Distribuição de requisições em clusters de web services: uma abordagem flexível, dinâmica e transparente

Bruno Squizato Faiçal

Distribuição de requisições em clusters de web services: uma abordagem flexível, dinâmica e transparente

Bruno Squizato Faiçal

Orientador: **Prof. Dr. Paulo Sérgio Lopes de Souza**

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA.*

USP – São Carlos
Julho de 2012

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Faiçal, Bruno Squizato
F159d Distribuição de requisições em clusters de web services: uma abordagem flexível, dinâmica e transparente / Bruno Squizato Faiçal; orientador Paulo Sérgio Lopes de Souza. -- São Carlos, 2012.
118 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2012.

1. Web services. 2. Cluster. 3. Políticas de distribuição. 4. SOA. I. Souza, Paulo Sérgio Lopes de, orient. II. Título.

“A cabeça de uma pessoa faz dela um rei.”
Provérbio Yorubá

Agradecimentos

Agradeço primeiro, aos Orixás por serem o pilar central da minha vida e um abrigo nos momentos de dificuldade. Aos meus pais Roberto e Cecília, que além do apoio financeiro, sempre me deram apoio moral e incentivo durante todo o mestrado. A sua preocupação com meus estudos e bem estar exigiu muito do trabalho e vida de ambos, a quem serei eternamente grato.

A minha esposa Rafaela, que esteve ao meu lado em todos os momentos de dificuldade durante o mestrado, compreendendo as minhas ausências, me alegrando com suas palavras e atos. Nunca me esquecerei dos momentos de dedicação e apoio ao nosso bem estar.

Ao meu orientador, Prof. Paulo Sérgio Lopes de Souza, que contribuiu muito para minha formação pessoal e profissional. Agradeço a máxima dedicação do Prof. Paulo para me ajudar em todos os sentidos durante o mestrado e com muita paciência me conduziu ao longo desta jornada. Com certeza, sua conduta e dedicação serão motivos de inspiração para minha vida profissional e pessoal.

Aos meus sogros, Rafael e Sirley pelo apoio que sempre me deram em todos os momentos da minha vida. A minha Mãe de Santo, Tereza de Ayra, pelas palavras de ensinamentos que foram de extrema importância para o meu comportamento durante o desenvolvimento do Mestrado. Ao meu irmão, Roberto, as minhas cunhadas Francielly e Gabriela e as minhas sobrinhas Izadora e Anna Luiza pelos momentos de lazer.

Agradeço ao meu tio Julio e minha tia Ruth por todo o apoio financeiro e moral ao longo dessa caminhada. Por toda ajuda que me deram desde a graduação e pelas conversas de apoio, participando efetivamente da minha formação profissional e moral.

Aos meus amigos Douglas Rodrigues, Luis Nakamura, Edwin Luis, Prof. Julio Cezar Estrella, Roni Guillermo, Prof. Regina Helena e todos os alunos e professores do laboratório LaSDPC pelas conversas, conselhos, convivência e auxílio durante o mestrado.

Agradeço a agencia de fomento FAPESP pelo financiamento deste projeto.

RESUMO

Esta dissertação de mestrado propõe uma nova política de distribuição de requisições em *cluster* de *web services*, denominada Política *Performance*. Esta política provê uma distribuição transparente, flexível e dinâmica das requisições na plataforma em que é executada. Um estudo sistemático também é realizado para analisar a qualidade dos índices de carga empregados no contexto de *web services* e propõe um novo índice capaz de representar fielmente o desempenho dos *web services* e encapsular a complexidade estrutural da plataforma. Também é proposto um Módulo Gerenciador de Energia capaz de prover sustentabilidade à plataforma, reduzindo o consumo de energia elétrica sem prejudicar a alta confiabilidade na distribuição das requisições e com baixo impacto no tempo médio de resposta. Os estudos experimentais realizados neste trabalho mostraram que a Política *Performance* permitiu um melhor desempenho no atendimento das requisições realizadas à plataforma. Estes resultados referem-se a um desempenho superior a 70% no tempo médio de resposta, quando comparado ao desempenho demonstrado pela política padrão do *Mod_cluster*. O Módulo Gerenciador de Energia proporcionou uma redução de aproximadamente 30% no consumo de energia da plataforma mantendo a alta confiabilidade na distribuição das requisições.

ABSTRACT

This Master's dissertation proposes a new policy for distribution of requests in cluster of web services, named policy Performance. This policy provides a transparent, flexible and dynamic distribution of requests on the platform. A systematic study is also conducted to examine the quality of load indices used in the context of web services, and proposes a new index that accurately represent the performance of web services and encapsulate the complexity structural of the platform. Also proposed is an Energy Manager Module capable of providing sustainability to the platform, reducing power consumption without sacrificing high reliability in the distribution of requests and low impact on the average response time. Our main results show that the policy Performance has a better performance in handling requests sent to the platform. Our results show a gain of performance higher than 70% in average response time when compared to the performance demonstrated by the default policy Mod_cluster. The Power Manager Module reduced by approximately 30% the energy consumption of the platform, even keeping the high reliability in the distribution of requests.

Sumário

1 INTRODUÇÃO 1

1.1 CONSIDERAÇÕES INICIAIS 1
1.2 CONTEXTUALIZAÇÃO 1
1.3 MOTIVAÇÃO E OBJETIVOS 3

2 FERRAMENTAS DE APOIO À WEB SERVICES 7

2.1 CONSIDERAÇÕES INICIAIS 7
2.2 REVISÃO SISTEMÁTICA 7
2.2.1 Planejamento 7
2.2.2 Condução da Revisão Sistemática 10
2.2.3 Análise dos Resultados 14
2.2.4 Conclusão 17
2.3 APACHE TOMCAT 18
2.3.1 O Módulo Mod_jk 19
2.4 JBOSS 21
2.4.1 O Modulo Mod_cluster 24
2.5 GANGLIA 27
2.6 O PROTÓTIPO JERRYMOUSE 28
2.6.1 Distribuição de requisições utilizando Jerrymouse 31
2.7 CONSIDERAÇÕES FINAIS 33

3 ÍNDICE DE CARGA 35

3.1 CONSIDERAÇÕES INICIAIS 35
3.2 MOTIVAÇÃO 35
3.3 MONITORANDO WEB SERVICES 36
3.4 ÍNDICES DE CARGA 38
3.5 ESTUDO EXPERIMENTAL 40
3.6 RESULTADOS 45
3.6.1 Percentual de CPU ociosa (CPU_Idle) 45
3.6.2 Percentual de CPU esperando E/S (CPU_WIO) 46
3.6.3 Free Memory 46
3.6.4 Swap Used e Memory Saturation 47
3.6.5 Bytes-in e Bytes-out 47
3.6.6 Ready Processes 48
3.6.7 Total Amount of Submitted Requests 48
3.6.8 Client Response-Time e Average-JMX_Time 49
3.7 CONCLUSÕES 49
3.8 CONSIDERAÇÕES FINAIS 51

4 POLÍTICAS DE DISTRIBUIÇÃO DE REQUISIÇÕES 53

4.1 CONSIDERAÇÕES INICIAIS 53
4.2 POLÍTICA PERFORMANCE 53
4.3 MÓDULO GERENCIADOR DE ENERGIA 59
4.4 CONSIDERAÇÕES FINAIS 63

5 ESTUDOS EXPERIMENTAIS COM AS POLÍTICAS DE DISTRIBUIÇÃO 65

5.1 CONSIDERAÇÕES INICIAIS 65
5.2 PLANEJAMENTO DOS EXPERIMENTOS 65
5.3 ANÁLISE DE RESULTADOS 69
5.4 CONSIDERAÇÕES FINAIS 83

6 CONCLUSÕES 85

6.1 CONSIDERAÇÕES FINAIS 85
6.2 CONTRIBUIÇÕES 87
6.3 TRABALHOS FUTUROS 89
REFERÊNCIAS BIBLIOGRÁFICAS 91
ANEXO I – RESULTADOS DA ANÁLISE SOBRE ÍNDICES DE CARGA 97

LISTA DE FIGURAS

FIGURA 1: PERCENTUAL DOS TRABALHOS REALIZADOS ENTRE OS ANOS DE 2006 E 2009.....18
FIGURA 2: ARQUITETURA UTILIZANDO INTERCEPTORES *CLIENT-SIDE* 22
FIGURA 3: ARQUITETURA UTILIZANDO BALANCEADOR DE CARGA EXTERNO.....24
FIGURA 4: ESTRUTURA DO *JERRYMOUSE* COM TODOS OS SEUS ELEMENTOS E A POSSÍVEL CONEXÃO ENTRE DOIS *JERRYMOUSE*, CASO O CLUSTER POSSUA MAIS DE UM *FRONT-END (MATOS, 2009)*. 29
FIGURA 5: EXECUÇÃO DA ARQUITETURA - BASEADA EM MATOS (2009) 33
FIGURA 6: PLATAFORMA UTILIZADA NO ESTUDO EXPERIMENTAL 41
FIGURA 7: JANELAS DESLIZANTES UTILIZADAS PARA AVALIAR O ÍNDICE *AVERAGE-JMX_TIME*. AS ÁREAS CINZENTAS INDICAM AS JANELAS DESLIZANTES USADAS COM O ÍNDICE TEMPO TOTAL DE EXECUÇÃO DO SERVIÇO POR *JMX* E O ÍNDICE QUANTIDADE TOTAL DE REQUISIÇÕES CONCLUÍDAS. 44
FIGURA 8: PSEUDOCÓDIGO DA POLÍTICA *PERFORMANCE* 55
FIGURA 9: REPRESENTAÇÃO DA LISTA DURANTE O MÓDULO DE INFORMAÇÃO 56
FIGURA 10: REPRESENTAÇÃO DA LISTA DURANTE O MÓDULO DE ANÁLISE 57
FIGURA 11: REPRESENTAÇÃO DA LISTA APÓS O MÓDULO DE SELEÇÃO SELECIONAR DESTINO PARA A REQUISIÇÃO 58
FIGURA 12: ESTRUTURA DO *JERRYMOUSE* COM ACRÉSCIMO DO MÓDULO GERENCIADOR DE ENERGIA 60
FIGURA 13: CONFIGURAÇÃO HETEROGÊNEA DO GERENCIADOR DE ENERGIA 62
FIGURA 14: REPRESENTAÇÃO DA LIGAÇÃO LÓGICA ENTRE OS ELEMENTOS DO *CLUSTER* COM 3 NÓS, SEM EXPLICITAR OS DISPOSITIVOS ESPECÍFICOS DE REDE. 66
FIGURA 15: REPRESENTAÇÃO DA LIGAÇÃO LÓGICA ENTRE OS ELEMENTOS DO CLUSTER COM 9 NÓS, SEM EXPLICITAR OS DISPOSITIVOS ESPECÍFICOS DE REDE. 67
FIGURA 16: TEMPO MÉDIO DE RESPOSTA ENTRE AS DISTRIBUIÇÕES REALIZADAS PELA POLÍTICA *PERFORMANCE* E A POLÍTICA PADRÃO DO *MOD_CLUSTER*.....70
FIGURA 17: COMPARAÇÃO ENTRE OS TEMPOS MÉDIOS DE RESPOSTA OBTIDOS COM AS DISTRIBUIÇÕES DAS DUAS POLÍTICAS UTILIZADAS. 71
FIGURA 18: QUANTIDADE DE REQUISIÇÕES ATENDIDAS POR SEGUNDO 72
FIGURA 19: QUANTIDADE DE REQUISIÇÕES ATENDIDAS COM SUCESSO. 72
FIGURA 20: TEMPO MÉDIO DE EXECUÇÃO DAS APLICAÇÕES CLIENTES 73
FIGURA 21: TEMPOS MÉDIOS DE RESPOSTA OBTIDOS NOS EXPERIMENTOS COM 2, 4 E 8 NÓS *BACK-ENDS* ...74
FIGURA 22: COMPARAÇÃO DOS TEMPOS MÉDIOS DE RESPOSTA DOS EXPERIMENTOS NA PLATAFORMA B..76
FIGURA 23: QUANTIDADE DE REQUISIÇÕES ATENDIDAS POR SEGUNDO NA PLATAFORMA B.....76
FIGURA 24: TEMPO DE EXECUÇÃO DAS APLICAÇÕES CLIENTES DURANTE OS EXPERIMENTOS NA PLATAFORMA B 77
FIGURA 25: TOTAL DE REQUISIÇÕES COMPLETADAS COM SUCESSO NA PLATAFORMA B 78
FIGURA 26: CÁLCULO DA INFLUÊNCIA DOS FATORES CONSIDERANDO OS RESULTADOS MAIS EXPRESSIVOS DA.....78
FIGURA 27: REPRESENTAÇÃO DO COMPORTAMENTO DO ÍNDICE DE CARGA E NA ANÁLISE NOS EXPERIMENTOS COM O MÓDULO GERENCIADOR DE ENERGIA NA PLATAFORMA B 80
FIGURA 28: TEMPOS MÉDIOS DE RESPOSTAS OBTIDOS NOS EXPERIMENTOS COM O MÓDULO GERENCIADOR DE ENERGIA E COM NÓS *BACK-ENDS* LIGADOS PERMANENTEMENTE NA PLATAFORMA B.....81
FIGURA 29: RESULTADOS PARA OS ÍNDICES DE *CPU*, CONSIDERANDO O SERVIÇO 98
FIGURA 30: RESULTADOS PARA OS ÍNDICES DE *CPU*, CONSIDERANDO O SERVIÇO 98
FIGURA 31: RESULTADOS PARA OS ÍNDICES DE *CPU*, CONSIDERANDO O SERVIÇO DE 99
FIGURA 32: RESULTADOS PARA OS ÍNDICES DE *CPU*, CONSIDERANDO O SERVIÇO DE 99
FIGURA 33: RESULTADOS DOS ÍNDICES DE *CPU*, CONSIDERANDO OS QUATRO SERVIÇOS.....100
FIGURA 34: RESULTADOS PARA OS ÍNDICES DE MEMÓRIA, CONSIDERANDO O SERVIÇO.....101
FIGURA 35: RESULTADOS PARA OS ÍNDICES DE MEMÓRIA, CONSIDERANDO O SERVIÇO.....101
FIGURA 36: RESULTADOS PARA OS ÍNDICES DE MEMÓRIA, CONSIDERANDO O SERVIÇO.....102
FIGURA 37: RESULTADOS PARA OS ÍNDICES DE MEMÓRIA, CONSIDERANDO O SERVIÇO DE.....102
FIGURA 38: RESULTADOS PARA OS ÍNDICES DE MEMÓRIA CONSIDERANDO OS QUATRO 103
FIGURA 39: RESULTADOS PARA O ÍNDICE DE SATURAÇÃO DO SISTEMA DE MEMÓRIA,.....104
FIGURA 40: RESULTADOS PARA O ÍNDICE DE SATURAÇÃO DO SISTEMA CONSIDERANDO O SERVIÇO DE *MEMORY-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 104
FIGURA 41: RESULTADOS PARA O ÍNDICE DE SATURAÇÃO DO SISTEMA, CONSIDERANDO O SERVIÇO DE *NETWORK-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 105

FIGURA 42: RESULTADOS PARA O ÍNDICE DE SATURAÇÃO DO SISTEMA CONSIDERANDO O SERVIÇO DE BANCO DE DADOS PARA 1, 2, 4 E 8 CLIENTES.....105
FIGURA 43: RESULTADOS PARA O ÍNDICE DE SATURAÇÃO DO SISTEMA CONSIDERANDO TODOS OS SERVIÇOS PARA 1, 2, 4 E 8 CLIENTES. 106
FIGURA 44: RESULTADOS PARA OS ÍNDICES DE REDE CONSIDERANDO O SERVIÇO *CPU-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 107
FIGURA 45: RESULTADOS PARA OS ÍNDICES DE REDE CONSIDERANDO O SERVIÇO DE *MEMORY-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 107
FIGURA 46: RESULTADOS PARA OS ÍNDICES DE REDE CONSIDERANDO O SERVIÇO DE *NETWORK-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 108
FIGURA 47: RESULTADOS PARA OS ÍNDICES DE REDE CONSIDERANDO O SERVIÇO DE BANCO DE DADOS PARA 1, 2, 4 E 8 CLIENTES. 108
FIGURA 48: RESULTADOS PARA OS ÍNDICES DE REDE, CONSIDERANDO TODOS OS SERVIÇOS PARA 1, 2, 4 E 8 CLIENTES. 109
FIGURA 49: RESULTADOS PARA O ÍNDICE DE *READY PROCESS* CONSIDERANDO O SERVIÇO *CPU-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 110
FIGURA 50: RESULTADOS PARA O ÍNDICE DE *READY PROCESS* CONSIDERANDO O SERVIÇO *MEMORY-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 110
FIGURA 51: RESULTADOS PARA O ÍNDICE DE *READY PROCESS* CONSIDERANDO O SERVIÇO *NETWORK-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 111
FIGURA 52: RESULTADOS PARA O ÍNDICE DE *READY PROCESS* CONSIDERANDO O SERVIÇO DE BANCO DE DADOS PARA 1, 2, 4 E 8 CLIENTES. 111
FIGURA 53: RESULTADOS PARA O ÍNDICE DE *READY PROCESS* CONSIDERANDO TODOS OS SERVIÇOS EXECUTADOS 112
FIGURA 54: RESULTADOS PARA O ÍNDICE DE QUANTIDADE DE REQUISIÇÕES CONSIDERANDO O SERVIÇO *CPU-BOUND* PARA 1, 2, 4 E 8 CLIENTES.....113
FIGURA 55: RESULTADOS PARA O ÍNDICE QUANTIDADE DE REQUISIÇÕES CONSIDERANDO O SERVIÇO DE *MEMORY-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 113
FIGURA 56: RESULTADOS PARA O ÍNDICE QUANTIDADE DE REQUISIÇÕES CONSIDERANDO O SERVIÇO DE *NETWORK-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 114
FIGURA 57: RESULTADOS PARA O ÍNDICE QUANTIDADE DE REQUISIÇÕES CONSIDERANDO O SERVIÇO DE BANCO DE DADOS PARA 1, 2, 4 E 8 CLIENTES.....114
FIGURA 58: RESULTADOS PARA O ÍNDICE QUANTIDADE DE REQUISIÇÕES CONSIDERANDO TODOS OS SERVIÇOS EM EXECUÇÃO. 115
FIGURA 59: RESULTADOS PARA OS ÍNDICES BASEADOS EM TEMPO CONSIDERANDO O SERVIÇO *CPU-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 116
FIGURA 60: RESULTADOS PARA OS ÍNDICES BASEADOS EM TEMPO CONSIDERANDO O SERVIÇO DE *MEMORY-BOUND* PARA 1, 2, 4 E 8 CLIENTES. 116
FIGURA 61: RESULTADOS PARA OS ÍNDICES BASEADOS EM TEMPO CONSIDERANDO O SERVIÇO *NETWORK-BOUND* PARA 1, 2, 4 E 8 CLIENTES.....117
FIGURA 62: RESULTADOS PARA OS ÍNDICES BASEADOS EM TEMPO CONSIDERANDO O SERVIÇO DE BANCO DE DADOS PARA 1, 2, 4 E 8 CLIENTES. 117
FIGURA 63: RESULTADOS PARA OS ÍNDICES BASEADOS EM TEMPO CONSIDERANDO TODOS OS SERVIÇOS EM EXECUÇÃO. 118

Lista de Tabelas

TABELA 1: ESTUDOS SELECIONADOS NA BUSCA SOBRE INFLUÊNCIA DOS SOFTWARES 13
TABELA 2: ÍNDICES DE CARGA ANALISADOS NOS EXPERIMENTOS. 43
TABELA 3: TEMPO MÉDIO DE RESPOSTA, CONSIDERANDO TODOS OS NÍVEIS DE CLIENTES CONCORRENTES DOS EXPERIMENTOS REALIZADOS COM O MÓDULO GERENCIADOR DE ENERGIA NA PLATAFORMA B 82
TABELA 4: TEMPO DOS NÓS DA PLATAFORMA B QUE PERMANECERAM LIGADOS DURANTE O EXPERIMENTO COM A POLÍTICA *PERFORMACE*/MGE 82
TABELA 5: TEMPO DE EXECUÇÃO QUANDO EXECUTADO OS QUATRO SERVIÇOS SEPARADAMENTE. 97
TABELA 6: ÍNDICES DE *CPU* 97
TABELA 7: ÍNDICES DE MEMÓRIA 100
TABELA 8: ÍNDICE *SATURATION* 103
TABELA 9: ÍNDICES DE REDE 106
TABELA 10: ÍNDICES DE *READY-PROCESS* 109
TABELA 11: ÍNDICE AMOUNT SUBMITTED-REQUESTS 112
TABELA 12: ÍNDICES BASEADOS EM TEMPO DE EXECUÇÃO DO SERVIÇO 115

1 Introdução

1.1 Considerações Iniciais

O objetivo deste capítulo é apresentar a contextualização e a motivação para o desenvolvimento deste trabalho de mestrado intitulado “Distribuição de Requisições em *Clusters* de *Web Services*: uma abordagem flexível, dinâmica e transparente”.

1.2 Contextualização

A possibilidade de dividir grandes sistemas computacionais em pequenos softwares específicos e interligar esses diversos softwares menores (serviços) de diversas formas (inclusive pela *web*) para compor novos softwares, apresenta-se como uma alternativa ao paradigma de grandes softwares isolados. A decomposição e a interligação possibilitam a utilização de partes de outros *softwares* sem precisar reescrever e duplicar programas. Além disso, as informações podem ser acessadas por todos os sistemas de uma única forma, através de um único serviço que as acesse. Segundo Carter (2007), a flexibilidade é o principal combustível para o crescimento dos negócios no cenário atual e ela é proporcionada no setor de tecnologia da informação com a decomposição e interligação dos sistemas. Para dar o suporte a esse novo paradigma, os sistemas são decompostos em pequenos serviços, os quais contam com novos aspectos conceituais sobre a sua estruturação e disponibilização na *web*. Tais aspectos conceituais são denominados atualmente de *SOA* (*Service Oriented Architecture*).

SOA representa um meio de organizar *softwares* criados previamente. Josuttis (2007) define *SOA* não como uma arquitetura real, mas "algo" que leva a uma arquitetura real. Ainda para Josuttis (2007), pode-se chamá-la de um estilo, paradigma, conceito, perspectiva, filosofia ou representação. Isto é, *SOA* não é uma ferramenta concreta ou *framework* que pode-se comprar. É uma abordagem, uma maneira de pensar, um sistema de valores que leva a decisões concretas na concepção de uma arquitetura de um *software*. Definições de *SOA* podem ser encontradas em Brooks (2008), Schulte et al. (2008), Tseng e Huang (2008).

SOA utiliza uma estrutura de apoio aos serviços disponibilizados, cada um acessível por meio de uma interface padrão e protocolos de passagem de mensagem. As aplicações podem acessar os serviços disponíveis de outra empresa, sem precisar criar soluções ponto-a-ponto entre ambas. Essa abordagem também é aplicável a múltiplas instâncias de uma aplicação executando em diferentes plataformas (Papazoglou; Georgakopoulos, 2003). *SOA* viabiliza a flexibilidade e a redução de custos, definindo metodologias para reutilizar componentes de *software* e processos de negócio.

