ULBRA
CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Recredenciado pela Portaria Ministerial nº 3.607 - D.O.U. nº 202 de 20/10/2005

Daniel Álvares Montes

Balanceamento de carga em Aplicações *Web*: estudo de caso *Tomcat*

Palmas
2013

Daniel Álvares Montes

Balanceamento de carga em Aplicações *Web*: estudo de caso *Tomcat*

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA), orientado pelo Professor Mestre Madianita Bogo Marioti.

Palmas
2013

Daniel Álvares Montes

Balanceamento de carga em Aplicações *Web*: estudo de caso *Tomcat*

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA), orientado pelo Professor Mestre Madianita Bogo Marioti.

Aprovada em junho de 2013.

BANCA EXAMINADORA

_____

Prof. M.Sc. Madianita Bogo Marioti

Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Fernando Luiz de Oliveira

Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Jackson Gomes

Centro Universitário Luterano de Palmas

Palmas
2013

Resumo

Com o grande crescimento da *Web*, sua popularização e sua grande importância como meio de comunicação global, cada vez mais a atenção se volta para questões de desempenho e disponibilidade de aplicações *Web*. Dessa forma, o objetivo deste trabalho foi construir um *cluster* Tomcat para balanceamento de carga de aplicações *Web*, visando aumentar o desempenho e a disponibilidade. Como resultado, é apresentado um comparativo entre dois cenários, um sem a utilização do *cluster* e outro com a utilização do *cluster* e, além disso, uma explanação sobre as vantagens de se utilizar *cluster* de balanceamento de carga para aplicações *Web*.

LISTA DE FIGURAS

LISTA DE TABELA

SUMÁRIO

1. INTRODUÇÃO

O ano de 2000 foi marcado pela euforia com crescimento exponencial do uso da internet, de forma que as empresas se apressaram em garantir seu espaço na *Web*. Garantir espaço no mundo virtual era primordial e necessário para as empresas, por vários motivos, mas, o fator determinante refere-se ao alcance a um número muito grande de usuários.

Consequentemente, com o grande número de usuários, há também maior quantidade de acessos às aplicações das empresas e, com isso, a necessidade de garantir a disponibilidade e a rapidez do acesso aos serviços oferecidos através dessas aplicações.

O aumento de forma exponencial do tráfego na internet traz, consequentemente, o crescimento da taxa de requisições sobre as aplicações disponibilizadas pelas empresas, o que pode ocasionar demora nas respostas ou até congestionamentos, quando são solicitadas inúmeras requisições ao mesmo tempo.

Hoje em dia não é tolerável para empresas que querem ter uma opinião favorável de seus clientes, disponibilizar um serviço na *Web* sem que o mesmo seja rápido e consiga suportar picos de acessos que venham a surgir.

Devido a isso, um dos grandes desafios enfrentados pelos responsáveis por administrar os serviços disponíveis na *Web* é realizar o dimensionamento do tráfego e fazer uma estimativa de evolução dos acessos. Assim, é possível implantar uma estrutura que venha a atender os requisitos fundamentais de bom desempenho, escalabilidade e disponibilidade dos serviços, utilizando os recursos computacionais para suprir a demanda e disponibilizar o serviço de forma eficiente aos usuários.

Uma forma de fazer isso é utilizando *cluster*, que é um conjunto de vários computadores conectados e que dividem as tarefas entre si como se fossem um único computador. Uma forma de construir um *cluster* é utilizando o *Apache Tomcat,* que é uma ferramenta que dá suporte a configuração e manutenção de *clusters* de computadores.

Nesse contexto, o objetivo desse trabalho foi implementar um *cluster Apache Tomcat* para realizar balanceamento de carga, visando a garantir um melhor

desempenho e disponibilidade de serviços *Web*. Para isso, foram realizados e são apresentados nesse trabalho os seguintes procedimentos:

- Implantar um *cluster* de balanceamento de carga utilizando o *Apache Tomcat*;
- Realizar testes de desempenho e disponibilidade sobre uma aplicação *Web*, com e sem o uso do *cluster*;
- Descrever resultados de testes de desempenho e disponibilidade de uma aplicação *Web* executando em um servidor único;
- Descrever os resultados dos testes com o *cluster* estruturado e em funcionamento;
- Apresentar um paralelo entre o servidor único e o *cluster*, fazendo uma análise nos quesitos: desempenho e disponibilidade.

Este trabalho tem como principal objetivo criar um cluster de balanceamento de carga para aplicação Web, visando obter melhoria no desempenho e na disponibilidade do serviço. A tecnologia utilizada para a construção do cluster será Apache Tomcat.

Este trabalho está sequenciado da seguinte forma: Capítulo 2 – Referencial Teórico; Capítulo 3 – Materiais e Métodos; Capítulo 4 – Resultados e Discussões; Capítulo 5 – Considerações Finais e Capítulo 6 – Referências Bibliográficas.

2. REFERENCIAL TEÓRICO

Nesta seção são abordados os conceitos de arquiteturas paralelas e *clusters,* fundamentais para a compreensão e execução do projeto. São apresentados os diferentes tipos de *clusters*, com uma abordagem mais ampla em *cluster* de balanceamento de carga, pois este trabalho tem como foco o balanceamento de carga de aplicações *Web*. Por fim, será descrito o *Apache Tomcat*, que é a tecnologia utilizada nesse trabalho para a construção do *cluster*.

2.1 Arquiteturas Paralelas

Com o avanço da tecnologia os computadores estão ficando cada vez mais velozes, porém, as "exigências sobre eles crescem no mínimo mais rápido do que a velocidade de operação" (TANENBAUM, 2001, pg. 45), por isso a utilização de arquiteturas paralelas é cada vez mais interessante.

Uma das características deste tipo de arquitetura é a utilização de várias unidades para realizar processamento, ou seja, ao invés de se utilizar um único computador para realizar o processamento das requisições solicitadas, utilizam-se dois ou mais computadores interligados de forma paralela, assim, a execução de uma tarefa pode ser dividida por mais de um processador/computador.

As arquiteturas paralelas trazem algumas vantagens, entre elas: maior desempenho, confiabilidade e escalabilidade (OLIVEIRA, 2001, pg. 37). Consegue-se maior desempenho com uma arquitetura paralela se a mesma for comparada a uma aplicação que seja fornecida por um único servidor, pois ao se ter o paralelismo as tarefas são realizadas de forma compartilhada entre dois servidores ou mais. A confiabilidade é garantida por acontecer a replicação de dados críticos da aplicação ou de funções, em vários discos. Por fim, a escalabilidade torna possível a adição de novos servidores no sistema paralelo também a adição de novos recursos ao sistema computacional, de acordo com as necessidades, que são vistas através do aumento de usuários e consequentemente o número de acessos ao serviço.

Existe, também, uma motivação financeira para utilizar essas arquiteturas, pois podem ser usados computadores comuns. Quando se fala em “comuns” é que não têm a mesma arquitetura de servidores, estes são interligados por uma rede de comunicação, para que seja feita a divisão de tarefas entre eles, ou seja, o sistema paralelo é capaz de dividir o programa em partes e atribuí-las a diferentes

processadores. Outros pontos importantes são a comunicação e o sincronismo entre os processos para que não se perca informações e nem o sincronismo das ações. Nada impede também que nesses *cluster*s possam ser utilizados servidores de alta capacidade.

Os dados e as instruções exigidas na execução das tarefas da aplicação são repassados na comunicação que ocorre entre os computadores interligados. De acordo com a unicidade ou multiplicidade de dados e instruções, foi proposta a taxonomia de Flynn que apresenta quatro tipos de arquiteturas, no qual seguem as descrições (ROQUE, 2012, online):

- SISD (*Single Instruction Single Data*): essa é uma arquitetura de um computador utilizado por muitos, ou seja, um computador comum, neste tipo de arquitetura só é possível realizar uma instrução no conjunto de dados por vez.
- SIMD (*Single Instruction Multiple Data*): esse é um dos tipos de computação paralela, nele uma única instrução recebe vários conjuntos de dados, a instrução é executada em um coprocessador de um sistema.
- MISD (*Multiple Instruction Single Data*): nesse tipo, várias instruções executam sobre um único conjunto de dados. Não existe uma máquina que seja contemplada com esse tipo de arquitetura, sua ideia de várias instruções sobre um conjunto de dados é algo ainda difícil de imaginar.
- MIMD (*Multiple Instruction Multiple Data*): neste tipo de arquitetura, várias instruções executam sobre vários tipos de dados, na qual se enquadram a maioria das máquinas paralelas, tanto com multiprocessadores quanto com sistemas de vários computadores como os *Clusters*. Máquinas usando MIMD têm vários processadores que funcionam de maneira assíncrona e de forma independente.

A Figura 1 apresenta a relação entre as instruções executadas em paralelo e os dados tratados em paralelo das arquiteturas da Taxonomia de Flynn, ou seja, SISD, SIMD, MISD, MIMD.

Figura 1 - Taxonomia de Flynn - (ROQUE, 2012, online)

Conforme ilustra a Figura 1, a arquitetura SISD corresponde ao computador convencional, no qual ocorre a relação é de 1 para 1, uma única instrução para um único fluxo de dados. Já na arquitetura SIMD a relação que ocorre é de 1 para múltiplas, uma única instrução para múltiplos fluxo de dados. No MISD a relação é de múltiplas para 1, múltiplas instruções executadas em paralelo para um único fluxo de dados, arquitetura para a qual, como já informado, não existem computadores. Por fim, na arquitetura MIMD, que a relação é de múltiplas para múltiplas, executando múltiplas instruções para tratar múltiplos dados.

Como este trabalho tem como objetivo a construção de um *cluster,* que é um tipo de arquitetura MIMD, de maneira específica o *cluster* para balanceamento de carga *Web*, este tema será abordado mais detalhadamente na próxima seção.

2.2 *Cluster*

Segundo *Sterling e Hawick et al. (*1999*, online) cluster* é um sistema de computação que possui vários computadores, conhecidos como nós, interligados por uma rede local, em que, normalmente, os nós são gerenciados por uma máquina, que é o nó que gerencia o *cluster.* Ou seja, um *Cluster* de computadores nada mais é do que um aglomerado de computadores conectados em rede local, que se comunicam e trabalham de maneira integrada para realizar uma determinada atividade de forma que pareça que a execução das tarefas esteja sendo realizada em uma única máquina.

Segundo Gorino (2006, online) os *cluster*s são projetados para garantir alta disponibilidade dos nós e alto desempenho no processamento das tarefas, oferecendo maior acesso aos canais de E/S por estes canais poderem ser compartilhados mesmo que fisicamente. São exemplos comuns de dispositivos de E/S mouse, teclado etc. Os dispositivos de E/S (Entrada/Saída) têm como funções básicas a comunicação do usuário com o computador, a comunicação do computador com o meio ambiente e armazenamento (gravação) de dados. Porém, a utilização dos *cluster*s não se restringe à busca de alto desempenho e disponibilidade, estendendo sua aplicação para prover o balanceamento de carga, que se refere a computadores interligados entre si, ou seja, interligação de nós.

2.2.1 Interligação dos nós

Como os *clusters* são construídos por vários computadores comuns, esses computadores devem ser conectados através de uma rede para que seja possível realizar as trocas de informações entre os computadores (nós) do *cluster*. "Os nós podem ser conectados de duas formas: ponto-a-ponto ou através de um *switch"* (LINS DA ROCHA, 2004, online).

Switch: os computadores possuem uma única placa de rede e são todos interligados através de um *switch*. Descartando, assim, drivers específicos e placas de rede dual nos nós, porém tem um custo mais elevado. A Figura 2 ilustra essa conexão:

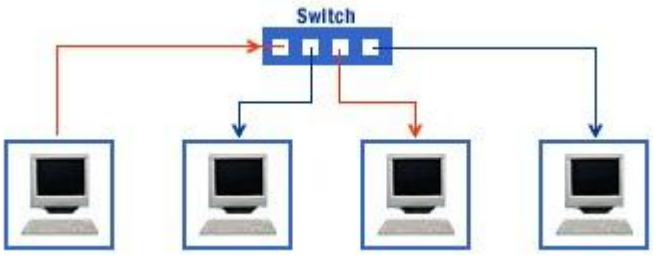


Figura 2 - Rede com Switch - (MICROSOFT, 2012, online)

A Figura 2 mostra uma rede onde os computadores estão conectados através do *switch*, uma ponta cabo vai na placa de rede do computador e a outra ponta no *switch*, caso seja necessário adicionar mais computadores é só conectar o novo computador ao *switch*.

Ponto-a-ponto: nesse tipo de conexão cada computador possui duas placas de rede. Desde o primeiro nó até o último nó são conectados através dessas placas de rede. Uma das placas recebe o cabo de rede do nó anterior e a outra conecta-se o cabo para conectar com o nó posterior. Devido ao gerenciamento das entregas de mensagens, exige-se um *software* de roteamento de mensagem mais complexo. A Figura 3 exemplifica essa conexão:

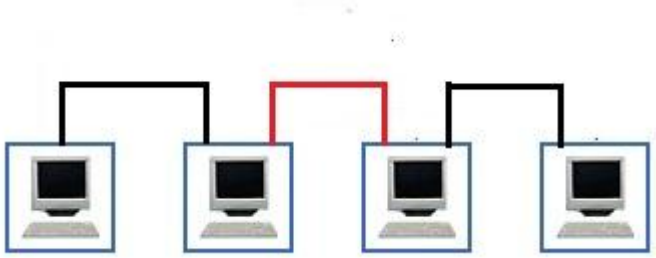


Figura 3 - Conexão ponto-a-ponto

A Figura 3 apresenta uma rede de computadores ponto-a-ponto onde os computadores são interligados entre si para isso utiliza-se duas placas de rede em cada computador e liga-se um no outro, caso seja necessário acrescentar mais computadores a essa rede o novo computador deve possuir duas placas de rede e será conectado ao último computador da rede.

No que tange a facilidade na adição de novos computadores ao *cluster* (escalabilidade), o mais indicado para o uso é a conexão feita pelo *switch,* que abrange tanto poucos computadores quanto muitos computadores, diferente da conexão ponto-a-ponto que é indicado somente para *clusters* com poucos nós.

Foram demonstrados então os dois tipos de rede que normalmente usa-se na construção de um *cluster*, a seguir serão apresentados as possíveis estruturas para a construção desse *cluster*.

2.2.2 Estruturas de *cluster*

Em relação a estrutura, existem algumas formas de se construir um *cluster*, aqui serão demonstradas três formas. As diferenças básicas entre essas estruturas de *clusters* é o recurso computacional dedicado ou não ao processamento do *cluster* e a posição física dos nós (GORINO, 2006, online):

Pilha de PCs: essa estrutura é utilizada, normalmente, nas comunidades científicas, na qual encontra-se um único computador, conhecido como mestre (servidor), que

gerencia todos os outros computadores que compõem o *cluster* (chamados escravos), estes computadores escravos não possuem teclados, mouses e vídeos conectados a eles e são empilhados junto ao servidor mestre. Ao contrário dos nós escravos, o mestre possui mouse, teclado e vídeo. Normalmente, é de responsabilidade do mestre realizar a divisão de tarefas entre os escravos, e estes enviarão as mensagens com a resolução das tarefas ao servidor, que são montadas e apresentadas de volta ao cliente. Na Figura 4 segue a representação do *cluster* com a estrutura pilha de PCs.

- Esse tipo pode ser utilizado como, por exemplo, num balanceamento de carga para *Web*, que é o objetivo deste trabalho.

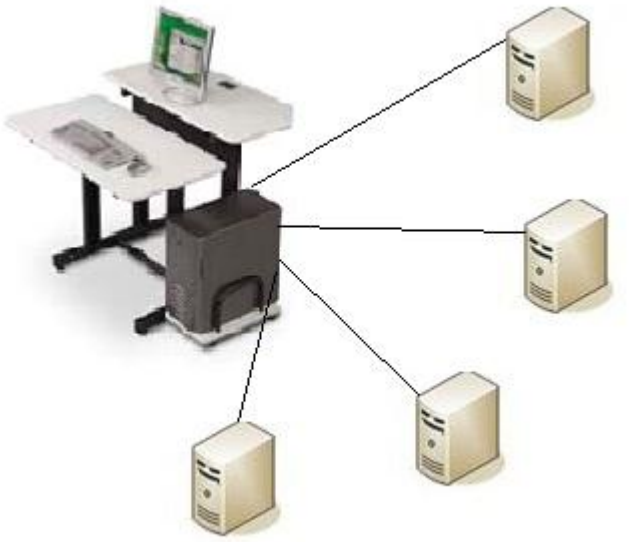


Figura 4 - Pilha de PCs.

Cluster de workstations: nessa estrutura também existe o servidor mestre e os escravos, e todos os nós possuem mouses, teclados, vídeos, entre outros dispositivos de E /S. Assim, os escravos podem ser utilizados diretamente pelos usuários para realizar todo tipo de trabalho como, por exemplo, processamento de textos e acesso à internet. Assim, os escravos são usados como nós do *cluster* apenas quando estão ociosos. O *cluster* trabalha com fila, formada através do agendamento de tarefas solicitadas pelos usuários que são executadas uma a uma pelo *cluster*, assim que estiver em funcionamento e logo após é devolvida pelo escravo já executada para o usuário. Na Figura 5 segue a representação do *cluster* de *workstations*.

- Nesta estrutura, é aconselhável que o servidor mestre não esteja no mesmo espaço físico dos nós escravos e que fique protegido por esquemas de segurança.

Figura 5 - *Cluster* Workstation (Thomas Computer Corporation, 2002, online)

Cluster distribuído ou Grid: essa estrutura é constituída de vários computadores que são interligados através de uma rede não local (WAN[1]), ou seja, os computadores que compõe esse tipo de *cluster* se localizam em diferentes regiões do mundo. Os dois tipos de estrutura anteriores pode ser utilizados para estruturar os *clusters* existentes no Grid, ou seja, a estrutura dos *clusters* na grid pode ser de pilhas de PCs ou *cluster* de workstations.

Essa seção apresentou as estruturas possíveis de utilizar o *cluster*, na próxima seção serão apresentados quais são os tipos de *cluster*s possíveis de construir utilizando-se tais estruturas.

2.2.3 Tipos de *Clusters*

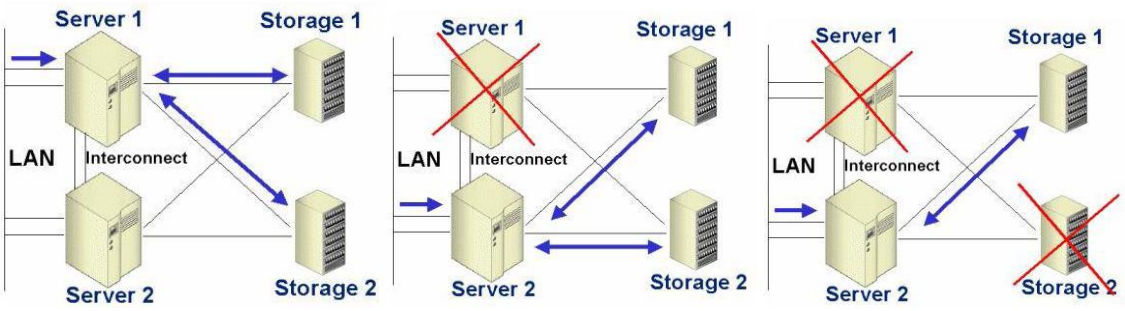
As três funcionalidades que, normalmente, são buscadas ao se projetar os *clusters* são: alta disponibilidade, balanceamento de carga e de alto desempenho, por isso, pode-se dizer que existem três tipos de *cluster* (GORINO, 2006, online):

Clusters de alta disponibilidade: são construídos com a intenção de garantir, ao máximo, a disponibilidade dos serviços instalados. Sua arquitetura é baseada em

[1] (WAN) **Wide Area Network**

nós com máquinas redundantes, que executam o mesmo serviço. Assim, caso um dos nós sofra qualquer tipo de falha, outro nó entra em funcionamento tentando evitar que haja qualquer perda, sendo necessário haver, pelo menos, duas máquinas para fornecer redundância. Durante as pesquisas, não foi encontrado um número máximo de nós que pode ser utilizado na construção desse tipo de *cluster*.

A figura 6 seguinte (a) demonstra o sistema atuando normalmente, de forma que o fluxo de dados que vem da rede é tratado pelo *Server1* e o armazenamento das informações necessárias para a replicação é feito em ambos os *Storage*; Em (b) é simulada a falha do *Server* 1, nesse caso, o *Server* 2 assume sem maiores problemas, e o armazenamento continua sendo replicado; Em (c) tem-se um caso de falha no *Server1* e em uma das mídias de armazenamento, mesmo assim, o *Server2* continuará atendendo as requisições, pois está utilizando os dados que haviam sido replicados.



(a) (b) (c)

Figura 6 - *Cluster*s de Alta Disponibilidade (Stor IT Back - Ihr Speicherspezialist, 2011, online).

Clusters de alto desempenho: esse tipo é conceitualmente simples: dado um problema complexo e identificado como “paralelizável”, um servidor (mestre) deve ser responsável por dividir este problema em inúmeras tarefas a serem processadas em nós escravos (nós dedicados ao processamento). "Assim que cada nodo (nó) encontrar a sua solução, ele a envia ao servidor para que a solução completa do problema possa ser montada" (GORINO, 2006, online). Esse tipo de *cluster* é utilizado normalmente em sistemas de previsão do tempo e sistemas de simulação, tarefas típicas que exigem alto poder de processamento. A Figura 7 apresenta a distribuição das tarefas para cada nó do *cluster,* que é feita de maneira transparente ao usuário que está utilizando a aplicação.

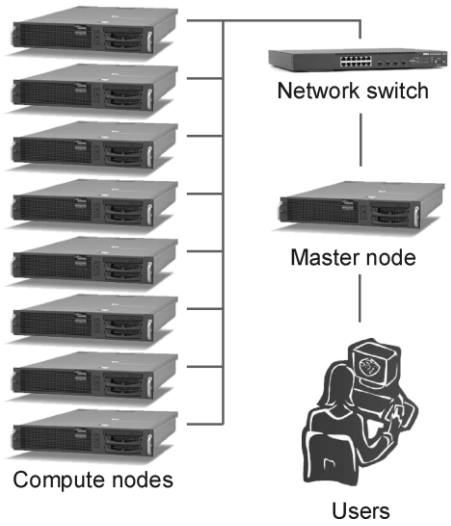


Figura 7 - *Cluster* de alto desempenho (ILRI High Performance Computing, 2005, online)

*Cluster*s de balanceamento de carga: essa topologia consiste na divisão de requisições entre vários servidores, mas, ao contrário do *cluster* de alto desempenho, o objetivo principal não é realizar um processamento de forma distribuída, visando a alcançar um ponto mais elevado em velocidade, e sim dividir os serviços de um sistema (dividir tarefas) de forma justa, para que não sobrecarregue um computador mais do que o outro. Um exemplo em que o uso deste tipo de *cluster* seria interessante é em *sites* nos quais existe um grande número de acessos simultâneos. Por exemplo, no site de busca google.com, o excesso de acessos poderia acarretar em um problema no qual o servidor poderia negar serviço a alguns usuários, por estar ocupado com os outros usuários no sistema. Com o uso do balanceamento de carga, em que as requisições seriam divididas entre vários computadores, trazendo assim uma maior disponibilidade do serviço.

- A Figura 8 apresenta um *cluster* para balanceamento de carga e como as requisições são feitas e distribuídas entre os servidores, Servidor 1 e o Servidor 2, pelo balanceador de carga. Na Figura 8 pode-se observar que os terminais 1 e 3 fizeram requisição ao balanceador que escolheu o Servidor 2 para responder as essas requisições, e que os terminais 2 e 4 tiveram suas requisições processadas e atendidas pelo servidor 1,

ocorrendo então o balanceamento de carga, que consistiu na distribuição de tarefas entre os dois servidores.

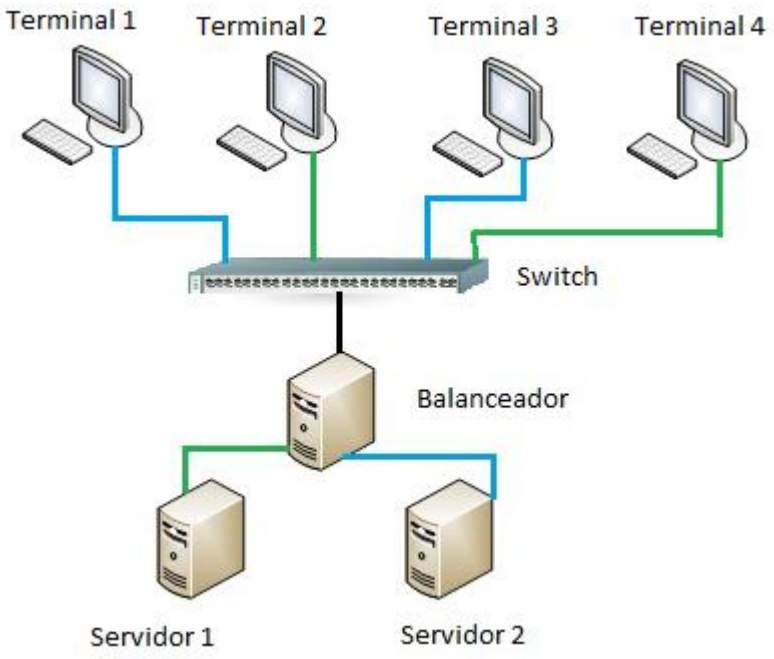


Figura 8 - *Cluster* de balanceamento de Carga

Fazendo então uma abordagem de forma mais sucinta sobre os tipos de *cluster* são eles três: primeiramente o alto desempenho normalmente é mais barato pois utiliza-se de sistemas operacionais gratuitos e máquinas comuns. Em segundo alta disponibilidade, são *cluster*s que consegue através de um computador controlador manter o seu funcionamento por um longo tempo de maneira a nunca parar seu funcionamento e, além disso, este controlador é capaz de detectar possíveis falhas. E por fim o *cluster* para balanceamento de carga, cuja principal função é distribuir carga entre os nós do *cluster* de maneira a conseguir-se um equilíbrio de processamento, porém é necessário um constante monitoramento nos mecanismos de redundância e na sua comunicação, pois caso haja alguma falha, irá interromper o seu funcionamento.

Na próxima seção serão apresentados alguns motivos para que sejam utilizados *cluster*s tanto para serviços *Web* quanto para serviços *Desktop*.

2.2.4 Motivações

Alguns benefícios que podem ser obtidos na utilização de *cluster*s são descritos logo abaixo:

- Escalabilidade Absoluta: se refere à criação de um *cluster* de uma grande proporção, ou seja, com um grande números de máquinas que interligadas terão uma capacidade de computação bem maior se comparado a um supercomputador, em outras palavras é possível criar um *cluster* com uma grande capacidade de computação que ultrapassa a capacidade de computação da melhor máquina individual;
- Escalabilidade Incremental: é possível adicionar novos nós ao sistema de acordo com a necessidade do usuário, sem que haja complicações ou que tenha que ser feita alteração na estrutura para isso;
- Alta disponibilidade: a falha é tratada pela redundância, na falta de um nó não ocorrerá a interrupção do serviço, pois outro nó assumirá as requisições enviadas pelos usuários;

Como o objetivo deste trabalho é construir um *cluster* de balanceamento de carga para aplicações *Web*, utilizando como estudo de caso o *Apache Tomcat*, a próxima seção discorre sobre balanceamento de carga de uma maneira mais detalhada.

2.3 Balanceamento de carga

"O balanceamento de carga tenta distribuir a carga de trabalho de acordo com a disponibilidade de processamento e recursos de cada máquina no sistema computacional. Esta distribuição visa maximizar a utilização dos recursos, possibilitando um melhor desempenho do sistema. O balanceamento de carga deve ser aplicado em pontos que podem se tornar um gargalo no sistema" (GORINO, 2006, online).

