figura 24 – Gráfico de Latência (Matriz 64) – 2 Nós



Fonte: Do Autor.

Figura 25 – Gráfico de Tempo de Execução (Matriz 64) – 2 Nós



Fonte: Do Autor.

Tabela 4 – Tabela de Número Máximo de Usuários (Matriz 64) – 2 Nós

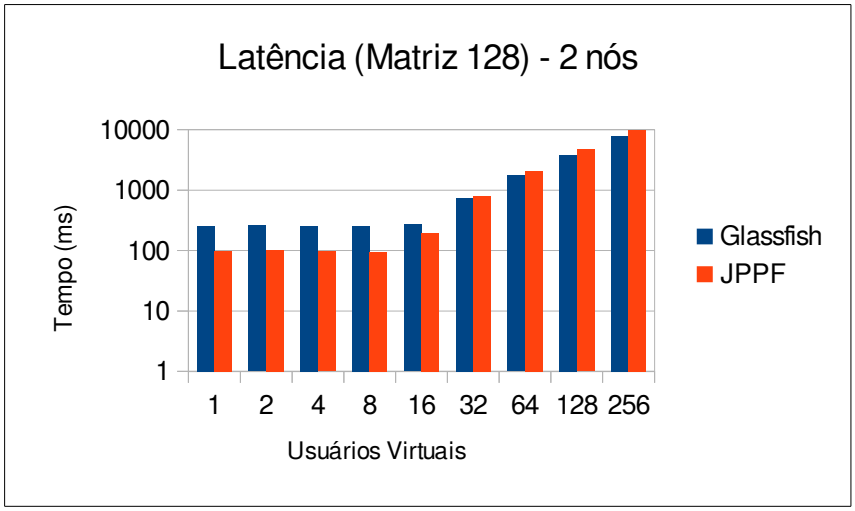
Usuários	Glassfish	JPPF
1	1	1
2	2	1
4	2	1
8	2	1
16	2	1
32	2	2
64	4	11
128	24	75
256	153	200

Fonte: Do Autor.

Com a utilização de dois nós, verificou-se uma ligeira melhora na latência e no tempo de resposta do modelo JPPF, entretanto, não foi o suficiente para alterar o padrão observado com a utilização de um nó individual. O cluster com Glassfish não obteve nenhuma melhora no processamento da aplicação.

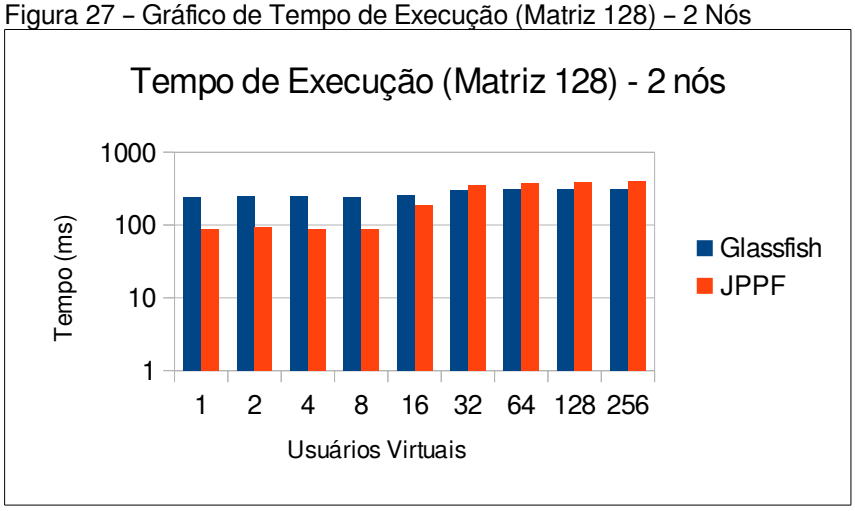
Em seguida, foram realizados testes com matrizes 128x128:

Figura 26 – Gráfico de Latência (Matriz 128) – 2 Nós



Fonte: Do Autor.

Figura 27 – Gráfico de Tempo de Execução (Matriz 128) – 2 Nós



Fonte: Do Autor.

Tabela 5 – Tabela de Número Máximo de Usuários (Matriz 128) – 2 Nós

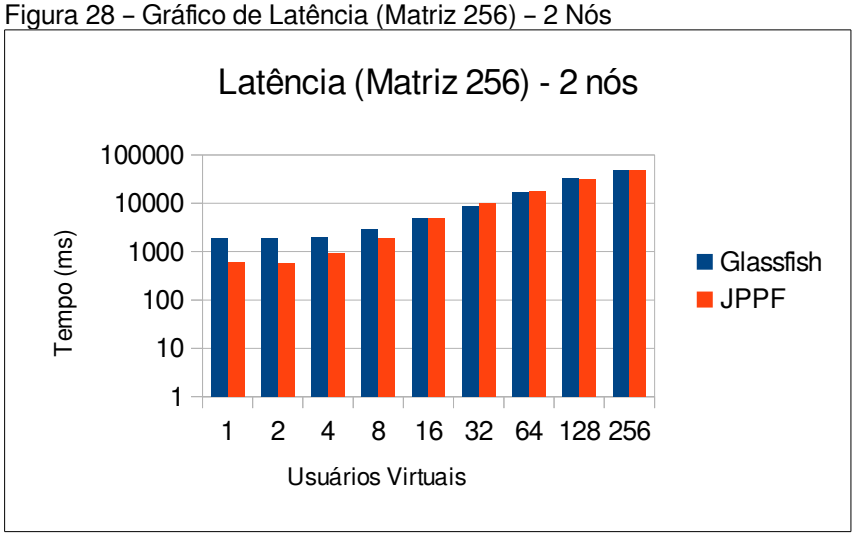
Usuários	Glassfish	JPPF
1	1	1
2	2	1
4	2	1
8	4	1
16	6	5
32	21	20
64	52	52
128	115	116
256	244	243

Fonte: Do Autor.

Na execução da tarefa com matrizes de 128x128, ainda que o tempo de execução nas configurações acima de 32 usuários tenha sido maior no modelo JPPF, este mesmo modelo conseguiu uma performance muito mais próxima do modelo Glassfish. A tabela 5 ainda demonstra o equilíbrio de ambos os modelos no processamento e resposta dos usuários ativos.

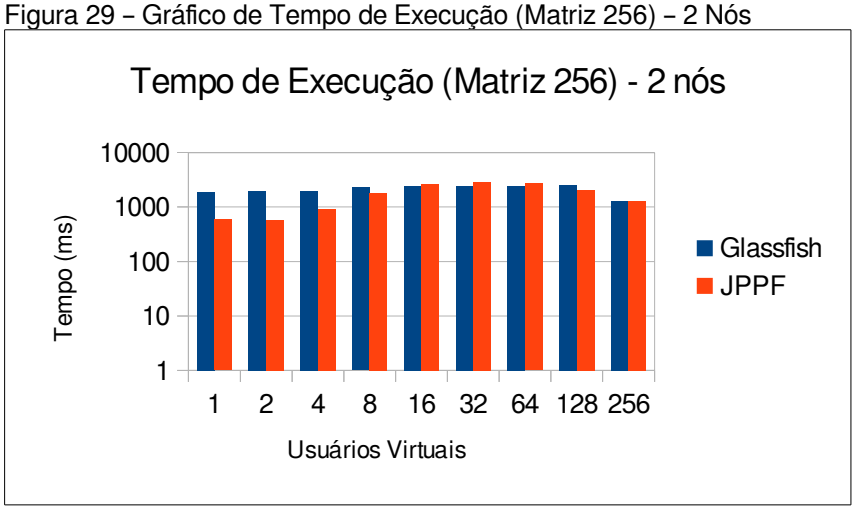
Por fim, no modelo com dois nós, foram realizados testes com matrizes de tamanho 256x256:

Figura 28 – Gráfico de Latência (Matriz 256) – 2 Nós



Fonte: Do Autor.