SOA também pode ser instanciada como uma maneira lógica de projetar sistemas, disponibilizando serviços para aplicações por meio de interfaces de publicação e de descoberta. *SOA* define a interação entre agentes de *software* como uma troca de mensagens entre clientes que requisitam serviços e provedores destes. Os provedores publicam uma descrição de seus serviços e os clientes devem encontrar a descrição para a conexão. *SOA* não é formada somente por serviços, mas principalmente pela relação entre cliente, agente de descoberta e provedor. O agente de descoberta possui uma lista dos serviços oferecidos em provedores conhecidos. Na implementação de *SOA* por meio de *web services*, as descrições de interfaces dos *web services* existentes nessa lista podem ser expressas usando a linguagem *WSDL* (*Web Service Description Language*) (Newcomer, 2002).

Os *web services* usam a *URI* (*Uniform Resource Identifier*), um identificador com interfaces e ligações definidas, descritas e descobertas através do padrão *XML* (Ferris; Farrell, 2003). As interações entre *web services* ocorrem como chamadas *SOAP* (*Simple Object Access Protocol*), um protocolo de comunicação baseado em *XML* para a interação de aplicações (Papazoglou; Georgakopoulos, 2003). *SOAP* é amplamente empregado por uma nova geração de aplicações de computação distribuída, independente de plataforma e de linguagens.

A crescente integração das aplicações faz aumentar significativamente a quantidade de informações a serem processadas. Para atender tal demanda, vários *softwares* que dão suporte aos *web services* permitem o agrupamento de várias instâncias desses serviços, a fim de gerar um aglomerado de servidores ou *cluster*. O *cluster* tem potencial para melhorar o desempenho no atendimento às requisições, além de possibilitar outros benefícios como alta disponibilidade e tolerância a falhas. Os *softwares* mais usados em *clusters* de provedores de *web services* são: *XFire*, *Celtix*, *JBoss*, *WebSphere*, *Jetty*, *Apache Tomcat* e *Apache Axis 2* (Matos, 2009). Apesar de existirem alternativas, esses *softwares* citados estão em

desenvolvimento por mais tempo, com amplos grupos de usuários. Estudos preliminares já realizados no grupo de pesquisa em Sistemas Distribuídos e Programação Concorrente do ICMC/USP mostram que o *Apache Tomcat* apresenta boas características quanto ao desempenho no atendimento de requisições, destacando-se dos demais (Matos, 2009). Ele é desenvolvido em Linguagem *Java* e possui uma estrutura modular.

Desejam-se em *SOA* características como: transparência à aplicação, uso de plataformas heterogêneas, alta confiabilidade e alto desempenho aos serviços oferecidos. As duas últimas características são viabilizadas graças a uma distribuição eficiente de requisições, dentro do módulo provedor. Essa distribuição é realizada de diferentes maneiras, envolvendo desde *web switches* até *clusters* de provedores de *web services*. Quando o distribuidor das requisições fica distante dos provedores de *web services*, ou seja, é implementado nos *web switches* ou nos servidores *web*, a distribuição limita-se a usar pouca ou nenhuma informação precisa sobre os serviços sendo requisitados (Conocchioli, 2006). Dessa forma, *clusters* de servidores de *web services* apresentam-se como uma alternativa flexível, de alto desempenho, robusta e altamente utilizada para atender os requisitos de interligação dos sistemas disponíveis na *web*.

1.3 Motivação e Objetivos

Considerando o contexto exposto acima, este projeto de mestrado tem como objetivo desenvolver políticas para distribuição de requisições em *cluster* de *web services* flexíveis, dinâmicas e transparentes que forneçam alto desempenho, alta confiabilidade e economia de energia ao *cluster* de *web services*. Como principal produto final deste mestrado tem-se, portanto, os protótipos de novas políticas de distribuição de requisições voltadas para a melhoria do desempenho na distribuição de requisições em *clusters* de *web services*. Um módulo de Economia de Energia foi associado a estas políticas, o qual é acoplado diretamente ao *Jerrymouse* e permite gerenciar quais nós do *cluster* permanecerão ligados ou desligados frente à demanda gerada sobre o *cluster*.

Outro ponto forte deste projeto de mestrado, relacionado ao desenvolvimento das políticas de distribuição de requisições, é a investigação da eficiência dos índices de carga voltados para *web services*, para predizer o desempenho futuro de serviços a serem requisitados (Souza et. al., 2011b). Um novo índice de carga foi desenvolvido, a partir dessas

investigações, baseado no tempo médio de execução do serviço no servidor. Os estudos experimentais realizados com este índice mostraram que ele atua de maneira transparente à arquitetura e aos serviços executados, com baixa intrusão, estabilidade e fidelidade na representação do desempenho verificado com a execução do serviço monitorado.

Encontra-se em desenvolvimento no Laboratório de Sistemas Distribuídos e Programação Concorrente do ICMC/USP, o projeto de pesquisa intitulado "Otimizando a Distribuição de Requisições em *Clusters* de *Web Services*: uma abordagem flexível, dinâmica e transparente" (processo FAPESP número 2008/00553-1). Esse projeto visa, principalmente, otimizar a distribuição de requisições em *clusters* de *web services*, no contexto de *SOA*, através do desenvolvimento de uma nova arquitetura de *software* e um protótipo de ferramenta. De ordem prática, esta nova arquitetura e protótipo atuam como um *middleware* capaz de incluir a atividade de distribuição de requisições diretamente nos servidores de *web services* de maneira flexível, dinâmica, transparente e tolerante a falhas, viabilizando com isso um uso mais eficiente da plataforma computacional distribuída. Atualmente, as melhores políticas encontradas neste contexto estão disponíveis no *JBoss* através do módulo *mod_cluster* e representam variantes de uma política *Round-Robin*. Elas podem utilizar índices voltados à representação de recursos da arquitetura, tais como: percentual de *CPU* utilizado e quantidade ou percentual memória utilizada do computador (Mod_cluster, 2011). Tais índices não representam de maneira fiel o desempenho no atendimento das requisições (Souza et. al., 2011b), pois os serviços requisitados geram demandas diferentes dos recursos monitorados no sistema. Outros índices, voltados para o contexto de *web services*, também são encontrados no *mod_cluster* (Mod_cluster, 2011), tais como: quantidade de sessões ativas, percentual de *threads* conectores ocupados e fluxo de entrada ou saída na placa de rede. No entanto, estes índices não consideram possíveis cargas de trabalho externas aos provedores de *web services* e também não representam fielmente o desempenho dos serviços requisitados (Souza et. al., 2011b).

Este trabalho de mestrado em particular é o segundo mestrado fortemente associado ao projeto citado, além de uma Iniciação Científica (ambos já concluídos). No primeiro mestrado, intitulado "Balanceamento de carga flexível e dinâmico para provedores de *web services*", desenvolvido por Jonathan de Matos (processo FAPESP número 2007/57971-1), definiu-se a nova arquitetura de distribuição de requisições em *clusters* de *web services* e, a partir dessa arquitetura proposta, houve a implementação de um protótipo de ferramenta de

software, chamado *Jerrymouse*. No entanto, esse primeiro projeto de mestrado não aprofundou as suas investigações no desenvolvimento de políticas de distribuição adequadas aos *web services*, limitando-se a criar a infra-estrutura básica necessária à execução dessas futuras políticas de distribuição (Matos, 2009) (Souza et. al., 2011).

A Iniciação Científica relacionada a esse projeto, intitulada "Definição de métricas para *web services*: uma contribuição usando monitoração" e desenvolvida pelo aluno Rodrigo Ferreira Ladeira (processo FAPESP número 2008/02588-7), teve como objetivo principal investigar métricas de desempenho em plataformas distribuídas destinadas ao suporte de *web services*, através do uso de ferramentas de monitoração que apresentem características de transparência, baixa intrusão e flexibilidade. Os resultados dessa Iniciação Científica contribuem para este projeto de mestrado indicando a possibilidade de uso da métrica "tempo médio" adquirido a partir de uma versão alterada do *JMX-console*, que une o *JMX*[1] ao *Ganglia*. *JMX-console* é uma ferramenta associada ao provedor de *web services JBoss* utilizada para gerenciar e disponibilizar, de forma amigável, diversas informações ao administrador do sistema. Os resultados obtidos com essa iniciação científica foram aprofundados durante este mestrado e os resultados de tais investigações são apresentados nesta dissertação.

Esta dissertação esta organizada em 6 capítulos. Após esta introdução que apresenta a contextualização e a motivação para o desenvolvimento deste projeto de Mestrado, o capítulo 2 descreve as principais ferramentas que fornecem suporte à execução de *web services* e o desenvolvimento de uma revisão sistemática neste contexto. A arquitetura de distribuição *Jerrymouse*, desenvolvida por (Matos, 2009), também é descrita nesse capítulo. No capítulo 3 é apresentado um estudo sobre índices de carga no contexto de *web services*, tal como a qualidade de suas aplicações neste contexto. O capítulo 4 descreve as políticas de distribuição de requisições em *cluster* de *web services* desenvolvidas neste mestrado. No capítulo 5, é detalhado o planejamento, os resultados e a análise dos estudos experimentais com estas políticas. No capítulo 6 estão detalhadas as conclusões, contribuições propostas resultantes deste projeto de mestrado e alguns trabalhos futuros.

[1] *JMX* (*Java Management Extensions*) é um *framework* que possibilita desenvolver ferramentas utilizando a linguagem *Java*. O *JMX* também realiza o gerenciamento e a monitoração do ambiente, tal como a máquina virtual *Java*, Sistema Operacional e recursos de *hardware*.

2 Ferramentas de Apoio à Web Services

2.1 Considerações Iniciais

O objetivo deste capítulo é apresentar a revisão sistemática realizada no contexto de distribuição de requisições em *cluster* de *web services* e a importância dos principais *softwares* utilizados em estudos sobre *SOA*, tais como os resultados e conclusões desses estudos. Este capítulo também aborda as principais ferramentas de apoio a *web services* utilizadas para o desenvolvimento deste projeto de mestrado. O protótipo de distribuição de requisições *Jerrymouse* também é abordado neste capítulo.

2.2 Revisão Sistemática

O objetivo desta seção é apresentar uma revisão sistemática das contribuições existentes na literatura, envolvendo o tema pesquisado neste mestrado. Para tanto, a seção aborda o planejamento, a condução, a análise dos resultados obtidos e, finalizando a seção, são apresentadas as conclusões obtidas nesta revisão sistemática. A revisão sistemática desenvolvida neste trabalho foi planejada baseando-se no modelo apresentado por Biolchini et al. (2005). Esta revisão sistemática apresenta duas frentes de pesquisa: 1) relacionada com objetivo desse trabalho de mestrado e 2) com os softwares utilizados pela arquitetura de distribuição **Jerrymouse**, proposta por Matos (2009).

2.2.1 Planejamento

Nesta primeira etapa, são definidos os objetivos principais da pesquisa bibliográfica e os métodos que norteiam todas as análises realizadas no material bibliográfico encontrado. Os objetivos da pesquisa realizada neste projeto são:

- Identificar e analisar métodos específicos para distribuição de requisições em *cluster* de *web services*;
- Identificar pesquisas relacionadas aos softwares *Ganglia*, *JBoss*, *Apache Tomcat* e *Axis2* no contexto de *Service-Oriented Architecture* (SOA);

- Identificar dificuldades na distribuição de requisições em *cluster* de *web services*;
- Identificar dificuldades na monitoração dos recursos de *clusters*.

Para esta revisão sistemática são propostas duas questões principais de pesquisa e quatro questões secundárias associadas aos objetivos. As questões primárias devem ser respondidas com informações essenciais (básicas) contidas nos trabalhos retornados pela busca. As questões secundárias podem ou não serem respondidas com informações mais específicas sobre a abordagem realizada pelo trabalho analisado.

Questão Primária 1: Quais métodos para distribuição de requisições em *cluster* de *web services* são utilizados?

Questão Primária 2: Qual a influência dos softwares *Ganglia*, *JBoss*, *Apache Tomcat* e *Axis2* em pesquisas na área de SOA?

Questão Secundária 1: Quais métodos para distribuição de requisições são específicos a *web services*?

Questão Secundária 2: Quais métodos para distribuição de requisições consideram o estado dos recursos do *cluster*?

Questão Secundária 3: Quais métodos para distribuição de requisições são tolerantes a falhas e/ou almejam balanceamento de carga?

Questão Secundária 4: Dentre os softwares encontrados, qual a influência nos trabalhos realizados dentro do contexto de *SOA*?

Seguindo o planejamento da revisão sistemática, a estratégia de busca para a seleção preliminar de trabalhos publicados é estabelecida considerando as opções descritas a seguir:

Fonte: As buscas são realizadas na máquina de busca *Engineering Village*[2] empregando a base de dados *Compendex*[3]. Utilizando esta ferramenta associada à base de dados, a busca é automatizada em diversos repositórios indexados, entre os quais podem ser

[2] *Engineering Village* é um serviço de informação baseado na web que oferece uma ampla gama de recursos para especialistas da informação, profissionais e pesquisadores que trabalham com a ciência aplicada e engenharia.
[3] *Compendex* é o mais abrangente banco de dados bibliográficos de pesquisa disponível na data da realização deste trabalho, contendo mais de dez milhões de referências e resumos, tomadas em mais de 5.600 revistas acadêmicas, revistas de negócios, conferências e relatórios técnicos (Engineering Village, 2009)

citados *ACM, IEEE, Springer*. Com isso, não é necessário a adaptação das *strings* de busca para realizar a pesquisa em cada repositório que se deseja abranger.

Idioma dos trabalhos: os idiomas selecionados para a busca por trabalhos são: inglês e português. A utilização do idioma inglês se justifica por ser internacionalmente aceito para a redação de trabalhos científicos. Enquanto o idioma português é o idioma oficial do país de origem dos autores da revisão sistemática e a sua exclusão automaticamente excluiria trabalhos relevantes de autores brasileiros, caso estes não estivessem no idioma inglês.

Palavras-chave e sinônimos: Para o idioma inglês: "*distribution*", "*scale*", "*request*", "*instantiates*", "*order*", "*demand*", "*scheduling*", "*balancing*", "*solicitation*", "*suit*", "*application*", "*web*", "*cluster*", "*ganglia*", "*jboss*", "*tomcat*" e "*axis2*". No idioma português: "distribuição", "escalonamento", "requisição", "instancia", "balanceamento", "pedido", "demanda", "solicitação", "*web*", "*cluster*". Estas palavras-chave definem o escopo que a revisão sistemática pretende abranger.

Estão definidos, a seguir, dois critérios de inclusão primários e quatro secundários com objetivo de selecionar apenas trabalhos que estejam relacionados ao contexto abordado nesta revisão sistemática. Estes critérios de inclusão respondem às questões de pesquisa definidas anteriormente. Os critérios primários devem ser satisfeitos pelo trabalho selecionado, enquanto que os critérios secundários podem não ser satisfeitos por serem específicos e mesmo assim o trabalho pode apresentar informações relevantes aos objetivos definidos.

Primário 1: Estudo de metodologias para distribuição de requisições em *cluster* de *web services*.

Primário 2: Influência dos softwares *Ganglia*, *JBoss*, *Apache Tomcat* e *Axis2* em pesquisas desenvolvidas na área de *Service-Oriented Architecture* (*SOA*).

Secundário 1: Estudo ou definição de métodos para distribuição de requisições especifica a *web services*.

Secundário 2: Estudo de métodos para distribuição de requisições considerando o estado dos recursos do *cluster*.

Secundário 3: Estudo de métodos para distribuição de requisições tolerantes a falhas e/ou balanceamento de carga.

Secundário 4: Influência dos *softwares* pesquisados na distribuição de requisições em *cluster* de *web services*.

Como critério de exclusão, caso o trabalho em análise não satisfaça nenhum dos critérios de inclusão já definidos, este é eliminado da revisão sistemática.

Ao término do processo de planejamento, inicia-se o processo de condução da revisão sistemática.

2.2.2 Condução da Revisão Sistemática

Esta revisão sistemática foi conduzida por um período de dois meses (setembro/2009 a outubro/2009). Foram recuperados 133 trabalhos e estes submetidos às etapas de seleção preliminar, seleção intermediária, seleção final e extração de resultados de acordo com os critérios de seleção. A seguir são apresentados mais detalhes das atividades realizadas, incluindo a estratégia adotada para construção das *strings* de busca e os resultados das buscas na fonte selecionada.

Seleção Preliminar

A seleção preliminar dos trabalhos é constituída por três etapas:

1. Construção das *strings* de busca;
2. Realização das buscas;
3. Seleção preliminar de trabalhos.

As atividades realizadas em cada etapa do processo de seleção preliminar são detalhadas a seguir.

Construção das Strings de Busca: de acordo com os objetivos propostos por esta revisão sistemática, duas frentes de pesquisa são definidas: (1) busca por métodos para distribuição de requisições em *cluster* de *web services*; (2) busca pela influência dos softwares *Ganglia*, *JBoss*, *Apache Tomcat* e *Axis2* nos trabalhos desenvolvidos dentro do contexto de *SOA*. Portanto, são construídas duas strings de busca para facilitar a realização das buscas e a posterior análise dos trabalhos obtidos. As *strings* de busca são formadas pela combinação dos sinônimos das palavras-chave identificados de maneira que retornem o maior volume

possível de trabalhos dentro deste contexto. Essas *strings* foram submetidas à máquina de busca relacionada e, logo após, foi realizada a leitura dos resumos dos trabalhos recuperados. Constatando-se a relevância de um trabalho, já destacada em seu resumo, e havendo um consenso entre os pesquisadores, este é pré-selecionado para ser lido na íntegra.

As *strings* possuem os conectores lógicos *AND* (e), *OR* (ou) e *NOT* (não) formando as condições necessárias para filtrar os trabalhos relevantes à revisão. As palavras-chave, definidas anteriormente, são utilizadas para restringir o foco da pesquisa dentro do contexto proposto. As palavras-chave podem conter caracteres especiais como o "*", para que seja selecionada qualquer palavra que inicie com as mesmas letras; por exemplo, no caso de *{scal*}* trabalhos que possuem as palavras *scale* e/ou *scaling* são recuperados como resultado. Para definir o campo onde a máquina de busca deve analisar a existência das palavras-chave o argumento "*WN*" é utilizado e em seguida o código referente ao campo onde a palavra-chave deve estar presente. O código referente a cada campo possível de análise é disponibilizado em uma tabela para consulta na própria página de busca no modo "pesquisa avançada". O código "*KY*" define os campos título, *abstract* e o assunto para análise, já o código "*LA*" define o campo idioma (*Language*).

Os termos foram organizados logicamente de maneira que recuperem trabalhos que possuam pelo menos um sinônimo de cada termo definido, mas não exige que estejam agrupados na mesma frase. Desta forma trabalhos relacionados aos objetivos deste trabalho são recuperados sem a necessidade de possuir frases semelhantes a deste trabalho. Por exemplo, frases como "Distribuição de requisições em *cluster* de *web services*", "Organização da demanda de *web services*", "Escalonamento de solicitações de *web services*", "Balanceamento de instancias de *web services*" e "Distribuição de pedidos a *web services*", são retornadas pela busca, entre outras frases que podem ser formadas utilizando os sinônimos contidos nas *strings* de busca.

O idioma definido aos termos utilizados nas *strings* de busca restringiu-se, na prática, ao inglês, devido à busca no idioma português não retornar nenhum resultado.

A *string1* é utilizada com objetivo de abranger os estudos sobre métodos para distribuição de requisições em *cluster* de *web services*. Considerando estas características, a *string1* é apresentada a seguir:

```
(
 (({distrib*} OR {sort} OR {sched*} OR {scal*} OR {balanc*}) WN KY)
 AND
 (({requ*} OR {instantiates} OR {order} OR {demand} OR {solicitation} OR {suit} OR {application}) WN KY)
 AND
 (({web} OR {cluster}) WN KY)
)
AND
(
(({English}) WN LA)
)
```

A *string2* é utilizada para abranger estudos que demonstrem a influência dos softwares *Ganglia*, *JBoss*, *Apache Tomcat* e *Axis2* na área de SOA. A *string2* está detalhada a seguir:

```
(
 (({SOA}) WN KY)
 AND
 (({ganglia} OR {jboss} OR {tomcat} OR {axis2}) WN KY)
)
AND
(
 (({English}) WN LA)
)
```

Realização das Buscas: as buscas com ambas as *strings* foram realizadas na máquina de busca *Engineering Village* no modo "pesquisa avançada" associadas a base de dados *Compendex*. A busca abrange estudos publicados entre os anos de 2000 a 2009, buscas em anos anteriores a estes não apresentam resultados relevantes à revisão. As demais opções são mantidas como padrão (*default*).

As buscas foram realizadas inicialmente em 20/10/2009. Importante citar que as buscas foram repetidas diversas vezes e em datas diferentes, durante a realização deste trabalho, para certificar-se que não houvesse alteração nos resultados das buscas devido a qualquer influência externa. A busca realizada com a *string1* recuperou 122 trabalhos. Enquanto a busca com a *string2* recuperou 11 trabalhos.

Seleção Preliminar de Trabalhos: ao término das buscas, os resumos dos trabalhos resultantes foram lidos. Apesar da busca realizada com a *string1* resultar 122 trabalhos, nenhum trabalho atendeu aos requisitos de inclusão. O contexto dos trabalhos analisados é diferente do foco proposto pela revisão sistemática. A busca realizada com a *string2* resultou

11 trabalhos, após a leitura e análise destes, 8 trabalhos foram identificados com contexto semelhante aos objetivos desta revisão sistemática e uma lista de referências de trabalhos pré-selecionados esta descrita na Tabela 1.

Seleção Intermediária de Trabalhos

Devido alguns trabalhos analisados não possuírem informações suficientes em seus resumos para considerá-los relevantes ou não, foi necessário utilizar o procedimento de Seleção Intermediária. Este procedimento foi empregado em paralelo com a Seleção Preliminar de Trabalhos.

Apesar deste procedimento ser empregado em diversos trabalhos, não houve aumento no número de trabalhos identificados como relevantes a esta revisão sistemática.

Tabela 1: Estudos selecionados na busca sobre influência dos softwares

ID	Títulos	Descrição
01	*An SOA based On-Demand Computation Framework for Spatial Information Processing.*	Implementação baseada em *Java* e *Apache Tomcat/Axis* para processamento de informações.
02	*Hierarchical modeling of the Axis2 web services framework with FMC-QE.*	Propõe uma metodologia de modelagem hierárquica como base para o sistema de modelagem quantitativa e avaliação de desempenho.
03	*Wasabi beans - SOA for collaborative learning and working systems.*	*Framework*, *Wasabi*, baseado em *EJB3* e *JBoss*, adaptável devido ao apoio de *web services*.
04	*A service-oriented architecture based vendor managed inventory system*	Apresenta uma arquitetura orientada a serviços para um sistema *VMI*. *Axis2* é utilizado para dar suporte a esta arquitetura.
05	*Grid service monitoring for grid market framework*	Implementação de um protótipo para monitoramento de serviços em arquitetura *Grid* utilizando *Ganglia*.
06	*A solution to block cross site scripting vulnerabilities based on service oriented architecture.*	Apresenta uma solução contra *XSS*. Esta foi testada em serviço desenvolvido com *JSP* e disponibilizado pelo *JBoss*.
07	*Comprehensive test mechanism to detect attack on web services.*	Apresenta um conjunto de testes para identificação de ataques de *hackers*. Os serviços utilizados para testes são disponibilizados pelo *Axis2*.
08	*A modular architecture for secure and reliable distributed communication.*	Implementação realizada em cima do *Axis2* que possibilita a troca de mensagens seguras e confiáveis em sistemas distribuídos.

Seleção Final e Extração dos Resultados

Nesta etapa, foi realizada a leitura completa dos 8 trabalhos selecionados com os procedimentos de Seleção Preliminar e Seleção Intermediária.