No balanceamento de carga, os hosts (nós) que compõem o *cluster* ficam em frequente comunicação e a todo instante trocam informações sobre estatísticas do desempenho e, também, sobre a divisão das solicitações recebidas (tarefas recebidas). Quando chega uma nova solicitação ela é vista por todos os servidores

do *cluster* e o próprio algoritmo do balanceamento de carga é quem define qual host (nó) terá responsabilidade sobre esta solicitação.

"Este tipo de *cluster* é utilizado quando um serviço é requisitado por muitos usuários e existem diversas máquinas responsáveis por responder pelo mesmo serviço. Para garantir que todas as máquinas recebam uma carga de trabalho equivalente, *daemons* (programas de monitoramento) estão ativos em cada servidor, monitorando a utilização da *CPU* e outros recursos. Com o resultado da monitoração, os servidores são capazes de realizar uma redistribuição da carga de tarefas, balanceando a utilização de todos os servidores. Se um nodo falhar, as requisições são redistribuídas entre os nodos disponíveis"(GORINO, 2006, online).

O balanceamento controla e distribui as requisições de vários clientes entre as máquinas do *Cluster* que disponibilizam o serviço, garantindo uma maior capacidade para atender solicitações de usuários, sendo que o processamento gerado por essas solicitações são divididos uniformemente entre os servidores que compõem o *Cluster*. O balanceamento de carga envolve a distribuição de tarefas entre os vários nós do *cluster*, para isso, se faz necessário definir uma política de escalonamento de tarefas. Um escalonador é um algoritmo que determina quando e por qual processador/nó será executada uma requisição. Existem os escalonadores locais e os globais (Schlemer, 1999, online). Os escalonadores locais dividem o tempo de *CPU* entre os processos contidos numa mesma máquina. Já os escalonadores globais distribuem as tarefas entre vários processadores.

Os escalonadores globais podem ser estáticos ou dinâmicos. Os estáticos são conhecidos previamente e não modificáveis. Os dinâmicos conseguem se auto avaliar e também são conhecidos como adaptativos. Os tipos de escalonadores globais serão detalhados a seguir.

- Estático: as regras de distribuição das tarefas não são modificáveis e com conhecimento prévio (OLIVEIRA , 2001, pg. 36). O seu uso implica em uma sobrecarga mínima, mas, deve ser utilizado quando o comportamento do sistema é conhecido antecipadamente.

Para os escalonadores estáticos, costuma-se empregar políticas deterministas ou probabilistas (GORINO, 2006, online): nas deterministas, existe uma lista com a relação dos nós aptos a receberem requisições ou tarefas; e nas probabilistas acontece um sorteio entre os nós, que é realizado pelo escalonador, para determinar quem irá receber a tarefa.

Nas duas políticas podem ser definidos rankings ou avaliações dos nós, proporcionando, assim, um aumento da possibilidade de selecionar um nó que seja considerado certo dentro da política utilizada. Nas avaliações utilizam-se critérios como capacidade de processamento e tempo de migração de uma tarefa. Observa-se que em algumas situações, dependendo do tipo do sistema, as políticas deterministas distribuem as requisições ou tarefas mais pesadas sempre para um mesmo grupo de nós, causando então a sobrecarga em determinado nó. Já nas probabilistas, no decorrer da execução do sistema as cargas são distribuídas uniformemente.

- No dinâmico, trabalha-se com uma constante avaliação da carga concentrada em cada nó. Nesse escalonamento observa-se em determinado momento a capacidade de cada nó, podendo assim determinar qual deles irão receber a tarefa. O escalonamento dinâmico recebe, também, classificação de adaptativo, tendo a capacidade de se auto avaliar. Após avaliação, caso seja identificada alguma sobrecarga em um dos nós, o escalonador redistribui as tarefas responsáveis pelo problema.

Caso seja feita a opção por utilizar o escalonamento dinâmico, devem ser definidos quais serão os critérios para realizar a avaliação e, posteriormente, fazer a seleção dos nós que irão receber as tarefas. Esses critérios também podem ser classificados como estáticos ou dinâmicos (OLIVEIRA , 2001, pg. 36). Os critérios estáticos consistem na capacidade de processamento de cada nó. Os critérios dinâmicos incluem a carga do processador e a carga da rede, de forma que a capacidade de processamento de um nó e a sua carga definem o tempo levado pelo nó para executar determinada tarefa e a capacidade da conexão de rede e a sua carga definem o tempo de transferência de uma tarefa até um nó específico.

Além de serem classificados em estáticos ou dinâmicos, os escalonadores globais podem ainda ser classificados como centralizados ou distribuídos (GORINO, 2006, online). No primeiro caso, apenas um dos nós do sistema decide como deve ocorrer a distribuição das requisições e no segundo caso cada nó tem autonomia para tomar decisões.

Se comparada aos escalonadores distribuídos, a abordagem centralizada tem o problema de possuir um ponto de falha único, ou seja, se o nó escalonador falhar, todo o sistema ficará indisponível. Além disso, o nó responsável pelo escalonamento

pode se tornar um gargalo do sistema, de tal forma que pode interromper o serviço por completo.

Já na abordagem distribuída cada nó deve ser capaz de se definir como receptor ou emissor de carga. Algumas implementações também consideram nós que não são nem receptores nem emissores de carga, ou seja, nós que estão no limite de sua carga. Os nós com sobrecarga são os emissores de carga e os nós com carga abaixo do normal são receptores de carga. Apesar de não depender de um ponto central, essa abordagem pode apresentar problemas ocasionados pela grande quantidade de informações que são trocadas constantemente entre os nós.

Os algoritmos de escalonamento distribuídos podem, por sua vez, ser classificados em iniciados pelo emissor ou iniciados pelo receptor de carga (OLIVEIRA , 2001, pg. 37). Na primeira situação, os nós sobrecarregados devem procurar um receptor para sua carga extra, sendo uma opção indicada quando o próprio nó emissor gera novas tarefas. Na segunda opção, os nós ociosos devem procurar por um nó doador de carga, sendo uma opção mais interessante, já que os nós ociosos possuem mais tempo para procurar nós doadores adequados. Além disso, podem existir algoritmos que são, ao mesmo tempo, iniciados pelos emissores e receptores.

Os escalonadores podem, ainda, ser classificados como preemptivo ou não preemptivo (OLIVEIRA , 2001, pg. 37). No escalonamento preemptivo, a tarefa que já está em execução pode ser escolhida para ser transferida para outro nó. Já no escalonamento não preemptivo, podem ser transferidas somente tarefas que não estão sendo executadas correntemente.

Vale ressaltar que, apesar das várias possibilidades, é difícil determinar qual é o melhor tipo de escalonador, de forma que a definição das características do escalonador que será usado depende, sobretudo, da natureza do sistema e das aplicações a serem consideradas.

Não existem grande diferenças entre balanceamento de carga para aplicações *Web* e o balanceamento de carga para o *desktop*, a diferença fica só por conta da aplicação e das ferramentas a serem utilizadas.

Na seção seguinte será apresentado o exemplo de um Cenário *Web* no qual é utilizado o balanceamento de carga.

2.3.2 Exemplo de um Cenário *Web* de utilização do balanceamento de carga

Existem portais corporativos e sites armazenam suas páginas em algum tipo de banco de dados, que traz de forma dinâmica as páginas cada vez que é recebido uma requisição de um cliente, juntando registros. Enquanto a quantidade de acessos for pequena, de 20 à 30 mil pageviews por dia, um servidor com uma configuração que seja considerada boa (supercomputador sozinho) conseguirá servir a todas as requisições sem grandes problemas. Mas, caso a quantidade de acessos aumentar muito, provavelmente, um único servidor não conseguirá suportar o tráfego.

Uma boa solução para este problema é utilizar balanceamento de carga, conectando vários servidores para receber as requisições dos usuários de forma distribuída, ou seja, os servidores dividirão o atendimento das solicitações para não haver sobrecarga sobre nenhum deles.

Para isso, todos os servidores devem manter uma cópia integral de todos os dados, pois devem estar aptos a atender as requisições (também podem acessar um compartilhamento ou sistema de arquivos distribuído/de rede). Um *software*(controlador) se encarrega de fazer a sincronização de forma automatizada entre os servidores.

Caso um servidor seja desligado, por um erro, para manutenção, ou por outro motivo qualquer, os demais continuam recebendo as requisições e retornando as páginas normalmente para os usuários. Ao ser religado, o *software* de sincronização se encarrega de sincronizar as mudanças obtidas. A Figura 9 apresenta a estrutura de um balanceamento de carga *Web*.

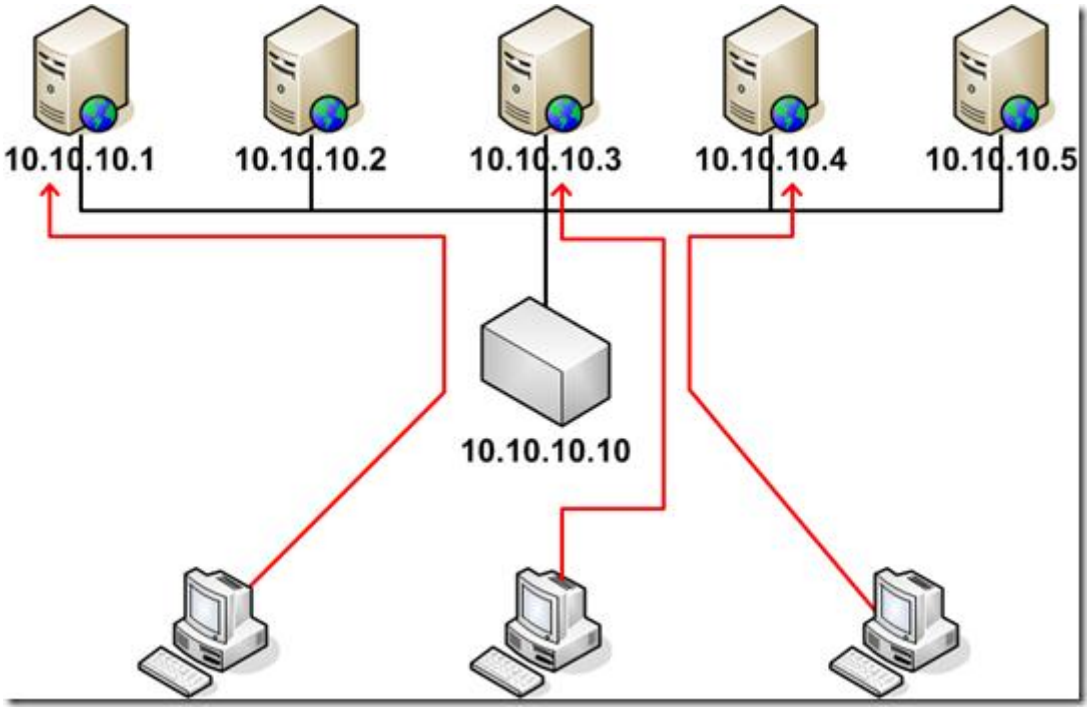


Figura 9 - Balanceamento de carga *Web*

Na Figura 9 os servidores *Web* estão recebendo requisições HTTP (Hypertext Transfer Protocol) originadas dos browsers. O cliente envia a requisição digitando, por exemplo, através do navegador (browser) o endereço “www.ulbra-to.br”. O balanceamento através do controlador (10.10.10.10) se encarrega de distribuir essa requisição de maneira rápida para um servidor que esteja desocupado ou que a fila de requisições pendentes seja menor para que o cliente receba a resposta o quanto antes.

Existem algumas ferramentas que oferecem a configuração do balanceamento de carga de aplicações *Web*, citando dois tipo de ferramentas mais conhecidas a primeira é o *JBoss* a segunda é o *Apache Tomcat*. Como o *Apache Tomcat* é o estudo de caso proposto por esse trabalho a próxima seção irá descrever sobre *Apache Tomcat*.

2.4 *Apache Tomcat*

Apache Tomcat é uma implementação open source das tecnologias *Java Servlet* e *JavaServer Pages*, que são especificações desenvolvidas pela *Java Community Process* (*The Apache Software Foudantion*, 2009, online). *Apache Tomcat* é desenvolvido em um ambiente aberto e participativo e liberado sob

a licença *Apache* versão 2 . *Apache Tomcat* é destinado a ser uma colaboração de desenvolvedores *best-of-breed* (melhores da categoria) de todo o mundo. *Apache Tomcat* possui numerosos poderes em grande escala, de missão crítica aplicações *Web* em uma ampla variedade de indústrias e organizações (*The Apache Software Foudantion*, 2009, online). O *Apache Tomcat* é um *container Web* no qual os *servlets são* executados, um *servlet* necessita de um *container Web* para ser executado.

O *Apache Tomcat* oferece execução para serviços de *clusters*, oferecendo suporte à replicação de sessões e à publicação automática entre os nós que formam o *cluster* (*The Apache Software Foudantion*, 2009, online).

A replicação das sessões acontece para que possa continuar com a mesma interação realizada com o cliente, mesmo que o nó atual vem a ficar indisponível e o outro nó possa terminar de servir o cliente. As interações entre o servidor *Apache Tomcat* e as requisições enviadas pelos clientes ocasionam a geração de algumas sessões, como conexões com banco de dados ou variáveis, essas são feitas pelo *servlet*, os quais podem ser armazenados e recuperados à medida que o cliente navega entres as páginas dinâmicas. As requisições dos usuários e as navegações são armazenados em sessões, assim, caso ocorra a interrupção do funcionamento de um dos nós do *cluster*, é possível manter o funcionamento de uma sessão, porém, todos os nós pertencentes ao *cluster* devem ter as sessões de diferentes usuários armazenadas. A partir daí então acontece a replicação, no qual as sessões de um nó são replicadas automaticamente para os demais nós que formam o *cluster*.

Além do suporte a replicação de sessões em *clusters*, o *Apache Tomcat* fornece a publicação automática, ou seja, uma vez que uma aplicação é publicada em um nó, essa publicação é repassada aos outros que também fazem a publicação da aplicação (Moodie, 2008, pg. 325).

Na sessão a seguir será apresentada como é feito a conexão entre o *Apache HTTP Server* e o *Apache Tomcat* através do *mod_jk* e em que tipo de caso é necessário fazer essa conexão.

2.4.1 O Módulo *mod_jk*

O *Apache Tomcat* oferece um método de distribuição de requisições através da sua ligação com o *Apache HTTP Server,* pelo *AJP* (*Apache Jserv Protocol*) (*The Apache Software Foudantion*, 2009, online). Essa ligação está presente em estruturas cujo intuito é fornecer páginas *Web* com conteúdo estático ou dinâmico.

Quando o *Apache HTTP Server* recebe uma requisição para uma página dinâmica, este ao envia para o *Apache Tomcat* através do módulo *mod_jk*, que é um conector no qual está agregado ao *Apache HTTP Server* através de uma biblioteca de ligação dinâmica.

Quando o servidor *HTTP* recebe uma requisição de uma página dinâmica, essa requisição é repassada para o *mod_jk* e, de acordo com as configurações, é realizado o encaminhamento para o *Apache Tomcat*. O módulo *mod_jk* pode ser configurado para trabalhar em *clusters* com a distribuição de requisições, de forma a realizar o balanceamento de carga.

Porém, na distribuição das requisições, não é possível obter informações de forma direta sobre os nós do *cluster* pelo *mod_jk*, como por exemplo, a utilização de memória, processador ou recursos computacionais. Assim, a política adotada por esse módulo para distribuir as requisições (carga) recebidas considera, no máximo, o número de requisições em atendimento, bem como o número de interações realizadas, ou seja, considera a demanda sobre a rede.

De acordo com *The Apache Software Foundation* (2010), o *mod_jk* possui algumas políticas de distribuições de requisições disponíveis, sendo elas (*The Apache Software Foudantion*, 2009, online) :

- *Request*: esta política analisa num determinado tempo a quantidade de requisições que foram distribuídas a todos os nós do *cluster*, tendo como objetivo eleger o nó que receberá a requisição, leva-se em consideração o parâmetro configurado no *lbfactor*, o *Ibfactor* carrega uma valor que corresponde a prioridade do nó. O nó que possuir de forma proporcional ao valor contido no parâmetro do *Ibfactor* a quantidade menor de requisições, este será escolhido como o nó de destino.
- *Session*: existe uma semelhança de comportamento com a política *Request*, entretanto, é utilizado como métrica para definir o nó de destino da requisição a quantidade de sessões. Esta política contabiliza todos os acessos que não possui *cookie*[2] de sessão, como sendo uma nova sessão, por não conhecer a situação atual de nenhuma sessão. Devido a não ter conhecimento se a sessão é válida ou não, não conhecer o tempo da sessão e muitas vezes não ter o

[2] ***Cookie*** é um grupo de dados trocados entre o navegador e o servidor de páginas, colocado num arquivo de texto criado no computador do utilizador. A sua função principal é a de manter a persistência de sessões HTTP.

conhecimento dos nós sobrecarregados, é indicado usar essa política quando o fator taxativo do *cluster* forem as sessões. Um exemplo disso, quando for necessário o uso de uma grande quantidade de memória para as sessões.

- *Traffic*: essa política leva em conta as informações sobre o tráfego de rede que ocorre entre o *mod_jk* e o *Apache Tomcat* em um determinado tempo, levando em conta as prioridades, essas são definidas no parâmetro *lbfactor*. Com isso, o nó com o menor tráfego de rede é eleito como o nó de destino, pois será o mais adequado a atendê-la. Nesta política o tráfego concorrente entre o *mod_jk* e o *Apache Tomcat* é ignorado, isso compromete na decisão de envio da requisição ao nó de destino.
- *Busyness*: o seu principal objetivo é escolher o nó menos ocupado para que seja definido como destino das requisições, isso é verificado no *lbfactor* de cada nó na fila de requisições pendentes. Tal informação pode ser obtida, verificando a quantidade de requisições que foram destinadas a cada nó e dessas quais ainda estão pendentes através do valor assumido no *Ibfactor*. Para que seja decidido para qual nó será encaminhado a requisição, será verificado o resultado do cálculo requisições recebidas x requisições pendentes. Então a escolha será definida para o nó que tiver a menor fila de pendências, pois será considerado o nó menos ocupado.

Algumas influências externas ao *mod_jk* como, por exemplo, processos agendados ou processos em execução no segundo plano, podem concorrer com o *Apache Tomcat* nos nós do *cluster*, ocasionando, de forma eventual, o retardo do processamento dos serviços.

Esta seção trouxe informações de como ocorre a conexão entre o *Apache HTTP* e o *Apache Tomcat*, a seção seguinte será descrito sobre *servlets* que é o grande responsável pela replicação que é realizada em *clusters* com *Apache Tomcat*.

2.4.2 *Servlet*

Servlet é uma tecnologia para replicação e reforço de servidores *Web,* baseada em componentes, método independente de plataforma para a construção de aplicativos baseados na *Web e* sem as limitações de desempenho de programas CGI (Oracle, 2012, online).

Por ser independentemente da plataforma, o desenvolvedor fica livre para escolher um "*best of breed*" estratégia para os seus servidores, plataformas e ferramentas. Ou seja, este termo "*best of breed*" é usado para descrever um produto que é o melhor em uma determinada categoria de produtos, podendo assim escolher o melhor servidor para aplicação, a melhor plataforma e por último a melhor ferramenta.

Os *Servlets* possuem acesso a toda a família de *APIs Java*, incluindo a *API JDBC* para acessar bancos de dados corporativos e uma biblioteca de *HTTP* específicos chamadas e receber todos os benefícios da linguagem *Java* maduro, incluindo portabilidade, desempenho, reutilização e proteção acidente.

Servlets de hoje são uma escolha popular para a construção de aplicações *Web* interativas. S*ervlet* estão disponíveis para o *Apache Web Server*, *Microsoft IIS*, e outros. Recipientes de *servlet* geralmente são um componente de *Web* e servidores de aplicação, tais como *BEA WebLogic Application Server*, *IBM WebSphere*, *Sun Java System Web Server*, *Sun Java System Application Server*, entre outros.

A próxima irá apresentar os materiais e métodos utilizados para realizar este trabalho.

3. MATERIAIS E MÉTODOS

Nesta seção são apresentados os materiais e a metodologia utilizada para a realização deste trabalho, que consistiu em construir um *cluster Tomcat* para balanceamento de carga de aplicações *Web*, visando verificar o desempenho e a disponibilidade da aplicação *Web*, com e sem o uso do *cluster*.

3.1 Materiais

Esta seção apresenta os *hardwares* e *softwares* utilizados para o desenvolvimento deste trabalho.

3.1.1. *Hardwares*

Para verificar se haveria melhora na disponibilidade e no desempenho das aplicações *Web* ao utilizar um *cluster Tomcat*, foram criado dois ambientes: um com dois computadores (sem uso do *cluster*) e outro com no mínimo três computadores (com o uso do *cluster*).

Por não possuir os equipamentos necessários para construir um ambiente real, optou-se por criar ambientes virtuais:

- Ambiente 1: composto por dois computadores, uma máquina virtual e a máquina real, foi utilizado para realizar os testes sem a utilização do *cluster.* Na máquina virtual, papel de servidor, foi executada a aplicação *Web* sobre o *Tomcat*. Na máquina real foi executada uma aplicação para simular os acessos dos clientes à aplicação *Web* (*Jmeter*);
- Ambiente 2: ambiente composto por três computadores, duas máquinas virtuais e a máquina real, criado para realizar os testes com o *cluster* em funcionamento. Nas duas máquinas virtuais, que tiveram o papel de servidores do *cluster*, foi executada a aplicação *Web* sobre o *Tomcat* e denominam-se nós do *cluster*. A máquina real, além de executar *software* para simular os acessos dos clientes à aplicação *Web*, teve o papel de balanceador do *cluster*, ou seja, foi a responsável por fazer a distribuição das requisições recebidas.

O equipamento utilizado para criação dos ambientes virtuais contém as seguintes especificações: processador Core I5 2,67 GHz, HD de 500 GB e 8GB de RAM. Sobre este foram executadas máquinas virtuais com a seguinte configuração: processador core I5, HD de 20 GB e 1GB de memória RAM.

Na próxima seção serão apresentados os *software*s que foram utilizados para a realização deste trabalho, são eles: *software* utilizado para virtualizar os servidores (VMWARE), aplicação *Web* utilizada nos testes (*JAMWIKI*), *software* que realiza o serviço de balanceamento das requisições recebidas (*Apache* HTTP), *software* que simula os clientes acessando os servidores (*Jmeter*) e o responsável pela ligação do *Tomcat* com servidor *Web Apache* HTTP (*Apache* Conector).

3.1.2.*Softwares*

A seguir é apresentada a relação dos *softwares* utilizados para a realização deste projeto, bem como a finalidade de cada um:

- **VMWARE 9.0:** utilizado para criação dos ambientes virtuais. Essa ferramenta foi escolhida sem a realização de um estudo sobre outras ferramentas, por já existir um conhecimento prévio da mesma. O sistema operacional utilizado nas máquinas foi o *Windows* Sete;
- ***JAMWIKI* 1.3.1:** como o objetivo do trabalho foi analisar o *Tomcat,* optou-se por não implementar uma aplicação *Web* e sim buscar um aplicação pronta, disponível na Internet. Foi escolhida a *JAMWIKI* por ser desenvolvida em Java, já que o *Apache Tomcat* dá suporte a aplicações desenvolvidas nessa linguagem, por ter um processamento adequado para realização dos testes e por estar disponível para *download* no endereço http://downloads.sourceforge.net/jamwiki/jamwiki-1.3.1.war;
- ***Apache Tomcat* 7.0:** tecnologia que foi o estudo de caso deste trabalho. O *Tomcat* é um *container* que foi desenvolvido pela *Apache Software Foundation* para executar aplicações *Web* desenvolvidas em Java. Foi utilizado para a criação do *cluster Tomcat*, e assim verificar se haveria ganho de disponibilidade e desempenho. Como o estudo de caso deste trabalho é o uso do *Tomcat*, este está descrito com mais profundidade no referencial teórico, seção 2.4;

- ***Apache* Http 2.2.2:** servidor *Web*, que foi utilizado para fazer o balanceamento da carga do *cluster*, ou seja, foi configurado de forma a distribuir as requisições entre os dois nós do *cluster*. Este serviço tem uma ligação com as instâncias *Tomcat* (nós) do *cluster*.
- ***Apache Jmeter* 2.9:** utilizada para realizar os testes de *stress* sobre a aplicação, a sua funcionalidade principal é simular usuários virtuais. Com esta ferramenta foram simulados atendimentos a quantidades distintas de usuários, nos dois ambientes. Através desta simulação de acessos de vários usuários foi possível colher informações de desempenho da aplicação. Existem outras ferramentas que possuem a mesma funcionalidade, mas esta foi escolhida por ter sido desenvolvida pela mesma fundação do *Tomcat*, *Apache Software Foudantion*;
- ***Apache Tomcat* Conector 1.2.37:** utilizada para criar a ligação entre o *Apache* HTTP e os servidores com as instâncias *Tomcat* conhecido como módulo *mod_jk*.so, possui também disponível em *mod_jk* a opção de verificar a situação do *cluster* em tempo de execução. Disponível para download no site http://www.apache.org/dist/tomcat/tomcat-connectors/jk/binaries/windows/tomcat-connectors-1.2.37-windows-i386-httpd-2.2.x.zip.

Os *softwares* serão descritos com mais detalhes nas próximas seções; eles foram utilizados nos dois ambientes virtuais, com exceção do *Apache HTTP e Apache Tomcat Conector,* que foi utilizado somente no segundo ambiente, já que esta ferramenta teve o papel de balanceador do *cluster*.

3.1.2.1 Aplicação Utilizada – *JAMWIKI*

A aplicação *JAMWIKI* é uma aplicação *Web* baseada em Wiki, desenvolvida em Java, que tem como objetivo ser um repositório de informações que oferece funcionalidades ricas e extensíveis. O seu código fonte é aberto e está disponível para utilização, podendo ser utilizado tanto em empresas privadas quanto em empresas públicas (*JAMWIKI*, 2013, online).

Como o código fonte é aberto, podem ser feitas adaptações de acordo com a necessidade. Como o objetivo do trabalho é balancear as requisições solicitadas

para aplicação e a mesma possui páginas dinâmicas, então a aplicação é suficiente para a execução dos testes e, por isto, não foram feitas alterações na aplicação.

A Figura 10 apresenta a aplicação já instalada e sendo executada sua página inicial.

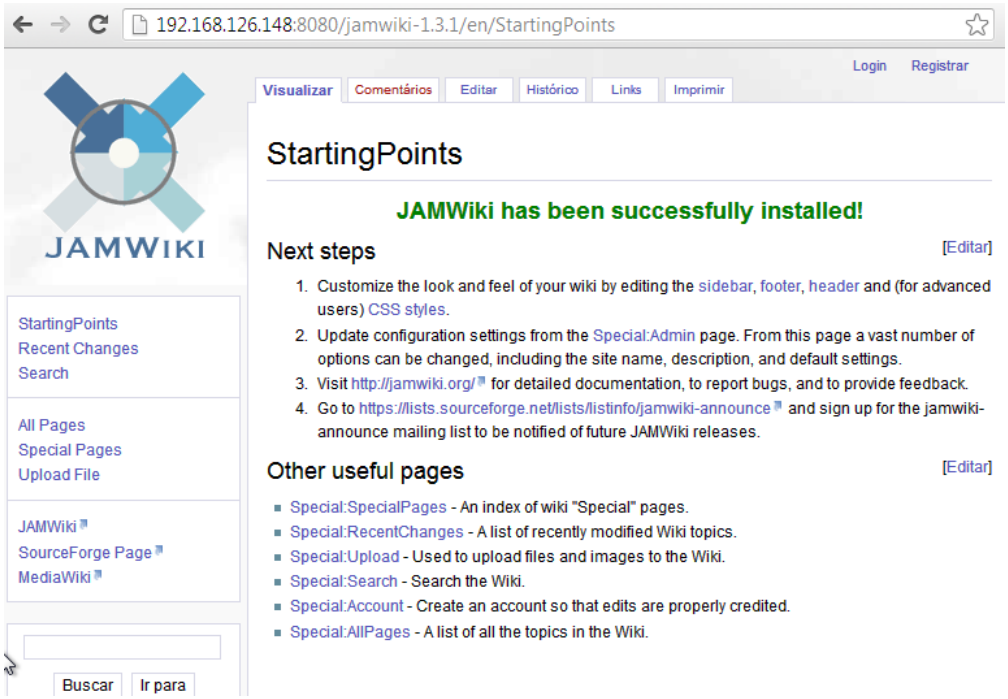


Figura 10 - Aplicação em execução

Essa aplicação foi utilizada em todos os testes, onde os usuários se conectavam realizavam *login* exigindo assim um processamento de conferência de usuário e senha e carregamento de seu perfil. Depois o usuário realizava o *logout*.