Figura 29 – Gráfico de Tempo de Execução (Matriz 256) – 2 Nós



Fonte: Do Autor.

Tabela 6 – Tabela de Número Máximo de Usuários (Matriz 256) – 2 Nós

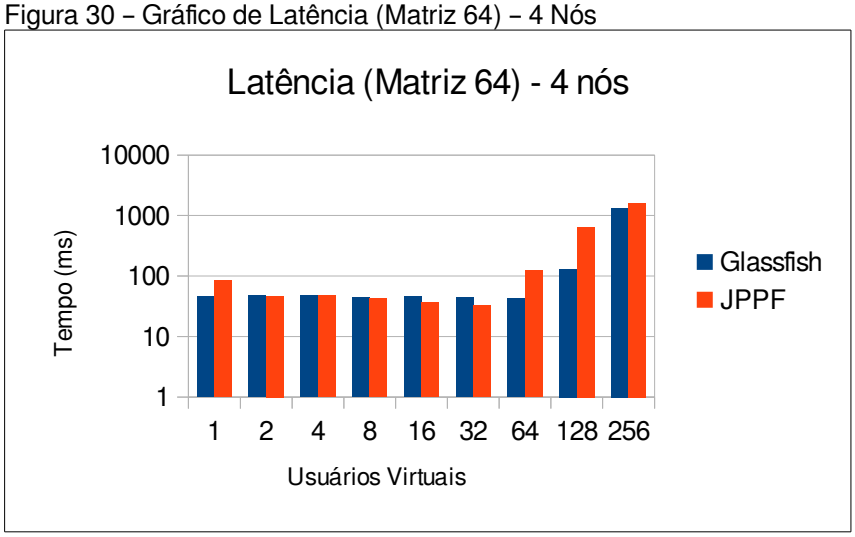
Usuários	Glassfish	JPPF
1	1	1
2	2	2
4	4	3
8	8	7
16	16	15
32	32	31
64	64	63
128	128	127
256	256	255

Fonte: Do Autor.

Nesta configuração, ambos os modelos foram bastante simulares, sendo o modelo JPPF ainda de maior performance nos casos com poucos usuários. Assim como no ambiente com um único nó, diversas requisições não foram correspondidas corretamente em ambos os modelos, no entanto, este problema foi observado somente com cargas de 128 e 256 usuários.

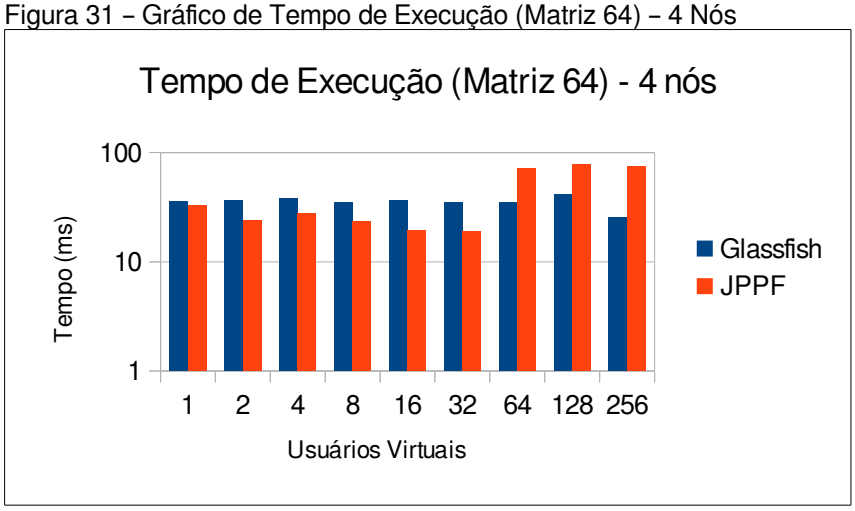
Como último ambiente testado, foram executados modelos com 4 nós de processamento, sendo, no modelo JPPF, um deles implementado juntamente na máquina executante do servidor de aplicação. Seguindo a disposição dos testes, foram realizadas, inicialmente, avaliações com matrizes de 64x64:

Figura 30 – Gráfico de Latência (Matriz 64) – 4 Nós



Fonte: Do Autor.

Figura 31 – Gráfico de Tempo de Execução (Matriz 64) – 4 Nós



Fonte: Do Autor.

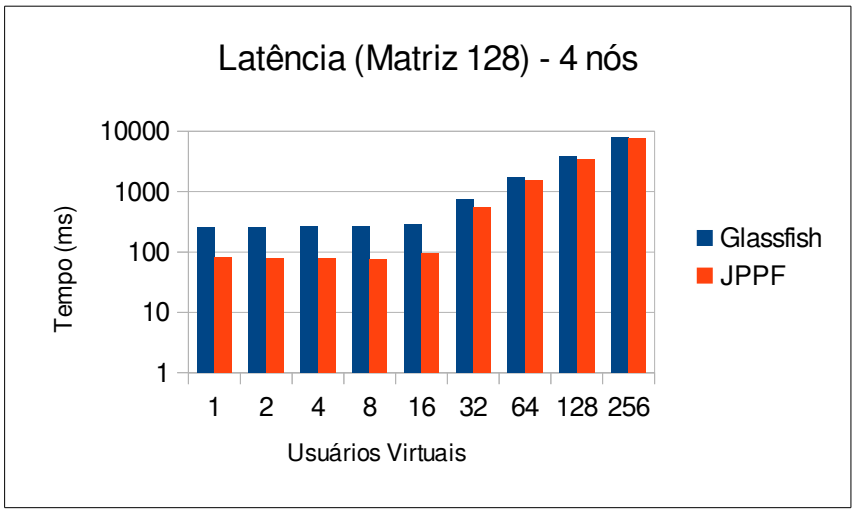
Tabela 7 – Tabela de Número Máximo de Usuários (Matriz 64) – 4 Nós

Usuários	Glassfish	JPPF
1	1	1
2	2	1
4	4	1
8	4	1
16	4	1
32	4	2
64	4	13
128	25	75
256	150	197

Fonte: Do Autor.

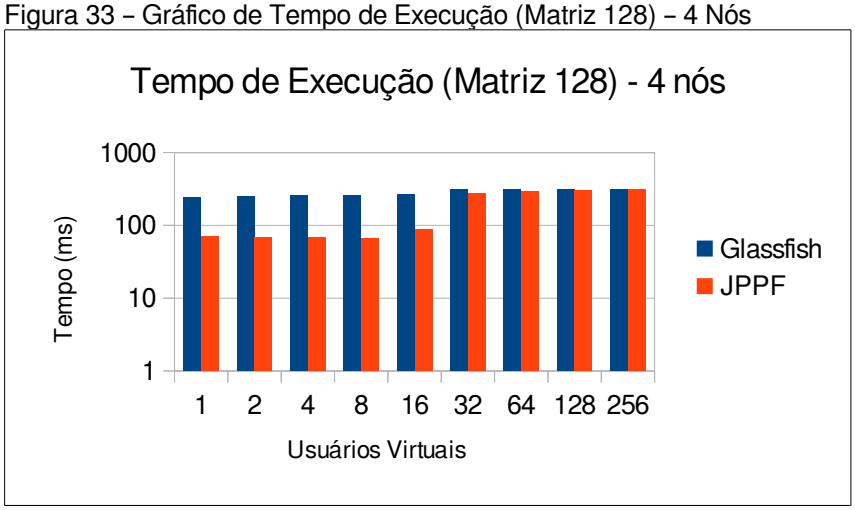
Esta configuração demonstrou os mesmos padrões observados nos ambientes com um e dois nós, não havendo nenhuma particularidade a ser observada. Foram analisados, então, execução de matrizes com cargas de 128x128:

Figura 32 – Gráfico de Latência (Matriz 128) – 4 Nós



Fonte: Do Autor.