A leitura completa dos trabalhos possibilitou a extração das informações relevantes, definidas na seção de planejamento da revisão sistemática. Foi elaborado um resumo para cada trabalho selecionado demonstrando o contexto dos trabalhos. Estes resumos encontram-se na seção de análise dos resultados.

2.2.3 Análise dos Resultados

Esta seção apresenta um detalhamento dos resultados obtidos, separando-os em dois focos principais: (1) pesquisas sobre metodologias para distribuição de requisições em *cluster* de *web services* e (2) pesquisas sobre a influência dos *softwares Ganglia*, *Apache Tomcat*, *Axis2* e *JBoss* em trabalhos na área de *SOA*.

Síntese dos Trabalhos Selecionados

Provedores de serviços: os provedores de serviços exercem papel fundamental em ambiente *SOA*. Eles são responsáveis por disponibilizar os serviços e podem influenciar no desenvolvimento destes, ocasionalmente o *web service* possui dependência de alguma biblioteca de um provedor específico.

Em ZhenChun, GuoQing (2006) foi proposto um *framework* baseado na demanda computacional de *SOA* para oferecer maior flexibilidade, alto processamento e baixo *overhead* como, por exemplo, em processamento de informações. Para cumprir o objetivo proposto, o *framework* instanciado foi projetado e implementado com base no *Apache Tomcat/Axis*.

Sistemas Orientados a Serviços são sistemas complexos, sendo que, muitas vezes, suas características atuam como grandes facilitadores para a integração de sistemas empresariais existentes. O rápido ritmo de desenvolvimento possibilitado pela *SOA*, por possibilitar a reutilização de funcionalidades (serviços) já desenvolvidas, e a complexidade inerente destes sistemas, dificulta a manutenção dos diferentes componentes do sistema. Muitas vezes tais sistemas estão com problemas de desempenho inesperados, difíceis de prever e localizar. Com o objetivo de sanar esses problemas Copaciu et al. (2007), propõem o sistema *FMC-QE* (*Fundamental Modeling Concepts for Quantitative Evaluation*), que é um modelo matemático

de apoio usado para estimar o desempenho de uma avaliação. Para testar e validar o *FMC-QE* foi utilizado um sistema baseado no *Axis2*. O sistema foi disponibilizado através do *Axis2* implanto no *Apache Tomcat* como uma aplicação web.

Em Schulte et al. (2008), afirma-se que os principais sistemas *CSCW/CSCL*[4] utilizam um único servidor e por isso todos os serviços de colaboração são fornecidos por um sistema. O intuito é usar *SOA* como um princípio de *design* para integrar funcionalidades heterogêneas baseadas na web, permitindo também fácil adaptação. Para tanto, os autores apresentam o *framework Wasabi Beans*, desenvolvido no *EJB* e *JBoss* como uma solução altamente flexível e escalável para sistemas *CSCW/CSCL*. *Wasabi* fornece uma série de funcionalidades não disponíveis nos sistemas *CSCW/CSCL*, já que estes sistemas não são capazes de tratar recentes tecnologias como, por exemplo, *Web 2.0*; além de se adaptar facilmente a vários serviços e sistemas de forma flexível e escalável.

No estudo realizado por Tseng, Huang (2008) é apresentado o sistema *VMIWC* (*Vendor Managed Inventory WS-Coordination*) baseado em *SOA* para reunir requisitos sobre demanda em sistema VMI [5](*Vendor Managed Inventory*). Os componentes do serviço são implantados de maneira flexível no ambiente web através do *Apache Axis2*.

Shanmugam, Ponnavaikko (2007) observaram que cerca de 80% das aplicações *web* são vulneráveis à ameaça *Cross-site scripting* (*XSS*). Com isso, propuseram uma solução desenvolvida em .NET, XML (*Extensible Markup Language*) e *XSD* (*XML Schema Definitions*). Esta solução foi testada em aplicações web desenvolvidas em *JSP/Servlets* e disponibilizadas em um servidor *JBoss*. A arquitetura desenvolvida baseada em *SOA* foi testada em aplicações existentes e se mostrou eficiente no bloqueio de ameaças *XSS*.

Em Siddavatam; Gadge (2008) afirma-se que 70% dos *path* de ataque fechados pelos *firewalls* na última década serão novamente reabertos pelo *XML* utilizado nos *web services*. Por esse motivo foi proposto um conjunto de testes para detecção de ataques a *web services*. O sistema desenvolvido reúne requisições *SOAP* realizadas ao provedor de *web services* durante um determinado período de tempo e diversos testes são realizados. Isso ajuda a detectar se a requisição *SOAP* foi propensa a ataques ou pode criar um ataque. Todas as

[4] *CSCL* (*Computer-supported collaborative learning*) e *CSCW* (*Computer Supported Cooperative Work*) são áreas científicas interdisciplinares que estudam a forma como o trabalho em grupo pode ser auxiliado por tecnologias de informação e comunicação.
[5] *VMI* é um sistema em que o fornecedor se responsabiliza pela gestão dos níveis de estoque dos clientes.

requisições de teste SOAP que falham são separadas para que novas medidas possam ser tomadas. O sistema é desenvolvido em *Java*, juntamente com o *Axis2*.

O trabalho proposto em Jayalath; Fernando (2007) permite a troca de mensagens de forma segura e confiável em sistemas distribuídos. Os autores optaram por implementar tais funcionalidades de forma modular, através da *Sandesha2*, esta responsável pela confiabilidade, e através da *Apache Rampart*, responsável pela segurança das trocas de mensagens. Ambas as soluções foram desenvolvidas sobre o *Axis2* e podem ser utilizadas em conjunto como outros módulos do próprio *Axis2*, possibilitando assim a troca de mensagens de forma confiável e segura em sistemas distribuídos.

Monitor: a monitoração dos recursos computacionais é essencial em diversos aspectos como, por exemplo, fornecer dinamismo no comportamento de um sistema.

Em Peng et al. (2006) foi proposto um *framework* voltado a *Grids* para construção de um suporte à comercialização de *Grids*, a fim de permitir que organizações compartilhem recursos com fins lucrativos. Em tal ambiente voltado ao mercado, a monitoração de serviços do *Grid* é uma parte importante para a recuperação do estado dos serviços e realimentação das informações. Por isso foi abordada principalmente uma arquitetura para a monitoração de serviços do *Grid*. Um protótipo da arquitetura foi implementado com base no monitor *Ganglia*. Além da publicação de métricas sobre desempenho, o protótipo permitiu a publicação de outras métricas, classificadas pelos autores como *non-performance*.

Análise em Relação às Questões de Pesquisa

Influência dos softwares Ganglia, JBoss, Apache Tomcat e Axis2 em pesquisas desenvolvidas na área de SOA: pode-se observar que os softwares abordados possuem grande influência nos trabalhos em que são utilizados na área de SOA. Estes softwares influenciam nas metodologias propostas nos estudos e também em suas validações.

Estes softwares foram utilizados nos estudos de acordo com as naturezas de suas funcionalidades. Por exemplo, o *Ganglia* foi utilizado para monitoração do *status* dos serviços em *Grid* computacionais, enquanto o *JBoss*, *Apache Tomcat*, *Axis2* foram utilizados para disponibilizar serviços, auxiliar e validar as implementações realizadas nos trabalhos.

Estudo de metodologias para distribuição de requisições em cluster de web services:apesar da busca ter sido automatizada através da máquina de busca *Engineering Village* utilizando a base de dados *Compendex* e ter recuperado 122 trabalhos, ao aplicar os critérios de inclusão nenhum trabalho com foco na área de distribuição de requisições em *cluster* de *web services* foi identificado. Com isso, nenhum trabalho técnico/cientifico que aborde este tema pode ser analisado nesta seção.

Essa revisão sistemática demonstra a existência de uma lacuna por estudos em metodologias e análise científica na distribuição de requisições em *cluster* de *web services*.

2.2.4 Conclusão

A realização da busca em diversas bases de dados indexadas foi simpificada com a utilização da máquina de busca *Engineering Village*. Essa dificuldade (a busca em diversas bases), citada em Ferrari; Maldonado (2007), foi superada, pois tal máquina é responsável por automatizar a busca de forma que não seja necessário gerar uma nova *string* para cada base indexada.

Ao analisar os trabalhos recuperados com a *string2*, com o propósito de determinar a influência dos softwares *JBoss*, *Apache Tomcat*, *Axis2* e *Ganglia* em estudos realizados na área de *SOA*, pode-se concluir que os softwares pesquisados possuem alto nível de importância nos trabalhos em que estão inseridos. Os objetivos propostos nos trabalhos são baseados nestes *softwares* ou dependem diretamente de suas funcionalidades para serem alcançados e validados. Conforme os critérios utilizados para a seleção dos trabalhos, a Figura 1 demonstra o aumento de estudos realizados dentro do contexto de *SOA* com os softwares mencionados na área de *SOA*. Nenhum estudo, anterior ao ano de 2006, com este escopo foi retornado pela busca.

A busca sobre métodos para distribuição de requisições em *cluster* de *web services* não retornou nenhum trabalho publicado, permitindo, assim, concluir que existe uma lacuna nessa linha de pesquisa.

Apesar das semelhanças citadas na literatura entre escalonamento de processos em *clusters* e a distribuição de requisições em *clusters* de *web services*, esta revisão sistemática não encontrou nenhum trabalho técnico ou científico que tenha avaliado a distribuição de

requisições e suas implicações. Visto as semelhanças e também as peculiaridades existentes em cada uma dessas atividades, considera-se importante a realização de um estudo científico que avalie a influência e implicações da distribuição de requisições em *cluster* de *web services*.

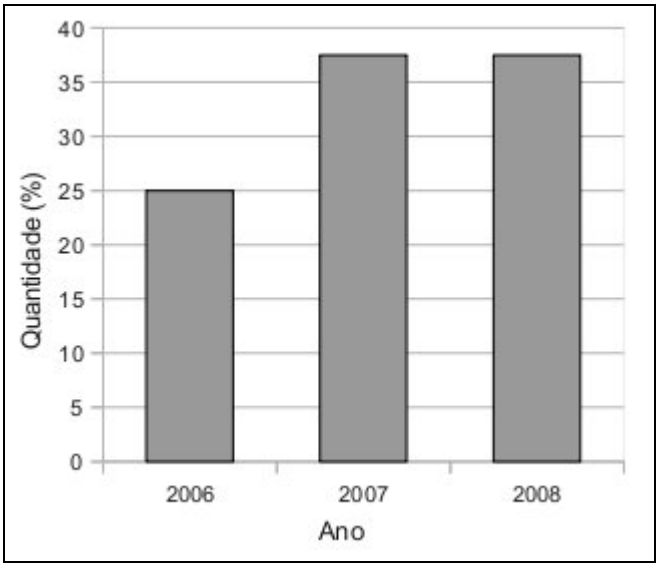

Figura 1: Percentual dos trabalhos realizados entre os anos de 2006 e 2009

2.3 Apache Tomcat

O *Apache Tomcat* é um *container* de *servlets* que possui a característica de ser a implementação de referência de *servlets*. O seu intuito é ser completamente compatível com a especificação de *servlets* definida pela *Sun Microsystems* (Moodie, 2008).

O *Apache Tomcat* foi formado originalmente por dois softwares, o *Java Web Server* desenvolvido pela *Sun Microsystems* e o *Jserv* criado pela *ASF* (The Apache Software Foundation, 2009). Ambos foram criados com o intuito de serem *containers* de *servlets*. O primeiro *software* não era suficientemente robusto para os padrões de seus criadores, o que fez com que seu código fosse doado para a *ASF*. Em 1999 houve a fusão do *Jserv* e do *Java Web Server*. A primeira versão do *Apache Tomcat*, derivada dessa fusão foi a versão 3.0. Desde 2001 foram lançadas as versões 4, 5 e 6 sendo que nas versões 4 e 5 houve reconstruções completas da arquitetura do *software* (Moodie, 2008). Atualmente, também se encontra disponível a versão 7 do *Apache Tomcat*. Esta versão implementa as especificações

da *Servlet 3.0* e *JavaServer Pages 2.2* da *Java Community Process* (The Apache Software Foundation, 2012).

O *Apache Tomcat* fornece suporte à execução em *clusters*. O suporte é focado em replicação de sessões e publicação automática entre os nós formadores do *cluster*. A replicação de sessões é responsável por continuar a mesma interação que um nó estava fazendo com um cliente mesmo que o nó se torne indisponível. As interações entre cliente e servidor no *Apache Tomcat* geram alguns dados, como variáveis ou conexões com banco de dados feito pelo *servlet*, os quais podem ser armazenados e recuperados à medida que o cliente navega entre páginas dinâmicas. Os dados gerados são armazenados em sessões. Para que se mantenha o funcionamento de uma sessão, mesmo que ocorra interrupção de funcionamento em um nó, os outros nós precisam também ter as sessões armazenadas. Surge então o conceito de replicação, pois as sessões iniciadas em um nó são replicadas em outros nós. Além do suporte a sessões em *clusters*, o *Apache Tomcat* fornece também a publicação automática, ou seja, uma vez que uma aplicação seja publicada em um nó, a publicação é repassada aos outros que também fazem a publicação da aplicação (Moodie, 2008) (The Apache Software Foundation, 2012).

O *Apache Tomcat* oferece um método de distribuição de requisições através da sua ligação com o *Apache HTTP Server* pelo *AJP* (*Apache Jserv Protocol*). Essa ligação está presente em estruturas cujo intuito é fornecer páginas de internet com conteúdo estático e dinâmico. Em estruturas como essas, o *Apache Tomcat* é usado para processar *servlets* ou *JSP*. Além de *servlets* e JSP, essa estrutura também pode atender requisições para conteúdo dinâmico processados por um interpretador *PHP* (The Apache Software Foundation, 2012).

Quando o *Apache HTTP Server* recebe uma requisição para uma página dinâmica (*servlet* ou JSP) e identifica que deve enviar para o *Apache Tomcat*, ele o faz pelo módulo *mod_jk*, este descrito a seguir.

2.3.1 O Módulo Mod_jk

Mod_jk é um conector que se agrega ao *Apache HTTP Server* através de uma biblioteca de ligação dinâmica.

O *mod_jk* recebe a requisição repassada pelo *servidor HTTP* e, de acordo com algumas configurações, realiza o encaminhamento para o *Apache Tomcat*. O módulo pode ser configurado para trabalhar em *clusters* com distribuição de requisições e balanceamento de

carga. Informações como utilização do processador, memória ou outros recursos computacionais não são obtidos diretamente dos nós do cluster pelo *mod_jk*. A política adotada pelo *mod_jk* para distribuir a carga de trabalho considera, no máximo, o número de requisições em atendimento nos *web services*, bem como o número de interações realizadas (demanda sobre a rede) no contexto de *web services*. Influências externas aos *mod_jk*, como processos em execução em segundo plano ou processos agendados, podem concorrer com o *Apache Tomcat* nos nós, eventualmente retardando o processamento dos serviços.

O módulo *mod_jk* não toma decisões considerando possíveis cargas de trabalhos concorrentes ao provedor de *web services*. De acordo com *The Apache Software Foundation* (2010), o módulo *mod_jk* possui algumas políticas para distribuição de requisições disponíveis, são elas:

Request: esta política utiliza a informação referente à quantidade de requisições distribuídas aos nós do *cluster* durante um espaço de tempo, a fim de determinar o destino da requisição, considerando o parâmetro configurável *lbfactor* (prioridade do nó). O nó que possuir a menor quantidade de requisições encaminhadas, de forma proporcional ao valor definido no parâmetro *lbfactor*, será eleito como destino da requisição. Este comportamento é encontrado em políticas baseadas em *round-robin*. A configuração simples do parâmetro *lbfactor* (com valores iguais) determina uma *round-robin* comum, enquanto que a existência de nós com maior prioridade consiste em uma *round-robin* ponderada. Esta é a política definida como padrão a ser utilizada para balanceamento de carga.

Session: possui comportamento semelhante à política *Request*, porém, utiliza a quantidade de sessões como métrica para definir o destino da requisição. Esta política não conhece o estado atual de nenhuma sessão, por isso, contabiliza cada acesso, sem um *cookie* de sessão (ou a codificação de *URL*), como uma nova sessão. Por não possuir conhecimento de quando uma sessão é invalidada, de qual é o tempo limite da sessão ou ainda quais são os nós sobrecarregados, a utilização desta política é indicada se as sessões forem o fator limitante do *cluster*. Isso ocorre, por exemplo, se as sessões necessitarem de grande quantidade de memória.

Traffic: considera informações sobre o tráfego de rede entre o *mod_jk* e o *Apache Tomcat* em um espaço de tempo, de acordo com prioridades definidas através do parâmetro *lbfactor* (semelhante à política *Request*). Assim, o nó que possuir menor tráfego de rede é

eleito como destino para a requisição, por ser considerado mais adequado a atendê-la. O tráfego de rede concorrente à interação entre o *mod_jk* e o *Apache Tomcat* é ignorado, comprometendo a decisão sobre o destino da requisição.

Busyness: têm como objetivo eleger o nó menos "ocupado" como destino das requisições, baseando-se na fila de requisições pendentes proporcional ao *lbfactor* de cada nó. Esta informação é obtida através da divisão da quantidade de requisições direcionadas ao nó, que ainda encontram-se pendentes, pelo valor do *lbfactor*. O resultado deste cálculo é o parâmetro considerado para eleger o nó de destino. Assim, o nó que possuir a menor fila de requisições pendentes é considerado o nó mais adequado (menos ocupado) a atender a requisição.

2.4 JBoss

O servidor de aplicações *JBoss* foi desenvolvido em 1999, baseado em *software* livre. Devido à sua popularização, em 2001 foi criado o *JBoss Group*, com objetivo de fornecer-lhe suporte especializado. No ano de 2004 o *JBoss Group* passa a ser uma empresa (Fleury et. al., 2005).

O *JBoss* é fornecido sob licença livre seguindo os padrões da *Red Hat* e apresenta versões gratuitas e versões empresarias (pagas). A diferença entre as versões é o suporte oferecido pela empresa *Red Hat*. O *JBoss* apresenta uma solução completa para integração de aplicações empresariais atuando sobre a maioria das tecnologias *Java* voltadas para esta finalidade. Este servidor de aplicações possui uma de suas frentes de atuação relacionada aos *web services*, fornecendo todo o ambiente necessário para publicação dos serviços e a interface de programação de aplicação, baseada atualmente em anotações. O *JBoss* possui o *Apache Tomcat* embarcado, possibilitando a execução de *web services* através deste último (Matos, 2009).

Em relação ao *Apache Tomcat* embarcado no *JBoss*, seus proponentes afirmam que esse *software* está no mesmo nível de outros servidores *web* de alto desempenho existente como *IIS* e *Apache HTTP Server* (Red Hat, 2009).

A existência de aplicações no mercado que funcionam com base em um servidor de aplicações utilizando *EJB*[6] (Matos, 2009), é considerado um fator positivo para a utilização do *JBoss*. Outra tecnologia utilizada no *JBoss* é persistência de dados. A persistência de dados evoluiu nos últimos tempos com o intuito de fornecer produtividade e otimizar a manipulação de dados na camada de aplicação. Um *framework* conhecido para esta finalidade é o ***Hibernate*** (Red Hat Middleware, 2009). A idéia por trás do *framework* de persistência é que o desenvolvedor da aplicação manipule somente objetos e o *framework* se encarregue de mapear os objetos ativos para um banco de dados relacional, cuidando de sua gravação, recuperação e atualização no banco de dados. É também responsável por garantir a consistência dos dados que estão sendo operados na memória. Considerando o *framework* em questão, ressalta-se que o *JBoss* possui suporte nativo (Matos, 2009).

O *JBoss* possui duas arquiteturas básicas para *clustering*: interceptores *client-side* (*proxies* inteligentes ou *stubs*) e balanceadores de carga externos (JBoss, 2012).

Na arquitetura "interceptores *client-side*", representada na Figura 2, os clientes possuem um *proxy* responsável por decidir o nó que irá atender a requisição. Este *proxy* é gerado pelo servidor e implementa a interface de negócio do serviço. O cliente realiza a chamada local ao objeto *proxy*, e por sua vez o *proxy* se responsabiliza em encaminhar automaticamente a chamada remota aos serviços gerenciados nos servidores (JBoss, 2012).

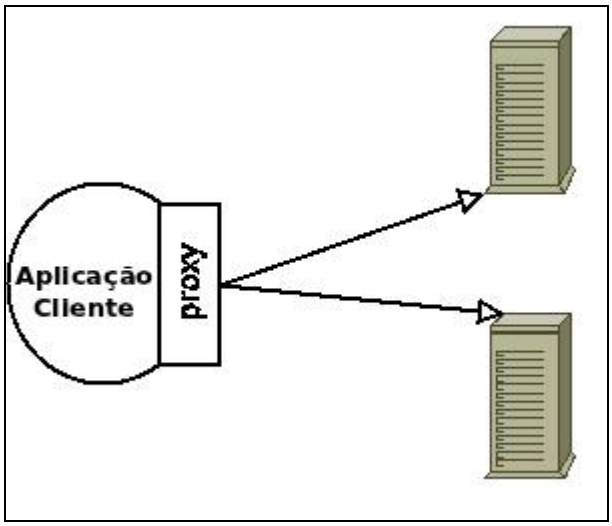

Figura 2: Arquitetura utilizando Interceptores *client-side*

[6] *Enterprise JavaBeans* é um dos principais elementos da plataforma *Java 2 Enterprise Edition*. É um componente do tipo servidor que executa no *container* do servidor de aplicação. A tecnologia *EJB* busca fornecer um desenvolvimento rápido e simplificado de aplicações *Java* distribuídas, transacionais, seguras e portáveis.

Nesta arquitetura estão disponíveis quatro políticas para distribuição de requisições aos nós do *cluster*:

- ***Round-Robin***: nesta política cada requisição é enviada para um novo nó com base no conceito de fila circular. O primeiro nó a receber a requisição é eleito de maneira aleatória;
- ***Random-Robin***: para cada requisição um nó é eleito aleatoriamente como destino;
- ***First Available***: cada *proxy* elege um nó disponível como destino principal para as requisições encaminhadas por ele. Este nó de destino é eleito aleatoriamente em uma lista de nós disponíveis no *cluster*. Caso ocorra alguma alteração na lista de nós disponíveis, esta política elege um novo nó de destino para suas requisições, exceto se a alteração não afetar o nó já eleito pelo *proxy*. Se um cliente utilizar dois *proxies* para realizar suas requisições, cada *proxy* poderá eleger um nó de destino diferente do outro;
- **First Available Identical All Proxies**: possui o comportamento semelhante à política "First Available", exceto que o nó destino eleito é compartilhado por todos os *proxies* no lado do cliente. Portanto, se um cliente utilizar dois *proxies* para realizar requisições a um serviço, todos os *proxies* utilizarão o mesmo nó destino. No caso de diferentes nós serem eleitos por *proxies* em clientes distintos para atenderem requisições referentes ao mesmo serviço, estes nós formaram um "grupo" ou "família" e serão compartilhados para atender as requisições dos clientes que também formarão um "grupo" requisitante de um determinado serviço. Desta forma, pode-se ter "N" nós fornecendo suporte ao serviço requisitado por "M" clientes.