Com esses passos descritos no parágrafo anterior gerou um processamento no qual foi possível colher informações para que fosse comprovada ou não a melhoria de desempenho e de disponibilidade da aplicação.

3.1.2.2 *Apache Jmeter*

Apache Jmeter é uma ferramenta de código aberto, desenvolvida pela *Apache Software Foundation*, que tem como uma das suas funções realizar teste de *stress* em aplicações *Web*, estáticas ou dinâmicas. Este teste consiste em utilizar usuários virtuais para realizar a simulação de uma carga pesada em um servidor de *Web* (*Apache Jmeter*, 2013, online). Com isso é possível verificar o desempenho de servidor, script e, também, o comportamento deste com uma carga pesada de forma concorrente.

A seguir são apresentadas as principais características do *Jmeter* (*Apache Jmeter*, 2013, online):

- Tipos de servidores que se pode realizar teste de carga e desempenho: *Web*, *SOAP*, bancos de dados via *JDBC*, *LDAP*, *JMS*, *Mail* e Comandos nativos ou *scripts shell*;
- Desenvolvida em Java e possui total portabilidade;
- Completa amostragem com vários usuários virtuais se conectando de forma simultânea;
- Repetição dos resultados de teste.

Como essa ferramenta é desenvolvida em Java, ela tem como pré-requisito a instalação da JVM, versão 6.0 ou superior.

Para realizar testes usando o *Jmeter* é necessário que seja criado um plano de teste, pois é no plano de teste que se definem os passos que os usuários virtuais devem executar pela aplicação *Web* e a quantidade de usuários virtuais.

Elementos de um plano de teste no *Jmeter*

Um plano de testes nada mais é do que a descrição de várias ações que serão executadas de forma automática pelo *Jmeter* (*Apache Jmeter*, 2013, online). Para que o plano de teste seja executado de forma correta, utiliza-se um ou mais grupo de usuários com controladores lógicos, controladores de geração de amostras, ouvintes, temporizadores, elementos de configuração e afirmações.

A Figura 11 apresenta a tela inicial da aplicação *Jmeter*, quando esta é executada pelo usuário.

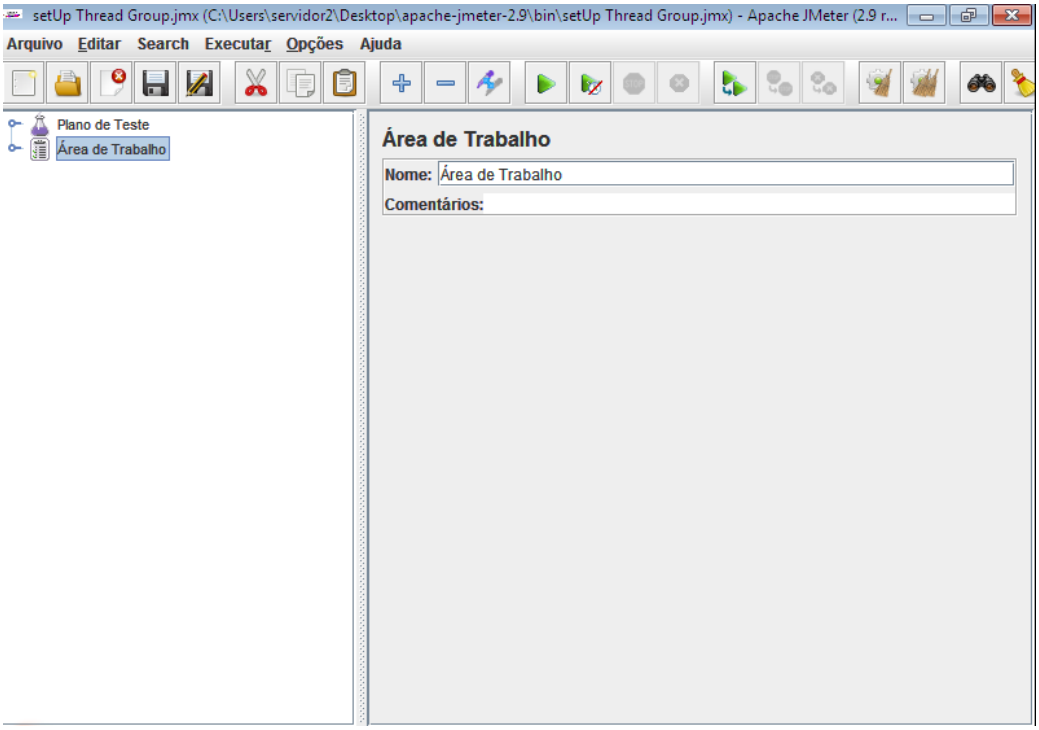


Figura 11 – Tela inicial do Jmeter.

Na Figura 11 pode-se ver o *Jmeter* em execução pronto para a criação de um plano de teste. No Plano de Teste é possível adicionar *Threads* (Usuários), fragmento de teste, elementos de configuração, temporizador, pré e pós-processadores, afirmações e ouvintes. A seguir cada um desses elementos é descrito de forma sucinta (*Apache Jmeter*, 2013,online) :

- ***Threads*:** esse elemento é o ponto de partida para criação do plano de teste, pois os controladores lógicos devem estar ligados ao *ThreadGroup* (Grupo de Usuários Virtuais), ou seja, é necessário existir usuários para acessar a aplicação. Nesse elemento configura-se o número de usuários virtuais que devem acessar a aplicação, cada usuário corresponde a uma *thread* que é lançada e controlada pelo *Jmeter* e que simula a navegação do usuário na aplicação sob teste.
- **Fragmento de Teste:** o elemento de fragmento de teste é um tipo especial de controle existente no mesmo nível do *ThreadGroup*, o qual só é executado quando mencionado ou por um módulo de controlador, ou seja, é possível definir pequenos fragmentos da aplicação a ser testada.
- **Elementos de Configuração:** são os controladores lógicos que serão definidos para que os usuários virtuais siga o que foi realizado, um exemplo é

o controlador de gravação, que grava o passo a passo sobre a aplicação e depois os usuários virtuais repete o procedimento gravado.

- **Temporizador:** por padrão, o *Jmeter* envia pedidos sem pausa entre cada solicitação, caso deseja utilizar uma pausa utiliza-se o temporizador para configurar esta pausa.
- **Pré-processador:** um pré-processador é mais frequentemente usado para modificar as configurações de um controlador lógico apenas antes de ser executado, ou atualizar as variáveis que não são extraídos do texto de resposta.
- **Pós-processador:** um pós-processador é mais frequentemente usado para processar os dados de resposta, muitas vezes para extrair valores a partir dele.
- **Afirmações:** afirmações permitem afirmar fatos sobre as respostas recebidas do servidor que está sendo testado. Usando uma afirmação, você pode essencialmente verificar que a sua aplicação está retornando os resultados que você esperava.
- **Ouvintes:** os ouvintes são os responsáveis por coletar as informações enquanto o plano de teste é executado e depois gerar gráficos com essas informações.

A Figura 12 apresenta o *Jmeter* com o plano de teste já configurado.

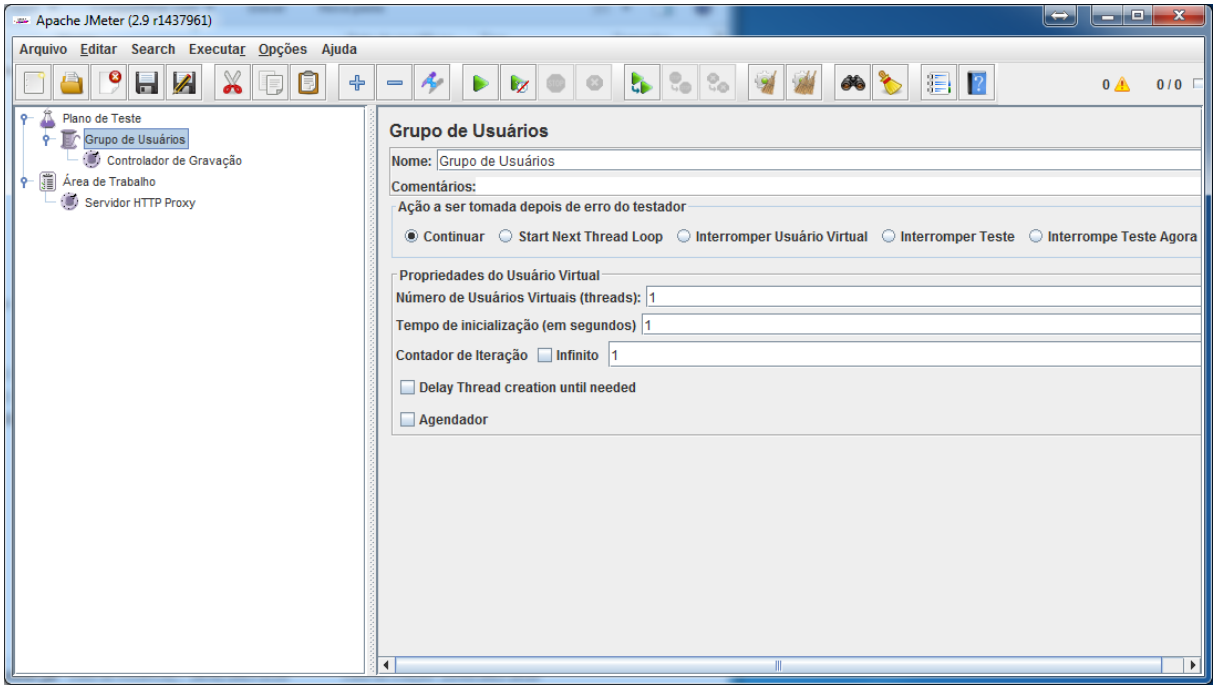


Figura 12 - Plano de Teste

Para que seja executado o *Jmeter*, o mesmo faz o papel de um servidor HTTP proxy e deve ser configurado no navegador utilizado para que toda conexão passe pelo *Jmeter*, os detalhes de configurações no navegador encontram-se em anexo.

Depois de configurado o plano de teste e também o navegador, é necessário ir até servidor HTTP Proxy e clicar em iniciar. Depois, deve-se navegar na aplicação para que o controlador de gravação grave os passos a serem executados pelos usuários virtuais.

A Figura 13 apresenta no controlador de gravação os passos executados na aplicação para que os usuários repitam os mesmos.

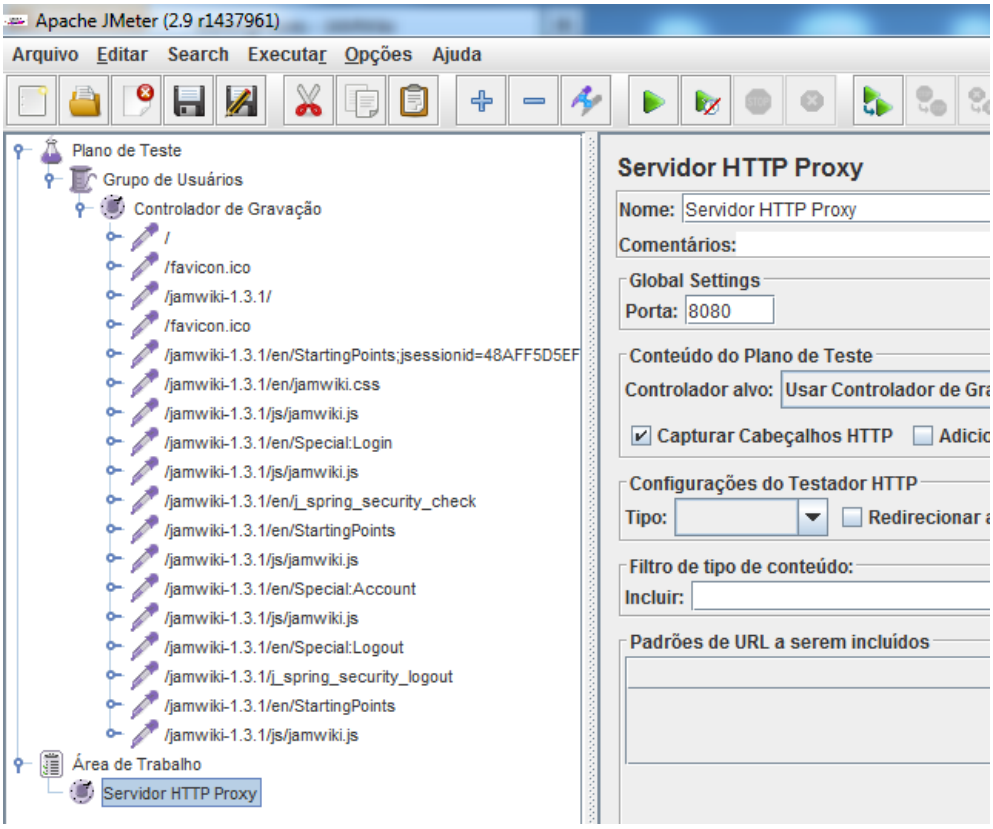


Figura 13 - Controlador de Gravação

Depois, basta definir a quantidade de usuários virtuais que irão repetir os passos apresentados no controlador de gravação, estes usuários irão executar de forma simultânea.

3.1.2.3 *Apache Tomcat Conector*

Conhecido como *mod_jk*, esse módulo de conexão tem como principal objetivo fazer a ligação do *Tomcat* há vários servidores *Web*, neste trabalho será realizada a conexão das instâncias *Tomcat* com o servidor *Web Apache* HTTP.

O conector *mod_jk* é um módulo do *Apache* HTTP que permite HTTP se comunicar com as instâncias do *Apache Tomcat* sobre o protocolo AJP. O módulo é utilizado em conjugação com o componente de conector AJP do *Tomcat*. AJP, um acrônimo para o *Apache* JServ protocolo, é uma versão binária do HTTP que é otimizado para comunicação entre *Apache* HTTP e *Tomcat* através de uma conexão TCP (*Tomcat Apache*, 2013, online).

Além disso, o *Tomcat* Conector tem também um tipo especial de trabalho, o chamado status do trabalhador. O estado do trabalhador não encaminha solicitações para o *Tomcat*. Em vez disso, permite recuperar o status e informações de

configuração em tempo de execução, e, além disso, para mudar muitos itens de configuração dinamicamente. Isto pode ser feito através de uma interface de *Web* simples incorporado ao *mod_jk*.

3.2. Métodos

Após fazer uma revisão de literatura sobre aplicações *Web*, sistemas distribuídos e balanceamento de carga, foi criado um *cluster* para realizar o balanceamento de carga em aplicações *Web* utilizando como estudo de caso o *Tomcat*.

Inicialmente, foram criados dois ambientes de testes, para realizar testes sem o *cluster* e, posteriormente, com o funcionamento do *cluster*. Com isso, foi possível realizar testes sobre os dois ambientes e comparar os resultados para verificar se há ganho de disponibilidade e desempenho no acesso à aplicação com o uso do *cluster Tomcat*.

A estrutura dos dois ambientes foi explicada na seção 3.1.1. A seguir é descrito como esses ambientes foram utilizados nos testes:

- Ambiente 1 – No primeiro ambiente foi disponibilizado um único servidor virtual (servidor 1), sem a utilização do *cluster*. Nesse servidor foi instalado o *Tomcat* e, sobre ele foi executada a aplicação *JAMWIKI*, sendo que todos os usuários se conectavam na aplicação *Web* executando sobre esse servidor. O detalhamento de como foram realizados esses testes encontra-se adiante nessa seção e os resultados obtidos encontram-se na seção 4.
- Ambiente 2 – No segundo ambiente foram disponibilizados três servidores, formando um *cluster*. Em dois nós foi instalado o *Tomcat* e executada a aplicação *JAMWIKI*. No nó balanceador foi instalado o *Apache HTTP* para dividir a carga de trabalho entre os outros dois nós, de forma que os usuários conectassem e eles fossem direcionados para as instâncias *Tomcat*. Com isso foram realizados testes com a utilização do *cluster* e o detalhamento destes encontra-se na seção 4.

Depois de definir como seriam os ambientes de teste, definiu-se quais recursos seriam analisados para verificar o desempenho da aplicação *Web*, são eles:

- *CPU:* foi observado o quanto de utilização da *CPU* foi usado para executar os testes, desta forma pode-se verificar o desempenho de cada servidor*;*

- Memória: foi observada a quantidade, em porcentagem, de utilização de memória, com isso é possível verificar o desempenho para atender certa quantidade de requisições nos servidores;
- Disco: foi verificado o quanto foi acessado e o tempo utilizado para escrita e leitura no disco.

Outro recurso que se costuma analisar em trabalhos que envolvem balanceamento de carga é a influência da rede no *cluster*, neste caso não foi analisado tal recurso por ter sido utilizado ambiente virtual, que não é adequado para analisar este recurso.

Depois de definir os recursos, foi definido que seriam realizados testes de *stress* nos dois ambientes, pois a partir do teste de *stress*[3] é possível verificar o desempenho da aplicação servida pelos servidores *Tomcat*.

Com a ferramenta *Jmeter*, foram realizados os seguintes testes de *stress*:

- Situação 1: 10 usuários entraram simultaneamente, fizeram *login navegaram pelo seu perfil* e depois fizeram *logout*;
- Situação 2: 100 usuários entraram simultaneamente, fizeram *login navegaram pelo seu perfil* e depois fizeram *logout*;
- Situação 3: 1000 usuários entraram simultaneamente, fizeram *login navegaram pelo seu perfil* e depois fizeram *logout*; e
- Situação 4: 10.000 usuários entraram simultaneamente, fizeram *login navegaram pelo seu perfil* e depois fizeram *logout*.

Cada uma das situações foi repetida por três vezes, em momentos diferentes. Isso foi necessário para que fossem coletadas as informações dos parâmetros definidos, feitas comparações dos três momentos e para comprovar que, mesmo com outros processos sendo executados nos servidores, o *cluster* é mais vantajoso.

A seguir, para verificar a disponibilidade da aplicação executada no *cluster Tomcat,* durante o acesso dos usuários, foi desligado um nó do *cluster. Isso foi feito para verificar* se os usuários continuariam acessando a aplicação sendo servidos por

[3] Em um teste de stress, além de uma grande carga disparada contra a aplicação, alguns cenários de crash da aplicação são testados, com o objetivo também de determinar a capacidade de recuperação e estabilidade do sistema.

um único nó. Esse teste foi realizado várias vezes e em momentos diferentes, desligando-se um nó e depois outro nó, sempre um único nó de cada vez.

Por fim, foi feito um paralelo entre o comportamento dos dois ambientes, a partir dos resultados obtidos nos testes realizados. Para isso, foi elaborada uma tabela comparativa, para representar este paralelo (CPU, Memória e DISCO) e facilitar a compreensão. Também, foi elaborada uma explanação sobre os benefícios de utilizar-se *cluster Tomcat*, no que diz respeito à disponibilidade e desempenho.

4. RESULTADOS E DISCUSSÃO

O objetivo deste trabalho foi realizar a implementação de um *cluster Apache Tomcat* para realizar o balanceamento de carga dos acessos a aplicação *Web*. O intuito foi promover, assim, uma maior disponibilidade da aplicação e também um melhor desempenho. Por isto, além de ter sido realizada a implantação do *cluster Tomcat* e dos testes básicos de comunicação, foram realizados experimentos para verificar se a utilização do *cluster* melhoraria de forma significativa o desempenho e a disponibilidade da aplicação. O tipo de *cluster* a ser utilizado neste trabalho conforme consta no referencial teórico é do tipo balanceamento de carga.

A estrutura do *cluster* é composta por três máquinas, na qual foram configuradas duas máquinas como servidores (nó1 e nó2), sendo que estes executarão a aplicação *Web* que atenderá as solicitações dos clientes. A terceira máquina foi configurada como servidor utilizado para ser o balanceador de carga, ou seja, o responsável por distribuir as requisições dos clientes entre os dois servidores nós do *cluster*. Por questões práticas, esta estrutura foi configurada em um ambiente virtual, conforme explicado na seção 3.1.1.

Como o objetivo foi verificar a melhoria de desempenho e disponibilidade proporcionada pelo *cluster Tomcat*, foram realizados testes sobre *cluster* e sobre um ambiente de servidor único, para que fosse possível fazer o comparativo entre as duas estruturas e verificar a melhoria proporcionada pelo uso do *cluster*.

Nesse contexto, essa seção apresenta:

- Estrutura do *cluster*: apresenta a arquitetura do *cluster*, uma visão geral sobre a sua criação e alguns detalhes do jkstatus (módulo mod_jk);
- Testes de verificação: testes de *stress* sobre a aplicação usando a ferramenta (*Jmeter*) para simular conexões de usuários de forma simultânea, realizados sem o uso *cluster* e com o uso *cluster*;
- Teste de disponibilidade: teste para verificar o comportamento após o desligamento das máquinas, simulando alguma falha com as mesmas;
- Paralelo entre os resultados dos testes com a utilização do *cluster* e sem a utilização do *cluster*;
- Discussão sobre os resultados obtidos.

4.1 – Estrutura do *cluster*

A estrutura do *cluster Tomcat*, apresentada na Figura 14, mostra a arquitetura que é composta por dois servidores de aplicação, o balanceador e os clientes. Para que fosse possível criar essa estrutura foi criado um ambiente virtual, criado com o *software* VMWARE, de forma que: os dois servidores (nós do *cluster*) são máquinas virtuais; balanceador e a ferramenta que simula os acessos dos clientes à aplicação são executados na máquina real. O sistema operacional em todas as máquinas é o *Windows* 7.

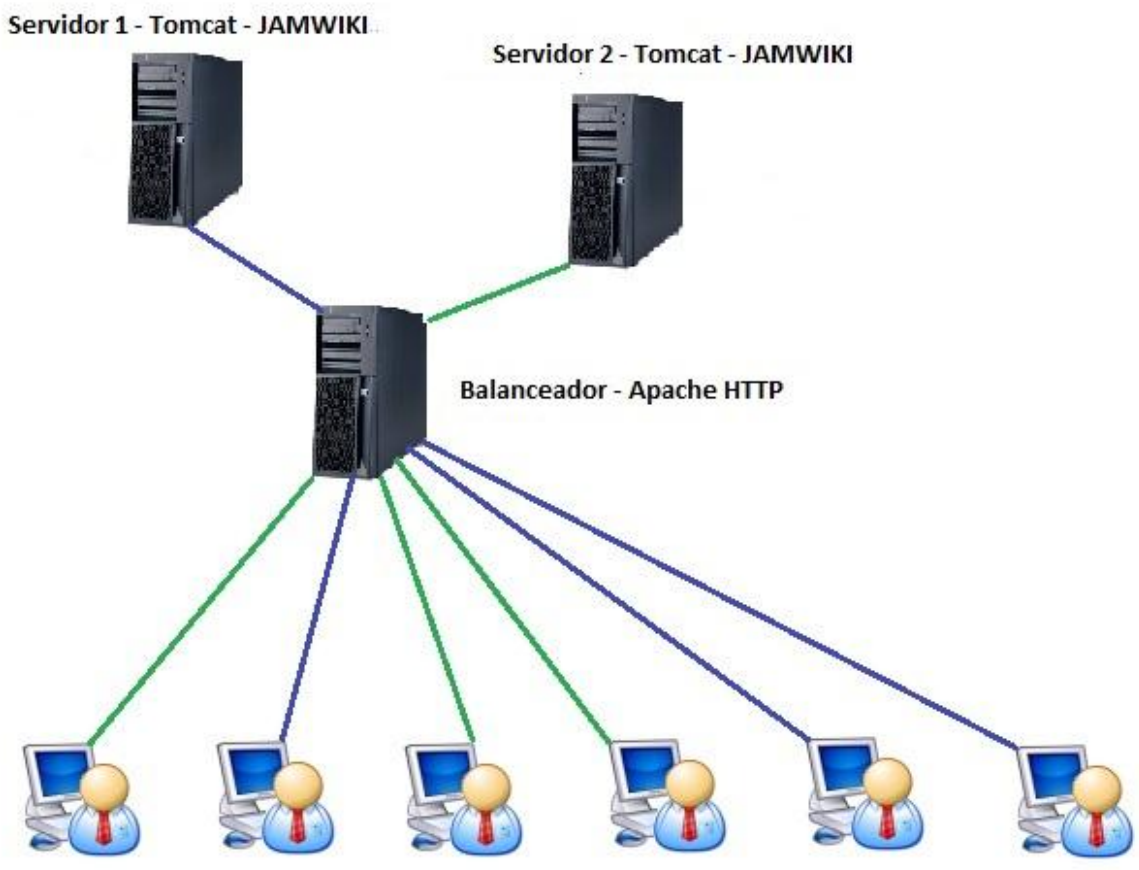


Figura 14 - Cluster Tomcat e os clientes

Para que fosse possível a criação desse *cluster,* foi configurado nos dois servidores virtuais o *Apache Tomcat* executando a aplicação *JAMWIKI* e na máquina real instalado o *Apache* HTTP com a ligação entre o balanceador (*Apache* HTTP) e os nós (*Apache Tomcat*), para a ligação foi utilizado o *mod_jk* (*Apache* Conector).

Nesta seção são apresentados detalhes importantes da configuração e a mesma, por completo, se encontra no apêndice C.

O servidor *Apache Tomcat* oferece suporte a implantação de aplicações de uma forma simples através da opção manager App, sendo que estas aplicações devem ter o formato *.war*.

Após a configuração das duas instâncias *Tomcat,* com a aplicação executando nos dois nós (servidores virtuais), uma configuração importante é a inclusão da classe cluster no arquivo *server.xml* para que a instância funcione no cluster. Para isso, foi adicionado ao arquivo *server.xml* um trecho de código necessário para o funcionamento do *cluster* em cada um dos nós, o arquivo *sever.xml* encontra-se na pasta raiz do *Tomcat* no diretório conf. A Figura 15 apresenta o código que foi adicionado no arquivo

server.xml.