Figura 33 – Gráfico de Tempo de Execução (Matriz 128) – 4 Nós



Fonte: Do Autor.

Tabela 8 – Tabela de Número Máximo de Usuários (Matriz 128) – 4 Nós

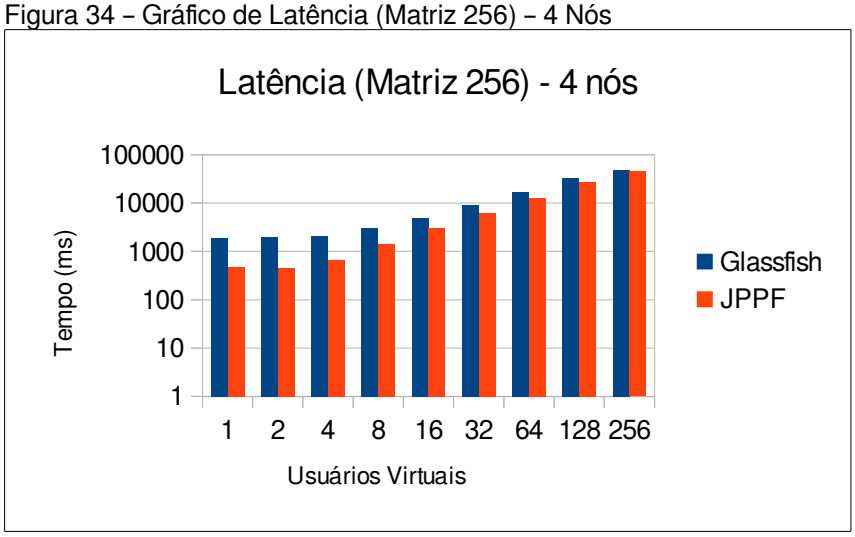
Usuários	Glassfish	JPPF
1	1	1
2	2	1
4	4	1
8	4	1
16	8	2
32	22	17
64	52	49
128	116	113
256	245	240

Fonte: Do Autor.

Ainda que em menor grau nos casos com maior número de usuários, pela primeira vez o modelo JPPF foi integralmente superior em termos de performance ao modelo Glassfish.

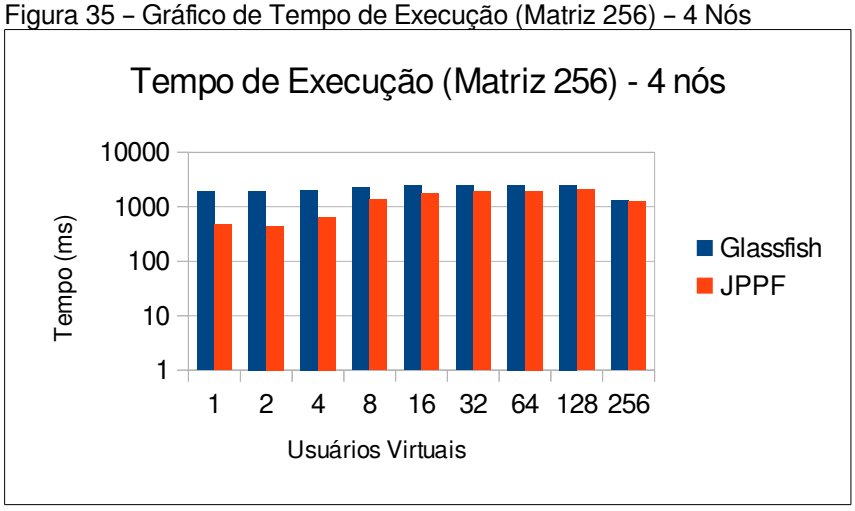
Por fim, foram realizados testes com matrizes de 256x256:

Figura 34 – Gráfico de Latência (Matriz 256) – 4 Nós



Fonte: Do Autor.

Figura 35 – Gráfico de Tempo de Execução (Matriz 256) – 4 Nós



Fonte: Do Autor.

Tabela 9 – Tabela de Número Máximo de Usuários (Matriz 256) – 4 Nós

Usuários	Glassfish	JPPF
1	1	1
2	2	1
4	4	3
8	8	6
16	16	14
32	32	30
64	64	62
128	128	126
256	256	254

Fonte: Do Autor.

Como já havia sido observado com matrizes de tamanho 128x128, o modelo JPPF demonstrou superioridade de desempenho com relação ao modelo Glassfish em todos os casos. Os problemas recorrentes de falha na resposta nos ambientes testados com um e dois nós também foram detectados neste caso, porém, apenas nas experimentações com carga de 256 usuários.

Este problema pode ser explicado pela utilização excessiva de memória pelo servidor, visto que foi observado principalmente com cargas mais altas de usuários e com menor assiduidade a medida que se aumentava o número de nós de processamento. Outro fator importante é de que, como constatado no modelo com um único nó, o modelo com a utilização do JPPF possuiu um número maior de respostas irregulares. Este comportamento também pode ser explicado de forma que a utilização do *framework* amplifica a utilização de memória por parte do servidor.

Outro problema observado durante a execução dos testes foi uma alta carga de tráfego de rede, principalmente nas maiores matrizes, na execução do modelo JPPF. Esta circunstância reflete a leitura, por meio da rede, das matrizes geradas na aplicação cliente pelos nós de processamento, ação necessária para realizar o cálculo corretamente.

Em termos de performance, pode-se constatar a superioridade conquistada pelo modelo JPPF na utilização de cargas pequenas de usuários, realidade que pode ser explicada pela implementação paralela desenvolvida através do framework. Este modelo foi superior também na utilização de quatro nós de processamento. Entretanto,

o modelo de utilização exclusivo com servidores Glassfish demonstrou uma maior consistência no gerenciamento das tarefas e uma maior estabilidade de respostas.

6.2.2 Avaliação das demais características

Além das métricas de performance avaliadas, foram observadas também outras características referentes à instalação e funcionamento dos modelos propostos com a finalidade de demonstrar pontos positivos e negativos destes modelos.

Com relação ao modelo Glassfish, foi observado facilidade na instalação e configuração dos clusters, bem como a instalação e configuração das aplicações nos mesmos. O servidor de aplicação oferece interfaces gráficas web ou linha de comando para o gerenciamento do serviço e inclusão de novos nós ao cluster. No entanto, na configuração realizada, cada máquina necessita de configuração particular e de um servidor SSH rodando, para que o servidor principal possa realizar operações remotas em cada instância do cluster.

Já no modelo que utilizou o *framework* JPPF, demonstrou uma facilidade maior de configuração do cluster, pois a inclusão de novos nós pode ser realizada através da execução do módulo de nós do JPPF. Dessa forma, é possível que novas máquinas sejam adicionadas ao cluster de forma transparente, sem a necessidade de configuração no servidor principal. Por meio do módulo de administração do JPPF foi possível identificar, ainda, uma grande quantidade de tráfego de rede decorrente dos dados enviados e recebidos entre a aplicação cliente e os nós remotos do cluster. Esse tráfego chegou a representar quase 50% do tempo de processamento das tarefas.