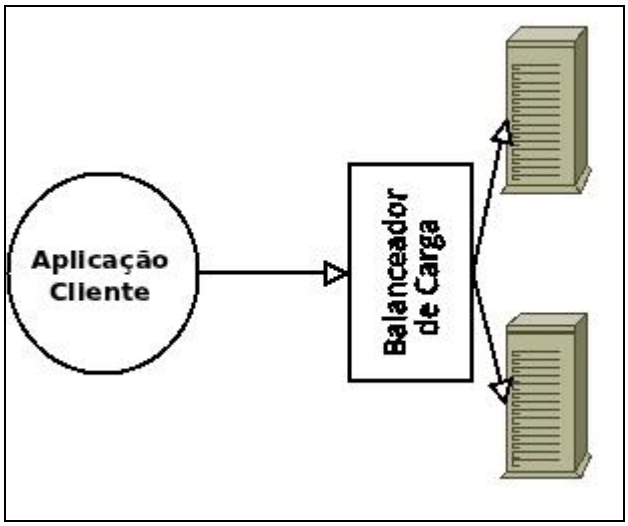

Figura 3: Arquitetura utilizando Balanceador de Carga Externo

Na segunda configuração de arquitetura básica citada para *clustering*, representada na Figura 3, são utilizados balanceadores de carga externos. Os balanceadores de carga externos fornecem seus próprios recursos para balanceamento de carga. Nesta opção de arquitetura podemos citar dois balanceadores de carga externos: *mod_jk*, descrito na seção 2.3.1 e mod_cluster, detalhado a seguir.

2.4.1 O Modulo Mod_cluster

Mod_cluster é um balanceador de carga baseado em httpd[7]. Assim como *mod_jk* e *mod_proxy*[8], *mod_cluster* usa um canal de comunicação para encaminhar as requisições do httpd para um conjunto de nós de servidores de aplicações. Ao contrário de *mod_jk* e *mod_proxy*, o *mod_cluster* utiliza uma conexão adicional entre os nós de servidores de aplicações e o *httpd*. Os servidores de aplicações utilizam este canal para transmitir métricas internas do servidor para balanceamento de carga e eventos do ciclo de vida para o *httpd*. Isso é feito através de um conjunto de métodos customizados *HTTP*, amigavelmente chamados de *Mod_Cluster Management Protocol* (MCMP). Este canal possibilita ao *mod_cluster* oferecer um nível de inteligência e granularidade não encontrado em outra solução para balanceamento de carga (Mod_cluster, 2011).

[7] *httpd* é um programa servidor de *HyperText Transfer Protocol* (*HTTP*), ou seja um servidor *web*. Ele é projetado para ser executado como um processo *daemon standalone* (processo executado em segundo plano de maneira independente).
[8] Este módulo implementa um *proxy*/*gateway* para o *Apache HTTP Server* (servidor *web*).

De acordo com (Mod_cluster, 2011), o *mod_cluster* apresenta as seguintes vantagens sobre os outros balanceadores de carga baseados em *httpd*:

- Configuração dinâmica dos nós *back-ends*;
- Cálculo da métrica de balanceamento de carga interna dos servidores de aplicações;
- Granulação fina no controle da vida de aplicações *web*;

Mod_cluster utiliza memória compartilhada para armazenar a descrição dos nós. Esta memória compartilhada é criada na inicialização do *httpd* e a estrutura para cara item é fixada.

Segundo (Mod_cluster, 2011), o *mod_cluster* possui as limitações a seguir:

- Tamanho máximo para *alias* (pseudônimo) de 40 caracteres;
- Tamanho máximo para contexto de 40 caracteres;
- Tamanho máximo para nome do balanceador de 40 caracteres;
- Tamanho máximo para *JVMRoute string*[9] de 80 caracteres;
- Tamanho máximo para o nome do grupo de balanceamento de carga de 20 caracteres;
- Tamanho máximo para o *hostname* de um nó de 64 caracteres;
- Tamanho máximo para o número da porta para um nó de 7 caracteres;
- Tamanho máximo para descrever o método de um nó de 6 caracteres (ex: *http*, *https*, *ajp*);
- Tamanho máximo para o nome de *cookie* de 30 caracteres;
- Tamanho máximo para *path* de 30 caracteres;
- Tamanho máximo para *sessionid* (identificação de sessão) de 120 caracteres.

Uma das principais características do *mod_cluster* é a capacidade de usar métricas de carga interna ao servidor de aplicações para determinar qual a melhor forma de balancear as requisições. A classe *DynamicLoadBalanceFactorProvider* é a responsável por calcular o

[9] *JVMRoute string* é um atributo único entre todas as instancias de servidores de aplicações que compõem o *cluster.*

fator de balanceamento de carga em um nó utilizando a métrica definida em sua configuração. A métrica pode ser configurada com um "peso" associado a sua importância.

O *mod_cluster* pode ser configurado para realizar composição de métricas (combinar mais de uma métrica) para o cálculo do fator de carga de cada nó. Cada métrica de carga fornece um valor para a taxa de ocupação do nó. Os fatores de carga resultantes de cada métrica são agregados de acordo com seus pesos, originando um fator de carga geral do nó. De modo geral, o fator de carga é dado pela fórmula: *(carga/capacidade) * peso/peso_total* (Mod_cluster, 2011).

A classe *DynamicLoadBalanceFactorProvider* aplica uma função de deterioração de tempo nas cargas retornadas por cada métrica, buscando manter um histórico do comportamento da carga dando maior importância para o histórico mais recente. A carga agregada, com relação a valores de carga anterior, pode ser expressa pela fórmula: $L=(\sum^{H}_{i=0} L_i/D^i) * (\sum^{H}_{i=0} D^i)$ (Mod_cluster, 2011).

Considera-se na fórmula que **L** determina o fator de carga, **D** o fator de deterioração e **H** o histórico. Se o atributo *history* for definido como "0" a função de deterioração do tempo é desativada e apenas a carga atual para cada fator de carga será considerada no cálculo do fator de balanceamento de carga. O *mod_cluster* espera que o fator de carga seja um inteiro entre 0 e 100, onde 0 indica carga máxima no nó e 100 indica nenhuma carga. Portanto, o fator de carga final enviado ao *mod_cluster* resulta da fórmula: $L_{Final} = 100 – (L * 100)$ (Mod_cluster, 2011).

São disponibilizadas oito métricas para utilização na classe *DynamicLoadBalanceFactorProvider*. Estas métricas estão divididas em três grupos: *Web Container metrics*, *System/JVM metrics* e *Other metrics*. *Web Container metrics* são métricas relacionadas ao servidor de aplicações (métricas internas do servidor de aplicações), o *System/JVM metrics* são métricas relacionadas ao sistema operacional e a *JVM* (*Java Virtual Machine*) e por fim, *Other metrics* são métricas consideradas não pertencentes a nenhum dos outros dois grupos (Mod_cluster, 2011).

Web Container metrics

ActiveSessionsLoadMetric: Análoga ao *method=S* do *mod_jk*, esta métrica retorna a quantidade de Sessões ativas. Ela também requer o atributo "*capacity*" definido de maneira explícita.

BusyConnectorsLoadMetric: Análoga ao *method=B* do *mod_jk*, esta métrica retorna o percentual de *threads* conectores do *pool* de *threads* que estão ocupados servindo requisições.

ReceiveTrafficLoadMetric e *SendTrafficLoadMetric*: Análogas ao *method=T* do *mod_jk*, estas métricas retornam em KB/segundos o tráfego recebido e enviado, respectivamente, entre os servidores de aplicações. Estas métricas requerem o atributo "*capacity*" definido de maneira explícita.

System/JVM metrics

AverageSystemLoadMetric: Retorna a carga de trabalho da CPU. Requer *Java 1.6* ou superior.

SystemMemoryUsageLoadMetric: Retorna a quantidade de memória utilizada do sistema de memória. Requer *JDK* (*Java SE Development Kit*) ou *OpenJDK* (*Open Java Development Kit*).

HeapMemoryUsageLoadMetric: Retorna o percentual de memória utilizada, baseado na quantidade máxima de memória que pode ser utilizada. Requer *Java 1.6* ou superior.

Other metrics

ConnectionPoolUsageLoadMetric: Retorna o percentual de conexões em uso da *JCA Connection Pool*.

2.5 Ganglia

Ganglia é uma ferramenta que permite a monitoração de diversas informações dos nós de um *cluster* de maneira simplificada (Massie et al., 2004), visto que proporciona grande flexibilidade. Foi desenvolvido na universidade de Berkeley e seu projeto tem como base a distribuição hierárquica de *clusters* formando federações. Seus desenvolvedores afirmam que o Ganglia pode ser utilizado em *clusters* formados por até dois mil nós, graças às estruturas de dados e algoritmos utilizados, que causam baixa sobrecarga. Existe uma lista de grandes *players* de mercado que utilizam o Ganglia para monitoração de seus *clusters*. Além de todo o seu poder de monitoração de grandes estruturas, a sua instalação e configuração são simples, apresentando bom desempenho também em pequenas estruturas (Matos, 2009).

O Ganglia é composto por um núcleo e algumas ferramentas. O seu núcleo é constituído pela ferramenta *gmond*. Este é um *daemon* que deve estar presente em todos os nós do *cluster*, coletando e publicando as informações sobre a sua carga. Esse *daemon* possui por padrão diferentes métricas, tais como: informações referentes à configuração do nó, a utilização do processador, quantidade de processos ativos/esperando E/S, uso de memória, tráfego de rede e uso do disco rígido.

A propagação das informações coletadas é realizada através de *multicast*[10]. Isso possibilita que todos os nós do cluster que contenham o *gmond* possam ter informações de todos os outros nós. Os *gmonds* possuem capacidade de coletar informações de outros nós e publicar as suas próprias informações. Por isso há um cuidado referente à propagação das informações. Quando um *gmond* recebe uma conexão ele envia todas as informações codificadas dentro de um documento *XML*. Isso simplifica a interface entre outros programas e o *gmond*, tornando possível que programas externos tenham acesso às informações realizando uma conexão *TCP* ao *gmond* e um processador de *XML* para interpretar o arquivo obtido. O Ganglia também possibilita uma configuração centralizada, onde as informações de todos os nós do *cluster* permanecem em um único nó. Assim, um *gmond* é determinado como principal e é configurado para apenas receber as informações; enquanto os outros são configurados para enviar suas informações para o *gmond* principal.

2.6 O Protótipo Jerrymouse

O protótipo *Jerrymouse* (também chamado de Distribuidor) é uma ferramenta de *software* que implementa arquitetura de distribuição de requisições para *cluster* de *web services* proposta em (Matos,2009). O principal objetivo desta arquitetura é possibilitar a distribuição de requisições em *clusters* de *web services* de maneira flexível e dinâmica (Matos, 2009). Centrada nesse objetivo maior, a distribuição pode ser realizada de diferentes formas, de acordo com objetivo específico da política aplicada às requisições atendidas pelo *cluster*.

O *Jerrymouse* pode ser utilizado com diferentes provedores de *web services*, desde que estes possibilitem a utilização de um balanceador de carga externo, fornecendo a

[10] *Multicast* é a entrega de informação para múltiplos destinatários simultaneamente utilizando a estratégia de que as mensagens só passam por um *link* uma única vez.

distribuição de requisições de forma flexível e dinâmica para *cluser* de *web services* (Matos, 2009).

A Figura 4 exibe a estrutura do *Jerrymouse* com todos os seus elementos que fornecem as funcionalidades de flexibilidade e dinamicidade. Os elementos são: núcleo, gerenciador de configuração, gerenciador de conectores, gerenciador de adaptadores para monitoração e gerenciador de políticas.

Núcleo: O núcleo é o elemento central do *Jerrymouse*. Ele é o primeiro elemento a ser executado e o responsável pelo gerenciamento dos demais componentes. O núcleo é a ponte de ligação entre os conectores e as políticas. No início da execução, o núcleo realiza o processo de carga das configurações. Após carregar as configurações, segue com o processo de carga de todos os gerenciadores, apresentados adiante. Os gerenciadores são executados de maneira paralela ao núcleo. Com os gerenciadores iniciados, o núcleo permanece em execução até que eles (gerenciadores) finalizem as suas execuções, retornando a linha de execução para o núcleo. O processo de tomada de decisões passa pelo núcleo, usando-o como uma ponte para os dados gerais do programa que contém as referências para as políticas (Matos, 2009).

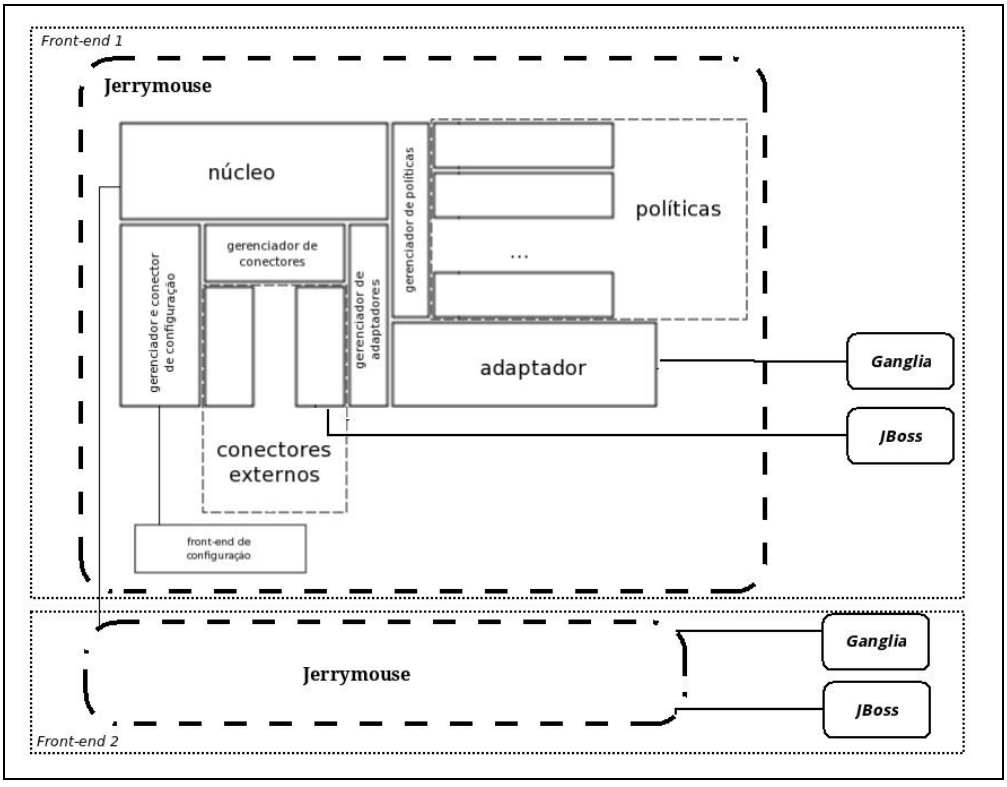

Figura 4: Estrutura do *Jerrymouse* com todos os seus elementos e a possível conexão entre dois *Jerrymouse*, caso o cluster possua mais de um *front-end (Matos, 2009)*.

Gerenciador de Configuração: O gerenciador de configuração atua como um conversor das configurações para que todos os componentes do distribuidor possam acessá-las de maneira universal. A função inicial dele é carregar o arquivo que contém toda a configuração do sistema, descrita em *XML*. A validação do *XML* é usada para minimizar possíveis erros na maneira que o administrador configura o *Jerrymouse*. Em Matos (2009), a validação do *XML* é feita através do *XML Schema* [11]e o *DTD*[12] (*Document Type Definition*), realizada de maneira automática pela *libxml2* (biblioteca para linguagem C utilizada no processamento de arquivos *XML*). Após a validação das configurações, o realizado o *parsing*, transformando as informações obtidas em uma representação binária que pode ser utilizada pelos componentes do *Jerrymouse*. Após terminar as etapas iniciais, o gerenciador de configuração permanece ativo e retorna a linha de execução para o núcleo. A sua atividade consiste em esperar por alguma interação entre o administrador e o *Jerrymouse* ou mesmo uma interação *Jerrymouse-Jerrymouse*. O gerenciador de configuração também possui a função de propagar as atualizações de configuração para outros *Jerrymouse* na rede (Matos, 2009).

Gerenciador de Conectores: O *Jerrymouse* é projetado para ter um acoplamento fraco com os provedores de *web services*. Dessa forma, é necessário que o *Jerrymouse* possua uma interface de interação com o provedor. Essa interface é viabilizada pelos conectores. Os conectores podem receber conexões de diferentes provedores. O gerenciador de conectores é o elemento que, executado a partir do núcleo, lê as configurações sobre os conectores já convertidas pelo gerenciador de configuração e instancia os conectores. Ele inicialmente coloca os conectores em execução que ficam então em funcionamento paralelamente a ele, esperando por conexões externas. Como o ambiente é dinâmico, é gerenciador de conectores precisa manter uma referência para todos os conectores ativos. Quando uma mensagem chega a este gerenciador, partindo do gerenciador de configuração, ele se encarrega de notificar o conector. Os conectores podem ser ativados, desativados (permanecendo na memória), adicionados ou então removidos (Matos, 2009).

Gerenciador de Adaptadores para Monitoração: Os adaptadores para monitoração atuam como um *link* entre o *Jerrymouse* e a ferramenta de monitoração usada. O gerenciador de monitoração é responsável por carregar os adaptadores que serão ligados aos monitores

[11] *XML Schema* é uma linguagem baseada no formato *XML* para definição de regras de validação ("esquemas") em documentos no formato *XML*.
[12] *DTD* define quais as *tags* que podem ser usadas em um documento *XML* e quais os valores válidos.

presentes no sistema. Ele (Gerenciador de Adaptadores para Monitoração) pode ativar, desativar, adicionar e remover os adaptadores do *Jerrymouse*. Além da função de gerenciamento, ele também possui a função de ser uma ponte de ligação entre a política e os adaptadores. Os adaptadores são iniciados por esse gerenciador e permanecem ativos paralelamente até que uma notificação de encerramento seja realizada (Matos, 2009).

Gerenciador de Políticas: Este gerenciador é responsável por gerenciar todas as políticas durante a execução do *Jerrymouse*. Com isso, é possível existir dentro do *cluster* diversas políticas ativas em paralelo. Tais políticas são fundamentais para que o *Jerrymouse* proporcione uma distribuição flexível e dinâmica das requisições aos nós do *cluster* (Matos, 2009). Quando o gerenciador de políticas é iniciado, ele utiliza os dados presentes na memória do *Jerrymouse* para buscar e carregar as políticas. As políticas, diferentemente dos conectores, não permanecem ativas. Elas são executadas apenas quando uma decisão é requisitada ao *Jerrymouse*. As políticas podem ser ativadas, desativadas, adicionadas e removidas por meio desse gerenciador (Matos, 2009).

2.6.1 Distribuição de requisições utilizando Jerrymouse

A estrutura básica para a distribuição das requisições é composta por um distribuidor de requisições (*Jerrymouse*), um provedor de serviços atuando como *front-end* (*Apache Tomcat* ou *JBoss*) e um sistema de monitoração (*Ganglia*); todos apresentados na Figura 5. As requisições alcançam a arquitetura através de um link de conexão utilizando o protocolo *SOAP* (passo 1) e são recebidas por um *front-end*, o qual implementa um provedor de serviços e a processa para então tomar uma decisão de distribuição. É possível que exista mais de um *front-end* para atender as requisições submetidas à arquitetura, levando em consideração as necessidades em relação à carga do sistema. Na Figura 5, a requisição *SOAP* é atendida pelo nó (ou *host*) de número 2. Após a requisição ter sido recebida pelo provedor de serviços (passo 1), este consulta o distribuidor de requisições (passo 2), o qual retorna o endereço *IP* do nó para onde a requisição deve ser encaminhada. A escolha do nó do *cluster* é realizada de acordo com a política de distribuição aplicada pelo distribuidor, esta vinculada ao serviço requisitado. Cada serviço disponibilizado pelo *cluster* pode conter uma política específica para a distribuição de suas requisições. Considerando a proposta de distribuição dinâmica, outro elemento envolvido no processo de escolha do destino da requisição é o monitor. Esse elemento deve estar presente em todos os possíveis nós de destino para que as informações

sobre seus recursos sejam publicadas e as políticas de distribuição possam acessá-las. Quando o provedor de serviços recebe o endereço *IP* do distribuidor de requisições (passo 3), ele redireciona a requisição para o nó de destino eleito (passo 4). Em Matos (2009) o reenvio da requisição é efetivado como uma reconstrução da requisição, porém, em nome do *front-end*. O ciclo de atendimento da requisição é finalizado (passo 5) com o atendimento da requisição pelo provedor de *web services* em execução no nó de destino (*back-end*) eleito. Após o término da execução do *web service*, a resposta da requisição é enviada ao *front-end* (passo 6). Ao receber a resposta, o *front-end* re-encaminha a resposta ao cliente que realizou a requisição (passo 7) (Matos, 2009). Desta forma, o uso do *Jerrymouse* para distribuição das requisições no *cluster* de *web services* é transparente ao usuário por encapsular a maneira que a requisição é atendida no *cluster*.

Ocasionalmente, o próprio *front-end* pode ser eleito para atender a requisição em questão, caso a política aplicada considere que este seja o nó mais indicado. Isso é possível porque o distribuidor de requisições possui acesso às informações sobre a disponibilidade dos recursos de cada nó, publicadas pelo monitor.

Como pode ser observado na Figura 5, o intuito é que o *hardware* possua o menor acoplamento possível com o software, possibilitando assim diversas configurações e alcançando alta flexibilidade. O distribuidor de requisições é colocado no mesmo nó que o provedor de serviços com o objetivo de diminuir a sobrecarga na comunicação. Porém é possível realizar uma configuração em que ambos estejam em nós diferentes. Visto que o acoplamento do distribuidor com o provedor de serviços utiliza poucas informações, a comunicação entre esses dois componentes é realizada de maneira flexível, possibilitando assim várias configurações arquiteturais (Matos, 2009).

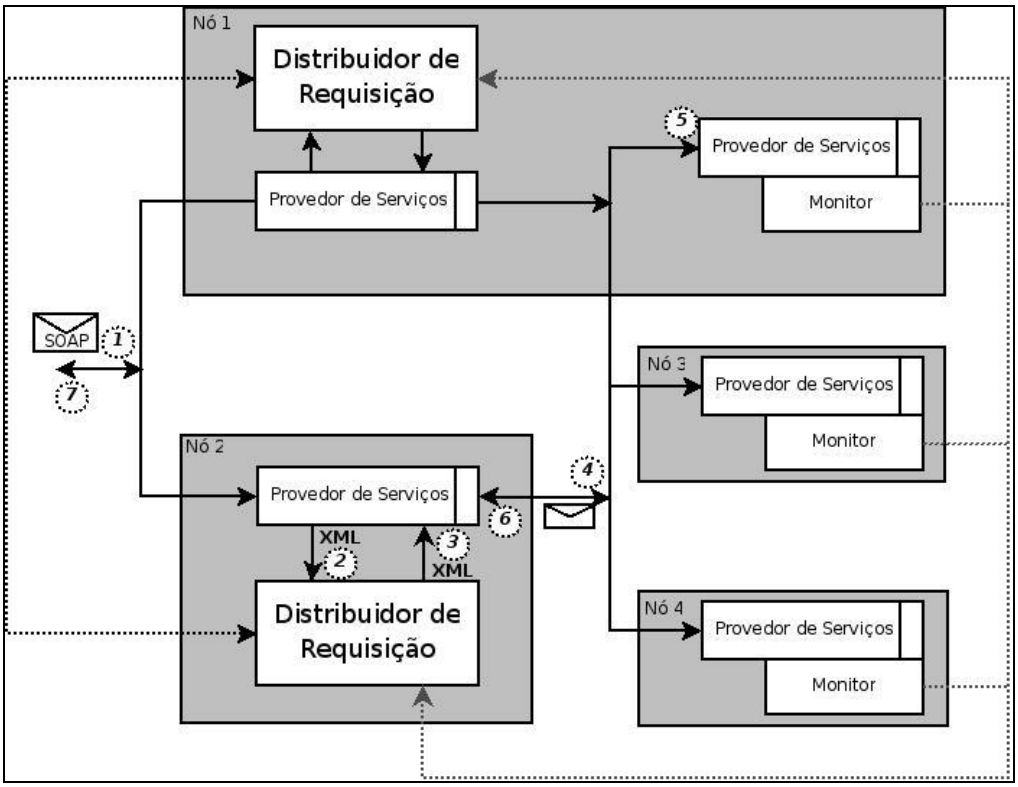

Figura 5: Execução da Arquitetura - baseada em Matos (2009)

2.7 Considerações Finais

Foram vistos nesse capítulo os resultados obtidos com a revisão sistemática realizada com duas frentes de pesquisas, uma sobre a importância dos *softwares* relacionados a estudos no contexto de *SOA* e a segunda frente de pesquisa sobre políticas para distribuição de requisições em *cluster* de *web services*. Também foram abordados os principais *softwares* que fornecem suporte à *web services*, assim como o funcionamento e os detalhes de cada componente do protótipo de distribuição de requisições *Jerrymouse*.