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
                 channelSendOptions="8">

          <Manager className="org.apache.catalina.ha.session.DeltaManag
er"
                   expireSessionsOnShutdown="false"
                   notifyListenersOnReplication="true"/>

          <Channel className="org.apache.catalina.tribes.group.GroupCha
nnel">
            <Membership className="org.apache.catalina.tribes.membershi
p.McastService"
                        address="228.0.0.4"
                        port="45564"
                        frequency="500"
                        dropTime="3000"/>
            <Receiver className="org.apache.catalina.tribes.transport.n
io.NioReceiver"
                      address="auto"
                      port="4000"
                      autoBind="100"
                      selectorTimeout="5000"
                      maxThreads="6"/>

            <Sender className="org.apache.catalina.tribes.transport.Rep
licationTransmitter">
              <Transport className="org.apache.catalina.tribes.transpor
t.nio.PooledParallelSender"/>
            </Sender>
            <Interceptor className="org.apache.catalina.tribes.group.in
terceptors.TcpFailureDetector"/>
            <Interceptor className="org.apache.catalina.tribes.group.in
terceptors.MessageDispatch15Interceptor"/>
          </Channel>

          <Valve className="org.apache.catalina.ha.tcp.ReplicationValve
"
                 filter=""/>
          <Valve className="org.apache.catalina.ha.session.JvmRouteBind
erValve"/>

          <Deployer className="org.apache.catalina.ha.deploy.FarmWarDep
loyer"
                    tempDir="/tmp/war-temp/"
                    deployDir="/tmp/war-deploy/"
                    watchDir="/tmp/war-listen/"
                    watchEnabled="false"/>

          <ClusterListener className="org.apache.catalina.ha.session.Jv
mRouteSessionIDBinderListener"/>
          <ClusterListener className="org.apache.catalina.ha.session.Cl
usterSessionListener"/>
        </Cluster>
```

Figura 15 - Código cluster adicionado nos nós (Apache Conector, 2013, online)

A Figura 15 apresenta o trecho que implementa a classe *cluster*, que foi inserido no arquivo *server.xml* de cada um dos nós do *cluster*, servidores 1 e 2. Após

isso, os dois nós do *cluster* estão prontos para receber as requisições do balanceador.

Na máquina real foi instalado o *Apache* HTTP, que faz a função de balanceador do *cluster*, ou seja, os clientes irão acessar na máquina real na porta 80 e o *Apache* HTTP irá distribuir as solicitações entre os servidores.

Para que o balanceador funcione adequadamente, é necessário realizar alguns ajustes na máquina real, como:

- Inserir o arquivo (*mod_jk*.so) dentro da pasta raiz do *Apache* HTTP no diretório modules;
- Editar o arquivo httpd.conf e adicionar o trecho de código apresentado na Figura 16.

```
492  LoadModule jk_module modules/mod_jk.so
493  JkWorkersFile   conf/workers.properties
494  JkLogLevel      debug
495  JkLogFile       logs/mod_jk.log
496  JkMount         /jamwiki-1.3.1/* lb
497
498  <Location /jkmanager/>
499      JkMount jkstatus
500      Order deny,allow
501      Allow from all
502  </Location>
```

Figura 16 - Configuração httpd.conf

A Figura 16 apresenta: na linha 492, a informação da localização do arquivo (*mod_jk*.so); na linha 493, o caminho do arquivo que deve ser criado com as informações dos nós do *cluster* workers.properties (arquivo com as informações dos servidores nós do *cluster*); na linha 495, o local dos logs; na linha 496, o caminho da aplicação renomeado para lb; e; por fim; na linha 499, configura-se o *Jk_status*, que possibilita acompanhar o *cluster* em tempo de execução.

Foi criado o arquivo workers.properties, o qual contém as informações dos servidores nós que compõem o *cluster.* A Figura 17 traz o trecho de código que deve ser inserido.