De maneira geral, todas as ferramentas utilizadas para a construção dos modelos e execução dos testes possuíam documentação atualizada e disponível publicamente. Sendo assim, ambos os modelos demonstraram viabilidade de implementação e facilidade de configuração, além das questões de desempenho apresentadas anteriormente.

7 CONCLUSÃO

Ao longo desta pesquisa foram estudados e abordados os conceitos relacionados à computação distribuída, sistemas web e a ferramenta JPPF, necessários para o entendimento e desdobramento do trabalho realizado.

Os estudos realizados validaram a importância da utilização de técnicas de programação paralela e distribuída para a construção de aplicações de alto desempenho, com o menor tempo de resposta possível. Da mesma maneira, a importância e a utilização de servidores web também foram apresentados no trabalho destacando-se a complexidade computacional das aplicações construídas para estes ambientes.

A ferramenta JPPF foi apresentada como alternativa para a construção de aplicações web distribuídas, sendo abordado as principais características da ferramenta e sua interface de programação.

Para a construção do modelo de testes, realizou-se o desenvolvimento de uma aplicação com tarefas para geração e multiplicação de matrizes. Para isso, foram produzidos dois métodos distintos: um destinado à execução não paralela, e outro se utilizando de uma implementação baseada na interface de programação JPPF. A aplicação foi desenvolvida na linguagem java, utilizando a especificação JSF.

Foram elaborados dois modelos de clusters: um utilizando o *framework* JPPF para o processamento das tarefas desenvolvidas na aplicação, e outro utilizando apenas um cluster com servidores de aplicação Glassfish. Ambos os modelos foram avaliados utilizando-se da ferramenta JMeter para testes tendo como base a carga de usuários acessando a aplicação, número de nós do cluster e o tamanho das matrizes. Dessa forma, registrou-se os dados de acordo com métricas de performance como latência e tempo de execução das tarefas. A análise dos dados foram realizados através de gráficos comparativos entre os modelos propostos.

Por meio desta análise, pode-se chegar a conclusão de que o modelo com a utilização do JPPF foi superior nos casos onde o número de usuários era reduzido (até 32 usuários). Entretanto, com o aumento do número de nós, este modelo teve um

melhor desempenho até mesmo nos casos com um número maior de usuários (mais que 32 usuários).

O modelo Glassfish, em contra partida, teve um melhor desempenho nas situações com um maior número de usuários, na utilização de um ou dois nós no cluster. Essa diferença foi acentuada nos testes realizados com matrizes de tamanho 64x64.

Ainda que o modelo JPPF tenha apresentado uma performance geral superior ao modelo Glassfish (principalmente nos testes com clusters de 4 nós), boa parte de seu sucesso pode ser atribuída ao modelo de programação paralela sob o qual foram construídos os métodos para multiplicação de matrizes na aplicação. Logo, de forma que os modelos tenham sido melhores do ponto de vista de performance em determinados casos, conclui-se que a utilização de ambos de forma conjunta pode acrescentar desempenho no processamento de tarefas de alta performance.

Outro fator observado durante a instalação e configuração dos modelos foi a facilidade de implementação de um cluster através do *framework* JPPF. Dado que a inclusão de novos nós para processamento de tarefas não é (necessariamente) gerenciada pelo servidor, pode-se sugerir a criação de clusters não dedicados para processamento de tarefas de aplicações web. Esse tipo de construção possibilita que máquinas que permanecem ociosas durante algum tempo em uma determinada rede de computadores possam colaborar com o cluster, anexando o poder computacional destas máquinas no processamento das aplicações de um servidor de aplicação.

De forma a manter o desenvolvimento dos assuntos abordados neste trabalho, sugere-se, para pesquisa, os seguintes trabalhos:

a) realizar um estudo mais aprofundado da utilização da plataforma JPPF em ambiente não-dedicado;
b) realizar um estudo mais aprofundado da clusterização de servidores de aplicação;
c) validar os dados coletados por meio da aplicação de métodos estatísticos.

REFERÊNCIAS

APACHE SOFTWARE FOUNDATION. **Apache JMeter**. 2014. Disponível em: <http://jmeter.apache.org/index.html> Acesso em: 1 jun 2014.

APACHE. **Apache Tomcat**. 2014. Disponível em: <http://tomcat.apache.org/> Acesso em: 4 maio 2014.

AYDIN, Semra; BAY, Omer Faruk. Building a high performance computing clusters to use in computing course applications. **Procedia - Social and Behavioral Sciences**, v.1, n.1, Pages 2396-2401, 2009.

BELAPURKAR, A. et al. **Distributed systems security issues, processes, and solutions**. Hoboken, NJ: John Wiley & Sons, 2009.

BUYYA, R. **High performance cluster computing**: Programming and Applications. Upper Saddle River, N.J: Prentice Hall PTR, 1999.

COHEN, L. **Java Paralel Processing Framework**, 2013. Disponível em: <http://jppf.org>. Acesso em: 28 out. 2013

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas Distribuídos: Conceitos e Projeto**. Traducao João Tortello. Porto Alegre: Bookman, 2007.

DANTAS, M. **Computação distribuída de alto desempenho: redes, clusters e grids computacionais.** Rio de Janeiro: Axcel Books, 2005.

DEITEL, H. M.; DEITEL, P. J.; CHOFFNES, D. R. **Sistemas operacionais**. Traducao Arlete Simillle Marques. 3. ed. São Paulo: Pearson Prentice Hall, 2005.

FLYNN, M. J. Some Computer Organizations and Their Effectiveness. **IEEE Transactions on Computers**, v. C-21, n. 9, p. 948–960, set. 1972.

FORD, N. **Art of Java web development**: Struts, Tapestry, Commons, Velocity, JUnit, Axis, Cocoon, InternetBeans, WebWorks. Greenwich, CT: Manning, 2004.

GRIGORIK, I. **High-performance browser networking**. Sebastopol: O'Reilly Media. 2013.

HADDAD, S. et al. **Distributed systems: design and algorithms**. Hoboken, NJ: Wiley, 2011.

HWANG, K. et al. Designing SSI clusters with hierarchical checkpointing and single I/O space. **IEEE Concurrency**, v. 7, n. 1, p. 60–69, mar. 1999.

ISO/IEC. **Open Distributed Processing Reference Model: Architecture**, 1996. Disponível em: <http://www.joaquin.net/ODP/Part3/toc.html>. Acesso em: 25 out. 2013

KUROSE, J. F.; ROSS, K. W. **Computer networking**: a top-down approach featuring the Internet. 2nd ed ed. Boston: Addison-Wesley, 2003.

LI, Yan Fang; DAS, Paramjit; DOWE, David L. Two decades of Web application testing – A survey of recent advances. **Information Systems**, v. 43, n. 0, p. 20–54, jul. 2014.

MOORE, G. E. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. **IEEE Solid-State Circuits Newsleter**, v. 20, n. 3, p. 33–35, set. 2006.

ORACLE. **GlassFish Server Documentation**. 2013. Disponível em: <http://www.theserverside.com/news/1363671/What-is-an-App-Server> Acesso em: 4 maio 2014.

OTTINGER, Joseph. **What is an App Server?**. 2008. Disponível em: <http://www.theserverside.com/news/1363671/What-is-an-App-Server>. Acesso em: 4 maio 2014.

PEÑA-ORTIZ, R.; GIL, J. A.; SAHUQUILLO, J.; PONT, A. Analyzing web server performance under dynamic user workloads. **Computer Communications**, v. 36, n. 4, p. 386–395, 2013. Acesso em: 10/6/2014.

RED HAT. **What is WildFly?**. 2014. Disponível em: <http://wildfly.org/about/> Acesso em: 4 maio 2014.