No próximo capítulo é apresentada uma avaliação sistemática de diversos índices de carga. O principal objetivo desta avaliação é analisar os aspectos qualitativos de diferentes índices sob diferentes demandas geradas por serviços *web*.

3 Índice de Carga

3.1 Considerações Iniciais

Diversas métricas são utilizadas como índice de carga para demanda gerada por *web services* sem avaliação de sua qualidade neste contexto (Marzolla; Mirandola, 2007) (Al-Masri; Mahmoud, 2007) (Liu; Gorton; Zhu, 2007) (Porter; Katz, 2006). Esta seção apresenta uma avaliação sistemática de diversos índices de carga conhecidos no contexto de otimização do escalonamento de processos e na distribuição de requisições em *clusters* de *web services*. O principal objetivo da avaliação descrita neste capítulo é analisar o comportamento de diferentes índices de carga frente ao desempenho observado nos serviços executados e sob diferentes demandas geradas por tais serviços.

3.2 Motivação

Os requisitos operacionais de alta confiabilidade, alta segurança e menor tempo de execução dos serviços são objetivos que a maioria dos servidores *web* buscam otimizar, normalmente através da adoção de técnicas como, por exemplo, balanceamento da carga gerada pelas requisições aos *web services* (Rosenberg; Platzer; Dustdar, 2006). A carga de trabalho é uma abstração representada por um índice de carga, usado por políticas responsáveis pela distribuição das requisições em servidores individuais em um *cluster* de provedores. Com isso, o objetivo passa a ser escolher o provedor alvo mais adequado para minimizar o tempo de resposta de um *web service* individual (Ardagna; Pernici, 2006).

A otimização do balanceamento de carga é um problema com dificuldade já reconhecida. Os níveis de desempenho em servidores de *web service*s variam em função de diferentes fatores, dentre eles o volume de trabalho gerado dinamicamente pelas requisições dos serviços. Além disso, a carga simultânea gerada pelos provedores de *web services*, como o *Apache Tomcat* (The Apache Software Foundation, 2009) e *JBoss* (Red Hat, 2011), também afetam o desempenho geral do sistema. A utilização destes sistemas em *clusters* de provedores apresenta diferenças significativas na operação do sistema, quando comparado com outros contextos, tais como *High Performance Computing* (*HPC*), bancos de dados distribuídos e servidores *web* com conteúdo estático. Tais diferenças impõem-se pelas

diferentes camadas de *software* existente para dar suporte à execução dos *web services*, bem como à natureza heterogênea dos serviços que serão executados.

Um índice de carga, normalmente, representa a quantidade de carga de trabalho em um servidor. Ferrari e Zhou (1987) definem um índice de carga como uma variável não-negativa, iniciando do zero (para representar um recurso ocioso) e aumentando seu valor conforme o aumento da carga de trabalho. Apesar da definição apresentada, a representação pode ser inversa à definição, onde quanto menor o valor do índice de carga maior a carga de trabalho (ex: memória disponível). Tal índice tem objetivo de representar uma carga de trabalho passada e prever o desempenho esperado do servidor em um futuro próximo. Vários índices foram adotados para este fim no contexto de composição de *web services*, como por exemplo, o número de requisições encaminhadas, o tamanho da fila de processos-prontos, a quantidade de memória livre e o tempo de execução dos processos balanceados na plataforma (Dyachuk; Deters, 2007). No entanto, na maioria dos casos, índices de carga têm sido adaptados sem uma análise completa da adequação e eficiência resultante de seu uso no contexto de distribuição das requisições em *cluster* de *web services*.

Outra característica relevante para índices de carga é a baixa intrusão na sua obtenção. Esta característica está relacionada com a maneira que é realizada a monitoração dos recursos, a forma que o índice é obtido e de como é realizada sua divulgação para ser usado. Este conjunto de tarefas é essencial para determinar a qualidade de um índice de carga.

3.3 Monitorando Web Services

A monitoração de *web services* fornece diversas informações que possibilitam estimar o desempenho em um futuro próximo, evitar gargalos, otimizar o uso dos recursos do sistema e melhorar a relação custo-benefício global (Li; Jin; Han, 2006).

O servidor de aplicações *JBoss* foi desenvolvido com estrutura modular e utilizando a *JMX* (*Java Management Extensions*) (JBoss, 2010). *JMX* é uma *API* (*Application Programming Interface*) que permite gerenciar e monitorar a *Java VM* (*Java Virtual Machine*). Ela permite monitorar qualquer aplicação Java, desde que instrumentada adequadamente. A *JMX* fornece ferramentas para a construção de soluções distribuídas modulares e dinâmicas baseadas na *web* para gerenciar e monitorar dispositivos, aplicações e serviços orientados a rede. A *JMX* possibilita instrumentar o código *Java*, criar agentes *Java*

inteligentes, implementar um *middleware* de gerenciamento distribuído, e integrar suavemente essas soluções no gerenciamento de sistemas de monitoramento já desenvolvidos (Oracle, 2012).

Através da *JMX* embarcada no *JBoss* é possível gerenciar e monitorar tanto o servidor de aplicações quanto os *web services* disponibilizados nele (JBoss, 2010). Assim, é possível obter diversas informações sobre o provedor de aplicações e também dos *web services* disponibilizados. Entre as informações obtidas da *JMX* pode-se citar, por exemplo, o tempo (em milissegundos) decorrido desde a inicialização do servidor de aplicação, quantidade de requisições recebidas por um *web service*, tempo total (em milissegundos) de processamento dos *web services* no atendimento das requisições recebidas, quantidade de requisições que não foram atendidas corretamente, lista dos *web services* disponíveis no servidor de aplicações e tempo gasto para tornar o *web service* disponível à requisições (carregar o *web service*).

A *JMX* fornece uma interface para se comunicar com outras aplicações através de conectores. Esta interface permite interagir de forma transparente com um agente *Java* e os seus recursos *JMX* administráveis (Oracle, 2012) (JBoss, 2010). Utilizando esta interação é possível obter informações disponíveis com a monitoração de maneira transparente, flexível e com baixa intrusão. Os conectores suportam conexões locais e/ou remotas, aumentando sua flexibilidade na comunicação com aplicações em execução paralela em relação ao servidor de aplicações.

O suporte à interação com outros *softwares* também é uma característica encontrada no *Ganglia*. A monitoração feita pelo *Ganglia* (este já descrito anteriormente neste trabalho) possui funcionalidades que vão ao encontro da demanda gerada pelos *web services* nos servidores. Outro aspecto positivo é a interação com outros *softwares* que complementam esta monitoração no contexto de *web services* (como a *JMX*). Através da ferramenta *gmetric* (interna ao *Ganglia*) é possível incluir e/ou atualizar novos índices de carga na lista de divulgação do *Ganglia*. A comunicação entre a ferramenta *gmetric* e o *daemon gmond* (responsável pela comunicação entre os demais *daemons* em execução no *cluster*) é realizada por meio de mensagens *TCP/IP*.

A interligação entre o *JMX* e o *Ganglia* permite ao último gerenciar e publicar novos índices de carga, proporcionando um método flexível e eficiente para monitorar servidores de *web service*s. Considerando as características encontradas no *JMX* e no *Ganglia*, um módulo

associado ao *gmetric* foi desenvolvido para coletar índices a partir do *JMX* existente no *JBoss* e enviá-los para o *gmond*. Neste caso o *gmetric* que age como um elo entre esses dois ambientes: *JBoss*/*JMX* e *gmond*. O *gmond* se responsabiliza em distribuir os valores dos seus índices para as outras instâncias do *gmond* em execução em outros nós do *cluster*. Esta estrutura distribuída permite que todo o *cluster* mantenha informações atualizadas sobre o estado da plataforma.

A combinação entre *Ganglia* e *JMX* fornece um quadro bastante robusto para a monitoração de *web services*. No entanto, uma questão importante no monitoramento é determinar qual o índice de carga se deve monitorar para representar a carga de trabalho na plataforma. Esta não é uma questão trivial de ser respondida, devido à diversidade das demandas impostas por *web services*. Na prática, as diferentes implementações selecionam diferentes índices de carga para representar a carga de trabalho de *web services*. A próxima seção descreve os principais índices de carga atualmente utilizados no contexto de *web services*.

3.4 Índices de Carga

A correta utilização de um índice de carga deve considerar os objetivos gerais da plataforma e métricas de desempenho associadas a cada objetivo. Alguns exemplos de objetivos utilizados são (Porter; Katz, 2006): reduzir o tempo de resposta, aumentar o *throughput* da plataforma, melhorar o balanceamento de carga e proporcionar maior tolerância a falhas. Destes, a redução do tempo de execução dos serviços é um objetivo comum e, em muitos casos, até mesmo ortogonal a outros objetivos. O índice de carga coleta dados da plataforma para quantificar o comportamento geral do sistema no contexto dos objetivos. Eles também podem estimar um "futuro próximo" baseado no desempenho de um "passado recente". É neste ponto em particular de estimar um desempenho futuro que o uso dos índices de carga apresenta comumente distorções de interpretação. A qualidade de um índice de carga pode ser determinada através de critérios definidos e propriedades desejáveis, tais como (Ferrari; Zhou, 1987): 1) a precisão suficiente para representar a carga de trabalho de forma adequada, mesmo sob diferentes demandas computacionais, 2) uma relação direta com as métricas de desempenho (idealmente linear) de tal forma que o índice de carga pode ser facilmente aplicado para prever o provável desempenho futuro, 3) a estabilidade, em particular, suavizando os picos de carga de trabalho; 4) escalabilidade, permitindo um baixo

custo de implementação para ser mantida mesmo em plataformas de grande porte. Os custos de coleta do índice de carga e a estratégia adotada para transmiti-lo para outros nós são muito importantes porque afetam diretamente a escalabilidade.

Além de propriedades quantitativas, os índices de carga devem apresentar outras propriedades qualitativas, tais como a capacidade de encapsular detalhes de arquiteturas e sistemas operacionais (SO), capacidade de suportar a portabilidade entre plataformas heterogêneas e um objetivo geral de fornecer transparência às aplicações, provedores e serviços. Não é trivial satisfazer todas essas propriedades ao mesmo tempo, especialmente porque algumas delas são contraditórias. Por exemplo, aumentando-se a precisão do índice, pode-se aumentar o custo de obtenção dos dados; e reduzindo-se os falsos picos de desempenho em um índice geralmente diminui a sua precisão e a sua capacidade de prever o desempenho futuro.

Aplicativos no contexto de *HPC*, tradicionalmente usam diferentes índices de carga, muitos deles relacionados com o agendamento de processos e balanceamento de carga. Alguns exemplos clássicos são (Chen et. al., 2008): o número de processos prontos, o percentual de tempo ocioso da CPU, o percentual de uso de memória e o número de conexões de rede ativas. O número de processos prontos tem sido amplamente utilizado como índice no contexto *HPC*, porque pode ser facilmente extraído do sistema operacional e encapsula diversos detalhes relacionados ao desempenho geral do sistema (Ferrari; Zhou, 1987).

Normalmente, *web services* utilizam índices de carga para prever o desempenho em pelo menos dois casos: durante a fase de descoberta e na distribuição das requisições para os nós *back-ends* do *cluster* receptor das requisições (Marzolla; Mirandola, 2007). Os algoritmos na fase de descoberta usam índices de carga para selecionar as melhores opções de serviços, registradas nos *brokers* pelos provedores através do protocolo *UDDI* (Al-Masri; Mahmoud, 2007). Índices de carga também são utilizados após a fase de descoberta, quando o nó *front-end* de um provedor de *web services* deve distribuir um pedido recém-chegado aos nós *back-end* de um *cluster* de provedores (Chen et. al., 2008).

Diferentes índices de carga, quantitativos e qualitativos, podem ser encontrados na literatura científica no contexto de *web services* (Liu; Gorton; Zhu, 2007). Alguns exemplos são: o número de requisições (recebido, finalizado ou esperando para ser atendido), o menor tempo de resposta; *throughput*; disponibilidade (*uptime*, percentual para um serviço em um

período); confiabilidade (a probabilidade de receber uma resposta correta em um tempo máximo previsto), segurança (que representa a garantia de confidencialidade e autenticação); precisão (a taxa de erro do serviço) e integridade (a capacidade de completar a correta execução de uma transação de serviços).

Em adição aos vários índices de carga descritos na literatura científica, provedores de serviços (por exemplo, *JBoss* (Red Hat, 2011) e *Apache Tomcat* (*The Apache Software Foundation*, 2010) usam na prática índices como: requisições enviadas, sessões ativas, tráfego de rede (entre os provedores de *web services*) e requisições pendentes.

Com objetivo de analisar e qualificar o comportamento de diversos índices de carga no contexto de *web services*, um estudo experimental foi realizado monitorando o estado do computador provedor de serviços. Este estudo é detalhado na próxima seção.

3.5 Estudo Experimental

Este estudo experimental considerou a execução de *web services* em uma plataforma formada por três nós com processador *Intel Core 2 Quad 2.66 GHZ* e *6MB cache*, *4GB DDR2 RAM* e um *Hard Disk Sata-2* de *500GB/7200RPM*. A base de dados usada pelos *web services* foi *MySQL*, versão 5.1. A rede usada para interligar os computadores é uma *Gigabit Ethernet* com um *switch TP-LINK TL-SL3428*. O sistema operacional é o *GNU/Linux*, *kernel 2.6.31-22*. Os compiladores usados são *gcc 4.3* e *Java 1.6.0_20*. O provedor de *web services* usado foi *JBoss 5.1.0.GA*. O *Ganglia* 3.1.2 e o *JMX* 5.1.0.GA com suas configurações padrões foram usados para monitorar a plataforma.

Embora a plataforma usada seja relativamente modesta (Figura 6), ela atende o objetivo deste estudo experimental de analisar e comparar o comportamento dos diversos índices de carga em um provedor de *web services*. Esta plataforma é capaz de suportar uma demanda controlada gerada a partir da execução de aplicações clientes (executadas em um primeiro nó da plataforma) que solicitam serviços hospedados em um servidor monitorado no segundo nó da plataforma. Os aplicativos clientes enviam as requisições para o nó provedor através de mensagens *HTTP* com encapsulamento *SOAP*. O nó provedor encaminha a mensagem *HTTP* ao seu provedor de *web services* local (*JBoss*), que executa o serviço solicitado. O terceiro nó hospeda um servidor de banco de dados (MySQL) que manipula as solicitações dos *web services*.

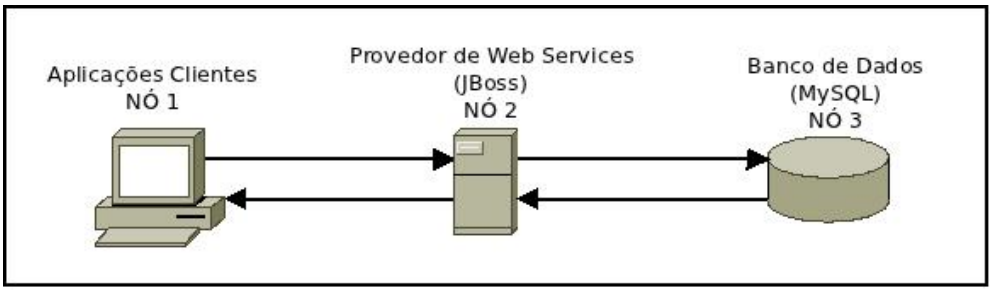

Figura 6: Plataforma utilizada no estudo experimental

Este estudo experimental considera a execução de três diferentes serviços no nó provedor, gerando cargas de trabalho controladas e específicas sobre a CPU (*CPU-bound*), memória (*Memory-bound*) e rede (*Network-bound*). Um quarto serviço gera carga de trabalho sobre a memória, disco rígido e interface de rede do servidor de banco de dados remoto, para simular um ambiente onde bancos de dados remotos são acessados pelos *web services* disponibilizados (OSDB, 2010).

O serviço *CPU-bound* executa instruções que realizam operações aritméticas em matrizes de inteiros e de ponto flutuante, estruturas de controle condicional, estruturas de repetição e pequenas chamadas de procedimentos (Sabetta; Koziolek, 2008). Esse serviço evita operações de E/S e as matrizes usadas possuem tamanho pequeno para serem armazenadas dentro do *cache* e, portanto, evitar acessos à memória principal. O serviço *CPU-bound* retorna a quantidade de interações e do tempo de execução.

O serviço *memory-bound* avalia a largura da banda de memória entre dois nós, utilizando o processamento de vetor com vetores de tamanho grande o bastante para eliminar os efeitos de *cache* e permitir que a descrição dos resultados seja a largura de banda contínua (Sabetta; Koziolek, 2008). Os principais resultados retornados por este serviço são: o tempo médio das operações, o *throughput*, menor tempo e o maior tempo.

O serviço *network-bound* gera demanda na rede entre o servidor de banco de dados e o servidor *front-end* (nó 1). Este é inspirado pelo *benchmark netperf*, que é baseado em um modelo cliente-servidor. *Netperf* foi originalmente desenvolvido pela *Hewlett-Packard* para medir o desempenho da rede (Binkert et. al., 2005). É constituída por uma pequena estrutura de repetição, que enche um *buffer* e o envia usando conexões *TCP*. Os principais resultados retornados por este serviço são: o total de *bytes* enviados/recebidos, o tempo decorrido e o *throughput*.

O serviço que requisita o banco de dados é baseado no *benchmark OSDB*, uma versão *open source* do *benchmark AS³AP*, que foi originalmente desenvolvido pela *Computer Corporation Compaq* (OSDB, 2010). Este serviço inclui a criação e acesso de tabelas com dados provenientes de arquivos de texto e também a criação de índices.

A avaliação do comportamento do índice de carga foi realizada sob três perspectivas diferentes (Ferrari; Zhou, 1987): a sua precisão para representar a carga de trabalho, sua relação com indicadores de desempenho (tempo de execução, no nosso caso) e sua estabilidade.

As experiências não consideram a fase de descoberta dos *web services*, uma vez que esta fase não afeta o comportamento do índice de carga no nó *front-end*. Cada instancia do aplicativo cliente realiza exatamente 1.000 iterações, cada um executando a seguinte seqüência: solicita o serviço, aguarda até que receba o resultado, em seguida, exibe o resultado na tela. Cada pedido demora cerca de 10s, quando executado sem concorrência de outros serviços.

Inicialmente, um, dois, quatro e oito *threads* do aplicativo cliente foram executados simultaneamente, solicitando cada serviço em separado. O experimento com quatro *threads* de aplicações, por exemplo, enviou quatro pedidos simultâneos para o mesmo serviço no servidor. Essa estratégia permitiu a análise dos índices de carga sob cargas de trabalho específicas e homogênea, o que foi o foco desta avaliação em particular. Após estas execuções, duas instâncias de cada serviço foram solicitadas simultaneamente, através de oito *threads* dos aplicativos clientes em execuções simultâneas no nó cliente. Este segundo conjunto de execuções permitiu a análise do comportamento dos índices de carga sob cargas heterogêneas.

A Tabela 2 mostra os índices de carga considerados nestes experimentos. A porcentagem de tempo ocioso da *CPU* (*CPU_Idle*), o tempo de espera da *CPU* por E/S (*CPU_WIO*), memória livre (*Free Memory*), a média de processos na fila de prontos no último minuto (*Ready Processes*) e os *bytes* enviados/recebidos para/pela rede (*Bytes In/Out*), são todos representados por índices. *System Saturation* (Voorsluys; Santana, 2010) indica se o subsistema de memória está sobrecarregado, considerando o número de páginas ativas em comparação com o número de páginas inativas. *Swap Used* também está relacionado à

utilização do subsistema de memória, no entanto, neste caso, considera a quantidade de *bytes* já utilizados para *swap* (Voorsluys; Santana, 2010).

Tabela 2: Índices de carga analisados nos experimentos.

Índice de carga	Descrição do índice de carga	Relacionado a
CPU_Idle	Percentual da CPU disponível	*CPU-bound*
CPU_WIO	Percentual da CPU aguardando E/S	*I/O-bound*
Free Memory	Memória livre em MB	*Memory-bound*
Swap Used	*Bytes* usados para *swap*	*Memory-bound*
System Saturation	Páginas ativas / páginas inativas	*Memory-bound*
Bytes in e *Bytes out*	Quantidade de *bystes* recebidos e enviados pela rede	*Network-bound*
Ready Processes	Quantidade média de processos na fila de prontos, considerando o último minuto	*Operating System*
Total Amount of Submitted Requests by JMX	Soma feita pelo *JMX* de todas as requisições recebidas pelo *web services*. Este índice é acumulativo.	*Web Service*
Requests Amount	Quantidade de requisições recebidas entre a penúltima e a última monitoração do total de requisições	*Web Service*
Client Response-Time	Tempo de resposta da requisição coletada diretamente na aplicação cliente	*Web Service*
Service Runtime	Tempo de execução no servidor coletada diretamente no código do serviço	*Web Service*
Service Total Runtime	Soma do total do tempo de execução do serviço feito pelo *JMX*. Este é um índice acumulativo.	*Web Service*
Service Average Runtime ***(Average-JMX_Time)***	Média aritmética usando janela deslizante com o total do tempo de execução pelo *JMX* e o total de requisições submetidas pelo *JMX*	*Web Service*

Total Amount of Submitted Requests by JMX determina o número acumulado de requisições enviadas a um serviço específico hospedado no servidor desde que este entrou em operação. O objetivo deste índice é simplesmente avaliar o índice de *Average-JMX_Time* (ou tempo médio de execução do serviço, explicado por último). A quantidade de pedidos recentemente apresentadas (*Requests Amount*) indica a diferença entre as duas últimas amostragens da quantidade total de requisições submetidas. Os dois índices, tempo de resposta para o cliente (*Client Response-Time*) e tempo de execução do serviço para o servidor (*Service Runtime*), foram usados principalmente para analisar o comportamento de outros índices, uma vez que o objetivo do aumento de desempenho é reduzir o tempo de execução de serviços em geral. *Service Total Runtime* é um índice acumulado, com a soma de todos os tempos de execução dos serviços desde que o servidor entrou em operação, avaliado pelo JMX. O novo índice *Service Average Runtime* (*Average-JMX_Time*) representa uma média das amostras recém-coletadas, considerando-se duas janelas deslizantes com cinco posições cada uma. Como mostrado na Equação 1, onde **N** é a quantidade total de amostras, que foram realizadas, **i** é a enésima amostra e **n** é o enésimo valor calculado para este índice.