```
1  worker.list=lb
2
3  worker.worker1.port=8009
4  worker.worker1.host=192.168.137.140
5  worker.worker1.type=ajp13
6  worker.worker1.lbfactor=1
7
8  worker.worker2.port=8009
9  worker.worker2.host=192.168.137.142
10 worker.worker2.type=ajp13
11 worker.worker2.lbfactor=1
12
13 worker.lb.type=lb
14 worker.lb.sticky_session=1
15 worker.lb.balanced_workers=worker1,worker2
16
17 worker.list=jkstatus
18 worker.jkstatus.type=status
```

Figura 17 - Arquivo workers.properties

A Figura 17 mostra as configurações realizadas, no nó1 e no nó 2, que foram feitas da seguinte forma:

- Linhas 3 até 6: apresenta as informações do nó1 , a porta de conexão, o ip o tipo de conexão e lbfactor (prioridade do nó);
- Linhas 8 até 11: apresenta as informações do nó1 , a porta de conexão, o ip o tipo de conexão e lbfactor (prioridade do nó);
- Linhas 13 até 15: apresenta a configuração dos servidores que irão receber as requisições do balanceador;
- Linhas 17 e 18: apresenta a ativação do *Jk_status*;
- Porta de conexão: porta que *mod_jk* ira se conectar;
- IP: apresenta o IP da máquina;
- Tipo de conector; é utilizado o “ajp13” para balanceamento;
- Lbfactor: é onde se define a prioridade de cada nó do *cluster*, ficando dessa forma “1” para cada nó, o balanceamento de carga distribuirá as requisições de forma igual para os dois servidores.

Conforme descrito anteriormente, o *mod_jk* possui o *Jk_status*, que possibilita a verificação do funcionamento do *cluster* em tempo de execução. A Figura 18 apresenta a tela do *Jk_status*, que é um módulo *Web*.

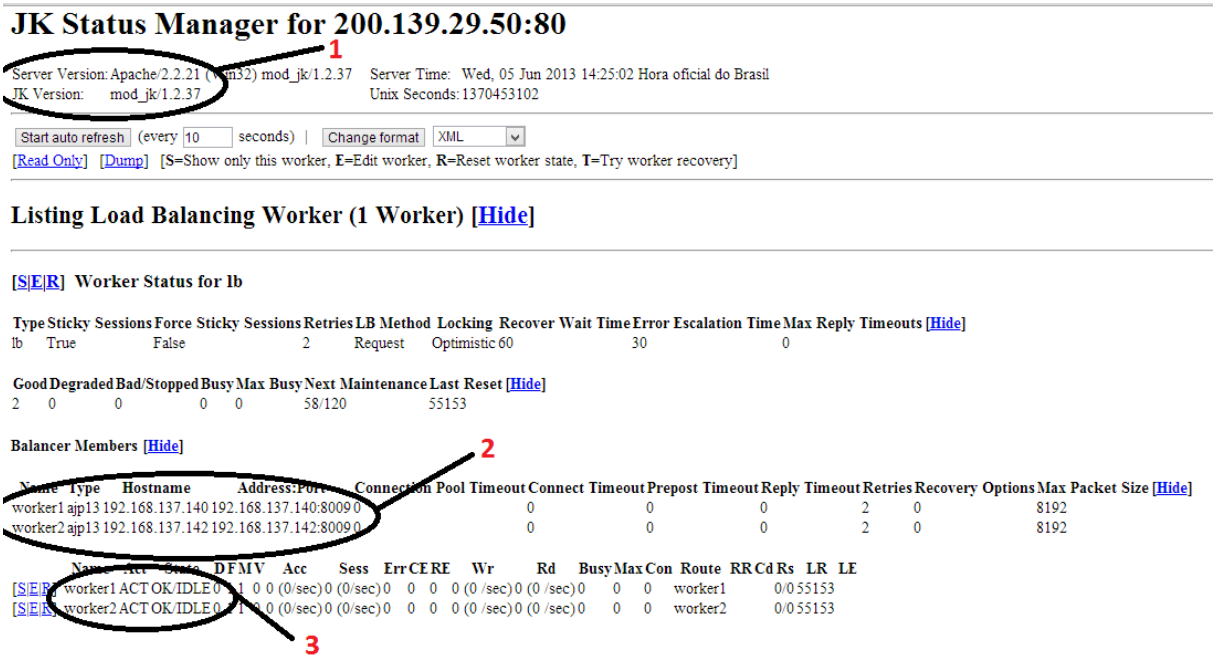
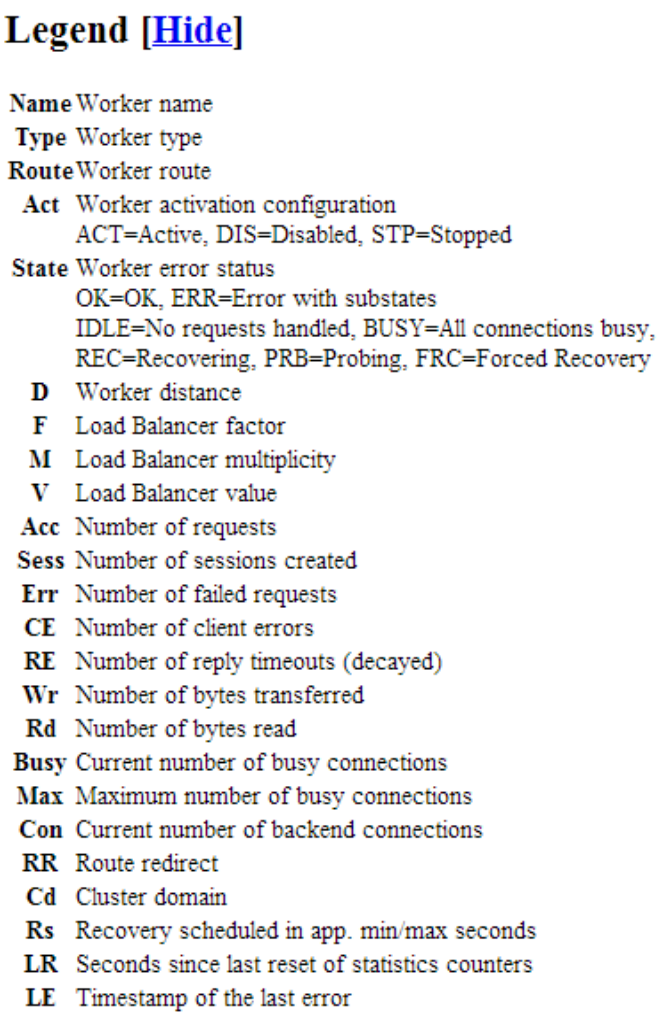


Figura 18 - Jk_status

Conforme mostra a Figura 18, o *Jk_status* apresenta diversas informações, como as que estão destacadas na imagem:

1. Versões do *Apache* HTTP e do *Apache* Conector, respectivamente 2.2.21 e 1.2.37;
2. Ele apresenta também os nós que compõem o *cluster*, mostrando o tipo, o nome e o endereço IP de cada nó;
3. Ele traz também a situação em que se encontra cada nó.

A Figura 19 apresenta os códigos e suas descrições que contém no *Jk_status*, para que seja possível identificar a situação de cada nó do *cluster.*



Legend [Hide]

Name	Worker name
Type	Worker type
Route	Worker route
Act	Worker activation configuration ACT=Active, DIS=Disabled, STP=Stopped
State	Worker error status OK=OK, ERR=Error with substates IDLE=No requests handled, BUSY=All connections busy, REC=Recovering, PRB=Probing, FRC=Forced Recovery
D	Worker distance
F	Load Balancer factor
M	Load Balancer multiplicity
V	Load Balancer value
Acc	Number of requests
Sess	Number of sessions created
Err	Number of failed requests
CE	Number of client errors
RE	Number of reply timeouts (decayed)
Wr	Number of bytes transferred
Rd	Number of bytes read
Busy	Current number of busy connections
Max	Maximum number of busy connections
Con	Current number of backend connections
RR	Route redirect
Cd	Cluster domain
Rs	Recovery scheduled in app. min/max seconds
LR	Seconds since last reset of statistics counters
LE	Timestamp of the last error

Figura 19 - Tabela de Situações Códigos

A Figura 19 apresenta uma lista com a codificação e a descrição de cada situação que pode ser apresenta nos nós do *cluster,* por exemplo, se um dos nós tiver com status ACT/IDLE significa que o nó está ativo e ocioso, ou seja, livre para receber requisições.

Com *Jk_status,* além de apresentar a situação que se encontra cada nó em tempo de execução, é possível também fazer algumas alterações nas configurações do *cluster*, conforme mostra Figura 20:

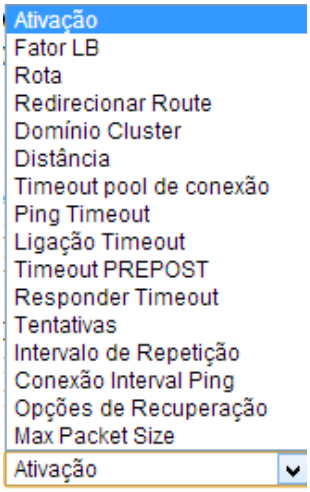


Figura 20 - Opções de configurações Jk_status

A Figura 20 apresenta algumas configurações que podem ser alteradas no *cluster* através do *Jk_status*, como por exemplo, alterar o fator LB de um dos nós ou até mesmo o domínio do *cluster*. O fator LB indica a prioridade de cada nó, ou seja, se o nó 1 tiver configurações superior do que o nó 2, pode-se configurar fator LB igual a 2 para o nó1 e 1 para nó 2, com isso o nó1 irá receber o dobro de requisições que o nó2. No *cluster* utilizou-se fator LB igual a 1 para os dois nós.

Na mesma máquina que foi configurado o balanceador, neste caso a máquina real, também foi o *software Jmeter*, para realizar os testes de acesso à aplicação pelos clientes. A próxima seção descreve os testes que foram realizados, tanto com o *cluster* em funcionamento quanto sem o uso *cluster*.

4.2 – Testes Realizados

Para comprovar que a utilização do *cluster*, mesmo com apenas dois nós servidores, proporcionaria um melhor desempenho e melhor disponibilidade da aplicação *Web*, foram realizados testes em dois Cenários:

- Cenário 1 - sem a utilização do *cluster*, ou seja, as requisições dos clientes foram atendidas por um único servidor;
- Cenário 2 - com o *cluster* em funcionamento, ou seja, o balanceador recebe as requisições dos clientes e a distribui entre o servidor 1 e servidor 2.

A ferramenta *Apache Jmeter* foi utilizada para realizar o teste de *stress* sobre a aplicação *Web*, nos dois Cenários. Com essa ferramenta foram criados usuários virtuais para simular os acessos à aplicação de forma simultânea. Os usuários

virtuais repetem as ações gravadas pelo controlador de gravação, conforme explicado na seção 3.1.2.2.

Nos testes apresentados cada usuário acessou sua conta visualizando assim o seu perfil, navegou pela aplicação por alguns segundos e logo depois saiu da aplicação.

O monitor de recursos do *Windows* foi a ferramenta utilizada para colher as informações relacionadas aos acessos dos usuários como processamento de *CPU,* uso de memória e tempo de resposta, informações essas necessárias para avaliar desempenho e disponibilidade.

Os testes de *stress*, realizados com intuito de obter informações sobre o desempenho, foram realizados em quatro situações:

- Situação 1 – 10 usuários virtuais acessando simultaneamente;
- Situação 2 – 100 usuários virtuais acessando simultaneamente;
- Situação 3 – 1.000 usuários virtuais acessando simultaneamente;
- Situação 4 – 10.000 usuários virtuais acessando simultaneamente.

Os mesmos testes foram realizados nos dois Cenários, sendo que cada situação foi aplicada em três momentos diferentes, para tornar os resultados mais consistentes. Os resultados desses testes são apresentados a seguir.

4.2.1. Testes sobre o Cenário 1

Em um primeiro momento foram realizados testes sobre o Cenário 1, sem a utilização do *cluster*. Assim, um único servidor ficou responsável por atender a todas as requisições dos usuários, conforme pode ser observado na Figura 21.

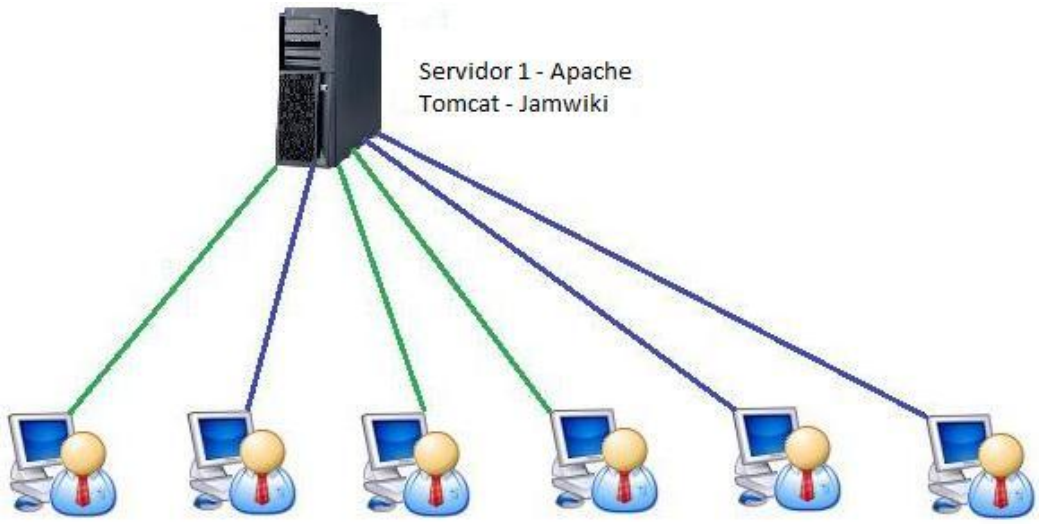


Figura 21 - Sem utilização do cluster

Para conseguir a situação ilustrada na Figura 21, o servidor que executa a aplicação *JAMWIKI* foi configurado em uma máquina virtual, na qual foi instalado e configurado o Apache Tomcat 7.0.

Foram aplicados testes nas quatro situações relacionadas na seção 4.2 e os resultados destes são apresentados a seguir. Vale ressaltar que, para cada situação, os testes foram repetidos em três momentos distintos.

Na primeira situação dez usuários virtuais acessaram simultaneamente a aplicação, efetuaram o acesso ao perfil, realizaram comandos de navegação por alguns segundos e saíram da aplicação. A Figura 22 ilustra a tela do *Jmeter*, na qual foram configuradas que ações os usuários realizariam e, em destaque, a configuração da quantidade de usuários virtuais que acessariam as informações.

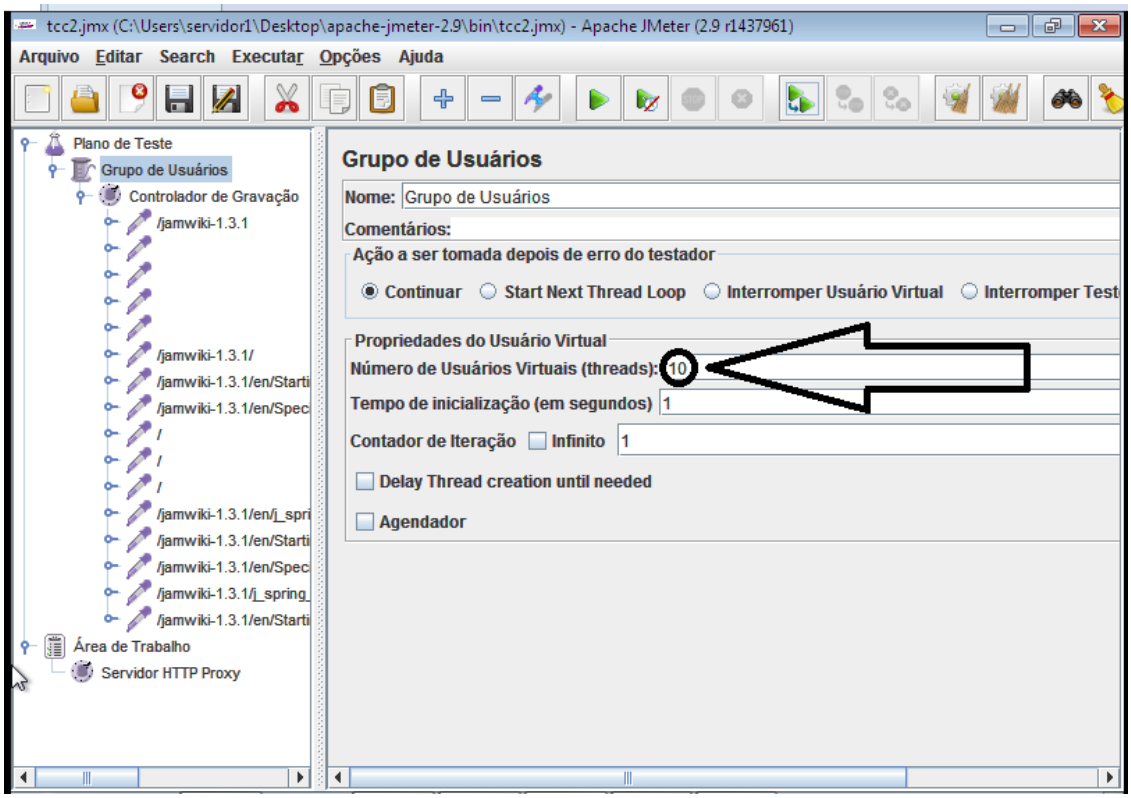


Figura 22 - Configuração de Usuários (Jmeter)

Na Figura 22, é possível identificar as ações executadas e gravadas pelo controlador de gravação, que aparece no item “Controlador de Gravação”, essas ações que serão executadas pelos usuários virtuais. Na Figura 22, mostra do lado esquerdo em controlador de gravação o acesso à aplicação, a navegação por alguns segundos e depois a saída da aplicação.

Para obter as informações do processamento e verificação do desempenho do servidor foi utilizado o monitor de recursos do *Windows*. A figura 23 apresenta os resultados do teste da situação 1, no primeiro momento.

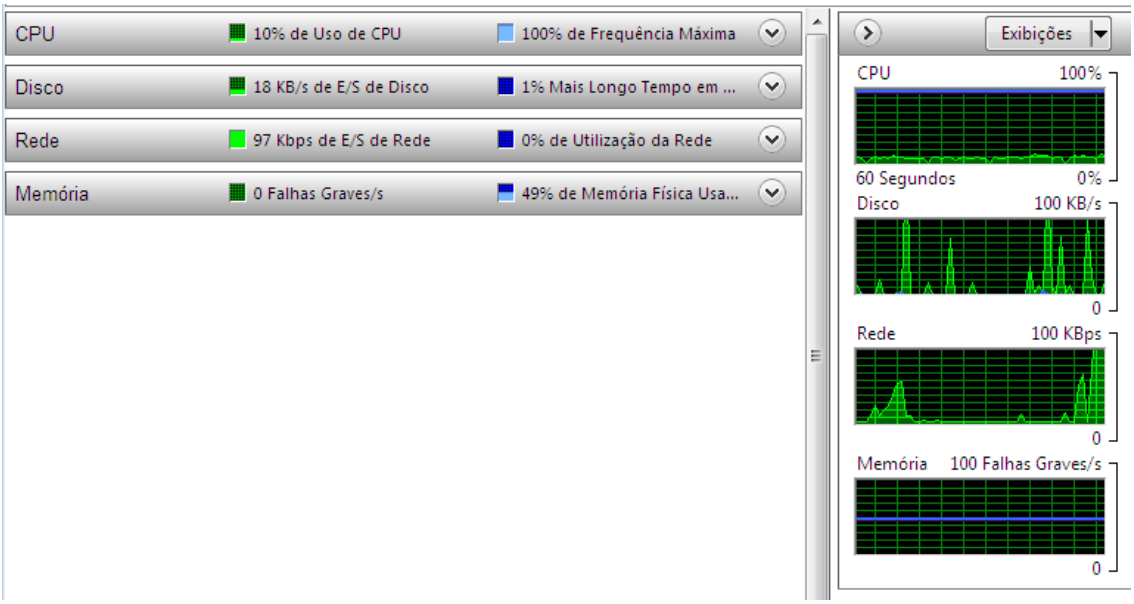


Figura 23 - 10 usuários - Primeiro Momento

A Figura 23 apresenta a tela do monitor de recurso do *Windows*, com as informações do primeiro momento, na qual se pode observar que: o processamento da *CPU* manteve-se com um nível baixo, somente 10% de uso; a memória atingiu 49% de utilização; e o disco apresentou uma taxa de leitura e escrita em torno de 18 KB/s.

Nos outros dois momentos da situação 1 e nos três momentos das situações 2, 3 e 4 os procedimentos de testes realizados foram os mesmos, sendo que os resultados são apresentados na Tabela 1. A descrição de todas as situações, nos três momentos, é apresentada de forma mais detalhada no apêndice A.

Tabela 1 - Cenário 1

RESULTADOS – CENÁRIO **1**					
SITUAÇÃO	**RECURSO**	**MOMENTO 1**	**MOMENTO 2**	**MOMENTO 3**	**MÉDIA**
10 Usuários	Uso de *CPU* (%)	10	11	9	10
	Uso de Memória (%)	49	48	49	48,66
	Fluxo de Disco (KB/s)	18	16	4	12,66
	Tempo de Resposta	1min 3s	1min 6s	1min 5s	1min 4s
100 Usuários	Uso de *CPU* (%)	58	28	52	46
	Uso de Memória (%)	60	57	60	59
	Fluxo de Disco	4	216	22	80,66

	(KB/s)				
	Tempo de Resposta	12min 15s	11min 35s	14min 10s	12min 50s
1000 Usuários	Uso de *CPU* (%)	100	100	100	100
	Uso de Memória (%)	61	79	79	73
	Fluxo de Disco (KB/s)	1033	3635	1172	1946,66
	Tempo de Resposta	1h 10min 20s	1h 15min 35s	1h 13min 45s	1h 12min 34s
10000 Usuários	Uso de *CPU* (%)	S/I	S/I	S/I	-
	Uso de Memória (%)	S/I	S/I	S/I	-
	Fluxo de Disco (KB/s)	S/I	S/I	S/I	-
	Tempo de Resposta	S/I	S/I	S/I	-

A Tabela 1 apresenta os valores dos requisitos que foram identificados no momento dos testes de cada situação e a média. Como pode ser observado, nos três momentos dos testes com 1.000 usuários, o uso de CPU atingiu 100%.

Ainda na Tabela 1, pode-se verificar que na situação em que 10.000 usuários virtuais se conectaram não foram apresentadas informações, pois nos três momentos dessa situação a máquina virtual e a máquina real entraram em pane.

Como explicado anteriormente, os mesmos testes realizados no Cenário 1 foram realizados nas máquinas do Cenário 2. A próxima seção apresenta os resultados dos testes realizados sobre o Cenário 2.

4.2.2. Testes sobre Cenário 2

Esta seção apresenta os testes que foram realizados sobre o Cenário 2, com a utilização do *cluster*, no qual os clientes se conectam a aplicação através do balanceador e o mesmo distribui as requisições entre os outros nós do *cluster* (servidor 1 e servidor 2). A Figura 14 apresentada na seção 4.1 ilustra o Cenário do *cluster*. Foram aplicados testes nas quatro situações relacionadas na seção 4.2 foram aplicadas, sendo que os testes de *stress* foram repetidos em três momentos distintos.