ROBISON, R. A. Moore's Law: Predictor and Driver of the Silicon Era. **World Neurosurgery**, v. 78, n. 5, p. 399–403, nov. 2012.

SATO, Jô; MARTINI, João Angelo; GONÇALVES, Ronaldo Augusto Lara. Análise do balanceamento de requisições em clusters Web não-dedicados. **Acta Scientiarum: Technology**, v. 33, n. 4, p. 393, 2011.

SATO, Jô. **Balanceamento de requisições em cluster de servidores web: uma extensão para o mod_proxy_balancer do apache**. 2007. 103 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Estadual de Maringá, Maringá, 2007.

SERGE, H. **Distributed systems:** design and algorithms. Hoboken, NJ: John Wiley & Sons, 2011.

SOUDERS, S. **Even faster web sites**. Sebastopol: O'Reilly, 2009.

STALLINGS, W. **Arquitetura e organização de computadores**. Traducao Daniel Vieira; Ivan Bosnic. 8. ed. São Paulo (SP): Pearson, 2010.

TANENBAUM, A. S. **Organização estruturada de computadores**. Traducao Arlete Simillle Marques. São Paulo: Pearson Prentice Hall, 2007.

TANENBAUM, A. S.; STEEN, M. VAN. **Sistemas distribuídos: princípios e paradigmas**. Traducao Arlete Simillle Marques. 4. ed. São Paulo: Pearson Prentice Hall, 2007.

XIONG, J.; WANG, J.; XU, J. **Research of Distributed Parallel Information Retrieval Based on JPPF.** IEEE, ago. 2010Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5573758>. Acesso em: 28 out. 2013

YOSHIDA, A. MOWS: distributed Web and cache server in Java. **Computer Networks and ISDN Systems**, v. 29, n. 8-13, p. 965–975, 1997.

APÊNDICES

CLUSTERS COMPUTACIONAIS NO PROCESSAMENTO DE TAREFAS DE APLICAÇÕES WEB

Ramon Venson[1], Paulo João Martins[1]

[1]Curso de Ciência da Computação – Universidade do Extremo Sul Catarinense (UNESC) – Criciúma – SC – Brasil

rvenson@riseup.net, pjm@unesc.net

Abstract. *This research set out to examine two distinct models of implementing a cluster for processing a web application: one using a traditional model, through simple clustering of a Glassfish application server; and the other using the resources of a framework for building distributed applications called Java Parallel Processing Framework (JPPF). Both models were analyzed based on performance metrics and management factors and tested in several different configurations, taking into account the number of users, number of nodes in the cluster and workload performed. Based on the achieved results, we observed the main advantages and disadvantages of the proposed models.*

Resumo. *Esta pesquisa se dispôs a analisar dois modelos distintos de implementação de um cluster para o processamento de um aplicativo web: um deles utilizando um modelo tradicional, através da simples clusterização de um servidor de aplicação Glassfish; e o outro utilizando dos recursos de um framework para a construção de aplicações distribuídas denominado JPPF. Ambos os modelos foram analisados com base em métricas de performance e funcionais e experimentados em diversas configurações distintas, levando-se em consideração o número de usuários, número de nós no cluster e carga de trabalho executada. Com base nos resultados atingidos, observou-se as principais vantagens e desvantagens dos modelos propostos.*

1. Introdução

Devido às limitações físicas atingidas no desenvolvimento da capacidade de processamento e ao aumento da complexidade dos algoritmos, o processamento distribuído representa uma resposta para a continuidade no avanço da capacidade de processamento com eficiência e confiabilidade, possibilitando a fragmentação do processamento e do acesso a serviços entre diversos computadores dispersos geograficamente (DANTAS, 2005).

Dentre os vários modelos desenvolvidos dentro da computação distribuída, os chamados aglomerados computacionais representam um agrupamento de computadores ligados por meio de uma rede. Estes ambientes, também conhecidos como clusters ou aglomerados computacionais, podem ultrapassar em muito o poder de máquinas isoladas, sendo possível configurá-lo de forma a permitir uma escalabilidade incremental. Dessa forma, o usuário pode começar com um ambiente modesto e, conforme a necessidade,

expandi-lo apenas adicionando novos dispositivos computacionais ao cluster (STALLINGS, 2010).

Dentre várias ferramentas disponíveis para a criação de ambientes distribuídos, encontra-se Java Parallel Processing Framework (JPPF). Uma ferramenta desenvolvida na linguagem Java e de código fonte aberto que permite aos programadores desenvolver e gerenciar aplicações com abordagem distribuída, provendo uma interface de desenvolvimento de aplicações acompanhada de um ambiente de distribuição de carga (COHEN, 2013).

De acordo com o criador e mantenedor do projeto Laurent Cohen (2013), o framework oferece três diferentes tipos de componentes que se comunicam para formar um ambiente de cluster. O cliente é o responsável pelo ponto de entrada no ambiente, permitindo aos desenvolvedores submeter determinadas tarefas da sua aplicação ao servidor por intermédio de uma interface de programação disponibilizada pelo JPPF. O servidor atua como o ponto central da rede, administrando e repassando as tarefas processadas de volta ao cliente e as não processadas aos nós, que por sua vez processam essas tarefas e retornam o resultado ao servidor.

Esse tipo de construção desempenha o papel de auxiliar no processamento de aplicações de alta complexidade computacional, tornando viável a implementação de algoritmos de alto desempenho que não teriam uma boa performance em uma abordagem não-distribuída ou mesmo utilizando supercomputadores multiprocessados. Além disso, a distribuição de carga de processamento entre sistemas computacionais colabora para o reaproveitamento de recursos físicos considerados obsoletos ou de baixa capacidade, tornando possível a execução dessas aplicações em hardware de baixo custo (AYDIN; BAY, 2009, tradução nossa).

Tão relevante quanto o conceito de utilizar recursos computacionais interconectados para prover processamento é a ideia da World Wide Web (Web). Originalmente desenvolvida para prover páginas estáticas, o progresso da tecnologia permite hoje a construção de aplicações de alta complexidade que podem ser acessadas mundialmente através da Internet por meio de navegadores (FORD, 2004).

Entretanto, a execução de aplicações Web complexas, assim como aplicações desktop, podem gerar um alto custo de processamento, como é o caso de algoritmos de encriptação complexa, renderização de gráficos ou manipulação de imagens. Estas tarefas necessitam de um grande esforço computacional, e podem causar problemas para sistemas que utilizam um servidor para processamento da aplicação que não atenda os requisitos de processamento da mesma (SOUDERS, 2009).

Entre as formas de contornar este problema, pode-se apontar a clusterização de servidores de aplicação Web, que compartilham do objetivo de sustentar as aplicações, oferecendo soluções para balanceamento da carga de trabalho, alta disponibilidade e alta performance. Todavia, não se desconsidera completamente a utilização de plataformas como o JPPF, onde a distribuição do processamento pode dar-se apenas em tarefas específicas da aplicação. Além disso, essa plataforma fornece uma maior flexibilidade na configuração de hardware dos dispositivos computacionais integrantes do cluster, encorajando um modelo de baixo custo financeiro.

Sendo assim, a proposta deste trabalho é explorar e analisar a utilização de modelos de plataformas de clusters computacionais no processamento de aplicações web. Para isso, pretende-se realizar a execução de uma aplicação com tarefas de alta carga de processamento em um servidor de aplicação configurado em cluster, comparando os resultados obtidos com outro modelo, baseado na plataforma JPPF para a execução apenas das tarefas de maior complexidade.

2. Justificativa

Os sistemas distribuídos consistem em um conjunto de computadores independentes que se apresentam ao usuário de maneira única e coerente. Essa definição não abrange o tipo de arquitetura dos computadores nem tampouco o meio pelo qual são interligados (TANEBAUM; STEEN; 2008).