Equação 1: Equação do índice *Average-JMX_Time*

$$Average - JMX\_Time_n = \frac{\sum_{i=N-4}^{N} Service\ Total\ Runtime\ by\ JMX_i}{\sum_{i=N-5}^{N-1} Total\ Amount\ of\ Submitted\ Requests\ by\ JMX_i}$$

O tamanho da janela de cinco posições foi empiricamente escolhido para reduzir os picos de desempenho, mantendo os dados mais recentes para a composição do índice. Uma janela mais estreita poderia melhorar a precisão do índice de carga enquanto que uma janela mais ampla iria melhorar a estabilidade, evitando picos de carga de trabalho indesejáveis. Estas janelas possuem um deslocamento de uma posição entre as elas, como apresentado na Figura 7. A quantidade total de requisições não considera a última amostragem, pois representa apenas as requisições submetidas e não as concluídas. O tempo de execução total do serviço, por outro lado, representa o tempo de execução das requisições finalizadas. Esta mudança minimiza o impacto das requisições submetidas e ainda não concluídas. Embora essa mudança não garanta que a média irá considerar somente solicitações concluídas, ela minimiza o problema. Os resultados empíricos mostram um excelente comportamento do índice *Average-JMX_Time* quando comparado com o tempo de execução de serviço coletadas diretamente do código do *web service* executado no servidor.

#	Tempo Total de Execução por JMX
n	$Tempo\ Total\ de\ Execução_n$
n-1	$Tempo\ Total\ de\ Execução_{n-1}$
n-2	$Tempo\ Total\ de\ Execução_{n-2}$
n-3	$Tempo\ Total\ de\ Execução_{n-3}$
n-4	$Tempo\ Total\ de\ Execução_{n-4}$
n-5	$Tempo\ Total\ de\ Execução_{n-5}$
...	...

#	Quantidade Total de Requisições por JMX
n	$Quantidade\ Total\ de\ Requisições_n$
n-1	$Quantidade\ Total\ de\ Requisições_{n-1}$
n-2	$Quantidade\ Total\ de\ Requisições_{n-2}$
n-3	$Quantidade\ Total\ de\ Requisições_{n-3}$
n-4	$Quantidade\ Total\ de\ Requisições_{n-4}$
n-5	$Quantidade\ Total\ de\ Requisições_{n-5}$
...	...

Figura 7: Janelas deslizantes utilizadas para avaliar o índice *Average-JMX_Time*. As áreas cinzentas indicam as janelas deslizantes usadas com o índice tempo total de execução do serviço por *JMX* e o índice quantidade total de requisições concluídas.

O *Ganglia* (Massie et al., 2004) e o *JMX* (Mahmoud, 2011) foram utilizados para coletar os índices de carga distintos nos nós, enquanto os serviços estavam em execução. O

Gmond em execução no nó *front-end* realizava amostragens a cada 8s e enviava as informações para o nó cliente. Um *script* no nó cliente coletava todos os índices publicados pelo *Ganglia* e armazenava em um arquivo de *log* a cada 10s. O desvio padrão (DP) e o intervalo de confiança de 99% foram calculados para todos os resultados.

Neste estudo experimental a Média Móvel Exponencial (MME) - ou *Exponential Moving Average* (*EMA*) - foi utilizada para todos os resultados, usando uma janela com as últimas cinco amostragens. O MME apresentado na Equação 2 é semelhante à média móvel simples, exceto que um peso maior é dado para as últimas amostragens. Na Equação 2, *Vt* é a última amostragem feita no tempo *t*, e *N* é o tamanho da janela (5, neste caso).

Equação 2: Equação do MME utilizado nos resultados

$$EMA_t = (V_t - EMA_{t-1}) \cdot \frac{2}{1+N} + EMA_{t-1}$$

3.6 Resultados

Os principais resultados obtidos nos experimentos realizados estão organizados pelos índices de carga nesta seção e podem ser visualizados da Figura 29 até a Figura 63 e da Tabela 5 até a Tabela 12 apresentadas no Anexo I – Resultados da Análise sobre Índices de Carga. Estes resultados foram apresentados considerando três fatores principais: a capacidade de representar a carga de trabalho real, a relação com tempo de execução do serviço (a métrica de desempenho adotada) e a sua estabilidade.

O *Ganglia* reuniu os índices localmente a partir do Sistema Operacional ou do *JMX* e, em seguida, a sua estrutura de difusão foi utilizada para minimizar os custos de publicação para os nós remotos.

Os principais resultados observados em nosso estudo experimental estão apresentadas nas próximas subseções, agrupados por índice de carga.

3.6.1 Percentual de CPU ociosa (CPU_Idle)

O índice *CPU_Idle* foi capaz de representar a carga de trabalho gerada na *CPU*, independentemente da demanda gerada pelo serviço (ver Tabela 6 e na Figura 29 até a Figura 33). Os resultados obtidos com a execução do serviço de base de dados apresentam um

comportamento estável, mesmo considerando cargas externas para a *CPU*. No entanto, este índice apresenta fraca estabilidade em outros casos, principalmente quando existem muitos processos na fila de processos-bloqueados durante um período significativo (ver Figura 29, a Figura 30 para 8 clientes e Figura 33). Nestes casos, esta instabilidade pode ser usada para indicar que o sistema foi saturado.

Os resultados com o índice *CPU_Idle* obtidos nestes experimentos, demonstram estar correlacionados com o desempenho do serviço de banco de dados durante a execução de demandas heterogêneas (Figura 33). Isso aconteceu porque esse índice foi capaz de indicar o aumento do seu valor no instante exato em que o serviço *cpu-bound* terminou e o tempo de execução do serviço de banco de dados diminuiu. No entanto, não têm o mesmo comportamento com outros serviços, em execução simultânea, neste experimento. O índice *CPU_Idle* não apresenta correlação com os tempos de execução quando demandas homogêneas foram geradas no nó *front-end*, mesmo para o serviço de base de dados (Figura 29 até a Figura 32). Estes tempos de execução foram influenciados devido a outras características da plataforma, tais como número de núcleos da *CPU* e *throughput* da rede.

3.6.2 Percentual de CPU esperando E/S (CPU_WIO)

Os valores do índice de *CPU_WIO* foram próximos de zero nos experimentos realizados (ver Figura 29 até a Figura 33 e Tabela 6) devido à demanda gerada, o qual não foi suficiente para gerar enfileiramento significativo de processos bloqueados à espera de E/S. A exceção neste caso foi o serviço de *memory-bound* com 8 clientes (Figura 30), quando o índice *CPU_WIO* teve uma elevada instabilidade, indicando possível saturação do sistema. Este índice foi capaz de representar a baixa carga de trabalho gerado, no entanto, não se correlacionou bem com os tempos de execução.

3.6.3 Free Memory

O índice *Free Memory* foi capaz de representar a carga de trabalho específica gerada pelo serviço *memory-bound* de forma estável, exceto quando o sistema de memória é saturado (Tabela 7 Figura 34 até à Figura 38). Os resultados deste serviço (Figura 35) tiveram uma maior instabilidade quando comparados a outros serviços, indicando a capacidade de alocar e liberar áreas de memória com freqüência.

O índice *Free Memory* não se correlacionou com os tempos de execução do serviço *cpu-bound*, serviço *network-bound* e serviço de banco de dados (Figura 34, Figura 36 e Figura 37), por estes não solicitarem memória suficiente para terem seu tempo de execução influenciados. O serviço de *memory-bound* (Figura 35), por outro lado, teve o seu tempo de execução influenciado. No entanto, a quantidade de memória usada é diferente da freqüência de acesso. O serviço considerado nos experimentos foi implementado para solicitar e liberar grandes quantidades de memória com grande freqüência. Não é esperado um impacto significativo no tempo de execução quando um serviço acessar uma grande quantidade de memória apenas ocasionalmente. Este índice não foi capaz de fornecer uma representação adequada da carga de trabalho sob demanda heterogênea (Figura 38).

3.6.4 Swap Used e *Memory Saturation*

O índice *Swap Used* manteve-se inalterado durante quase todas as execuções dos experimentos (da Figura 34 até Figura 38). A exceção foi na execução do serviço *memory-bound* com 8 clientes (Figura 35), quando indicou uma alta carga de trabalho acompanhada de instabilidade. Os resultados para o índice *Memory Saturation* permaneceram inalterados durante as execuções com cargas inferiores (a partir da Figura 39 até a Figura 43) e apresentaram uma elevada instabilidade com cargas de trabalho mais elevadas (Figura 40). Ambos os índices poderiam indicar uma sobrecarga do sistema, a partir do ponto de vista do sistema de memória.

3.6.5 Bytes-in e *Bytes-out*

Os índices *Bytes-in* e *Bytes-out* foram capazes de representar a carga de trabalho gerada na interface de rede, de forma independente do serviço (a partir da Figura 44 até a Figura 48 e Tabela 9). Serviços que requerem mais acessos à rede tiveram suas atividades representadas (ver Figura 46, Figura 47 e Figura 48); e serviços com menor demanda de rede também podem ser representados (Figura 44 e Figura 45). A quantidade de *bytes* enviados através da rede foi reduzida devido a demanda por *CPU*, como pode ser observado na Figura 44 para 8 clientes e Figura 48. Estes dois índices demonstram instabilidade principalmente sob alta carga de trabalho.

A carga de trabalho representada por estes dois índices não apresenta relação com o tempo de execução do serviço, nem para o serviço de *network-bound*, que foi especialmente

diferente da esperada. Outros recursos da plataforma não correlacionados com *Bytes-in* ou *Bytes-Out* influenciaram os tempos de execução neste caso.

3.6.6 Ready Processes

O índice *Ready Processes* foi capaz de representar a carga de trabalho dos serviços (Tabela 10 e na Figura 49 até a Figura 53). As cargas de trabalho do serviço *network-bound* e o serviço de banco de dados indicaram valores mais baixos e instáveis (ver o tempo médio na Tabela 10 e Figura 51 e Figura 52), porque estes serviços exigem operações de E/S e, por isso, eles foram transferidos para a fila de processos bloqueados durante a execução. Este índice foi capaz de representar a carga de trabalho heterogênea gerada durante a execução simultânea dos serviços, como pode ser observado na Figura 53. O índice relacionado a processos na fila de prontos indicou um comportamento instável em alta demanda, como observado quando 8 clientes realizaram requisições simultâneas para serviços diferentes e também para a execução simultânea dos serviços.

Este índice não indica uma relação direta com o tempo de execução dos serviços *cpu-bound*, *network-bound* e de banco de dados, como esperado, porque os seus tempos de execução foram afetados por outros recursos plataforma. Este comportamento pode ser comparado tanto com o serviço *memory-bound* e com a execução simultânea dos serviços, onde as mudanças de tempo de execução foram relacionadas com os valores do índice (ver Figura 50 e Figura 53 para o serviço de base de dados).

Estes resultados mostram que, neste caso, este índice pode fornecer resultados suficientemente precisos para estimar o desempenho futuro, se os serviços executados simultaneamente contiverem processos que permanecem em sua maioria na fila de pronto.

3.6.7 Total Amount of Submitted Requests

O índice *Total Amount of Submitted Requests* foi capaz de representar a carga de trabalho gerada por serviços nos experimentos (Tabela 11 e Figura 54 até Figura 58). No entanto, as medições efetuadas indicam alta instabilidade do índice.

Este índice não correlacionou bem com os tempos de execução do serviço, visto que, embora a carga de trabalho alterasse os respectivos tempos de execução, não tem um comportamento correspondente. Isto pode ser observado na Figura 54 para 1, 2 e 4 clientes, Figura 56 e Figura 57.

3.6.8 Client Response-Time e *Average-JMX_Time*

O índice *Client Response-Time* (Tabela 12 e as Figura 59 até Figura 63) esteve próximo dos tempos de execução do serviço em nossos experimentos, conforme o esperado. Esta relação de proximidade é devido à proximidade do nó cliente e do nó *front-end*, através de uma rede de comunicação *Ethernet Gigabit*. Os tempos de resposta para os clientes poderiam ser maiores se a tecnologia da rede de comunicação fosse mais lenta entre o cliente e nó servidor ou demanda sob a rede fosse maior. O índice de *Average-JMX_Time* possui o comportamento mais estável do que o comportamento apresentado pelo índice *Client Response-Time*, principalmente em alta carga de trabalho.

O índice *Average-JMX_Time* (Figura 59 até Figura 63) teve uma representação de carga de trabalho mais estável em nossos experimentos e, como esperado, teve uma relação linear com o tempo de execução dos serviços requisitados. Estes resultados demonstram um comportamento estável, que é independente de recursos da plataforma e da demanda.

3.7 Conclusões

Índices de carga para quantificar a carga de trabalho computacional são, geralmente, as métricas mais utilizadas para as heurísticas que tentam prever o desempenho em um futuro próximo da carga de trabalho em um passado recente. No entanto, nem sempre é simples usar índices de carga de forma eficiente no contexto de *web services*, porque eles precisam considerar uma série de diferentes fatores, a fim de representar adequadamente a carga de trabalho e prever o desempenho desejado. Em particular, o *hardware* da plataforma, os *softwares* do sistema, a demanda esperada do serviço e os objetivos gerais do sistema são fatores importantes que afetam a seleção dos índices de carga.

Este trabalho apresenta estudos experimentais realizados para avaliar o comportamento de diversos índices de carga, muitas vezes adotados no contexto de *web services*. Este estudo experimental foi baseado na execução de quatro serviços diferentes, cada um gerando diferentes e específicas cargas de trabalho destinadas a concentrar-se em uso de *CPU*, demanda por memória, tráfego de rede e acesso ao banco. Estes serviços foram executados individualmente e também simultaneamente, para gerar uma carga de trabalho heterogênea. Três diferentes critérios foram considerados durante a nossa análise: a

capacidade de representar a carga de trabalho, a relação da carga de trabalho com o desempenho do serviço e a estabilidade global do índice.

Os resultados demonstram que cada um dos índices analisados foi capaz de representar com precisão a carga de trabalho específica, que foram concebidos para monitorar. Alguns deles também foram capazes de fornecer uma boa representação de diferentes cargas de trabalho (ou seja, uma carga de trabalho diferente do que o índice havia sido projetado para monitorar). Dois exemplos de tais casos são os índices *cpu_idle* e *Ready Processes*, que são projetados para medir cargas de trabalho *cpu_bound*, mas também fornecem uma representação útil de outras demandas computacionais. Esses resultados confirmam que tais índices são especialmente úteis como heurísticas para algoritmos de balanceamento de carga.

No entanto, esses mesmos índices não fornecem uma medida precisa do desempenho global dos *web services* que estão sendo executados na plataforma. De fato, os resultados demonstram que os índices de carga mais utilizados na prática para o balanceamento de carga de *web services* não são capazes de fornecer uma perspectiva precisa de desempenho futuro dos serviços requisitados. Alguns exemplos de tais índices são cpu_idle, *Free Memory*, *Processos Ready* e *Total Amount of Submitted Requests*. Estes índices são altamente influenciados por diversos recursos particulares de *hardware* e a demanda pelo serviço global, e não pode ser aplicada com sucesso por si só, sem considerar esses outros fatores. Embora estes índices possam ser utilizados com sucesso para alguns tipos específicos de carga de trabalho e de *hardware*, qualquer tentativa de utilizá-los em um ambiente global apenas transfere para o utilizador final a responsabilidade de selecionar o índice correto para um serviço em particular. Tal opção não é viável para um sistema prático, uma vez que aumenta a complexidade global e não oferece transparência e portabilidade.

Outro aspecto negativo de tais índices são que eles geralmente não proporcionam a estabilidade necessária para evitar a representação de falsos picos de desempenho. Os valores médios com base em uma amostra maior poderiam ser utilizados para minimizar tais picos. No entanto, este método reduz a precisão do índice e também reduz a capacidade de estimar o desempenho futuro do serviço.

Nossos resultados mostram que muitos dos índices de carga comuns podem detectar corretamente as cargas de trabalho elevadas em um nó, mas que, apesar disso, o desempenho

geral do serviço permanece inalterado. A correta utilização e interpretação dos índices de carga deve permitir a redução do número de nós estabelecidos para uma carga de trabalho dado o objetivo da plataforma, o que pode resultar tanto em redução do consumo de energia como também no custo global de operação. A falta de precisão na previsão do desempenho do serviço pode levar a decisões erradas em relação ao uso de *web services*, comprometendo a qualidade oferecida e também gerar altos custos desnecessários para a plataforma.

Índices relacionados a *web services*, principalmente aqueles relacionados ao tempo de execução, foram capazes de encapsular com sucesso, diferentes aspectos do *hardware* e demanda de serviços. Por outro lado, os índices relacionados especificamente com o *hardware* não foram capazes de ocultar estes aspectos. Neste sentido, o índice *Average-JMX_Time* obteve o melhor resultado norteado pelos três critérios considerados: ser capaz de representar a carga de trabalho atual corretamente, ser capaz de prever com sucesso o desempenho do serviço em um futuro próximo e ter estabilidade necessária para evitar falsos picos de desempenho, elevados ou baixos. Outras características vantajosas do índice *Average-JMX_Time* são: a transparência dos serviços e clientes, a portabilidade entre diferentes plataformas e facilidade de implementação. Estas características são essenciais para um sistema prático realizar o melhor uso possível dos índices de carga de maneira eficaz.

Estes resultados foram aplicados ao projeto *Jerrymouse* (Souza et. al., 2011) para o desenvolvimento de política avançadas de distribuição de requisições em *clusters* de *web services*. O principal objetivo do projeto *Jerrymouse* é a adequação dos diferentes índices de cargas, com políticas de distribuição e uma investigação detalhada do desempenho resultante da integração entre os dois objetivos anteriores.

3.8 Considerações Finais

Nesta seção foi apresentada a avaliação do comportamento de diversos índices de carga no contexto de *web services*. Três diferentes critérios foram considerados durante esta avaliação: a capacidade de representar a carga de trabalho, a relação da carga de trabalho com o desempenho do serviço e a estabilidade global do índice.

Os resultados deste estudo demonstram que o índice *Average-JMX_Time* é capaz de encapsular com sucesso, diferentes aspectos do *hardware* e demanda de serviços. Enquanto

que os demais índices analisados, em um ambiente global apenas transferem para o utilizador final a responsabilidade de selecionar o índice correto para um serviço em particular. Esta falta de transparência ao se utilizar os índices de carga não é viável para um sistema prático, uma vez que aumenta a complexidade global e não oferece portabilidade.

No próximo capítulo é abordado o desenvolvimento das políticas de distribuição em *cluster* de *web services*, onde são usados índices de carga.

4 Políticas de Distribuição de Requisições

4.1 Considerações Iniciais

Existem poucas políticas desenvolvidas para a distribuição de requisições em *cluster* de *web services* na prática (Mod_cluster, 2011) (The Apache Software Foundation, 2010) . As poucas políticas existentes neste contexto apresentam pouca flexibilidade frente à variação dinâmica na demanda dos serviços requisitados. Dado este cenário, uma nova política de distribuição de requisições em *clusters* de *web services*, chamada *Performance*, foi proposta com a finalidade de prover uma distribuição dinâmica, flexível e transparente em *cluster* de *web services*. A política *Performance* é descrita na próxima seção. Um protótipo desta política foi implementado para avaliação do seu desempenho. Além da política *Performance*, desenvolveu-se também um módulo adicional ao *Jerrymouse* para prover economia de energia ao *cluster* de *web services*. Este módulo analisa a carga de trabalho da plataforma e decide por ligar ou desligar os nós do *cluster* conforme a demanda apresentada. O Módulo Gerenciador de Energia pode ser executado independentemente das políticas de distribuição aplicadas. A associação do Módulo Gerenciador de Energia com a política *Performance* permite minimizar o consumo de energia gasto com a plataforma, ao mesmo tempo em que fornece alta confiabilidade, sem perda de requisições durante a sua operação.

4.2 Política Performance

O principal objetivo da política *Performance* é oferecer uma distribuição dinâmica de requisições para os nós *back-ends* de um *cluster* de *web services*, de maneira mais flexível e transparente. Além disso, espera-se que a política possua uma execução estável, apresente execução rápida e seja capaz de distribuir todas as requisições de maneira confiável frente aos objetivos da plataforma. A execução exclusiva desta política mantém todos os nós do *cluster* ligados durante todo o tempo de sua operação. Dependendo de como as opções de configuração da política *Performance* são definidas ela pode apresentar diversas formas de execução.

A política *Performance* pode contemplar a heterogeneidade de desempenho da plataforma em que é executada, dependendo da sua configuração. A heterogeneidade é determinada previamente à execução dos serviços e fixada em um arquivo para posterior uso

durante a execução da *Performance*. Durante a distribuição das requisições, a diferença de desempenho dos nós é associada à carga computacional existente no momento da submissão (considerando um passado recente), permitindo, desta forma, que o índice de carga seja ponderado pela heterogeneidade.

A política *Performance* diferencia-se de uma *Round-Robin* devido à dinamicidade que trata a sua lista de elementos (nós). Estes são relacionados ao desempenho apresentado pelos nós e a ordem para a distribuição das requisições é estabelecida de maneira que os nós com melhor desempenho sejam os primeiros receptores. Ao contrário de uma política *Round-Robin* a qual não apresenta tais características. A Figura 8 apresenta o pseudocódigo da política *Performance* para auxiliar na compreensão da sua descrição no texto a seguir.