Serão apresentados os testes da situação 1, com 10 usuários, mostrando os resultados do balanceador, servidor 1 e servidor 2 para o primeiro momento, os demais resultados serão apresentados na Tabela 2.

A sequência seguida foi simular os acessos, no qual as solicitações são encaminhadas ao balanceador e o mesmo realiza a distribuição das requisições dos usuários entre o servidor 1 e servidor 2. As Figuras 24, Figura 25 e 26 apresentam os requisitos que foram analisados para avaliar o desempenho da aplicação, uso de *CPU*, memória e disco, respectivamente, no balanceador e nos nós (servidor 1 e servidor 2).

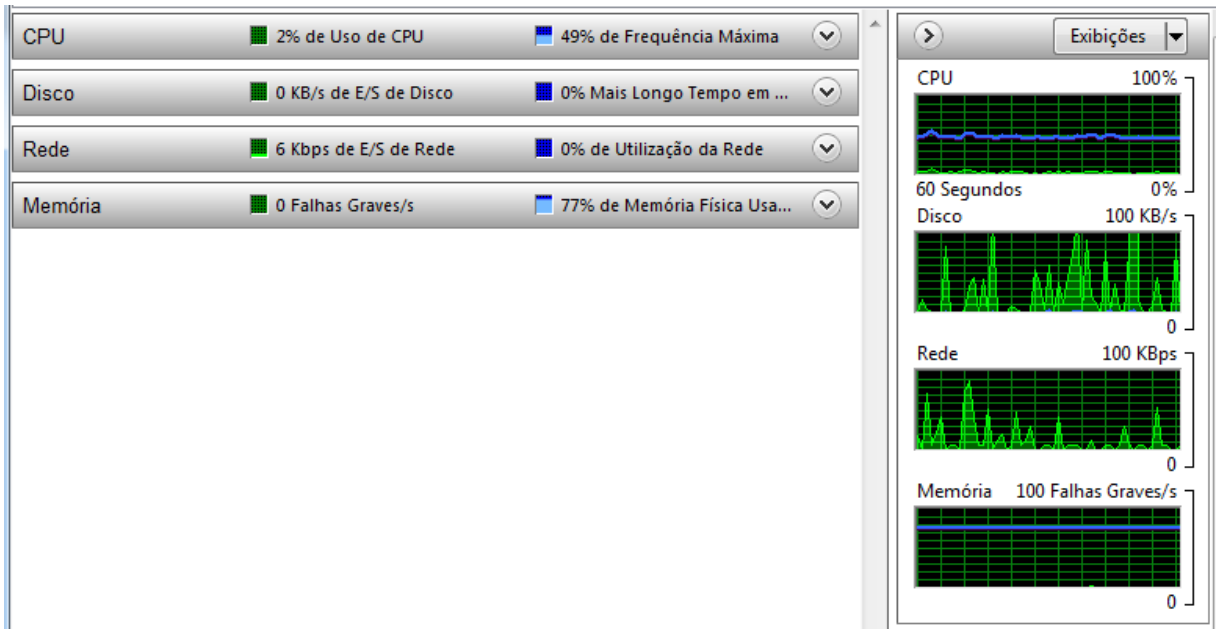


Figura 24 – 10 usuários – Balanceador

A Figura 24 apresenta a tela do monitor de recurso do *Windows*, com as informações do balanceador, no qual se pode observar que: 2% de uso de *CPU*, 77% de memória utilizada e fluxo de disco de 0 KB /s.

A Figura 25 apresenta as informações dos requisitos analisados ao ter o acesso dos 10 usuários simultâneos no servidor 1.

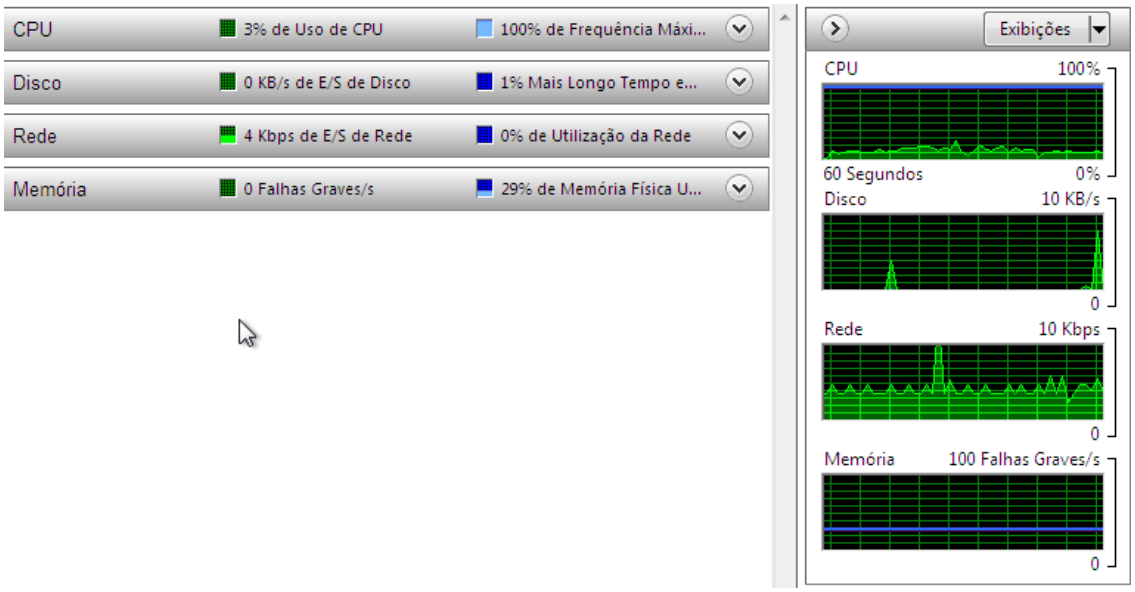


Figura 25 – 10 usuários - Servidor 1

A Figura 25 apresenta a tela do monitor de recurso do *Windows*, com as informações do servidor 1, no qual se pode observar que: o processamento da *CPU* tem o uso em 3%, memória com 29% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 0 KB/s.

A Figura 26 apresenta as informações dos requisitos analisados ao ter o acesso dos 10 usuários simultâneos no servidor 2, e não se observou grandes alterações dos requisitos analisados em relação ao balanceador e ao servidor 1.

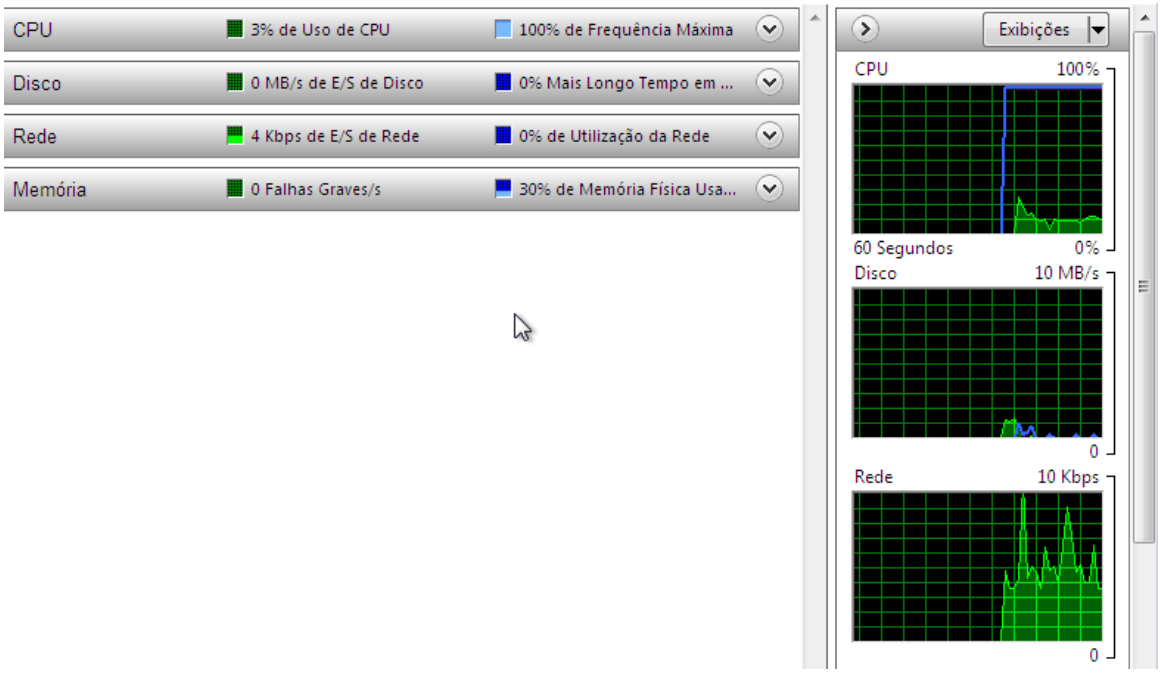


Figura 26 – 10 usuários - Servidor 2

A Figura 26 apresenta a tela do monitor de recurso do *Windows*, com as informações do servidor 2, no qual se pode observar que: o processamento da *CPU* tem o uso em 3%, memória com 30% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 0 KB/s.

O tempo gasto pela ferramenta que simula os usuários virtuais para gerar o *stress* (*Jmeter*) na aplicação, para atender todas as requisições dos 10 usuários utilizando o *cluster* foi de sete segundos.

Nos outros dois momentos da situação 1 e nos três momentos das situações 2, 3 e 4 os procedimentos de testes realizados foram os mesmos, sendo que os resultados são apresentados na Tabela 2 . A descrição de todas as situações, nos três momentos, é apresentada de forma mais detalhada no apêndice B.

Tabela 2 - Cenário 2

RESULTADOS – CENÁRIO **2**										
SITUAÇÃO	**RECURSO**	**MOMENTOS**								
		BALANCEADOR			**SERVIDOR 1**			**SERVIDOR 2**		
		1	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
10 Usuários	Uso de *CPU (%)*	2	3	2	3	4	3	3	5	4
	Uso de Memória *(%)*	77	75	76	29	31	30	30	32	29
	Fluxo de Disco (KB/s)	0	1	2	0	2	1	0	1	3
	Tempo de Resposta	7s	10s	8s	7s	10s	8s	7s	10s	8s
100 Usuários	Uso de *CPU (%)*	32	31	33	12	15	14	13	14	16
	Uso de Memória *(%)*	80	79	78	36	35	34	39	38	37
	Fluxo de Disco (KB/s)	16	14	20	0	1	2	8	10	7
	Tempo de Resposta	33s	36s	40s	33s	36s	40s	33s	36s	40s
1000 Usuários	Uso de *CPU (%)*	67	68	70	62	61	60	59	60	58
	Uso de Memória *(%)*	84	80	83	37	38	37	34	35	37
	Fluxo de Disco (KB/s)	3413	2350	3100	7	30	50	64	10	40
	Tempo de Resposta	3min 10s	3min 45s	3min 25s	3min 10s	3min 45s	3min 25s	3min 10s	3min 45s	3min 25s

10000 Usuários	Uso de *CPU (%)*	62	60	61	100	100	100	100	100	100
	Uso de Memória *(%)*	76	78	75	64	65	68	64	65	63
	Fluxo de Disco (KB/s)	943	345	465	1024	900	45	24	450	36
	Tempo de Resposta	16min 18s	20min 10s	19min 24s	16min 18s	20min 10s	19min 24s	16min 18s	20min 10s	19min 24s

A Tabela 2 apresenta os valores dos requisitos que foram identificados no momento dos testes de cada situação. Como pode ser observado, nos três momentos dos testes com 10.000 usuários, o uso de CPU atingiu 100% para os servidores 1 e 2.

Ainda na Tabela 2, pode-se verificar que apesar de ter o uso de CPU de 100% com 10.000 usuários, em todas as situações requisições foram atendidas.

Após a realização dos testes de *stress* sobre os servidores do Cenário 2, com uso do *cluster*, para que fosse feito um comparativo entre os requisitos analisados do Cenário 1 e Cenário 2, foram realizados os testes de disponibilidade, apresentados na próxima seção.

4.3 – Testes de Disponibilidade

Essa seção apresenta os testes de disponibilidade (*online*), ou seja, por quanto tempo permanecem ligados, que foram realizados sobre o Cenário 2, com a utilização do *cluster*. Em relação ao Cenário 1 não foi preciso realizar os testes de disponibilidade, pois se trata de um único servidor, caso o mesmo fosse desligado, com certeza, não responderia mais as requisições dos usuários.

Esse teste consistiu em desligar um dos servidores que estava executando a aplicação *JAMWIKI* e atendendo requisições dos clientes. Esse desligamento visa a simular uma falha ou pane na máquina, que faça com que esta não consiga mais atender requisições.

No cluster, as requisições dos usuários são atendidas pelo balanceador, que distribui as requisições entre dois nós (servidor 1 e servidor 2). Sobre esse cenário, foram realizados os seguintes testes:

- Desligou-se o servidor 1 – ao desligar o servidor 1, observou-se que todas as requisições recebidas pelo balanceador foram direcionadas apenas para o servidor 2 e no Jk_status o servidor 1 recebeu o status ERR(Erro);

- Desligou-se o servidor 2 - ao desligar o servidor 2, observou-se que todas as requisições recebidas pelo balanceador foram direcionadas apenas para o servidor 1 e no Jk_status o servidor 2 recebeu o status ERR(Erro);
- Desligou-se o balanceador – esse foi o último teste de disponibilidade, a partir do qual se observou que todas as requisições deixaram de ser atendidas.

Após realizar os testes de disponibilidade, observou-se que existe ganho na disponibilidade quando se utiliza o *cluster*, no que diz respeito aos servidores que disponibilizam a aplicação Web. Mas é importante enfatizar que existe um ponto de falha, que é o balanceador do *cluster*, pois, caso o mesmo fique indisponível, as requisições não serão encaminhadas para os servidores, de forma que todas elas deixarão de ser atendidas.

Para facilitar a compreensão dos resultados, a próxima seção apresenta um paralelo entre os testes realizados sobre o Cenário 1 e o Cenário 2, ou seja, sem a utilização do *cluster* e com a utilização do *cluster*.

4.4 – Paralelo entre os resultados dos testes sobre os dois cenários

Esta seção apresenta o comparativo dos parâmetros obtidos nos testes realizados sobre o Cenário 1 e o Cenário 2. Para facilitar a compreensão, recurso é apresentada uma tabela comparativa, que apresenta os valores obtidos. Os parâmetros analisados, que são apresentados na tabela, para fazer a comparação foram: o uso de *CPU, o uso* de memória, fluxo de escrita e leitura do disco e tempo de resposta. Os valores referentes a esses parâmetros, apresentados na Tabela 3, foram obtidos nos testes realizados e apresentados anteriormente nas Tabelas 1 e 2.

Como os testes foram realizados em três momentos distintos, apresentar todos os resultados seria inviável, pois a tabela ficaria muito extensa, definiu-se que a Tabela 3 apresentaria a média dos valores obtidos para os parâmetros, nos três momentos de cada cenário.

Tabela 3 – Cenário 1 X Cenário 2

QUANTIDADE DE USUÁRIOS	RECURSO	RESULTADOS		
		Média - Cenário 1	Média - Cenário 2	Ganho (%)
10 Usuários	Uso de *CPU (%)*	10	3	70
	Uso de Memória *(%)*	48	30	62
	Fluxo de Disco (KB/s)	12	1	
	Tempo de Resposta	1min 4s	8s	
100 Usuários	Uso de *CPU (%)*	46	14	70
	Uso de Memória *(%)*	59	36	61
	Fluxo de Disco (KB/s)	80	4	
	Tempo de Resposta	12min 50s	36s	
1000 Usuários	Uso de *CPU (%)*	100	60	40
	Uso de Memória *(%)*	73	36	49
	Fluxo de Disco (KB/s)	1946	33	
	Tempo de Resposta	1h 12min 34s	3min 26s	
10000 Usuários	Uso de *CPU (%)*	-	100	
	Uso de Memória *(%)*	-	64	
	Fluxo de Disco (KB/s)	-	413	
	Tempo de Resposta	-	18min 18s	
			Geral	=~ 40%

Conforme se pode observar na Tabela 3, existem vantagens na utilização do *cluster*, no que se refere a desempenho:

- O uso de CPU no *cluster* sempre foi menor do que sem a utilização do *cluster,* por exemplo, no acesso simultâneo dos 1.000 usuários sem a utilização do *cluster* o uso foi de 100% enquanto usando o *cluster* o uso apresentado foi de 62%;
- A memória utilizada também se obteve menor utilização, por exemplo, no acesso simultâneo dos 1.000 usuários sem a utilização do *cluster* o uso foi de 61% enquanto usando o *cluster* o uso apresentado foi de 37%;
- E, por fim, sem a utilização do *cluster*, o servidor único não conseguiu atender as requisições dos 10.000 usuários ao contrário da utilização do *cluster*.

Em relação à disponibilidade, foi apresentado anteriormente que existe um ganho ao se utilizar o *cluster* no que diz respeito aos servidores. Porém, há um ponto de falha, que é o balanceador.

Todos estes testes foram realizados em um ambiente virtual e também real no caso do balanceador, no qual as máquinas estavam executando apenas a aplicação e não havia tráfego de rede influenciando na comunicação do balanceador com os

servidores. Esse fato poderia alterar os resultados caso houvesse uma quantidade relevante de tráfego de rede. Isto porque o tráfego poderia deixar a comunicação entre o balanceador e os servidores mais lenta, diminuindo assim a diferença do tempo de resposta tanto sem *cluster* quanto no uso do *cluster*. Apesar disso, caso a arquitetura seja construída em um ambiente real, a tendência é que ofereça vantagens semelhantes às apresentadas nos resultados deste trabalho.

5. CONSIDERAÇÕES FINAIS

O propósito deste trabalho foi construir um *cluster* com *Apache Tomcat* para realizar o balanceamento de carga de requisições recebidas de uma aplicação *Web*. O intuito de se implantar uma estrutura em *cluster* é melhorar o desempenho e disponibilidade no acesso à aplicação. Com a realização deste trabalho pode-se verificar se a melhoria desejada seria realmente conseguida ao se utilizar um *cluster*.

Após a realização dos testes pode-se considerar que houve um aumento de aproximadamente 40% a mais de desempenho em relação à aplicação *Web* executada pelo o Apache Tomcat. Já no que se trata da disponibilidade como existe um ponto falho no *cluster*, que é o balanceador a disponibilidade deixou a desejar, o problema que antes era no servidor único, ao utilizar-se o *cluster* passou-se a ser no balanceador.

Um requisito que não foi analisado neste trabalho, mas é importante quando se fala de balanceamento de carga, foi o de rede, isso porque foram utilizados ambientes virtuais nos testes, sem simular ou considerar o tráfego de rede. Com isso, caso ocorresse um grande tráfego de rede poderia prejudicar assim o funcionamento do *cluster* deixando o acesso mais lento, mas mesmo assim a tendência é que com o uso do *cluster* melhore o desempenho.