Segundo Tanebaum e Steen (2008), os sistemas de computação distribuídos e aglomerados computacionais fazem parte dessa definição abrangente, sendo a última utilizada para executar um determinado programa em paralelo em diversas máquinas. Para Dantas (2005), um cluster computacional pode ser desenvolvido com máquinas de configurações distintas e relativamente pequenas, sendo a escalabilidade um fator de grande importância para que o desempenho geral do cluster cresça à medida que mais recursos se fazem necessários.

Uma das tecnologias distribuídas de maior sucesso no mundo é a Web, por onde circulam uma infinidade de conteúdos por meio de serviços interconectados. A web representa a plataforma de implementação mais comum disponível para desenvolvedores, atualmente: encontra-se em cada smarthphone, tablet, laptop, desktop ou outros dispositivos do meio. Projeções de crescimento revelam que, até 2020, cerca de 20 bilhões de dispositivos estejam conectados à Web através de navegadores, demonstrando a importância desse seguimento no desenvolvimento de novas aplicações (GRIGORIK, 2013).

Inicialmente, os sistemas projetados para a web eram escritos nas linguagens C e Perl e apresentavam basicamente páginas conectadas por hipertexto e estáticas. Com o tempo, novas tecnologias foram desenvolvidas com o intuito de gerar conteúdo dinâmico para a Web e, entre elas, destaca-se a linguagem java. Desenvolvida a princípio para a construção de aplicações tradicionais (paradigma desktop) e applets1, a linguagem rapidamente demonstrou potencial para o desenvolvimento Web e distribuído, primeiro através dos servlets2 e posteriormente com Java Server Pages3 (JSF) (FORD, 2004).

Neste contexto, surgem aplicações de maior complexidade que despendem de maiores recursos computacionais, fazendo-se necessário a composição de novos modelos de implementação para garantir a escalabilidade e performance do sistema. Um dos meios utilizados para contornar problemas de escalabilidade é a clusterização de um servidor de aplicação. Alguns servidores oferecem ferramentas para facilitar implementação de um cluster com o objetivo de melhorar o desempenho e a disponibilidade das aplicações alojadas. Entretanto, ao utilizar este recurso, o gerenciamento da distribuição de tarefas passa a ser realizado diretamente pelo servidor, sem a interferência do programador (ORACLE CORPORATION, 2002; RED HAT, 2014).

Permitindo também a construção de redes de processamento distribuído, o projeto Java Parallel Processing Framework utiliza a máquina virtual Java para oferecer uma interface de desenvolvimento distribuído voltada para a construção de aplicações que requerem uma grande quantidade de processamento. O JPPF permite dividir uma aplicação em pequenas partes que podem ser executadas simultaneamente em diferentes máquinas, por meio de chamadas de procedimento remoto. Uma das características de maior destaque nesta plataforma é sua facilidade de implementação e flexibilidade para criação de clusters com hardware de baixo custo e não-dedicado (COHEN, 2013).

Dado que uma aplicação Web construída na plataforma Java também pode utilizar dos recursos da plataforma JPPF, esta foi escolhida por ser uma implementação de código fonte aberto e que propicia facilidade de implementação de tarefas específicas da aplicação, em oposição à clusterização total do sistema por meio do servidor de aplicação. Dessa forma, propõe-se uma comparação entre os dois modelos de execução distribuída de uma aplicação Web de alta performance, com a finalidade de investigar e demonstrar as vantagens e desvantagens da utilização de ambos os modelos.

3. Metodologia

A metodologia de desenvolvimento do trabalho consistiu inicialmente na elaboração de dois modelos de cluster para a execução de aplicações web baseadas na plataforma JEE e no desenvolvimento de uma aplicação protótipo para a realização dos testes de performance. Para a realização dos testes, foi utilizado como ferramenta o software JMeter, pelo qual realizou-se a simulação de diferentes números de usuários e cargas de trabalho e a posterior coleta dos dados para análise.

3.1. Arquiteturas propostas para avaliação

Foram elaborados dois modelos de clusters para a execução de aplicações web de forma distribuída.

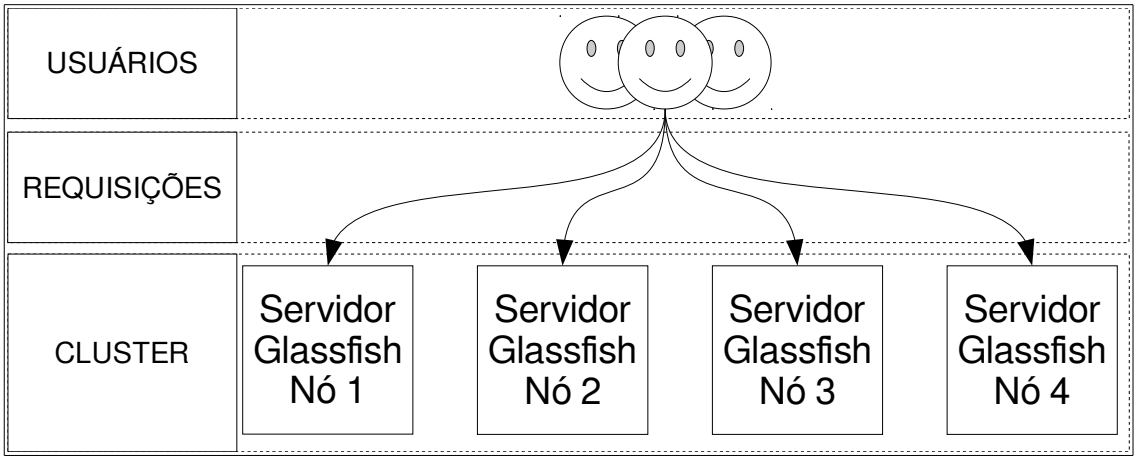


Figura 1. Modelo de arquitetura com cluster Glassfish

No primeiro modelo, utilizando o Glassfish Server Open Source Edition, utilizou-se do suporte para a clusterização de servidores web disponível na ferramenta para a criação do ambiente de execução. Foram determinadas três cenários diferentes neste modelo: um único servidor, dois servidores clusterizados e quatro servidores clusterizados.