O Módulo de Informação cria a lista de nós disponíveis no *cluster* e atribui uma quantidade de requisições para *warm-up* (aquecimento do índice de carga). O módulo de análise irá calcular a faixa de tolerância, responsável por estabelecer o valor limite para o índice de carga que definirá se o desempenho representado pelo índice considera o nó, em questão, apto a receber requisições. Caso a faixa de tolerância seja definido como 0% (sem faixa de tolerância) a política *Performance* irá trabalhar com rajadas de requisições para o nó com melhor desempenho representado pelo índice de carga, esse nó é eleito cada vez que o Módulo de Análise é executado. Ainda neste módulo a lista é ordenada de maneira decrescente do desempenho representado pelo índice de carga. A quantidade de requisições que nó irá receber é estabelecida neste módulo com base se o índice é ponderado ou não. Caso o nó seja ponderado, ele calcula quantas vezes o desempenho do nó é melhor do que o pior desempenho considerado apto a receber requisições. São considerados apenas os valores inteiros desse cálculo, por exemplo, se um nó apresentar um desempenho 2,5 vezes melhor do que outro nó, o nó mais rápido irá receber 2 requisições enquanto o nó com menor desempenho irá receber 1 requisição. O valor 0,5 (meio), desempenho melhor é desconsiderado. No Módulo de Seleção, a política *Performance* selecionará o nó destino para requisição em questão com base nas informações da lista. A seguir, o funcionamento da política *Performance* será descrito com maiores detalhes.

```
1  Inicia a Política Performance
2      Coleta Horário da execução da política
3      SE primeira execução OU Tempo da última atualização da lista de nós > TEMPO_ATUALIZA_LISTA
4          Coleta todos os endereços IPs dos nós do Cluster
5          PARA cada elemento da lista
6              Armazena cada endereço IP em um elemento da Lista
7              Insere a quantidade de requisições para warm-up
8          FIM-PARA
9      FIM-SE
10     SE primeira execução OU Tempo da última atualização informações da lista >
       TEMPO_ATUALIZA_INFORMAÇÕES
11         SE quantidade de requisições para warm-up > 0
12             PARA cada elemento da lista FAÇA
13                 Define Elemento como "Ativo"
14             FIM-PARA
15         SENÃO SE quantidade de requisições para warm-up == 0
16             Coleta/atualiza informações dos Nós
17             PARA cada elemento da lista FAÇA
18                 Armazena seu respectivo valor do índice de carga
19             FIM-PARA
20             Calcula a "Faixa de Tolerância"
21             PARA cada elemento da lista FAÇA
22                 SE valor do índice < LIMITE_MÁXIMO E valor do índice > LIMITE_MÍNIMO
23                     Define Elemento como "Ativo"
24                 SENÃO
25                     Define Elemento como "Inativo"
26                 FIM-SE
27             FIM-PARA
28             SE Índice Ponderado == FALSO
29                 PARA cada elemento da Lista FAÇA
30                     Insere a quantidade de requisições que cada Nó irá receber
31                 FIM-PARA
32             SENÃO SE Índice Ponderado == VERDADE
33                 PARA cada elemento da Lista FAÇA
34                     Calcula quantidade de requisições que cada Nó irá receber
35                     Insere a quantidade de requisições que cada Nó irá receber
36                 FIM-PARA
37             FIM-SE
38             Ordena lista em ordem decrescente do valor do índice de carga
39         FIM-SE
40     FIM-SE
41     SE não tiver elementos "ativos" na lista com requisições para receber
42         SE Índice Ponderado == FALSO
43             PARA cada elemento da Lista FAÇA
44                 Insere a quantidade de requisições que cada Nó irá receber
45             FIM-PARA
46         SENÃO SE Índice Ponderado == VERDADE
47             PARA cada elemento da Lista FAÇA
48                 Calcula quantidade de requisições que cada Nó irá receber
49                 Insere a quantidade de requisições que cada Nó irá receber
50             FIM-PARA
51         FIM-SE
52     SENÃO SE ainda tiver elementos "ativos" na lista com requisições para receber
53         SE quantidade de requisições para warm-up > 0
54             Seleciona o primeiro elemento que tiver maior quantidade de requisições para warm-up
55         SENÃO SE quantidade de requisições para warm-up == 0
56             Seleciona o primeiro elemento que ainda tiver requisições a serem recebidas
57         FIM-SE
58     FIM-SE
59 Fim da Política Performance
```

Figura 8: Pseudocódigo da política *Performance*

A política *Performance* é organizada em três módulos, como já citado: Módulo de Informação (da linha 03 até a 09), Módulo de Análise (da linha 10 até a 40) e Módulo de Seleção (da linha 41 até a 58). No Módulo de Informação, a política coleta todos os *IPs* dos

nós que constituem o *cluster*. No Módulo de Análise a política coleta os índices de carga e realiza a análise de possíveis nós de destino para a requisição em questão. No Módulo de Seleção, o nó de destino é selecionado e a requisição é encaminhada a ele. Esta estrutura fornece modularização e organização ao código do protótipo implementado.

O Módulo de Informação encarrega-se de criar uma lista onde cada elemento possui dados dos nós pertencentes à plataforma (linha 4 até a 8). Esta lista é alocada dinamicamente na memória do nó *front-end* e o endereço do primeiro elemento da lista é armazenado em um ponteiro genérico disponibilizado pelo *Jerrymouse*. Desta forma, esta informação poderá ser recuperada na próxima execução da política *Performance*. Inicialmente, a lista contém apenas o endereço *IP* de cada nó do *cluster* como identificador de cada elemento na lista e a quantidade de requisições que cada elemento receberá no processo de *warm-up* (este valor é configurável - ver Figura 9, estado da lista na linha 9). O processo de *warm-up* garante que todos os nós do *cluster* receberão uma certa quantidade mínima de requisições (linha 7), a fim de estabilizar o valor do índice de carga utilizado.

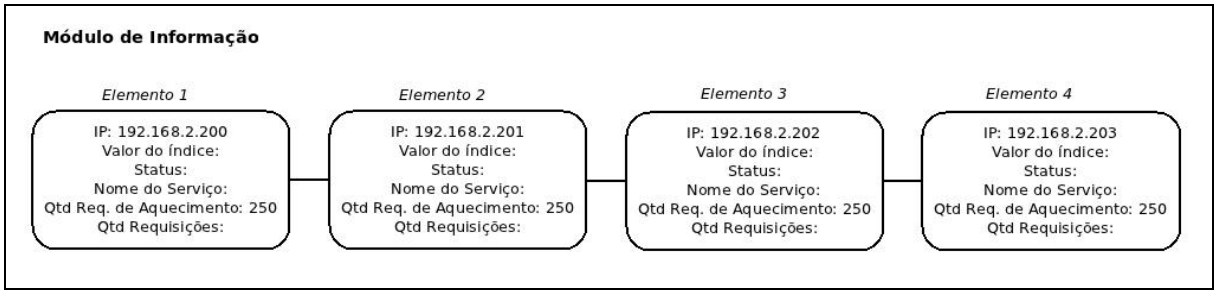

Figura 9: Representação da lista durante o Módulo de Informação

Na próxima fase de execução da política (Módulo de Análise, demonstrada na Figura 10), se o processo de *warm-up* já tiver concluído, os valores dos índices de carga de cada nó do *cluster* são coletados (linha 15 até a 39). O índice de carga é obtido pela política através de uma função disponibilizada pelo *Jerrymouse* (linha 16), este já em comunicação com o sistema de monitoração. A maneira que o índice de carga deve ser utilizado pela política pode ser configurada de acordo com as características do índice. A grandeza do índice (quanto maior melhor ou quanto menor melhor), o nome do índice de carga e se o valor do índice deve ser ponderado, são opções que podem ser configuradas. Após o processo de *warm-up*, o nó apenas receberá novas requisições se for definido como “ativo”.

Um nó é considerado “ativo” (linha 21 até a 27) para receber requisições de um *web service* em particular, se o valor do seu índice de carga estiver dentro de uma “Faixa de Tolerância” (explicada a seguir, linha 20), esta também configurável. O nó pode ser considerado “inativo” para requisições de um *web service* enquanto é considerado “ativo” para receber requisições de outro *web service*.

A “Faixa de Tolerância” é calculada (na linha 20) usando um valor (em percentual) configurável e o melhor valor do índice de carga obtido no *cluster*, de acordo com as configurações definidas na opção de grandeza do índice, resultando assim em um valor máximo que o índice de carga de cada nó deve ter para ser considerado “ativo”. É definida apenas a base de cálculo (valor percentual) para o valor máximo, já que o valor mínimo será o menor valor do índice de carga obtido no *cluster*. Dessa forma, se o índice de carga tiver seu valor entre o melhor valor do índice de carga coletado (menor valor obtido de algum nó do *cluster*) e o valor máximo resultante do cálculo da “Faixa de Tolerância” (melhor valor obtido mais a soma do seu percentual definido), o nó é considerado “ativo” para receber requisições do *web service* requisitado. Os limites da “Faixa de Tolerância” são dinâmicos, sendo calculados sempre que os valores dos índices de carga forem atualizados na lista de elementos da política. Após a definição dos nós “ativos”, a quantidade de requisições que cada nó deve receber é estabelecida, considerando a ponderação ou não do índice de carga (descrito a seguir e apresentado na Figura 8 nas linhas 28 até a 37), e armazenada no elemento da lista que corresponde ao nó. A “Faixa de Tolerância” é uma informação sobre o estado dos nós no *cluster* que a política *Performance* armazena em uma estrutura de dados paralela à lista de elementos do *cluster*. O conceito de “nós ativos” proporciona que um nó considerado “lento” para um tipo específico de *web service* não receba requisições para este serviço e, assim não eleve o desempenho médio da plataforma.

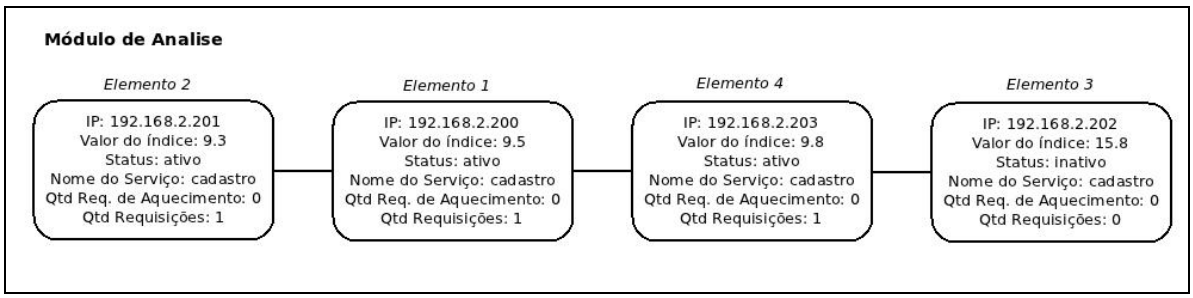

Figura 10: Representação da lista durante o Módulo de Análise

Caso a opção de “Ponderação do índice” esteja habilitada, a política calcula a quantidade de requisições que cada nó irá receber (linha 34 e 35) considerando que o nó “ativo” com menor desempenho irá receber uma requisição. Assim de acordo com o desempenho dos demais nós a quantidade de requisições que cada nó irá receber aumenta. Por exemplo, se um nó A possui o valor do seu índice de carga três vezes melhor do que o valor do índice de carga do nó B, ambos “ativos”, o nó A receberá três requisições enquanto o nó B receberá apenas uma requisição. Se a opção de “Ponderação do índice” estiver desabilitada, cada nó do *cluster* definido como “ativo” receberá a mesma quantidade de requisições (linha 30) cada vez que a lista for percorrida.

Independente da configuração da opção “Ponderação do índice” a lista é organizada em ordem decrescente de desempenho dos nós (linha 38), de maneira que os elementos da lista que correspondem os nós com melhor desempenho ocupem as primeiras posições da lista. Com isso, a política irá selecionar os nós com melhor desempenho para as primeiras requisições (Módulo de Seleção, ver Figura 11 – apresentada da linha 41 até a 58). A política *Performance* seleciona um nó “ativo” como destino de uma requisição em questão após seu antecessor ter recebido a quantidade de requisições destinadas a ele (da linha 53 até a 57). Quando o último elemento “ativo” da lista receber a última requisição destinada a ele, a política *Performance* atualiza a quantidade de requisições que cada elemento da lista deverá receber e retorna ao início da lista (linha 41 até a 51). A atualização da quantidade de requisições é realizada da mesma forma que descrito no Módulo de Análise, considera-se a ponderação (linha 47 até a 50) ou não (linha 44) do índice de carga e assim atribui a quantidade de requisições que cada nó deverá receber.

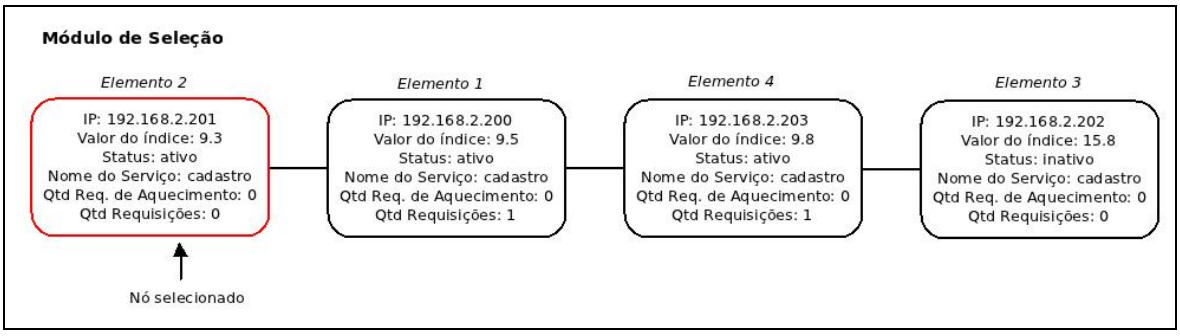

Figura 11: Representação da lista após o Módulo de Seleção selecionar destino para a requisição

O Módulo de Análise é executado a cada “X” unidades de tempo (em milissegundos), enquanto que o Módulo de Informação é executado a cada “Y” unidades de tempo (em

milissegundos), ambos os tempos são configuráveis. Caso o Módulo de Informação seja executado, o Módulo de Análise deverá ser executado para manter a consistência das informações contidas em cada elemento. Ainda no caso do Módulo de Informação for executado, todos os elementos recebem novamente a quantidade de requisições para a estabilização do índice.

Considerando que a política *Performance* permite definir qual o índice de carga deve ser utilizado, existe a opção de concatenar o nome do *web service* requisitado ao nome do índice de carga de cada elemento da lista. Esta opção é importante caso existam diversos índices de carga com nome igual no mesmo nó. Por exemplo, se o índice *Average-JMX_Time*, apresentado no capítulo 3 Índice de Carga, for utilizado pela política *Performance*, poderá existir diversos índices com o nome *Average-JMX_Time* para no mesmo nó, um para cada *web service*. Concatenando o nome do *web service* com o nome do índice de carga, o índice de carga se torna específico ao *web service*. Um ponto importante é que o índice de carga deve ser publicado pelo sistema de monitoração com a mesma estrutura em seu nome, porque senão apenas um índice de carga será publicado com o valor do último índice de carga informado ao sistema de monitoração.

A política *Performance* suporta a interação com o Módulo Gerenciador de Energia. Esta interação é descrita em detalhes na próxima seção.

4.3 Módulo Gerenciador de Energia

O Módulo Gerenciador de Energia é um módulo adicional que, acoplado ao *Jerrymouse* (Figura 12), tem como principal objetivo prover economia de energia de maneira estável e com alta confiabilidade para os objetivos da plataforma, durante toda a operação do *cluster*. O Gerenciador de Energia pode ser utilizado de forma independente da política de distribuição aplicada pelo *Jerrymouse* ou ser associada à política *Performance*, descrita na seção anterior. Esta integração (política *Performance* com o Módulo Gerenciador de Energia) é realizada através da disponibilização de um *web service* específico (este usado como destino para requisições identificadas, descritas a seguir) no provedor de *web services* em execução no nó *front-end* e o *Jerrymouse* deve ser configurado para que em todas requisições destinadas a este *web service* seja aplicada a política *Performance*. Com essa integração do

Módulo Gerenciador de Energia com a política *Performance* fornece suporte à confiabilidade proposta, de maneira que não ocorram falhas na distribuição das requisições durante o gerenciamento de energia do *cluster*.

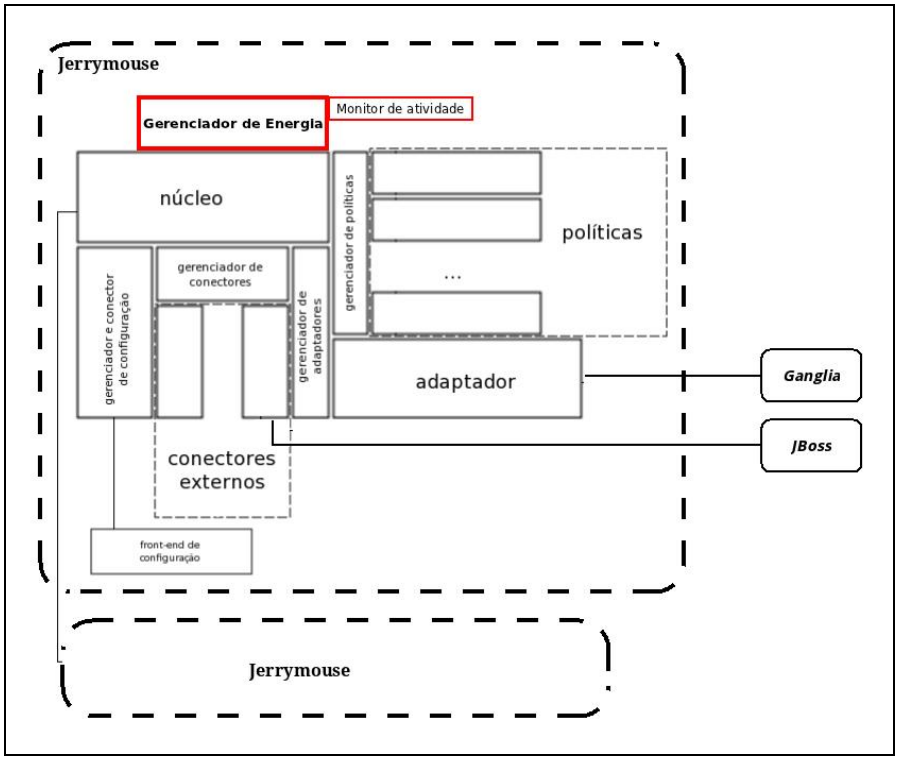

Figura 12: Estrutura do *Jerrymouse* com acréscimo do Módulo Gerenciador de Energia

Outra característica positiva do Módulo Gerenciador de Energia é sua transparência perante as aplicações clientes que realizam requisições aos *web services* disponibilizados pelo *cluster*. Estas aplicações não necessitam de nenhuma instrumentação em seu código ou conhecimento prévio do gerenciamento do *cluster*, apenas enviam suas requisições ao *cluster* o qual se encarrega de atendê-las e enviar suas respectivas respostas.

O Módulo Gerenciador de Energia pode ser configurado através de um arquivo de configuração onde é definido: se o módulo deve ser executado, o tempo (em segundos) entre as monitorações da plataforma, o nome do índice de carga a ser utilizado, o seu valor máximo, o seu valor mínimo (estes valores representam o limite máximo e mínimo da carga de trabalho adequada à plataforma) e a grandeza do índice de carga (quanto maior melhor ou quanto menor melhor). Estas configurações são carregadas no início de cada análise realizada do *cluster*; desta forma, caso ocorram alterações em suas configurações estas serão efetivadas em tempo de execução, inclusive o início da execução do Gerenciador de Energia.

A flexibilidade do Módulo Gerenciador de Energia possibilita uma configuração heterogênea dos nós do *cluster*. Essa configuração heterogênea contempla a formação de dois grupos de nós, representados na Figura 13. Um dos grupos mantém seus integrantes ligados durante toda a operação do *cluster*, enquanto que o segundo grupo é gerenciado pelo Módulo Gerenciador de Energia, o qual decide se seus integrantes devem ser ligados ou desligados. Para realizar essa configuração, basta associar o endereço *IP* e o endereço *MAC* da interface de rede de cada nó que será gerenciado pelo Módulo Gerenciador de Energia à possibilidade deste nó de ser desligado/ligado ou não. Desta forma, o Gerenciador de Energia utiliza o endereço *IP* e o endereço *MAC*[13] (*Media Access Control*) da interface de rede de cada nó gerenciado por este módulo. Para cada um deles é associada informação se o nó pode ser desligado/ligado automaticamente pelo módulo e se o nó está atualmente ligado ou desligado.

O Módulo Gerenciador de Energia realiza monitorações da carga de trabalho do *cluster* com intervalos de tempos definidos em seu arquivo de configuração. Esta monitoração coleta o valor do índice de carga de cada nó que constitui o *cluster* (gerenciados ou não, desta forma é possível calcular a carga de trabalho em imposta a toda plataforma) e calcula uma média aritmética da carga de trabalho no *cluster*. O valor resultante é amortizado utilizando a Média Móvel Exponencial (MME) com uma janela das últimas cinco amostragens, para evitar picos indesejáveis de desempenho. O Gerenciador de Energia analisa este valor final como índice de carga geral do *cluster*. Esta análise é realizada considerando os níveis definidos no arquivo de configuração e pode resultar em três estados de trabalho do *cluster*: carga de trabalho adequada, plataforma sobrecarregada e plataforma ociosa.

O estado "carga de trabalho adequada" representa que o nível da carga de trabalho está adequada para o *cluster*. Este estado de trabalho não resulta em nenhuma ação do Módulo Gerenciador de Energia.

O estado "plataforma sobrecarregada" representa que o nível da carga de trabalho está superior ao valor máximo definido no arquivo de configuração. Este estado de trabalho faz com que o Módulo Gerenciador de Energia inicie a atividade de *wake-up* em um nó desligado para auxiliar o *cluster* no atendimento das requisições. A partir de um arquivo de auxilio de gerenciamento (com informações dos nós do *cluster* ainda desligados) um nó é selecionado para integrar o *cluster*. Utilizando a tecnologia *Wake-On-Lan* (*WOL*) é enviado um pacote mágico (*magic packet*) (AMD, 1995) contendo o endereço *MAC* da interface de rede do nó

[13] O endereço *MAC* é o endereço físico de 48 *bits* da interface de rede.

que entrará em atividade. O pacote mágico é enviado para o endereço de *broadcast* da rede, desta forma as interfaces de rede de todos os nós que constituem o *cluster* (ligados ou não) irão receber este pacote, mas apenas o nó que possuir o endereço *MAC* da sua interface de rede igual ao do pacote mágico será ligado. Assim que o nó em questão entrar em atividade, o Gerenciador de Energia retoma suas monitorações e analisa para decidir se o nível da carga de trabalho e tornou adequada ou se é necessário acrescentar mais um nó ao *cluster*. Esta atividade é repetida até que o nível da carga de trabalho seja considerado adequado ao *cluster* ou não haja mais nós para serem ligados.

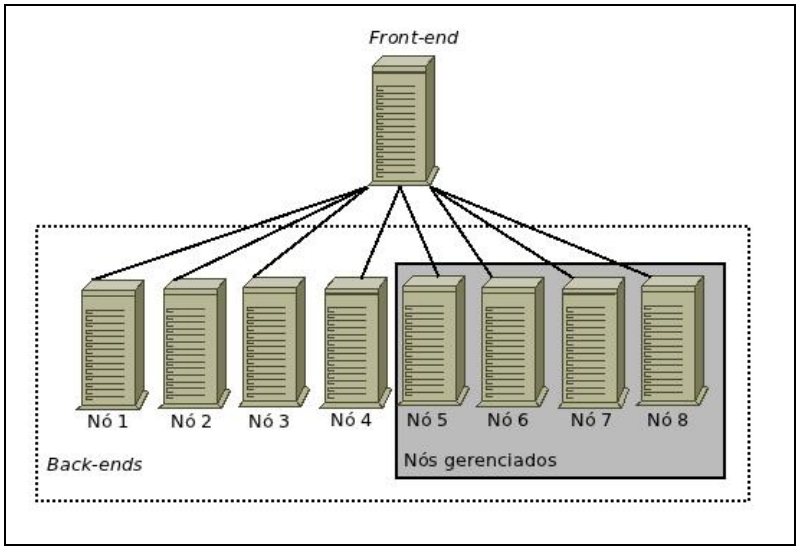

Figura 13: Configuração heterogênea do Gerenciador de Energia

O estado "plataforma ociosa" indica que o nível de carga de trabalho está inferior ao valor mínimo definido no arquivo de configuração. Este estado de trabalho faz com que o Módulo Gerenciador de Energia inicie a atividade de *shutdown* no último nó ligado. Essa ação busca economizar o consumo de energia do *cluster* e manter a sua carga de trabalho adequada aos objetivos da plataforma. Esta atividade conta com um *web service* que recebe uma requisição enviada pelo Gerenciador de Energia e executa, como um comando no sistema operacional, uma *string* passada como argumento na requisição. O Gerenciador de Energia seleciona o último nó acrescentado ao *cluster* para ser desligado e envia uma requisição com o comando para finalizar a execução do *Ganglia* deste nó. Após receber a resposta desta requisição, o Gerenciador de Energia obtém através do arquivo de configuração do *Ganglia* o tempo dos intervalos entre as atualizações das informações propagadas pelo

Ganglia e aguarda no mínimo um intervalo igual para prosseguir com a atividade de *shutdown*. Um valor de acréscimo a este intervalo pode ser configurado, dependendo do objetivo da plataforma. Isso garante que as informações sobre o nó em atividade de *shutdown* não permanecerão disponíveis no *Ganglia*. Depois desse intervalo, no caso de integração com a política *Performance*, o Gerenciador de Energia se comunica com a política *Performance* através de uma requisição identificada. Essa requisição informa à política que existe algum nó em processo de *shutdown*. A política, por sua vez, consulta um arquivo temporário que contém em seu interior o endereço *IP* do nó em questão e o retira da sua lista de possíveis nós receptores. Dessa forma, nenhuma nova requisição será enviada ao nó em processo de *shutdown*, no espaço de tempo em que as informações sobre o nó não estejam mais disponível no *Ganglia* e a lista de elementos da política não tenha sido reconstruída. Em seguida, o Gerenciador de Energia ativa um monitor de atividade para o *JBoss* do nó em processo de *shutdown*. Esse monitor se conecta remotamente ao *JMX* e permanece em monitoramento até que o nó não apresente nenhuma atividade de atendimento de requisições. Assim que o monitor identificar que o nó já terminou o atendimento de todas as requisições que lhe foram destinadas, o monitor envia uma requisição com o comando de *shutdown*. Após a conclusão do desligamento do nó, o Gerenciador de Energia retoma a monitoração para analisar se a carga de trabalho no *cluster* está adequada. O processo de *shutdown* pode se repetir até que a carga de trabalho seja analisada como adequada ou não haja mais nós para serem desligados, permanecendo ligado apenas o *Front-end*.