Um problema identificado ao realizar o trabalho foi à quantidade de material encontrado para realizar a construção do cluster no sistema operacional *Windows* escolhido para o trabalho, praticamente todos os materiais faziam referência ao sistema operacional *Linux*. Para realizar então os procedimentos de configuração do *cluster* no sistema *Windows* foi feita uma adaptação dos materiais encontrados para *Linux*.

Dois trabalhos que poderiam ser realizados futuramente são: instalar e configurar o *cluster Tomcat* em sistemas operacionais distintos e simulando um tráfego de rede caso fosse utilizado um ambiente virtual ou até mesmo realizar as configurações do *cluster* num ambiente real. Com isso seria interessante analisar os resultados do *cluster Tomcat* com sistemas operacionais variados, analisando assim o comportamento dos nós do *cluster*. Um terceiro trabalho também poderia ser realizado em cima da falha encontrada no balanceador do *cluster*, para que a mesma fosse solucionada.

6. REFERÊNCIAS BIBLIOGRÁFICAS

DOURADO FLYNN, M. J. Some computer organizations and their effectiveness. IEEETransactions on Computing, n. 9, p. 948–960, 1972.

GORINO, Fabio Valle Rego. Universidade Estadual de Maringá. Balanceamento de carga em *Cluster*s de alto desempenho: uma extensão para LAM/MPI. Disponível em: <http://www.din.uem.br/~mestrado/diss/2006/gorino.pdf>, Acesso em 15 de outubro de 2012.

HOGANSON, K. E. Workload execution strategies and parallel speedup on *cluster*ed computers. IEEE Transactions Computers. Volume 48, número 11, pág. 1173 – 1182, novembro de 1999.

ILRI High Performance Computing, 2005. Disponível em: <http://hpc.ilri.cgiar.org/img/hpc_graphic.gif>, Acesso em 10 de outubro de 2012.

Microsoft, 2012. Disponível em <http://support.microsoft.com/library/images/support/kbgraphics/public/EN-US/*Windows*XP/Networking/*hardware*006.gif>, Acesso em 25 de novembro de 2012.

MOODIE, Matthew. Pro *Apache Tomcat* 6.New York.Apress 2007.325p

NGUYEN, V. A. K.; PIERRE, S. Scalability of computer *cluster*s. Electrical and Computer Engineering. Canadian Conference on Volume 1, pág. 405 – 409, de 13 a 16 de maio de 2001.
SCHLEMER, E. Escalonamento em sistemas distribuídos: uma visão geral. TI-790 CPG-CC-UFRGS,1999.

Oracle. Java Servlet Technology Overview. Disponível em: <http://www.oracle.com/technetwork/java/overview-137084.html >, Acesso em 14 de outubro de 2012.

PILLA SMITH, Bianca. 2010. Disponível em: <http://online.unisc.br/seer/index.php/cepe/article/viewFile/1449/1236>, Acesso em 26 de outubro de 2012.

OLIVEIRA, Jorge Ricardo Souza de. ILES Santarém. Uma proposta para balanceamento de carga no CORBA. Espaço Científico, Santarém - Vol. 2 n 1,2 - jan. /dez. 2001 páginas 35 a 44.

ROQUE, Alexandre dos Santos. 2012. Taxonomia de Flynn. Disponível em: <http://srvapp2s.santoangelo.uri.br/labcd/wp-content/uploads/2012/04/Taxonomia_de_Flynn.pdf>, Acesso em 18 de novembro de 2012.

STERLING, T. An Introduction to PC *Cluster*s for High Performance Computing. Conforme consta em *Cluster* Computing White Paper, Final Release, Version 2.0, Editor Mark Baker. University of Portsmouth, UK. 2000.

Stor IT Back - Ihr Speicherspezialist, 2011. Disponível em: <www.storitback.de/anim_gif/*cluster*-ha.gif >, Acesso em 10 de outubro de 2012.

TANENBAUM, Andrew S. Organização Estruturada de Computadores. Rio de Janeiro, LTC, 2001

The *Apache Software* Foundation.*Apache Tomcat*.Disponível em: <http://*Tomcat*.*Apache*.org/>. Acesso em 20 de outubro de 2012.

Thomas Computer Corporation, 2002. Disponível em: <http://www.thomascomputer.com/catalog/furniture/balt/prod-503.html>, Acessado 12 de novembro de 2012.

JAMWIKI (2013) "*JAMWIKI* Java Wiki Engine". Disponível em: <http://*jamwiki*.org/ >, Acessado em 2013.

The *Apache Software* Foundation.*Apache Jmeter*. Disponível em: <http://*Jmeter*.*Apache*.org/>. Acesso em 2013.

APÊNDICE A

A Figura 27 apresenta as informações dos requisitos analisados ao ter o acesso dos 10 usuários simultâneos no segundo momento, onde observou-se pouca diferença entre os requisitos analisados.

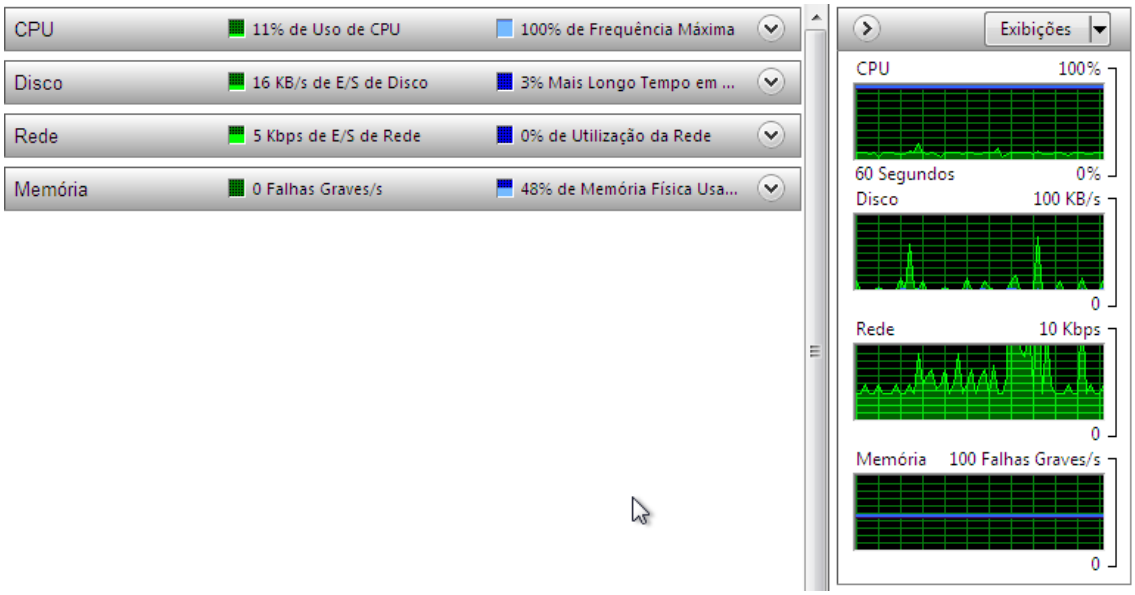


Figura 27 - 10 usuários - Segundo Momento

A Figura 27 apresenta as informações do segundo momento, onde o processamento da *CPU* manteve-se com baixo processamento somente 11% de uso, memória com 48% de utilização e não houve falhas e por fim disco com leitura e escrita em torno de 16 KB/s.

A Figura 28 apresenta as informações dos requisitos analisados ao ter o acesso dos 10 usuários simultâneos no terceiro momento, e não se observou grandes alterações dos requisitos analisados em relação ao primeiro e segundo momento.

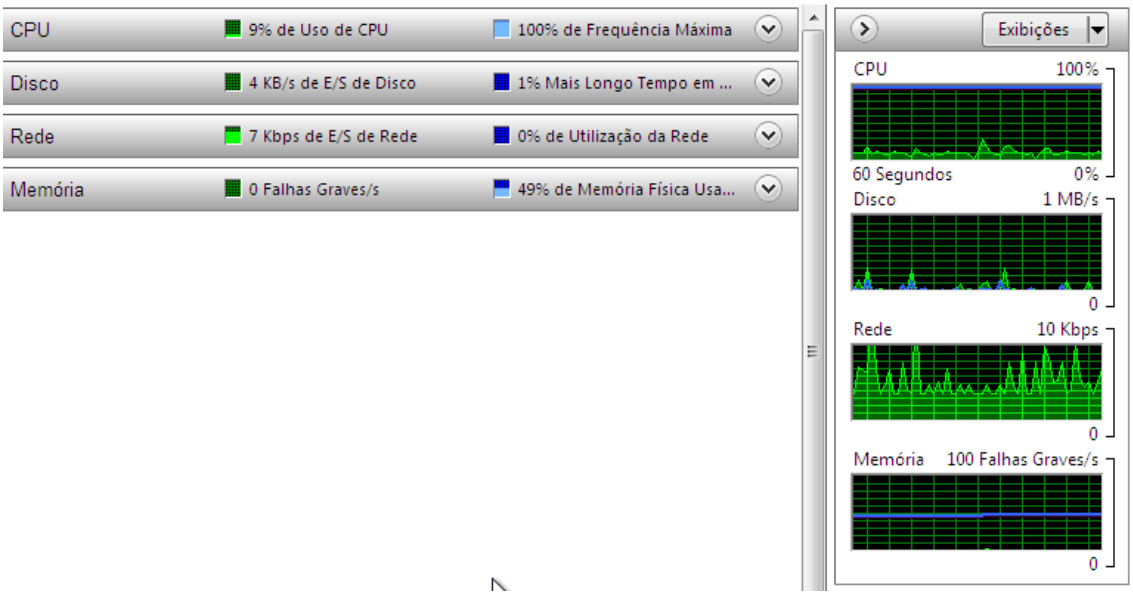


Figura 28 - 10 usuários - Terceiro Momento

A Figura 28 traz as informações do terceiro momento, onde observa-se o processamento da *CPU* manteve-se com baixo processamento somente 9% de uso, memória com 49% de utilização e não houve falhas e por fim disco com leitura e escrita em torno de 4 KB/s.

O tempo gasto pela ferramenta que simula os usuários virtuais para gerar o *stress* (*Jmeter*) na aplicação, para atender todas as requisições dos 10 usuários foi de um minuto e três segundos.

Situação 2 – 100 usuários virtuais

A sequência seguida com 100 usuários foi à mesma da anterior. As figuras 29, 30 e 31 apresentam resultados nos três momentos distintos para 100 usuários acessando simultaneamente:

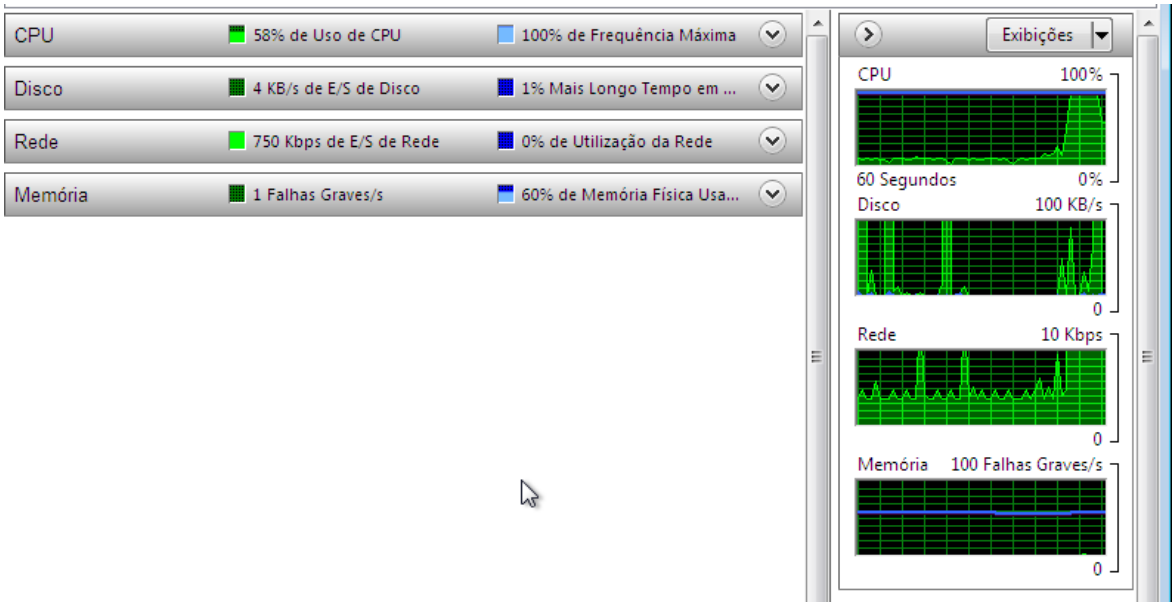


Figura 29 - 100 usuários - Primeiro Momento

A Figura 29 apresenta as informações do primeiro momento, onde se pode observar que o processamento da *CPU* já é maior que com 10 usuários no quesito uso de *CPU* é de 58% de uso, memória com 60% de utilização e houve uma falha grave e por fim leitura e escrita em disco com o valor de 4 KB/s.

A Figura 30 apresenta as informações dos requisitos analisados ao ter o acesso dos 100 usuários simultâneos no segundo momento, e não se observou grandes alterações dos requisitos analisados em relação ao primeiro momento.

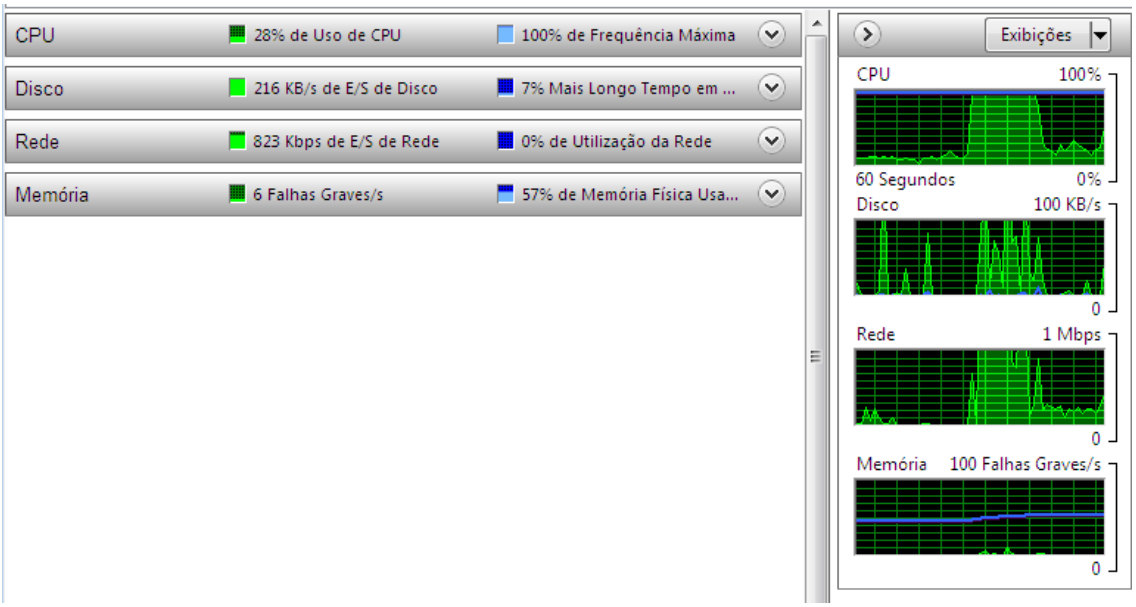


Figura 30 - 100 usuários - Segundo Momento

A Figura 30 apresenta as informações do segundo momento, onde se pode observar que o processamento da *CPU* é de 28% de uso, memória com 57% de utilização e houve seis falhas graves e por fim disco com leitura e escrita em torno de 216 KB/s.

A Figura 31 apresenta as informações dos requisitos analisados ao ter o acesso dos 100 usuários simultâneos no terceiro momento, e não se observou grandes alterações dos requisitos analisados em relação ao primeiro e segundo momento.

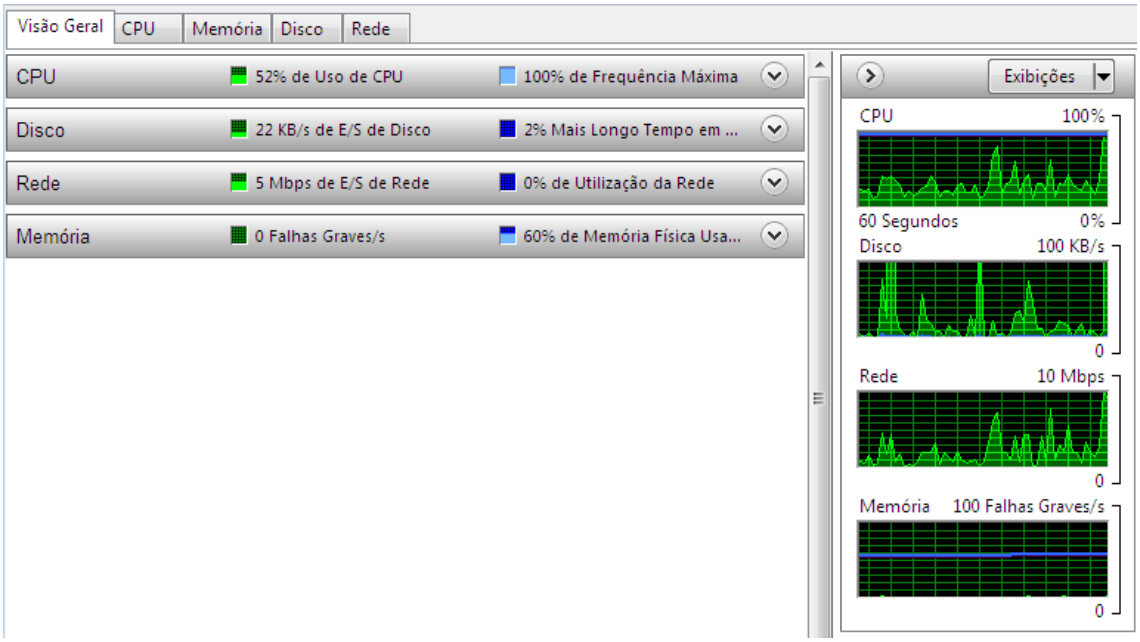


Figura 31 - 100 usuários - Terceiro Momento

A Figura 31 apresenta as informações do terceiro momento, onde se pode observar que o processamento da *CPU* é de 52% de uso, memória com 60% de utilização e não houve falhas graves e por fim disco com leitura e escrita em torno de 22 KB/s.

O tempo gasto pela ferramenta que simula os usuários virtuais para gerar o *stress* (*Jmeter*) na aplicação, para atender todas as requisições dos 100 usuários foi de doze minutos e quinze segundos.

Situação 3 – 1000 usuários virtuais

A sequência seguida com 1000 usuários foi à mesma utilizada com 10 usuários. As figuras 32, 33 e 34 apresentam resultados nos três momentos distintos para 1000 usuários acessando simultaneamente:

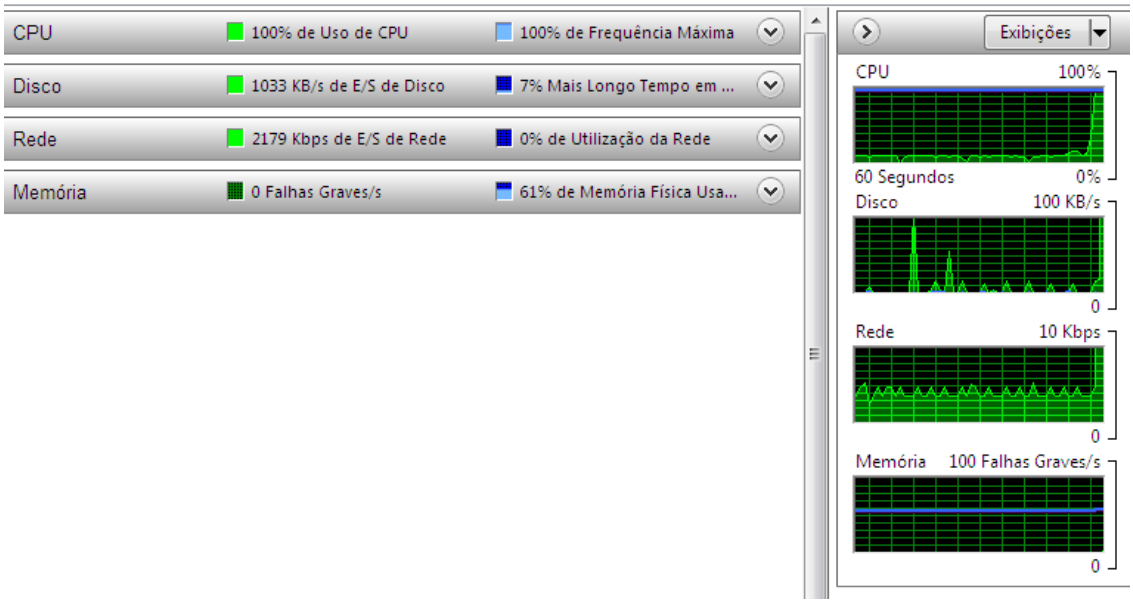


Figura 32 - 1000 usuários - Primeiro Momento

A Figura 32 apresenta as informações do primeiro momento, onde se pode observar que o processamento da *CPU* tem o uso de 100% , memória com 61% de utilização e por fim disco com fluxo de escrita e leitura de 1033 KB/s.

A Figura 33 apresenta as informações dos requisitos analisados ao ter o acesso dos 1000 usuários simultâneos no segundo momento, e não se observou grandes alterações dos requisitos analisados em relação ao primeiro momento.

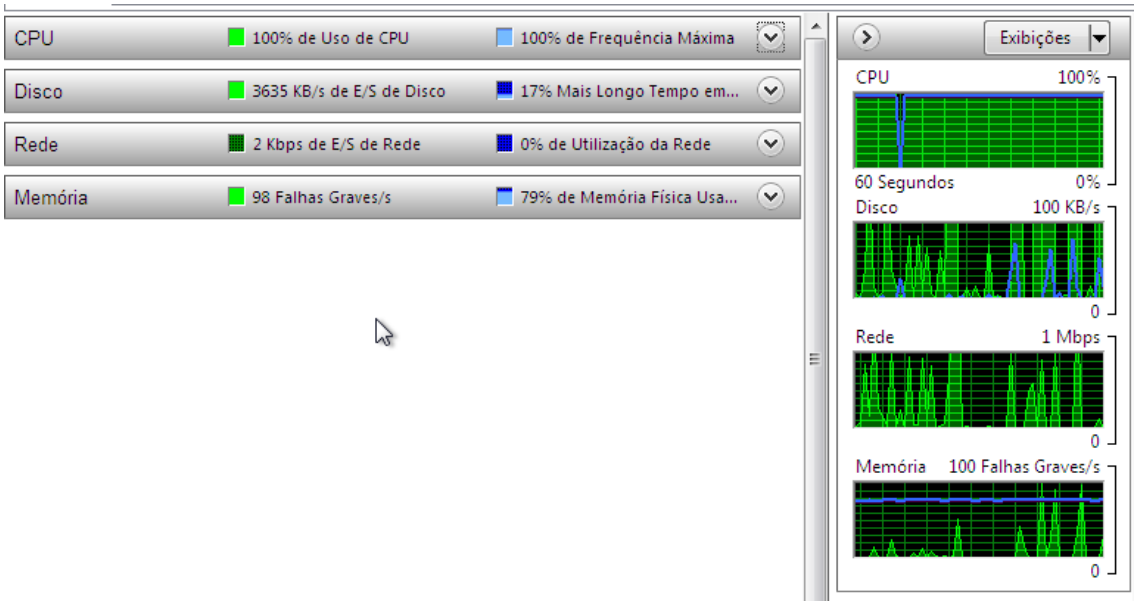


Figura 33 - 1000 usuários - Segundo Momento

A Figura 33 apresenta as informações do segundo momento, onde se pode observar que o processamento da *CPU* tem o uso em 100%, memória com 79% de utilização e 98 falhas graves e por fim disco com fluxo de leitura e escrita de 3635 KB/s.

A Figura 34 apresenta as informações dos requisitos analisados ao ter o acesso dos 1000 usuários simultâneos no terceiro momento, e não se observou grandes alterações dos requisitos analisados em relação ao primeiro e segundo momento.

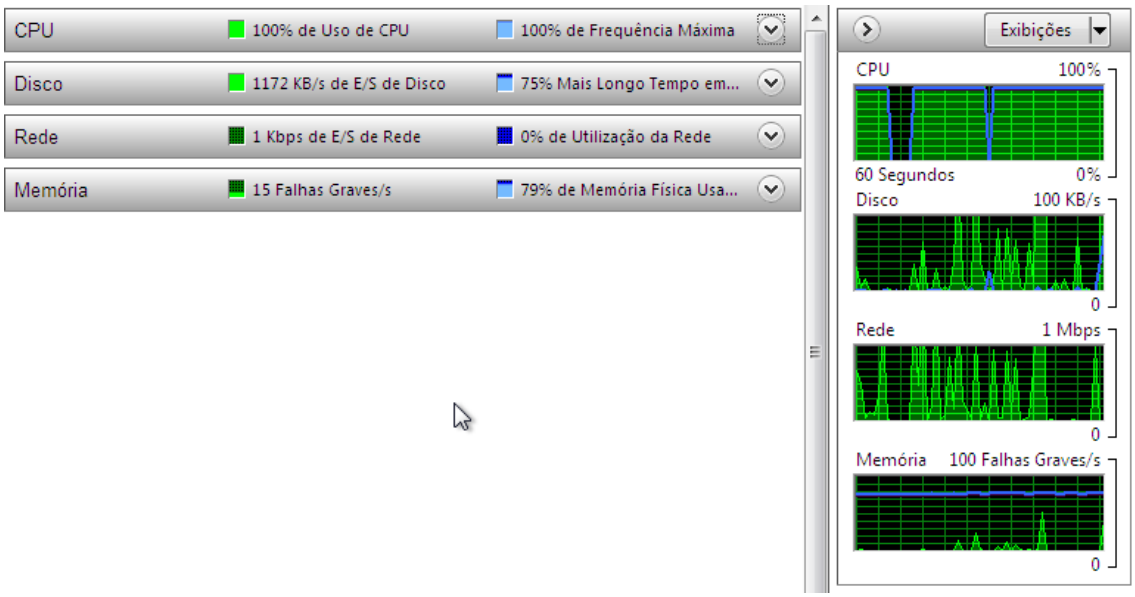


Figura 34 - 1000 usuários - Terceiro Momento

A Figura 34 apresenta as informações do terceiro momento, onde se pode observar que o processamento da *CPU* tem o uso em 100%, memória com 79% de utilização e 15 falhas graves e por fim disco com fluxo de leitura e escrita de 1172 KB/s.

O tempo gasto pela ferramenta que simula os usuários virtuais para gerar o *stress* (*Jmeter*) na aplicação, para atender todas as requisições dos 1000 usuários foi de uma hora, dez minutos e vinte segundos.

Situação 4 – 10.000 usuários virtuais

A sequência seguida com 10000 usuários foi à mesma utilizada com 10 usuários, mas infelizmente não foram possíveis colher informações do processamento e desempenho e muito menos o tempo gasto para atender as requisições, pois nos

três momentos que se tentou utilizar os 10.000 usuários virtuais para acessar simultaneamente, tanto a máquina virtual quanto a máquina real travaram e tiveram que ser reiniciadas.

APÊNDICE B

Situação 2 – 100 usuários virtuais

A sequência seguida foi simular os acessos, no qual as solicitações são encaminhadas ao balanceador e o mesmo realiza a distribuição das requisições dos usuários entre o servidor 1 e servidor 2. As Figuras 35, 36 e 37 apresentam os requisitos que foram analisados para avaliar o desempenho da aplicação, uso de *CPU*, memória e disco, respectivamente, no balanceador e nos nós (servidor 1 e servidor 2).

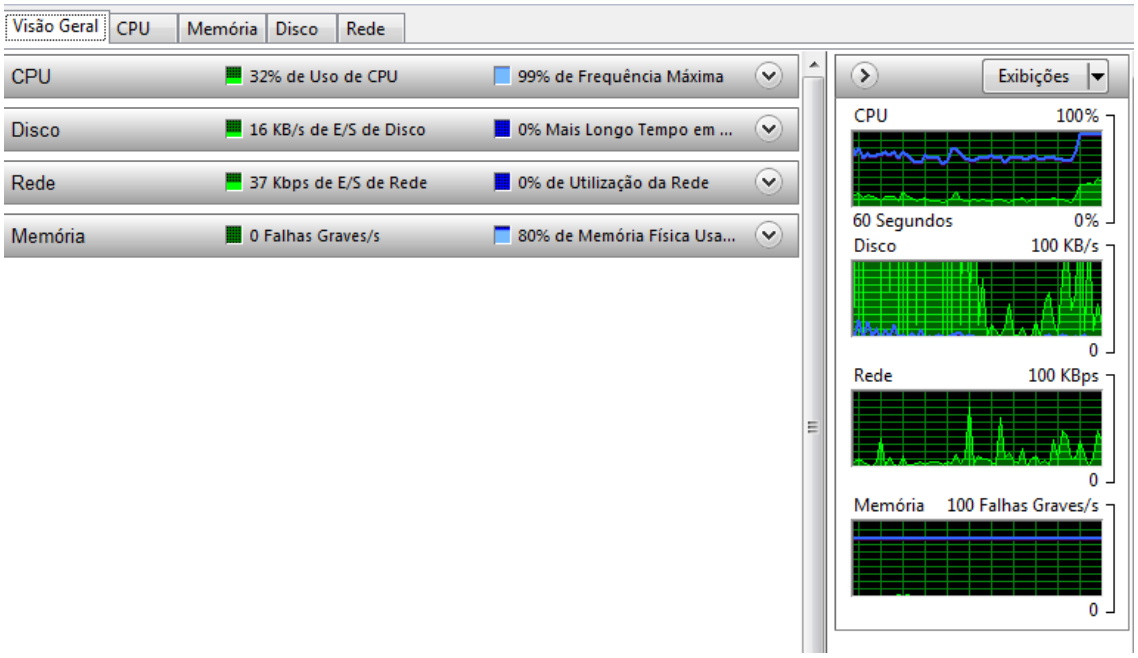


Figura 35 - 100 usuários – Balanceador

A Figura 35 apresenta as informações do balanceador, onde se pode observar que o processamento da *CPU* tem o uso em 32%, memória com 80% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 16 KB/s.

A Figura 36 apresenta as informações dos requisitos analisados ao ter o acesso dos 100 usuários simultâneos no servidor 1, e se observou algumas alterações dos requisitos analisados em relação ao balanceador.

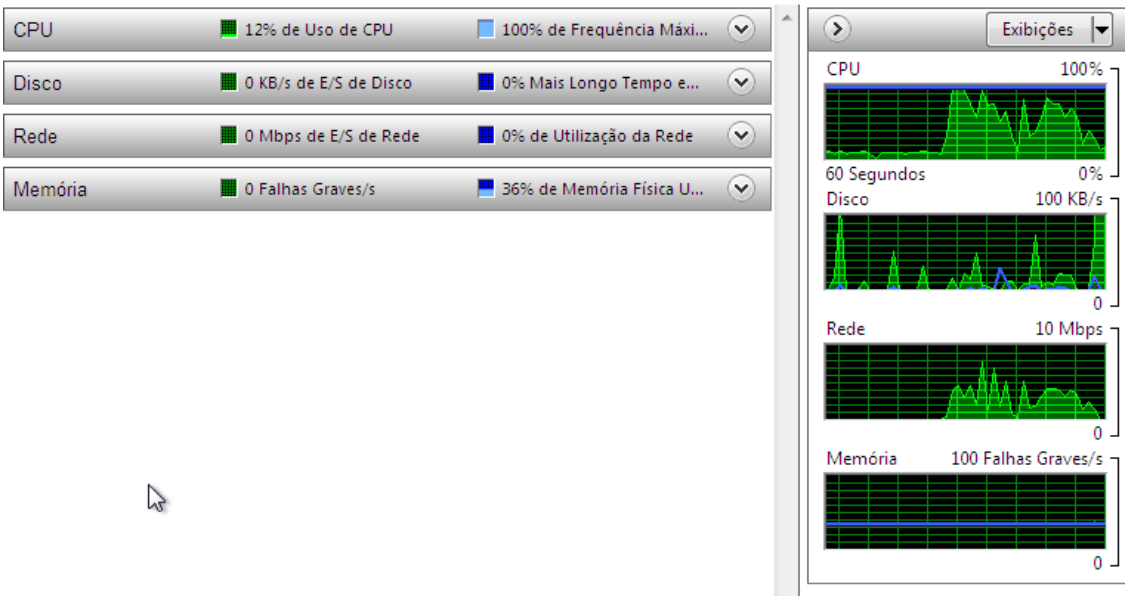


Figura 36 - 100 usuários - Servidor1

A Figura 36 apresenta as informações do servidor 1, onde se pode observar que o processamento da *CPU* tem o uso em 12%, memória com 36% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 0 KB/s.