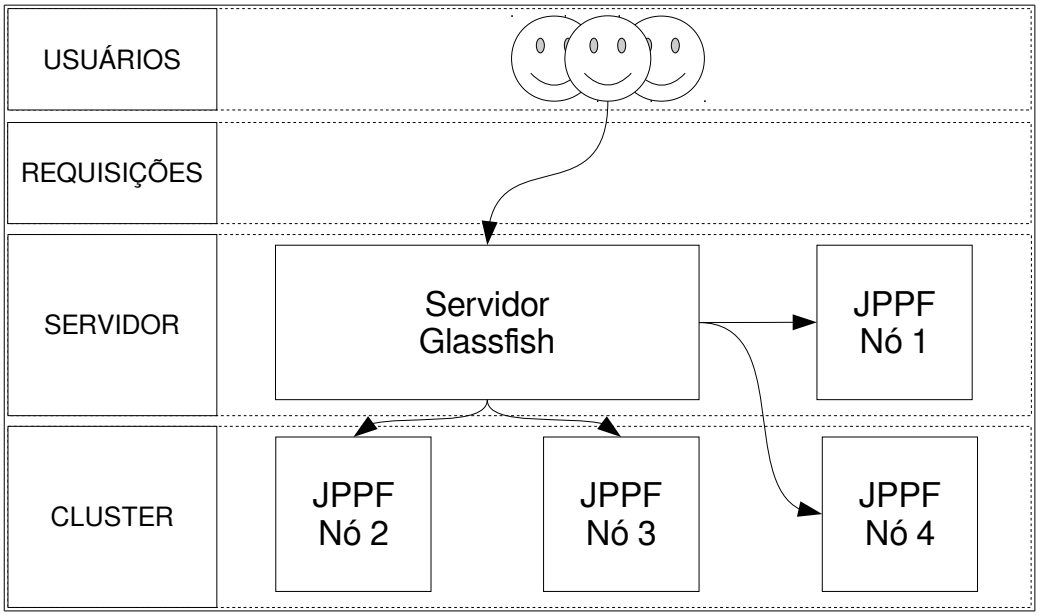


Figura 2. Modelo de arquitetura com cluster JPPF

No segundo modelo, utilizou-se de um único servidor Glassfish para realizar a execução da aplicação. Entretanto, o processamento das tarefas implementadas na aplicação ficou ao encargo dos nós de um cluster de computadores construído por intermédio do framework JPPF. Da mesma forma que o primeiro modelo, foram determinadas três cenários diferentes neste modelo: um único nó, dois nós e quatro nós. Um dos nós do cluster foi executado na mesma máquina do servidor de aplicação Glassfish com o intuito de diminuir o tráfego de rede nas tarefas de menor carga de trabalho.

3.2. Desenvolvimento da Aplicação

A aplicação protótipo desenvolvida para a realização dos testes foi feita com base na especificação JEE e utilização do framework JSP, que auxiliou no desenvolvimento e organização da aplicação web.

A tarefa escolhida para implementação foi a de multiplicação de matrizes, em sua forma intuitiva, devido à sua alta complexidade ($O(n^3)$) e perspectiva de paralelização do método. A execução da multiplicação é realizada através da soma dos produtos da linha da matriz A pela coluna da matriz B.

Uma classe denominada Matrix foi implementada para a geração aleatória de três matrizes quadráticas de um tamanho previamente informado, sendo as duas primeiras (matrixA e matrixB) os alvos da multiplicação, e a terceira (matrixC) o resultado dessa multiplicação. No entanto, foram implementados dois métodos distintos para a multiplicação das matrizes.

O primeiro método tinha como propósito a arquitetura de clusters somente com o servidor Glassfish. Não foram utilizadas neste método nenhuma técnica de programação paralela ou distribuída, de forma que a paralelização da tarefa ficasse por conta apenas do número de usuários realizando a requisição da tarefa.

O segundo método utilizou-se dos recursos da biblioteca JPPF para a construção de um modelo de processamento paralelizado e distribuído da tarefa de multiplicação de matrizes. Esse método foi implementado de forma que cada requisição efetuada por um usuário, gerasse um novo job composto por um número de tasks igual ao tamanho de cada matriz. Cada uma destas tasks recebia como parâmetro uma das linhas da matrixA, a ser multiplicada pelas colunas da matrixB, que pela natureza do processo de multiplicação, foi disponibilizada a cada task através do recurso DataProvider do JPPF, onde cada uma das tarefas possuía acesso de leitura à matriz referente através da rede. O resultado da multiplicação de cada linha era retornado para ser posteriormente agrupado na matriz de resposta matrixC.

Para gerenciar a conexão da aplicação com o servidor JPPF, foi implementada uma classe denominada JPPFController. Essa classe era responsável por realizar e estabelecer a conexão cliente/servidor apenas uma única vez, de modo a reduzir a carga de inicialização e execução das tarefas a cada requisição de um usuário à aplicação.

A interface gráfica da aplicação foi desenvolvida através de uma única página eXtensible HyperText Markup Language (XHML) com botões para acesso aos métodos citados durante a execução do script de testes, não sendo o escopo principal da implementação. As requisições realizadas através desta interface eram repassadas para a classe PrototipoMB, responsável pela execução principal da aplicação e pela coleta dos tempos de execução dos métodos chamados.

3.3. Roteiro de testes com JMeter

Para o desenvolvimento do roteiro dos testes, foi utilizado a ferramenta Apache JMeter, utilizada para testes de carga e performance com foco em aplicações web. O JMeter é um software desktop e de código fonte aberto e permite a criação de scripts (roteiros) com suporte multithread, onde é possível simular o acesso de um grande número de usuários virtuais realizando o acesso simultâneo a uma determinada aplicação (APACHE SOFTWARE FOUNDATION, 2014).

O primeiro componente adicionado ao plano de testes foi o Thread Group. Através desse componente é possível definir um número de threads (análogo ao conceito de usuários virtuais) que realizarão a execução das funções definidas no script. Dentro deste componente, foi inserido o componente Only Once Controller e dois componentes para realizar requisições HTTP: o First Request e o Button Request.

O First Request era responsável pelo acesso inicial à aplicação, onde, através do XPath Extractor, era possível guardar o atributo de ViewState de cada acesso, vinculado-o a cada usuário virtual. O Button Request era o componente efetivamente responsável pela requisição dos métodos implementados na aplicação por meio dos botões presentes na interface. A resposta do servidor à requisição deste componente continha o tempo de execução do método, sendo este extraído e incluído na tabela de resultados gerada pelo JMeter.

Por fim, os componentes HTTP URL Re-writing Modifier e HTTP Cookie Manager realizavam o armazenamento dos identificadores de sessão (JSESSIONID) e cookies.

Para a leitura dos dados, foi integrado ao script o componente Simple Data Writer, agente responsável por escrever em um arquivo os dados gerados pelo teste. Foi incluso neste arquivo a variável executionTime, extraída anteriormente pelo Button Request.

Para a execução dos testes no servidor clusterizado com Glassfish, criou-se quatro threadGroups distintos e o número total de usuários avaliado foi dividido igualmente entre todas as instâncias. Esse fato foi necessário dado que não houve a configuração de nenhuma ferramenta responsável pela distribuição de requisições entre os servidores Glassfish, circunstância não observada no cluster JPPF devido ao seu balanceamento de carga automatizado. Dessa forma, garantiu-se que todos os servidores receberiam exatamente o mesmo número de requisições de usuários.

3.4. Infraestrutura utilizada

Na realização dos testes, foram utilizados cinco computadores de arquitetura PC 64bits, equipados com processadores Intel I3 e quatro gigabytes de memória, localizados no Laboratório de Computação Distribuída do curso de Ciência da Computação da UNESC.

Os testes foram realizados em uma rede cabeada ethernet, de velocidade gigabit e por meio de um switch dedicado exclusivamente para o cluster.

3.5. Desenvolvimento dos testes

O roteiro executado pela ferramenta JMeter foi processado através de uma máquina adicional presente na mesma rede do ambiente de execução, sendo esta, responsável exclusivamente pela execução do roteiro e geração dos resultados.

O tamanho das matrizes foram determinados em três situações diferentes: matrizes 64x64, matrizes 128x128 e matrizes 256x256. Estes valores foram selecionados com o propósito de garantir consistência nos tempos retornados e diminuir a alta utilização de memória pela aplicação. O número de usuários seguiu uma lógica parecida, sendo determinado os valores de 1 a 256, com crescimento exponencial (1, 2, 4, 8, 16, 32, 64, 128, 256).

O plano de testes foi executado levando-se em conta:

a) os seis cenários elaborados (três cenários para o cluster Glassfish e três para o cluster JPPF);

b) três cargas de trabalho (tamanho das matrizes) distintas (64, 128 e 256);

c) e nove grupos de usuários (1, 2, 4, 8, 16, 32, 64, 128, 256).

Assim sendo, foram realizados 162 testes de configurações distintas, onde, cada teste, foi repetido 3 vezes com a finalidade de aprimorar a consistência dos dados coletados.

Dentre os problemas analisados durante o andamento dos testes, pode-se relatar a grande quantidade de requisições não respondidas pelo servidor nos casos onde houve grande quantidade de usuários (como 256 e 128) e matrizes de grande porte (256).

Após a execução dos testes, os dados coletados foram organizados de forma a extrair três métricas para a avaliação. As métricas escolhidas foram:

a) latência: refere-se ao tempo total entre o envio da requisição e o recebimento da resposta pelo usuário. Neste caso, a latência do componente First Request foram somados à latência total;

b) tempo de execução: tempo retornado pela resposta componente Button Request, equivalente ao tempo total de execução do método de multiplicação de matriz;

c) número máximo de usuários ativos: número máximo de usuários (threads) processadas pelo servidor simultaneamente.

Todos os dados foram analisados e separados por tamanho da matriz utilizada e do número de nós do cluster JPPF e Glassfish.

4. Resultados

Em termos de performance, pode-se constatar a superioridade conquistada pelo modelo JPPF na utilização de cargas pequenas de usuários, realidade que pode ser explicada pela implementação paralela desenvolvida através do framework. Este modelo foi superior também na utilização de quatro nós de processamento. Entretanto, o modelo de utilização exclusivo com servidores Glassfish demonstrou uma maior consistência no gerenciamento das tarefas e uma maior estabilidade de respostas.

4.2. Avaliação das demais características

Além das métricas de performance avaliadas, foram observadas também outras características referentes à instalação e funcionamento dos modelos propostos com a finalidade de demonstrar pontos positivos e negativos destes modelos.

Com relação ao modelo Glassfish, foi observado facilidade na instalação e configuração dos clusters, bem como a instalação e configuração das aplicações nos mesmos. O servidor de aplicação oferece interfaces gráficas web ou linha de comando para o gerenciamento do serviço e inclusão de novos nós ao cluster. No entanto, na configuração realizada, cada máquina necessita de configuração particular e de um servidor SSH rodando, para que o servidor principal possa realizar operações remotas em cada instância do cluster.

Já no modelo que utilizou o framework JPPF, demonstrou uma facilidade maior de configuração do cluster, pois a inclusão de novos nós pode ser realizada através da execução do módulo de nós do JPPF. Dessa forma, é possível que novas máquinas sejam adicionadas ao cluster de forma transparente, sem a necessidade de configuração no servidor principal. Por meio do módulo de administração do JPPF foi possível identificar, ainda, uma grande quantidade de tráfego de rede decorrente dos dados enviados e recebidos entre a aplicação cliente e os nós remotos do cluster. Esse tráfego chegou a representar quase 50% do tempo de processamento das tarefas.

De maneira geral, todas as ferramentas utilizadas para a construção dos modelos e execução dos testes possuíam documentação atualizada e disponível publicamente. Sendo assim, ambos os modelos demonstraram viabilidade de implementação e facilidade de configuração, além das questões de desempenho apresentadas anteriormente.

5. Conclusão

Ao longo desta pesquisa foram estudados e abordados os conceitos relacionados à computação distribuída, sistemas web e a ferramenta JPPF, necessários para o entendimento e desdobramento do trabalho realizado.

Os estudos realizados validaram a importância da utilização de técnicas de programação paralela e distribuída para a construção de aplicações de alto desempenho, com o menor tempo de resposta possível. Da mesma maneira, a importância e a utilização de servidores web também foram apresentados no trabalho destacando-se a complexidade computacional das aplicações construídas para estes ambientes.

A ferramenta JPPF foi apresentada como alternativa para a construção de aplicações web distribuídas, sendo abordado as principais características da ferramenta e sua interface de programação.

Para a construção do modelo de testes, realizou-se o desenvolvimento de uma aplicação com tarefas para geração e multiplicação de matrizes. Para isso, foram produzidos dois métodos distintos: um destinado à execução não paralela, e outro se utilizando de uma implementação baseada na interface de programação JPPF. A aplicação foi desenvolvida na linguagem java, utilizando a especificação JSF.

Foram elaborados dois modelos de clusters: um utilizando o framework JPPF para o processamento das tarefas desenvolvidas na aplicação, e outro utilizando apenas um cluster com servidores de aplicação Glassfish. Ambos os modelos foram avaliados utilizando-se da ferramenta JMeter para testes tendo como base a carga de usuários acessando a aplicação, número de nós do cluster e o tamanho das matrizes. Dessa forma, registrou-se os dados de acordo com métricas de performance como latência e tempo de execução das tarefas. A análise dos dados foram realizados através de gráficos comparativos entre os modelos propostos.

Por meio desta análise, pode-se chegar a conclusão de que o modelo com a utilização do JPPF foi superior nos casos onde o número de usuários era reduzido (até 32 usuários). Entretanto, com o aumento do número de nós, este modelo teve um melhor desempenho até mesmo nos casos com um número maior de usuários (mais que 32 usuários).

O modelo Glassfish, em contra partida, teve um melhor desempenho nas situações com um maior número de usuários, na utilização de um ou dois nós no cluster. Essa diferença foi acentuada nos testes realizados com matrizes de tamanho 64x64.

Ainda que o modelo JPPF tenha apresentado uma performance geral superior ao modelo Glassfish (principalmente nos testes com clusters de 4 nós), boa parte de seu sucesso pode ser atribuída ao modelo de programação paralela sob o qual foram construídos os métodos para multiplicação de matrizes na aplicação. Logo, de forma que os modelos tenham sido melhores do ponto de vista de performance em determinados casos, conclui-se que a utilização de ambos de forma conjunta pode acrescentar desempenho no processamento de tarefas de alta performance.

inclusão de novos nós para processamento de tarefas não é (necessariamente) gerenciada pelo servidor, pode-se sugerir a criação de clusters não dedicados para processamento de tarefas de aplicações web. Esse tipo de construção possibilita que máquinas que permanecem ociosas durante algum tempo em uma determinada rede de computadores possam colaborar com o cluster, anexando o poder computacional destas máquinas no processamento das aplicações de um servidor de aplicação.

Referências

AYDIN, Semra; BAY, Omer Faruk. Building a high performance computing clusters to use in computing course applications. Procedia - Social and Behavioral Sciences, v.1, n.1, Pages 2396-2401, 2009.

COHEN, L. Java Paralel Processing Framework, 2013. Disponível em: <http://jppf.org>. Acesso em: 28 out. 2013.

DANTAS, M. Computação distribuída de alto desempenho: redes, clusters e grids computacionais. Rio de Janeiro: Axcel Books, 2005.

FORD, N. Art of Java web development: Struts, Tapestry, Commons, Velocity, JUnit, Axis, Cocoon, InternetBeans, WebWorks. Greenwich, CT: Manning, 2004.

GRIGORIK, I. High-performance browser networking. Sebastopol: O'Reilly Media. 2013.

ORACLE. GlassFish Server Documentation. 2013. Disponível em: <http://www.theserverside.com/news/1363671/What-is-an-App-Server> Acesso em: 4 maio 2014.

RED HAT. What is WildFly?. 2014. Disponível em: <http://wildfly.org/about/> Acesso em: 4 maio 2014.

SOUDERS, S. Even faster web sites. Sebastopol: O'Reilly, 2009.

STALLINGS, W. Arquitetura e organização de computadores. Traducao Daniel Vieira; Ivan Bosnic. 8. ed. São Paulo (SP): Pearson, 2010

TANENBAUM, A. S.; STEEN, M. VAN. Sistemas distribuídos: princípios e paradigmas. Traducao Arlete Simillle Marques. 4. ed. São Paulo: Pearson Prentice Hall, 2007.