A Figura 37 apresenta as informações dos requisitos analisados ao ter o acesso dos 100 usuários simultâneos no servidor 2, e se observou algumas alterações dos requisitos analisados em relação ao balanceador e em relação ao servidor 1 não foi observado grandes alterações.

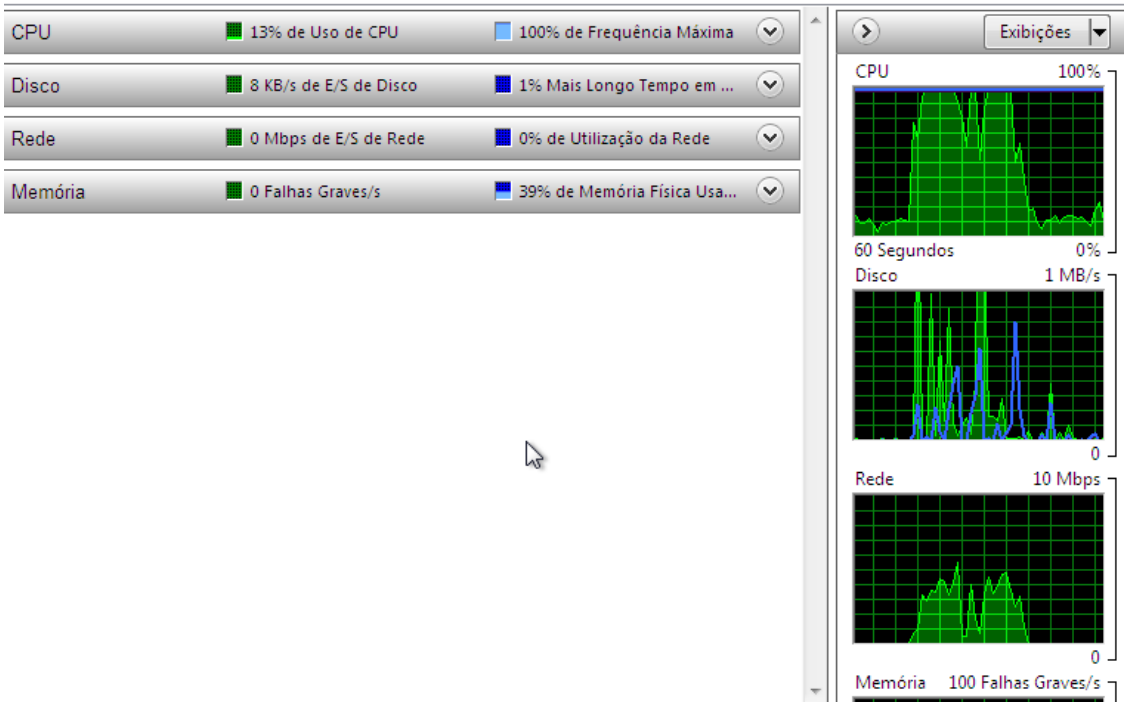


Figura 37 - 100 usuários - Servidor 2

A Figura 37 apresenta as informações do servidor 2, onde se pode observar que o processamento da *CPU* tem o uso em 13%, memória com 39% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 8 KB/s.

O tempo gasto pela ferramenta que simula os usuários virtuais para gerar o stress (*Jmeter*) na aplicação, para atender todas as requisições dos 100 usuários utilizando o *cluster* foi de trinta e três segundos.

Situação 3 – 1000 usuários virtuais

A sequência seguida foi simular os acessos, no qual as solicitações são encaminhadas ao balanceador e o mesmo realiza a distribuição das requisições dos usuários entre o servidor 1 e servidor 2. As Figura 38,Figura 39 e Figura 40 apresentam os requisitos que foram analisados para avaliar o desempenho da aplicação, uso de *CPU*, memória e disco, respectivamente, no balanceador e nos nós (servidor 1 e servidor 2).

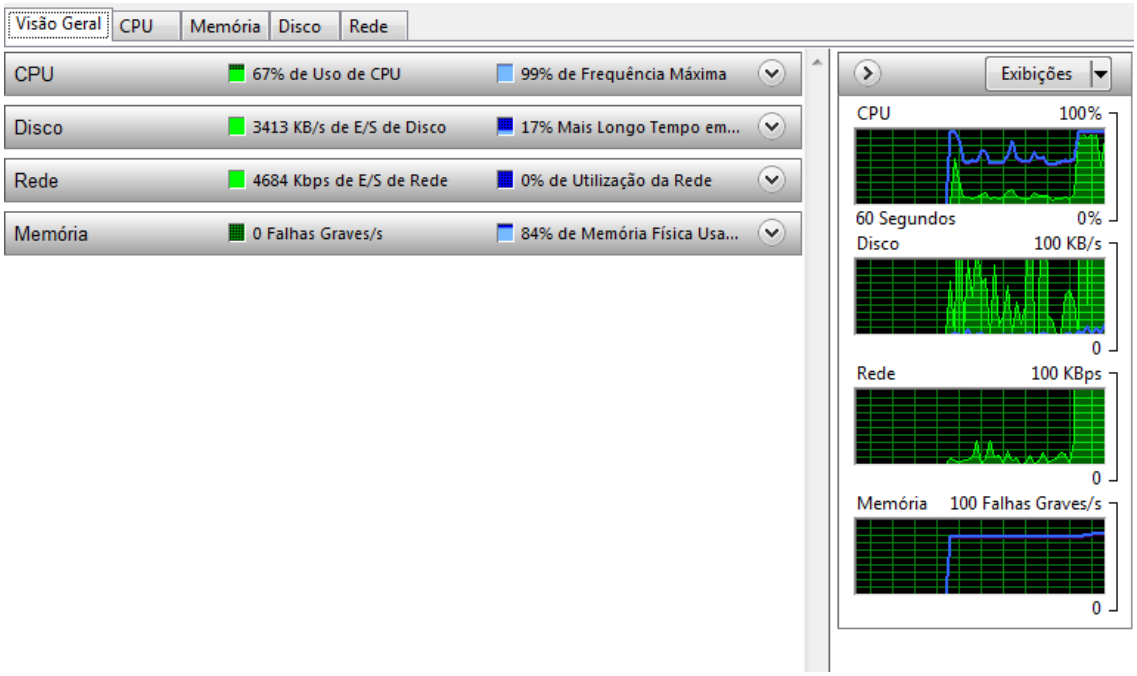


Figura 38 - 1000 usuários – Balanceador

A Figura 38 apresenta as informações do balanceador, onde se pode observar que o processamento da *CPU* tem o uso em 67%, memória com 84% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 3413 KB/s.

A Figura 39 apresenta as informações dos requisitos analisados ao ter o acesso dos 1000 usuários simultâneos no servidor 1, e se observou algumas alterações dos requisitos analisados em relação ao balanceador.

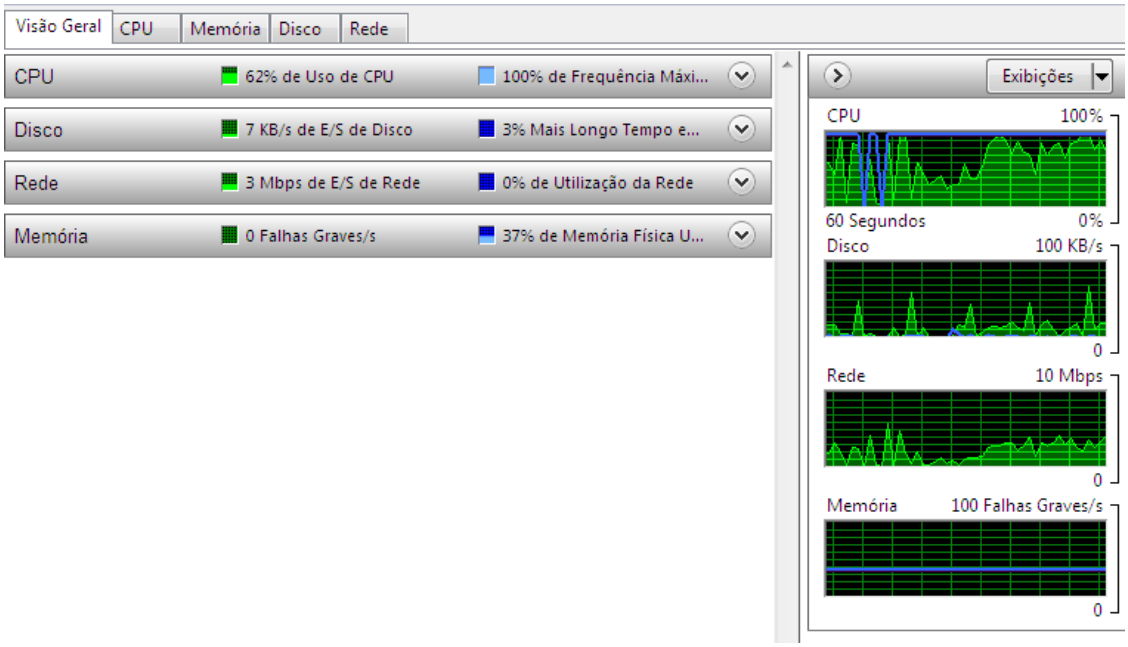


Figura 39 - 1000 usuários - Servidor 1

A Figura 39 apresenta as informações do servidor 1, onde se pode observar que o processamento da *CPU* tem o uso em 62%, memória com 37% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 7 KB/s.

A Figura 40 apresenta as informações dos requisitos analisados ao ter o acesso dos 1000 usuários simultâneos no servidor 2, e se observou algumas alterações dos requisitos analisados em relação ao balanceador e em relação ao servidor 1 não foi observado grandes alterações.

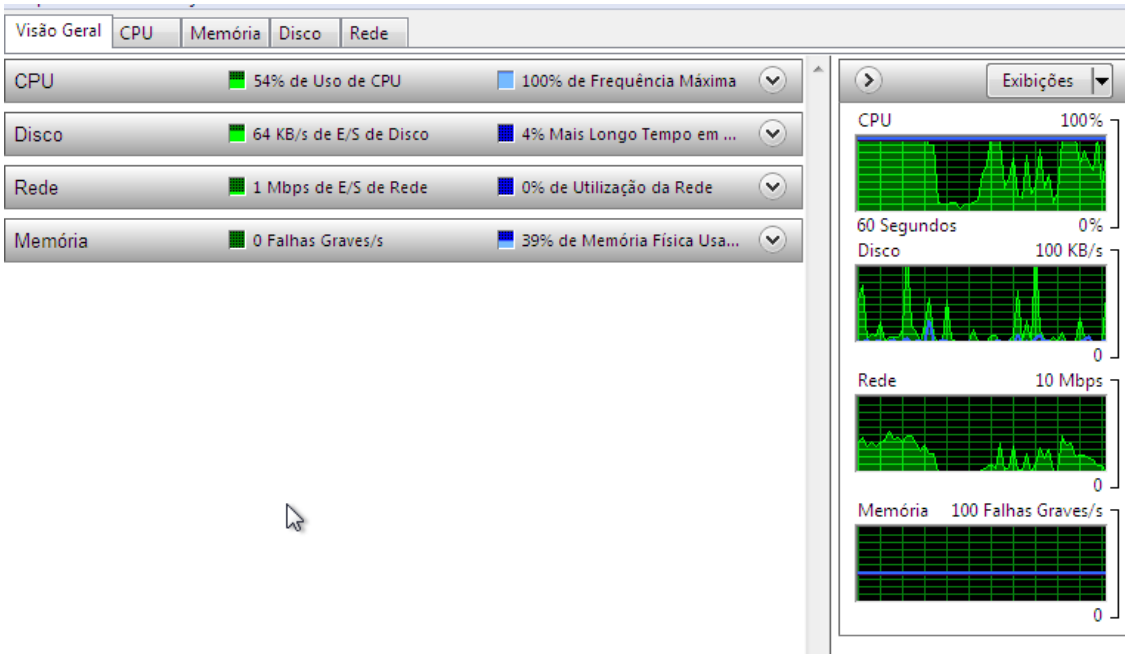


Figura 40 - 1000 usuários - Servidor 2

A Figura 40 apresenta as informações do servidor 2, onde se pode observar que o processamento da *CPU* tem o uso em 54%, memória com 39% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 64 KB/s.

O tempo gasto pela ferramenta que simula os usuários virtuais para gerar o stress (*Jmeter*) na aplicação, para atender todas as requisições dos 1000 usuários utilizando o *cluster* foi de três minutos e dez segundos.

Situação 4 – 10.000 usuários virtuais

A sequência seguida foi simular os acessos, no qual as solicitações são encaminhadas ao balanceador e o mesmo realiza a distribuição das requisições dos usuários entre o servidor 1 e servidor 2. As Figura 41,Figura 42 e Figura 43 apresentam os requisitos que foram analisados para avaliar o desempenho da aplicação, uso de *CPU*, memória e disco, respectivamente, no balanceador e nos nós (servidor 1 e servidor 2).

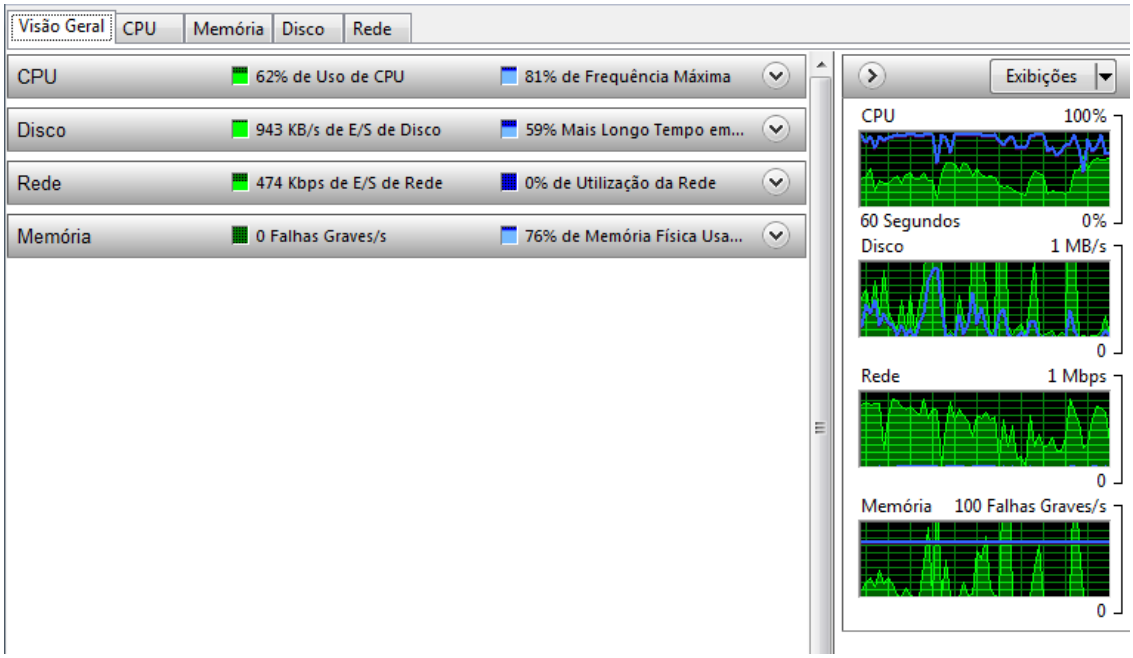


Figura 41- 10000 usuários - Balanceador

A Figura 41 apresenta as informações do balanceador, onde se pode observar que o processamento da *CPU* tem o uso em 62%, memória com 76% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 943 KB/s.

A Figura 42 apresenta as informações dos requisitos analisados ao ter o acesso dos 10000 usuários simultâneos no servidor 1, e se observou algumas alterações dos requisitos analisados em relação ao balanceador.

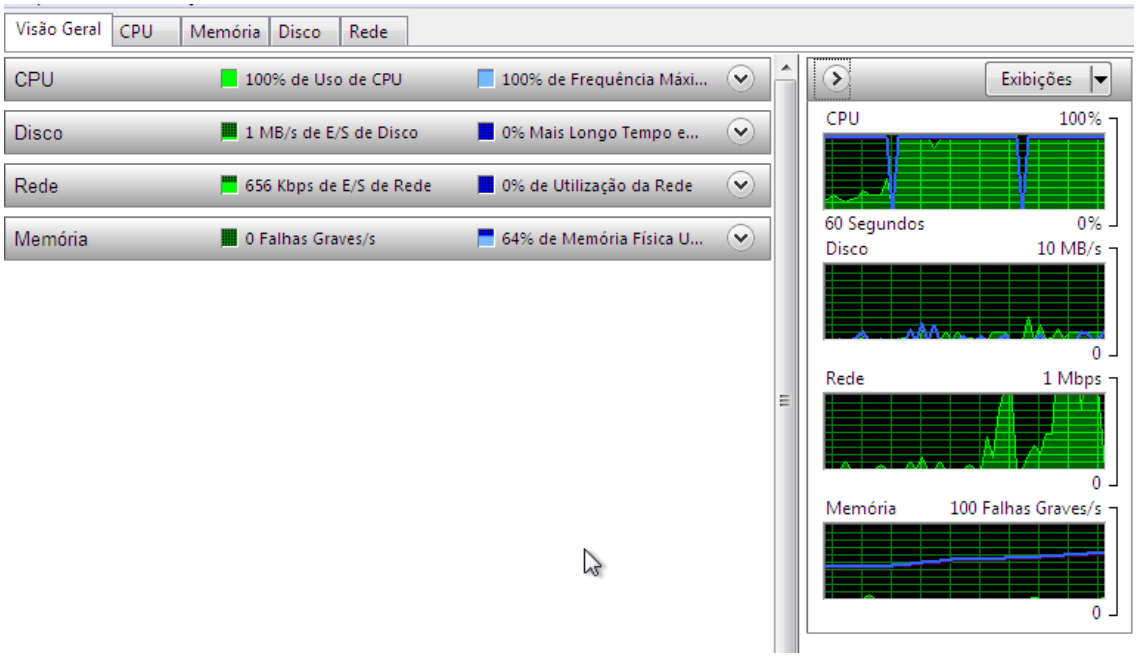


Figura 42- 10000 usuários - Servidor 1

A Figura 42 apresenta as informações do servidor 1, onde se pode observar que o processamento da *CPU* tem o uso em 100%, memória com 64% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 1 MB/s.

A Figura 43 apresenta as informações dos requisitos analisados ao ter o acesso dos 10000 usuários simultâneos no servidor 2, e se observou algumas alterações dos requisitos analisados em relação ao balanceador e em relação ao servidor 1 não foi observado grandes alterações.

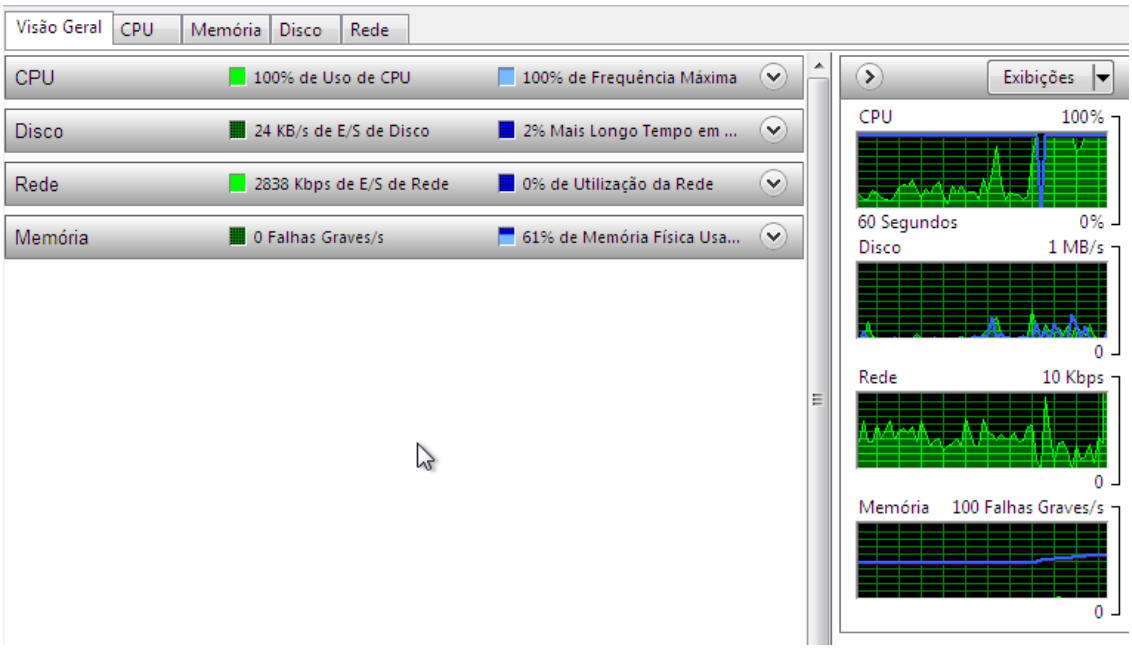


Figura 43- 10000 usuários - Servidor 2

A Figura 43 apresenta as informações do servidor 2, onde se pode observar que o processamento da *CPU* tem o uso em 100%, memória com 61% de utilização e nenhum falha grave e por fim disco com fluxo de leitura e escrita de 24 KB/s.

O tempo gasto pela ferramenta que simula os usuários virtuais para gerar o stress (*Jmeter*) na aplicação, para atender todas as requisições dos 10000 usuários utilizando o *cluster* foi de 16 minutos e dezoito segundos.

A próxima seção irá apresentar os testes de disponibilidade realizados neste trabalho.

APÊNDICE C

Este anexo mostrará os procedimentos necessários para a instalação e configuração do *Apache Tomcat* 7.0 no sistema operacional *Windows* Sete.

É pré-requisito da instalação do *Apache Tomcat*, ou seja, deve ser instalado primeiramente o JRE:

- **JRE[4](composto por bibliotecas JVM[5]) :** pré-requisito do *Apache Tomcat* e utilizado para que aplicações em Java possam ser executadas;

Instalação:

Depois de instalado o JRE, é o momento de instalar o *Apache Tomcat* o mesmo está disponível para download através do link http://*Tomcat*.*Apache*.org/download-70.cgi.

A Figura 44 - Tomcat - primeira telaFigura 44 apresenta a primeira tela da instalação com as explicações do *Apache Tomcat*, selecionar Next:

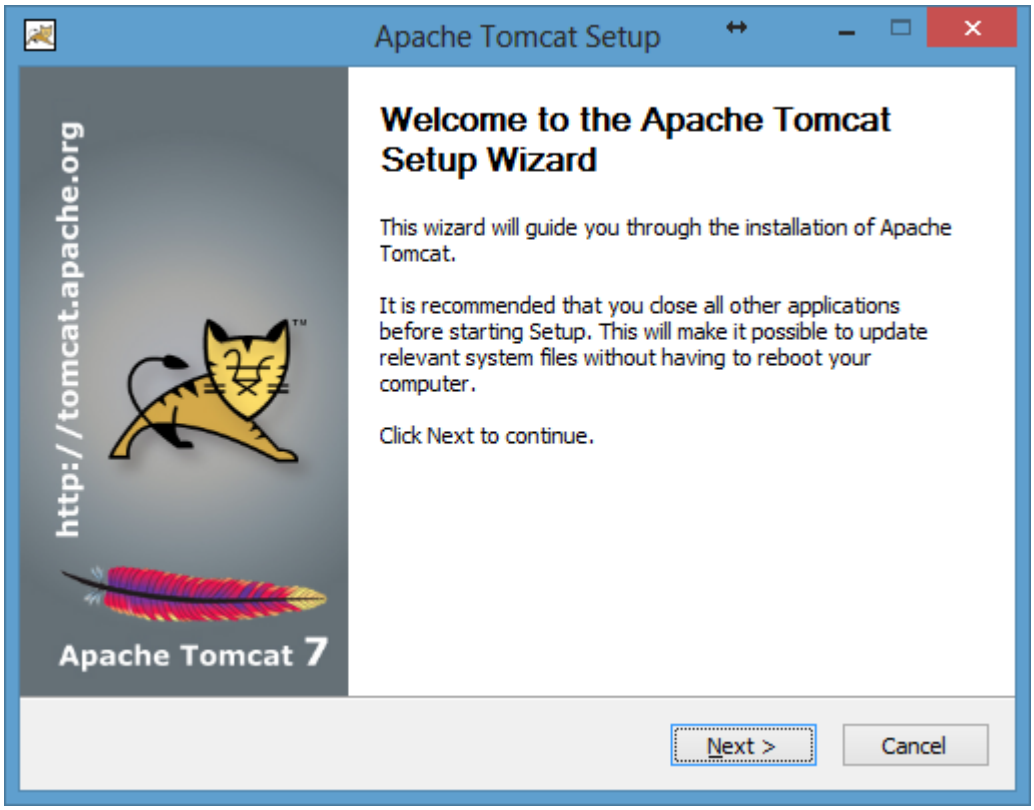


Figura 44 - Tomcat - primeira tela

[4] **Java Runtime Environment**
[5] **Máquina Virtual Java**

A Figura 45 apresenta a segunda tela da instalação apresenta o contrato de instalação do *Tomcat* pela *Apache Software* Foundantion, selecionar I Agree :

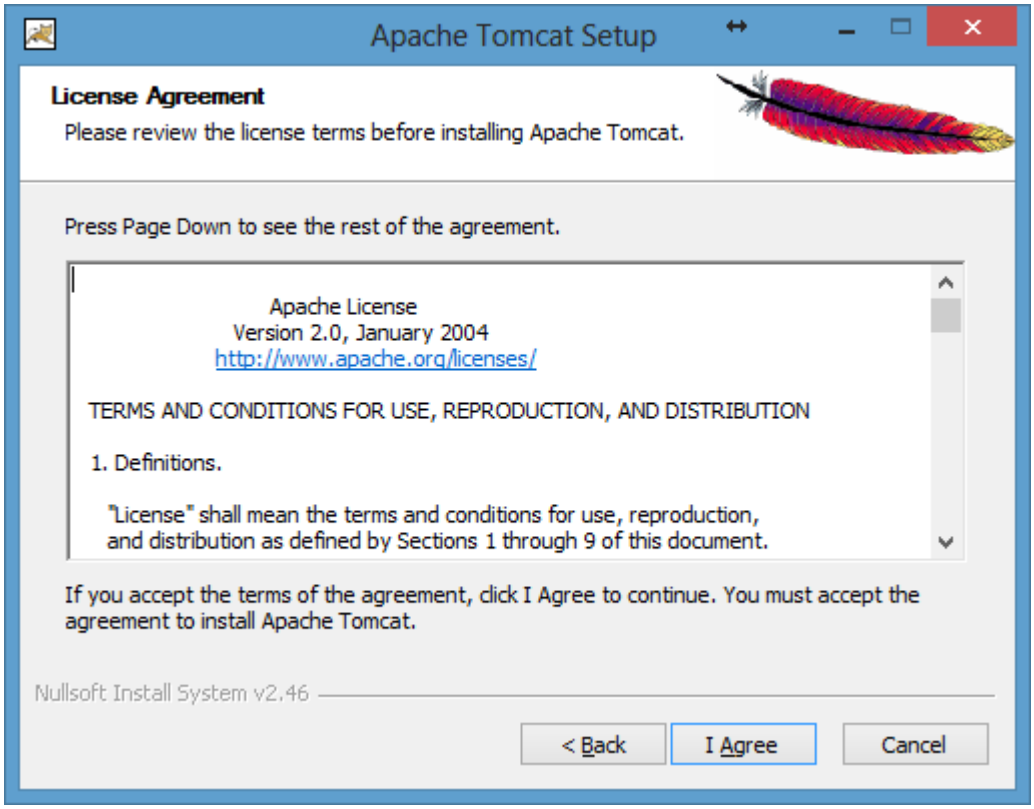


Figura 45 - Contrato Tomcat

A Figura 46 apresenta as configurações de componentes do *Tomcat***,** ou seja, aqui é definido o tipo de instalação, selecionar no caso a opção Next**:**

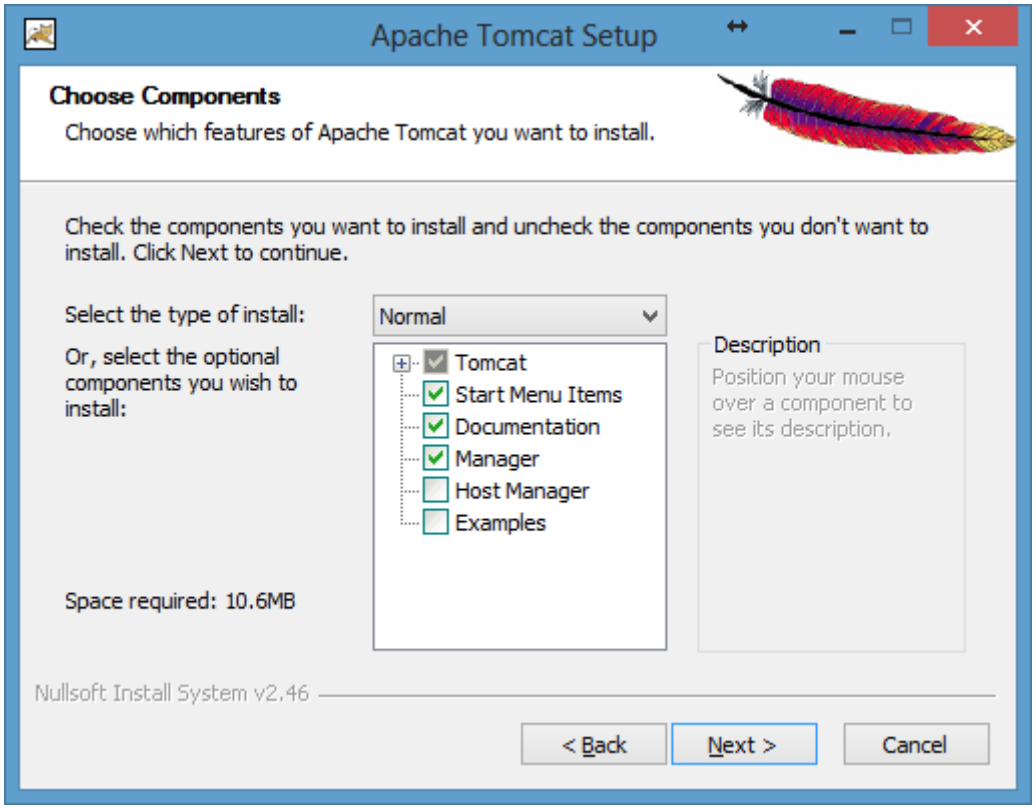


Figura 46 - componentes do Tomcat

A Figura 47 apresenta onde devem ser configuradas as portas de conexão, e também o usuário que vai ter acesso ao gerenciamento do *Tomcat*, foi definido usuário “admin” com a senha “admin” e selecionado Next:

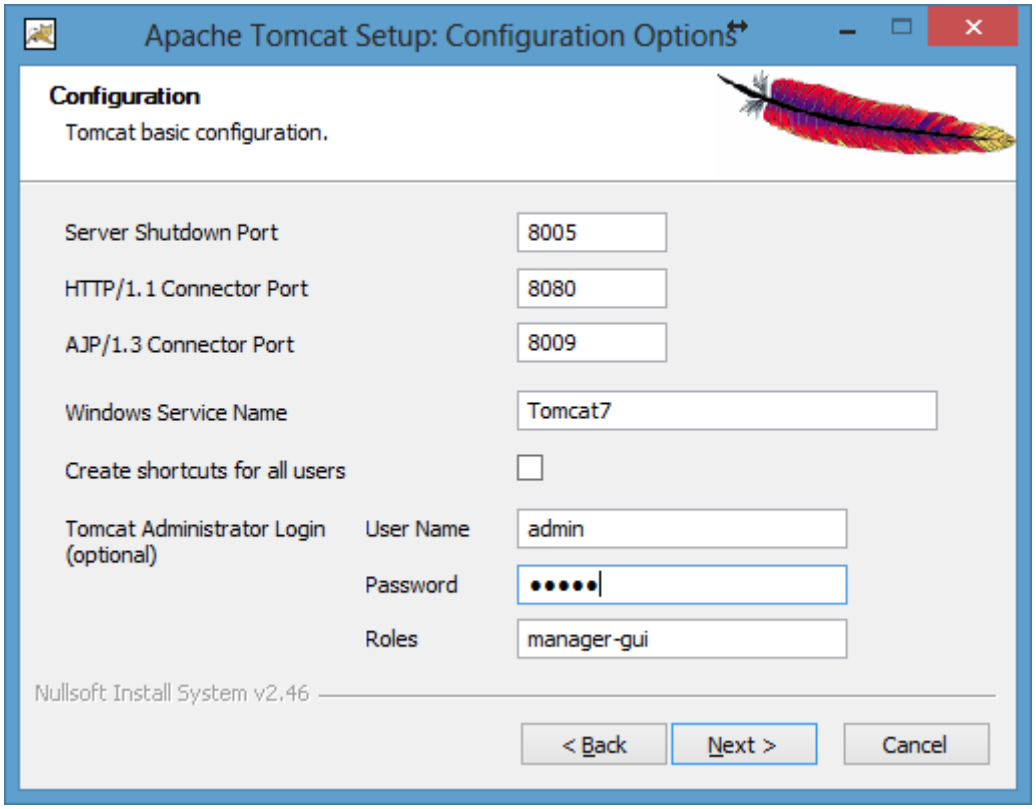


Figura 47 - Portas de Conexões Tomcat

A Figura 48 apresenta o diretório onde o JRE está instalado o caminho é: C:\Program Files\Java\jre7, selecionado Next:

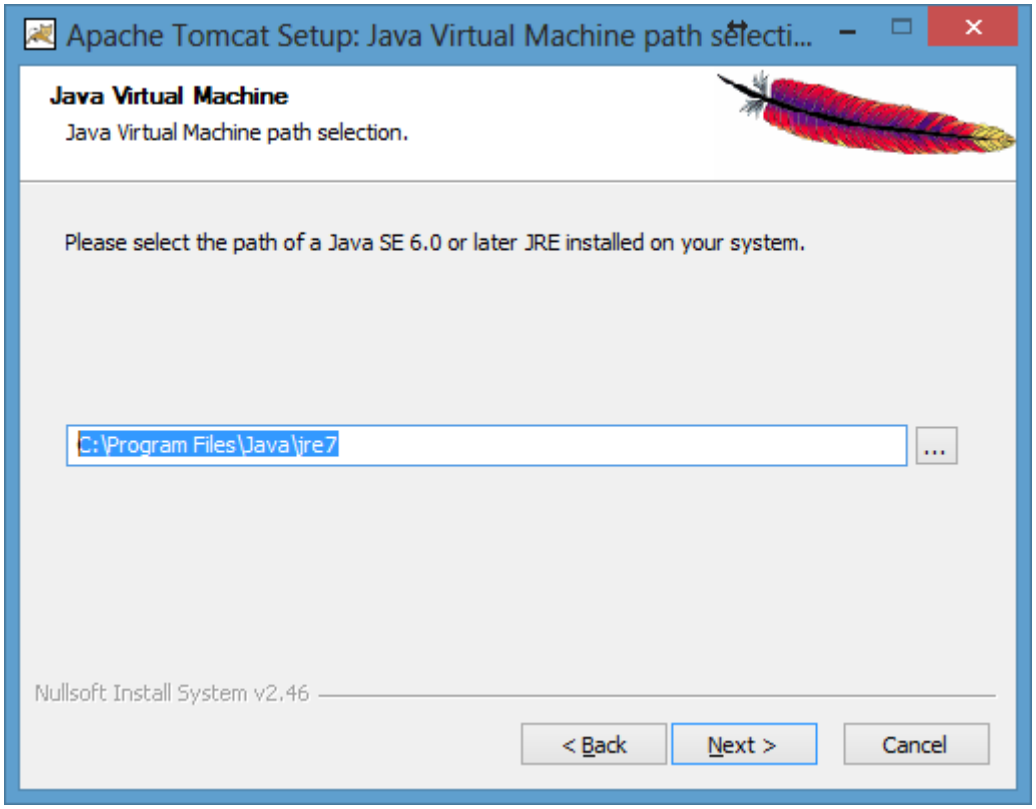


Figura 48 - Caminho JRE

A Figura 49 apresenta o diretório onde o *Tomcat* vai ser instalado, o caminho selecionado é: C:\Program Files*Apache Software* Foundation*Tomcat* 7.0, após isso selecione Install:

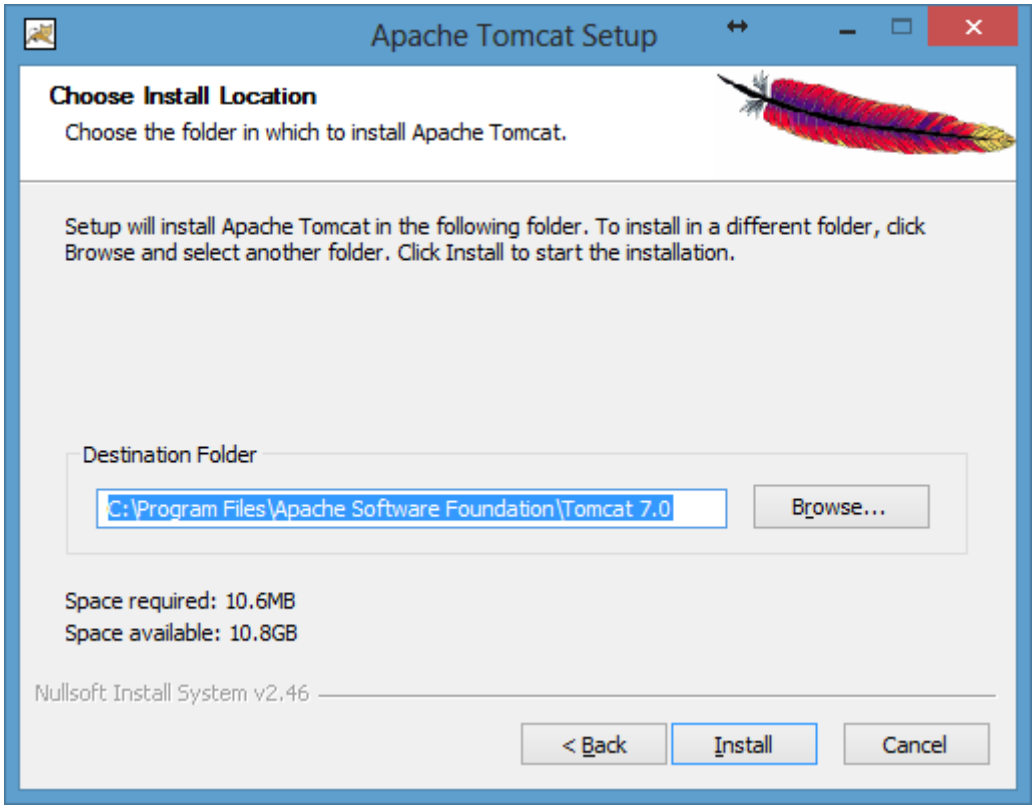


Figura 49 - Diretório Tomcat

A Figura 50 apresenta a tela com o processo de instalação do *Apache Tomcat*:

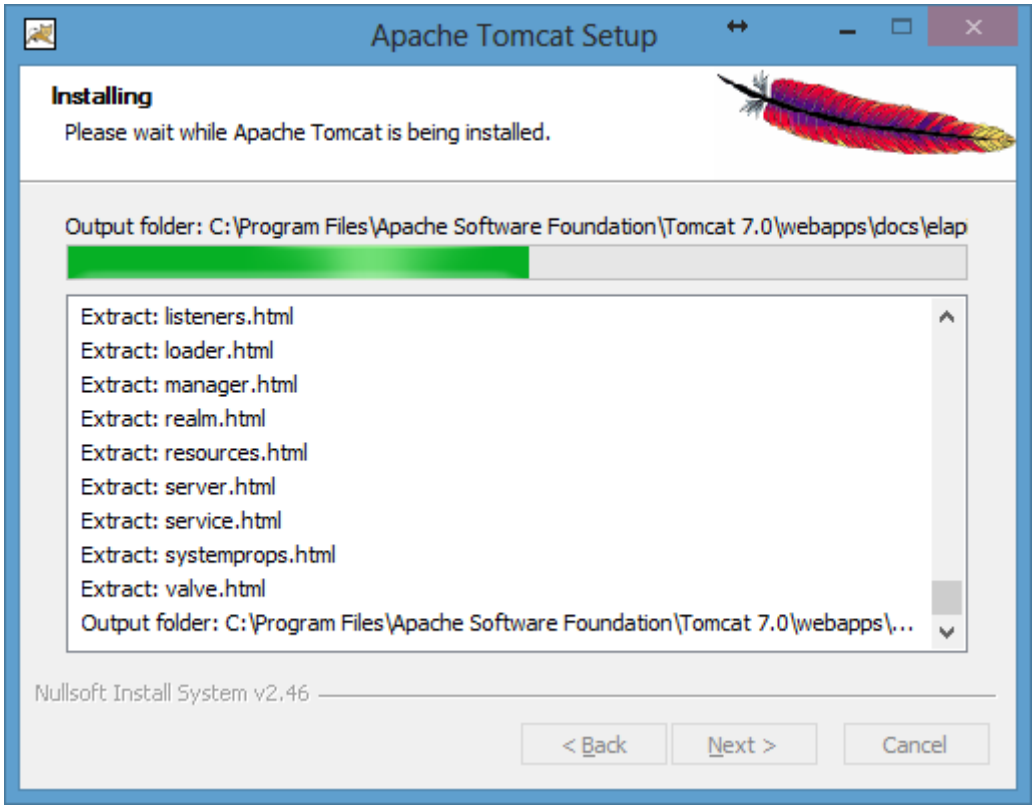


Figura 50 - Instalação Tomcat

E por fim, é apresentada Figura 51 com a última tela da instalação do *Apache Tomcat* 7.0, desmarque a opção Show Readme caso não tenha interesse de ler o manual e deixe a opção Run *Apache Tomcat* para executar o *Tomcat*, e clique em Finish:

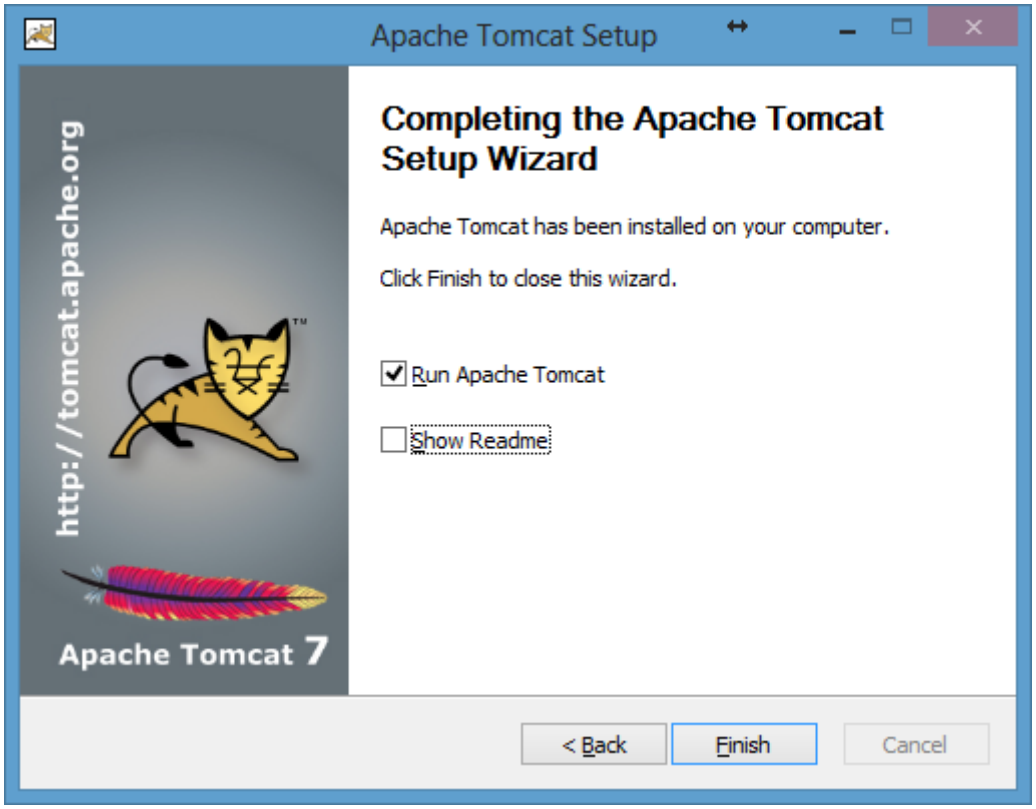


Figura 51 - Fim Instalação Tomcat

Configuração:

A Configuração do *Tomcat* é uma parte muito importante para o correto funcionamento do seu ambiente, essa configuração consiste na criação de uma variável de ambiente chamada CATALINA_HOME**.**

Para acessar as variáveis de ambiente clique com o botão direito do mouse no ícone Meu Computador, clique na aba Avançado, e clique no botão Variáveis de Ambiente a Figura 52 exibe a tela que será apresentada:

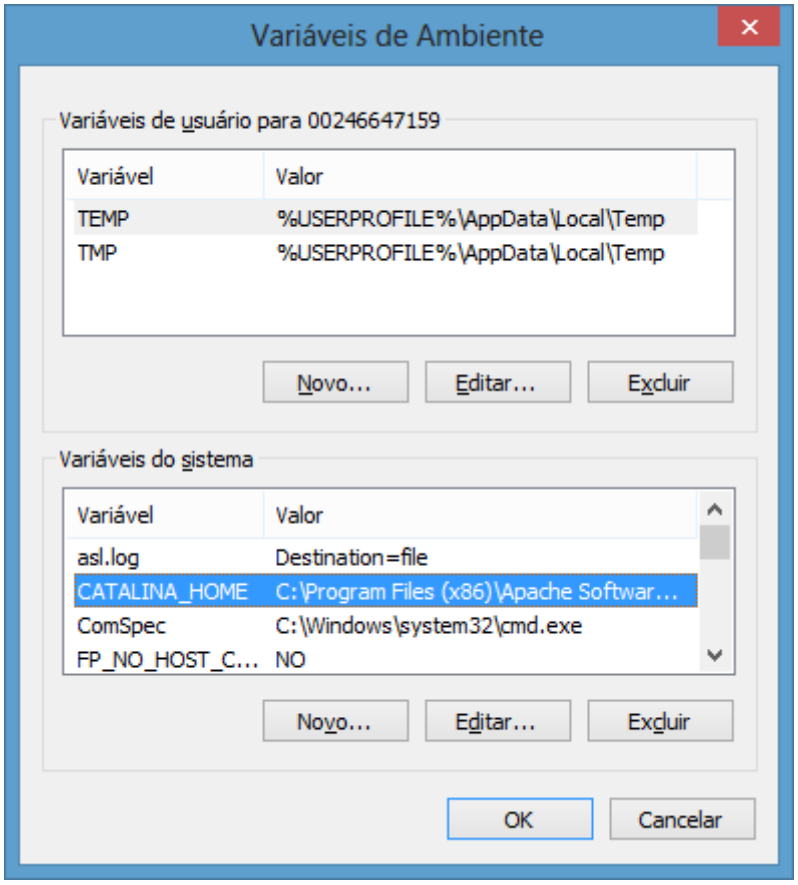


Figura 52 - Variáveis de Ambiente

Em Variáveis do sistema clicar na opção novo, em nome da variável colocar CATALINA_HOME e no valor da variável indicar o caminho onde está instalado o *Apache Tomcat* 7.0 conforme apresentado na Figura 53 :

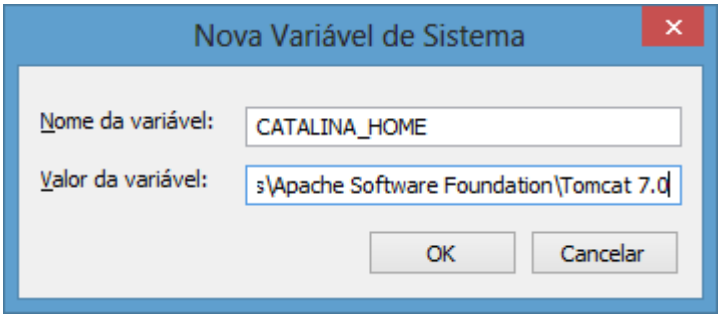


Figura 53 - Variável CATALINA_HOME

Após criar a variável de ambiente reinicie o serviço do *Tomcat* para que a alteração de configuração entre em vigor. Para testar o funcionamento do *Apache Tomcat*, acessa no navegador o endereço http://localhost:8080. A seguinte tela será apresentada conforme Figura 54:

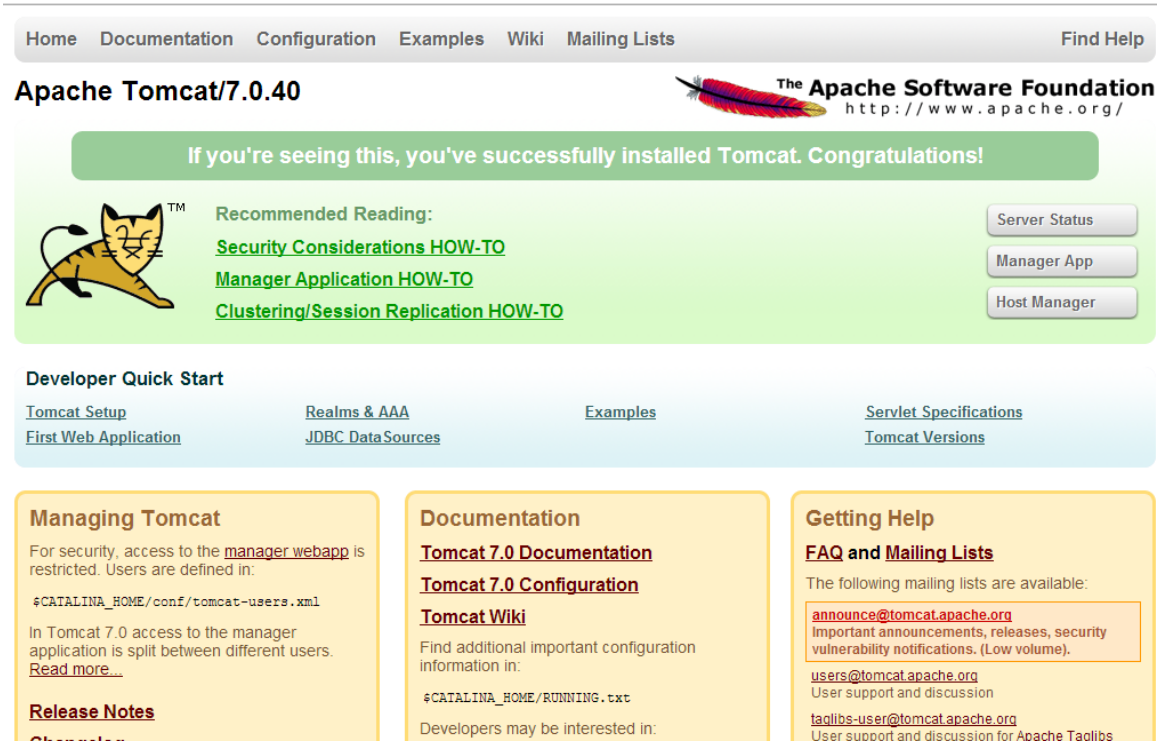


Figura 54 - Tela Inicial do Tomcat

Se essa tela for apresentada então a instalação do *Apache Tomcat* está correta e funcionando.

APÊNDICE D

Este anexo mostrará os procedimentos necessários para a instalação e configuração da aplicação *JAMWIKI* 1.3.1 no *container Apache Tomcat* 7.0. O download da aplicação pode ser feito através do link http://downloads.sourceforge.net/*JAMWIKI*/*JAMWIKI*-1.3.1.war.

Depois de instalado o *Tomcat* e com o arquivo *JAMWIKI*-1.3.1 baixado, acesse o link http://localhost:8080 e selecione a opção Manager App, serão solicitados o *login* e senha para gerenciar, esse *login* e senha foi criado durante a instalação do *Tomcat*.

A seguinte tela será apresentada, conforme Figura 55:

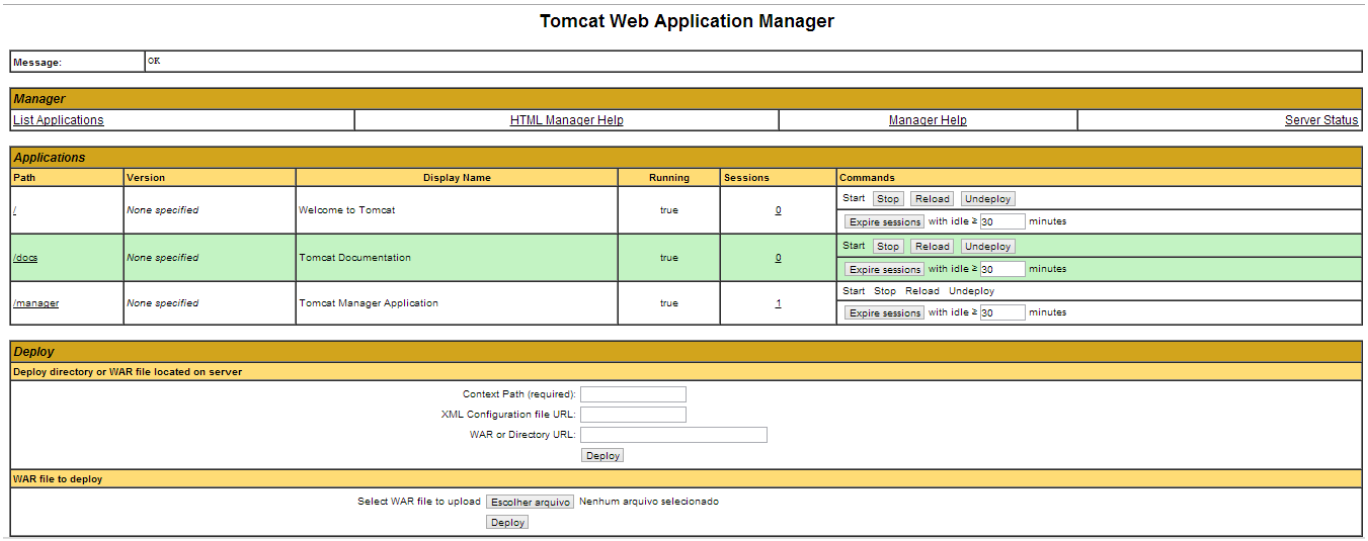


Figura 55 - Manager App Tomcat

No tópico “WAR file to deploy”, selecione a opção escolher arquivo e aponte para o arquivo *JAMWIKI*-1.3.1.war e depois acione o botão Deploy. Depois de acionado deploy entre na aplicação para configurá-la através do link http://localhost:8080/*JAMWIKI*-1.3.1/, a seguinte tela será apresentada, conforme Figura 56:

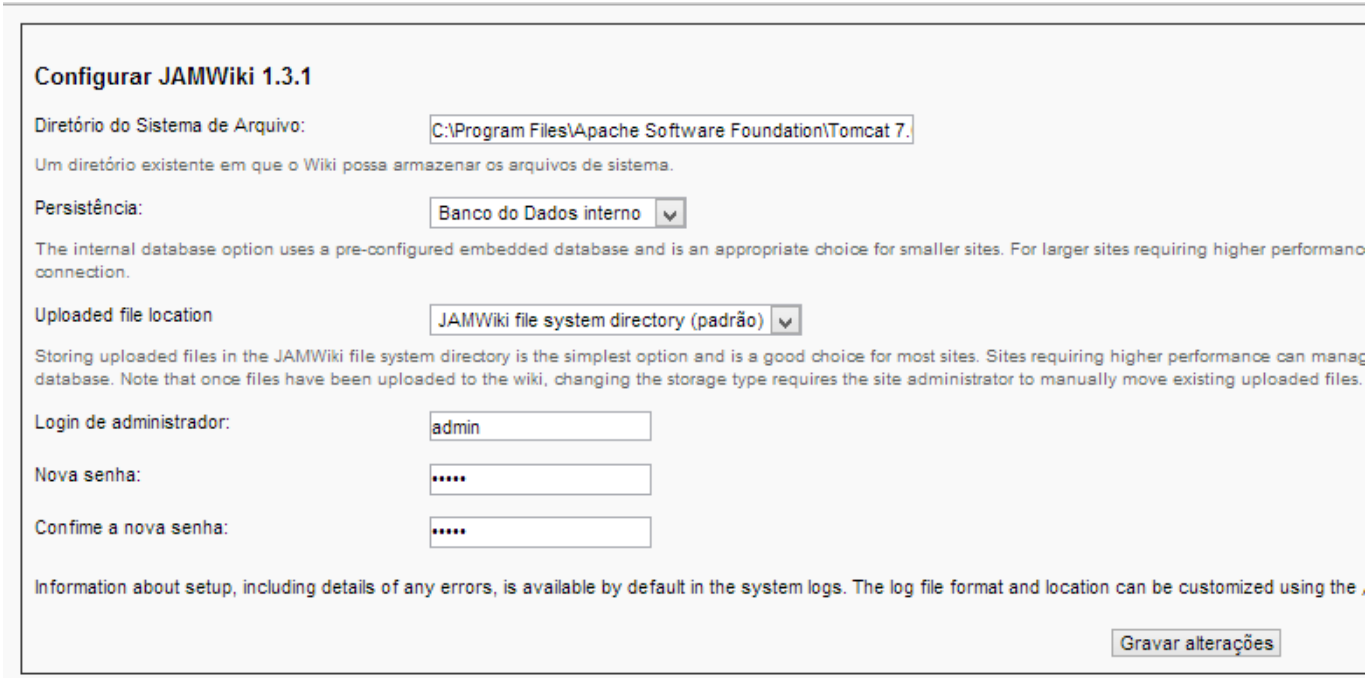


Figura 56 - Configuração JAMWIKI

Nesta tela apresentada serão configuradas as seguintes opções:

- Diretório do Sistema de Arquivo: será adicionado o local onde a aplicação irá armazenar os arquivos do sistema, neste caso foi utilizado o caminho C:\Program Files*Apache Software* Foundation*Tomcat* 7.0*Web*apps*JAMWIKI*-1.3.1;
- Persistência: será configurado o banco de dados que pode ser o interno onde as informações serão salvas na própria pasta do *JAMWIKI*, podendo ser externo indicando assim o caminho;
- Uploaded file location: nesta opção você pode selecionar onde os arquivos carregados para o sistema irão ficar, na própria pasta do *JAMWIKI* ou no banco de dados;
- *Login* de Administrador: será adicionado um *login* e senha para gerenciar a aplicação.

Todas estas alterações serão salvas na pasta raiz do *JAMWIKI* no diretório /*WEB*-INF/classes/ no arquivo logback.xml, caso deseja-se fazer alguma alteração posterior é possível editando o mesmo. Ao gravar as alterações realizadas a seguinte tela será apresentada indicando que a instalação e configuração estão prontas, conforme Figura 57:

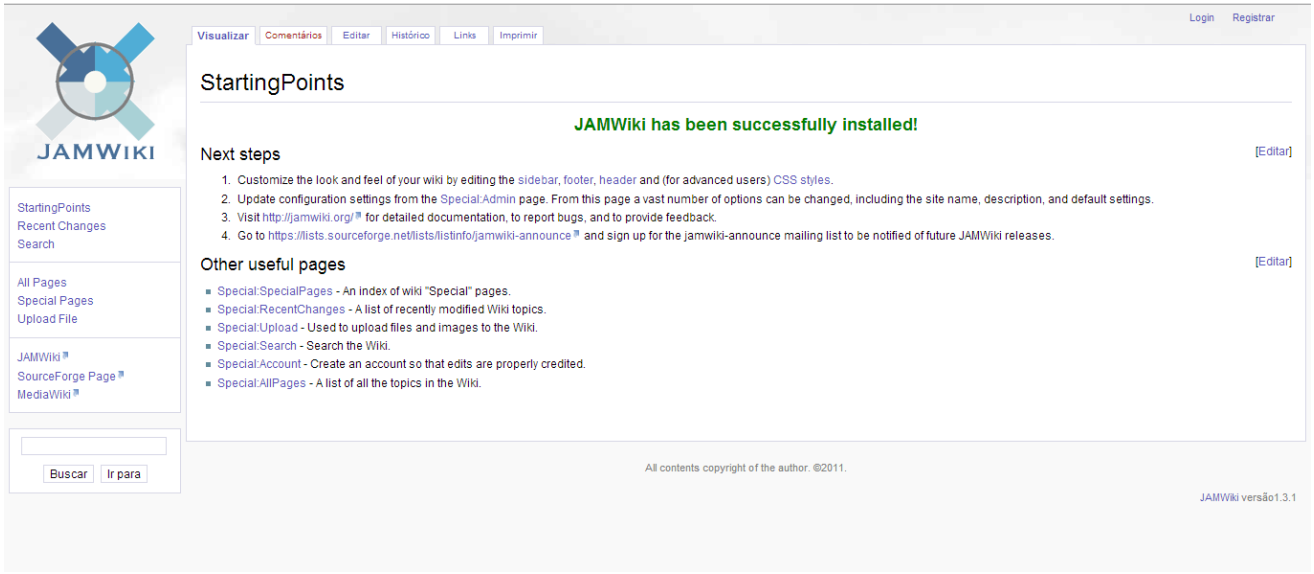


Figura 57 - JAMWIKI instalado