Caso a política utilizada para distribuição das requisições não seja a política *Performance*, esta deve tomar alguns cuidados a mais durante o processo de *shutdown* realizado pelo Gerenciador de Energia. Basicamente, a política deve atualizar a sua lista de nós sempre que um nó for desligado ou ligado. Essa atualização já é feita periodicamente pelo *Ganglia*, porém, as atividades de *shutdown* e de *wakeup* deveriam forçar novas atualizações.

4.4 Considerações Finais

Foram detalhados neste capítulo a política *Performance* e o Módulo Gerenciador de Energia. Assim como descrito, a política *Performance* tem como objetivo fornecer uma distribuição dinâmica, flexível e transparente das requisições em *cluster* de *web services*. A política *Performance* também permite aumentar o desempenho na distribuição das requisições nos nós do *cluster*, refletindo assim em um menor tempo de resposta para as aplicações

clientes. O Módulo Gerenciador de Energia tem como objetivo prover economia no consumo de energia do *cluster* mantendo a carga de trabalho do *cluster* adequada, conforme definido em seu arquivo de configuração.

No próximo capítulo são apresentados os estudos experimentais realizados com a política *Performance* e com o Módulo de Gerenciamento de Energia. Estes estudos buscam avaliar o desempenho da distribuição realizada pelo *Jerrymouse* empregando a política *Performance* frente à distribuição realizada pelo balanceador de carga *Mod_cluster* e também a economia no consumo de energia alcançada empregando o Módulo Gerenciador de Energia comparado a um mesmo ambiente sem o gerenciamento de energia.

5 Estudos Experimentais com as Políticas de Distribuição

5.1 Considerações Iniciais

Este capítulo apresenta o planejamento, os resultados e a análise dos resultados obtidos com estudos experimentais realizados com as políticas de distribuição desenvolvidos neste mestrado, confrontados com políticas de distribuição amplamente aplicados em *clusters* de *web services*. Os resultados obtidos com a política *Performance* apresentam desempenho superior na distribuição de requisições frente à política padrão do *Mod_cluster*. Por outro lado, o Módulo Gerenciador de Energia alcançou seu objetivo de economizar o consumo de energia elétrica pela plataforma mantendo alta confiabilidade no atendimento das requisições.

5.2 Planejamento dos Experimentos

Estes estudos experimentais consideraram a execução dos *web services* em duas plataformas distintas. Uma plataforma (**A**) é formada por três nós todos com processadores *Intel Core 2 Quad 2.66 GHZ* e *6MB cache*, *4GB DDR2 de RAM* e um *Hard Disk Sata-2* de *500GB/7200RPM* (Figura 14). A base de dados usada pelos *web services* é *MySQL*, versão 5.1. A rede usada para interligar os computadores é uma *Ethernet* de *100 Megabits* com um *switch TP-LINK TL-SL3428*. O sistema operacional é o *Ubuntu 9.04*, *kernel 2.6.31-22*. Os compiladores usados são *gcc 4.3* e *Java 1.6.0_20*. O provedor de *web services* é o *JBoss 6.0.0-Final*. O *Ganglia* 3.1.2 e o *JMX 6.0.0-Final* com suas configurações padrões monitoraram a plataforma.

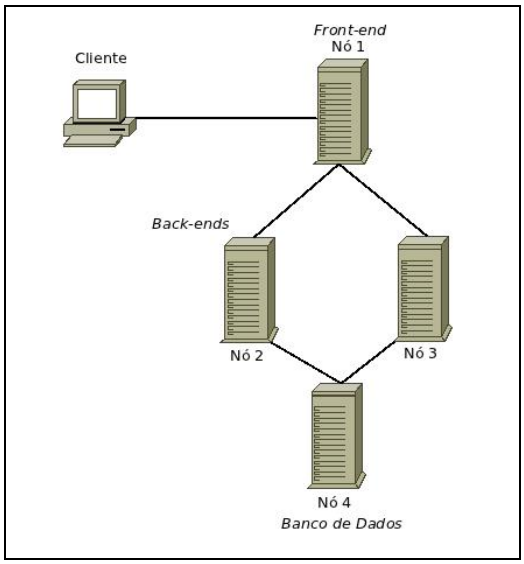

Figura 14: Representação da ligação lógica entre os elementos do *cluster* com 3 nós, sem explicitar os dispositivos específicos de rede.

A segunda plataforma (**B**) é formada por nove nós, todos com processadores *Intel Core 2 Quad Q6600 2.40 GHZ* e *8MB cache*, *2GB DDR2 de RAM* e um *Hard Disk Sata-2* de *140GB/7200RPM*. A rede usada para interligar os computadores é uma *Ethernet* de 1 *Gigabit* com um *switch TP-LINK TL-SL3428*. O sistema operacional é o *Ubuntu 10.04.3 LTS, Kernel 2.6.32-34-server*. Os compiladores usados são *gcc 4.3* e *Java 1.6.0_20*. O provedor de *web services* é o *JBoss 6.0.0-Final*. O *Ganglia* versão 3.1.2 e o *JMX 6.0.0-Final* monitoraram a plataforma. O *Ganglia* foi configurado para atualizar seus dados a cada segundo e, caso não receba uma nova atualização, manter as últimas informações recebidas por até 3 segundos. Se o *Ganglia* não receber novas atualizações até estes 3 segundos, as informações referentes ao nó que deixou de propagar suas informações são retiradas da lista de informações. Em ambas as plataformas, o banco de dados utilizado é executado em um nó à parte . Embora possua a mesma configuração, este nó não é contabilizado na quantidade total dos nós nas plataformas descritas.

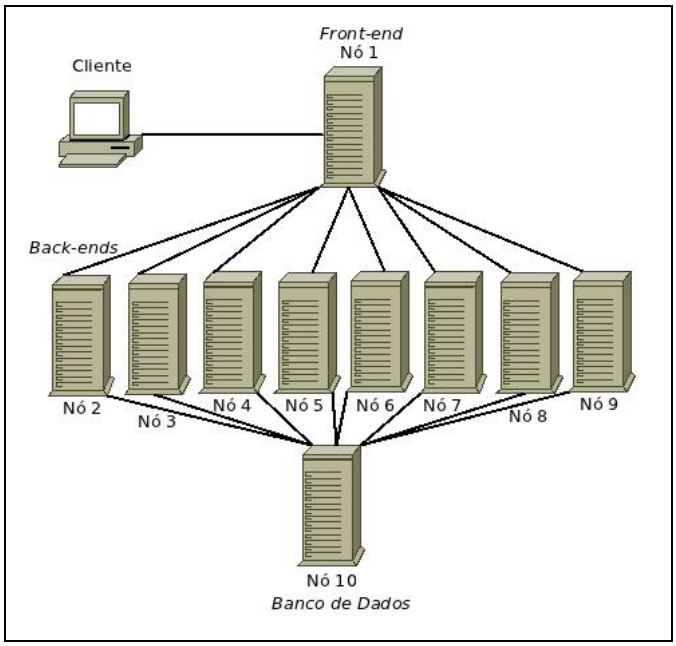

Figura 15: Representação da ligação lógica entre os elementos do cluster com 9 nós, sem explicitar os dispositivos específicos de rede.

Embora a plataforma **A** seja relativamente modesta, a mesma atende o objetivo deste estudo experimental de analisar e comparar o comportamento da distribuição de requisições aos nós *back-ends* do *cluster*. Os nós desta plataforma possuem Sistema Operacional, *hardwares* e provedores de serviços comuns em ambientes não dedicados ao atendimento de requisições aos *web services*. Esta configuração é comum em ambientes de *web services* menores. Por outro lado, a plataforma B possui Sistema Operacional, *hardwares* e provedores de serviços configurados como servidores de *web services* e, portanto, são destinados especialmente a esta finalidade. Os estudos experimentais foram executados em ambas as plataformas com objetivo de analisar a estabilidade das propostas realizadas neste projeto de mestrado para o atendimento das requisições, de maneira ortogonal a tais componentes.

Ambas as plataformas são capazes de suportar uma demanda controlada gerada a partir da execução de aplicações clientes (executadas em um computador à parte, este descrito na Figura 14 e Figura 15). Os aplicativos clientes enviam as requisições para o nó provedor através de mensagens *HTTP* com encapsulamento *SOAP*. O nó provedor encaminha a mensagem *HTTP* ao seu provedor de *web services* local (*JBoss*), que se comunica com o *Jerrymouse*, o qual escolhe o nó que deverá atender a requisição em questão, encaminhando a requisição a este nó. Após o atendimento da requisição, a resposta é encaminhada ao nó *front-end*, que por sua vez a enviará para a aplicação cliente que realizou a requisição.

Estes estudos experimentais consideram a execução de um *web service* responsável por realizar inserção de diversas informações em um banco de dados. Este *web service* foi escolhido por se assemelhar a diversos serviços utilizados por servidores de *web services*, os quais acessam bases de dados fazendo o armazenamento e a recuperação de informações. Outra característica importante deste serviço é seu baixo tempo de execução, em torno de 8 milissegundos.

Os estudos experimentais não consideram a fase de descoberta dos *web services*, uma vez que esta fase não afeta o comportamento da distribuição das requisições no nó *front-end*. Os estudos experimentais destinados à Política *Performance*, foram realizados nas plataformas **A** e **B**. Cada instância do aplicativo cliente realiza exatamente 10.000 iterações, cada requisição é realizada exatamente logo após a outra, sem aguardar a resposta das predecessoras. Foram executadas simultaneamente uma, duas, quatro, oito, dezesseis, trinta e duas e sessenta quatro *threads* concorrentes da aplicação cliente. A comparação entre a distribuição das requisições foi realizada pelos resultados obtidos com o provedor de *web service JBoss* 6.0-Final associado ao módulo *mod_cluster* e com o *JBoss* 6.0-Final associado ao *Jerrymouse*. O *JBoss* associado ao *mod_cluster* aplica a *DynamicLoadBalanceFactorProvider* (classe responsável por identificar o nó menos ocupado) considerando a composição dos índices de carga que representam o percentual de *CPU* utilizada e a quantidade de memória utilizada para identificar o nó mais adequado para receber a requisição e assim realizar a distribuição das requisições. O nó mais adequado é sempre o nó menos ocupado, de acordo com a análise aplicada. Na configuração utilizando o *JBoss* 6.0-Final associado ao *Jerrymouse*, aplicou-se a política *Performance* para a distribuição transparente e flexível das requisições considerando o índice de carga proposto *Average-JMX_Time*. Esta estratégia permitiu a análise da distribuição de requisições em *cluster* de *web services* sob cargas de trabalho específicas e homogêneas.

O estudo experimental destinado ao Módulo Gerenciador de Energia foi realizado na plataforma **B**. Este estudo foi realizado apenas em uma das arquiteturas, pois necessita de vários nós disponíveis para que os mesmos sejam ligados e desligados. Cada instância do aplicativo cliente realiza exatamente 40.000 iterações, realizada da mesma maneira que para os estudos experimentais da política *Performance*. Foram executadas duas, quatro, oito, dezesseis, oito, quatro, duas *threads* concorrentes da aplicação cliente. Esta estratégia permitiu a análise da distribuição de requisições em *cluster* de *web services* e a economia no

consumo de energia sob cargas de trabalho específicas e homogênea. Cada requisição demora cerca de 8 milissegundos em ambos os experimentos, quando o serviço é executado sem concorrência de outros serviços.

A análise considerou os resultados obtidos com a distribuição das requisições realizada pelo *JBoss* 6.0-Final associado ao *Jerrymouse* aplicando a política *Performance* e a distribuição das requisições realizada pelo *JBoss* 6.0-Final associado ao *Jerrymouse* aplicando a política *Performance* integrado ao Módulo Gerenciador de Energia. A configuração utilizando o *JBoss* 6.0-Final associado ao *Jerrymouse* aplicando a política *Performance* manteve todos os nós dos *cluster* ligados durante todo o período de atividade do *cluster*. A configuração utilizando o *JBoss* 6.0-Final associado ao *Jerrymouse* aplicando a política *Performance* e integrado ao Módulo Gerenciador de Energia inicia suas atividades apenas com o nó *front-end* ligado e somente liga os nós *Back-ends* caso seja necessário. O índice de carga proposto *Average-JMX_Time* foi utilizado em ambas às configurações.

O *Ganglia* (Massie et al., 2004) e o *JMX* (Mahmoud, 2011) foram utilizados para coletar os índices de carga nos nós do *cluster*, enquanto os serviços estavam em execução. Cada *gmond* em execução nos nós do *cluster* realizava amostragens a cada 1s e publicava as informações para os outros nós, caso a comunicação de algum *gmond* demora-se por mais de 3 segundos o *Ganglia* retirava suas informações das mensagens propagadas. Um *script* em cada nó do *cluster* calculava o índice de carga *Average-JMX_Time* e o inseria no *Ganglia* através da ferramenta *gmetric*. O desvio padrão (DP) e o intervalo de confiança de 99% foram calculados para todos os resultados. As variáveis de resposta analisadas são: o tempo de resposta da requisição, quantidade de requisições atendidas por segundo, quantidade de requisições completadas e tempo de execução dos clientes.

Nestes estudos experimentais a Média Móvel Exponencial (MME) - ou *Exponential Moving Average* (*EMA*) - foi utilizada para todos os resultados, Equação 2. A janela utilizada pela Equação 2 é de cinco amostras, escolhido empiricamente.

5.3 Análise de Resultados

Os resultados descritos nesta seção estão organizados em função da plataforma utilizada e podem ser visualizados da Figura 16 até a Figura 28.

A política *Performance* aplicada pelo *Jerrymouse* para a distribuição das requisições na Plataforma A apresenta desempenho superior quando comparada à distribuição realizada pelo *Mod_cluster*, aplicando sua política padrão e a classe *DynamicLoadBalanceFactorProvider*. A política *Performance* utilizou o índice de carga *Average-JMX_Time*, proposto neste mestrado, enquanto que o *Mod_cluster* realizou uma composição de índices que representavam a carga de trabalho na *CPU* e a quantidade de memória utilizada. Esta composição de índices está detalhada na subseção *2.4.1 O Modulo Mod_cluster*.

Apesar de a política *Performance* apresentar melhor desempenho em todos os níveis de concorrência dos clientes, demonstrado na Figura 16, a partir do nível de 8 clientes concorrentes a diferença entre os tempos de resposta começa a aumentar significativamente tornando-se visível o melhor desempenho da política *Performance* na distribuição das requisições sob alta demanda. A política padrão de o *Mod_cluster* distribuir as requisições baseada em rajadas, um nó é eleito para atender todas as requisições até que outro nó seja eleito como destino, essa distribuição não apresenta distribuição eficiente para que todos os recursos computacionais da plataforma sejam utilizados. Isso ocorre porque sob alta carga de trabalho o nó eleito é saturado enquanto os demais nós do *cluster* estão ociosos.

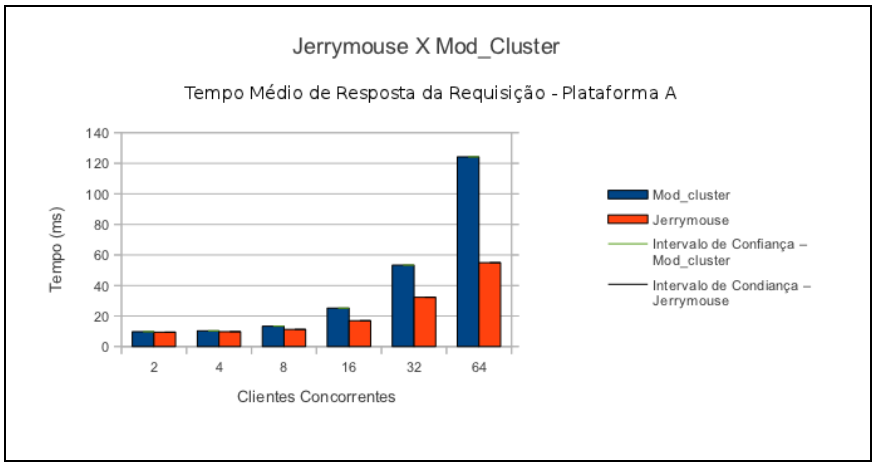

Figura 16: Tempo Médio de Resposta entre as distribuições realizadas pela política *Performance* e a política padrão do *Mod_cluster*

Os tempos médios de resposta, representados na Figura 16, são calculados utilizando os tempos das requisições completadas com sucesso. A comparação entre os tempos médios obtidos com as duas políticas de distribuição de requisições, Figura 17, demonstra que a distribuição realizada com a política padrão do *Mod_cluster* resulta em tempos de resposta

superiores a cinqüenta porcento (50%) em relação aos tempos de resposta obtidos com a distribuição realizada pela política *Performance*. Esta comparação é realizada subtraindo o tempo menor do tempo maior e dividindo o resultado pelo tempo maior.

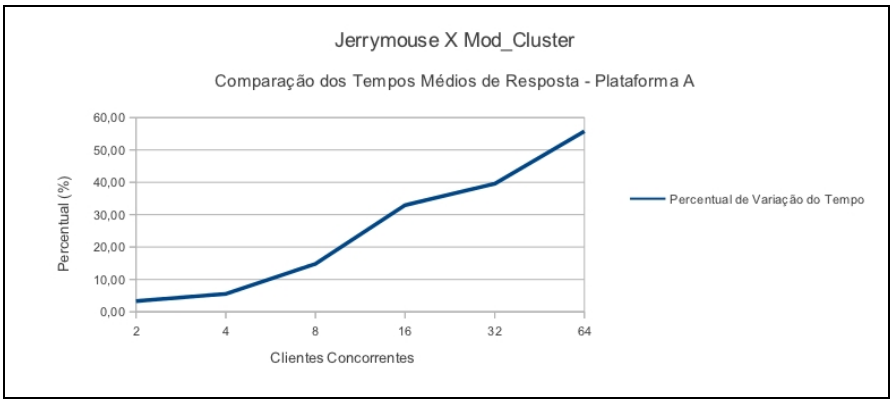

Figura 17: Comparação entre os tempos médios de resposta obtidos com as distribuições das duas políticas utilizadas.

A distribuição das requisições realizada pela política *Performance* apresenta maior quantidade de requisições atendidas por segundo do que a política padrão do *Mod_cluster*, ver Figura 18. A *Performance* possui maior estabilidade na sua execução, mesmo com o aumento na quantidade de requisições atendidas por segundo conforme a quantidade de clientes concorrentes aumenta, conseqüentemente, a quantidade de requisições realizadas à plataforma também aumenta. No entanto, a política padrão do *Mod_cluster* apresenta sobrecarga e instabilidade na sua execução a partir de 16 clientes concorrentes, demonstrando menor quantidade de requisições atendidas por segundo conforme a quantidade de clientes concorrentes aumenta. A redução na quantidade de requisições atendidas segue a sobrecarga apresentada no aumento no tempo de resposta, quanto maior o tempo de resposta menor a quantidade de requisições atendidas por segundo.

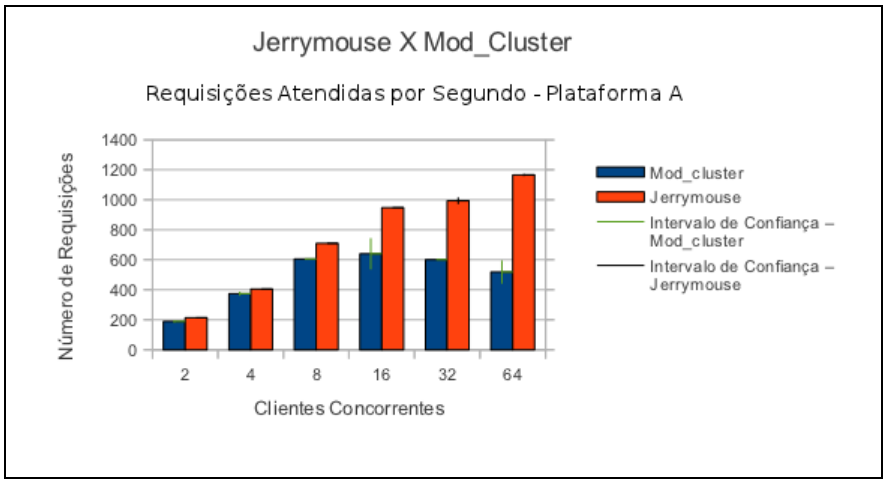

Figura 18: Quantidade de requisições atendidas por segundo

A política *Performance* distribuiu todas as requisições a que foi aplicada com resultados satisfatórios (Figura 19), quando tais resultados são comparados com a distribuição proporcionada pela política padrão do *Mod_cluster*, pois esta apresentou descartes de requisições quando demonstrou sobrecarga na distribuição das requisições. Apesar de apresentar perda de desempenho com 16 clientes concorrentes, a política padrão do *Mod_cluster* atendeu todas as requisições efetuadas nesse nível de concorrência, no entanto quando o nível passa a ser 32 clientes concorrentes a política realiza descarte de requisições e apresenta maior perda de desempenho. A Figura 19 apresenta o gráfico da quantidade de requisições completadas com sucesso de acordo com o nível de clientes concorrentes e seus respectivos percentuais.

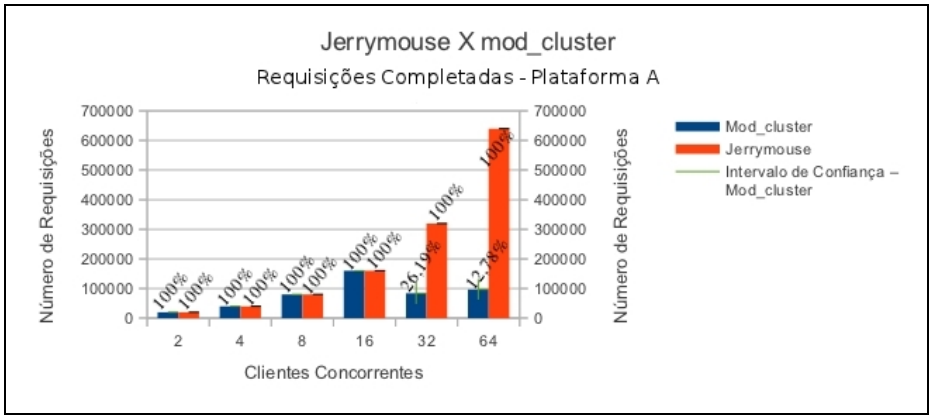

Figura 19: Quantidade de requisições atendidas com sucesso.

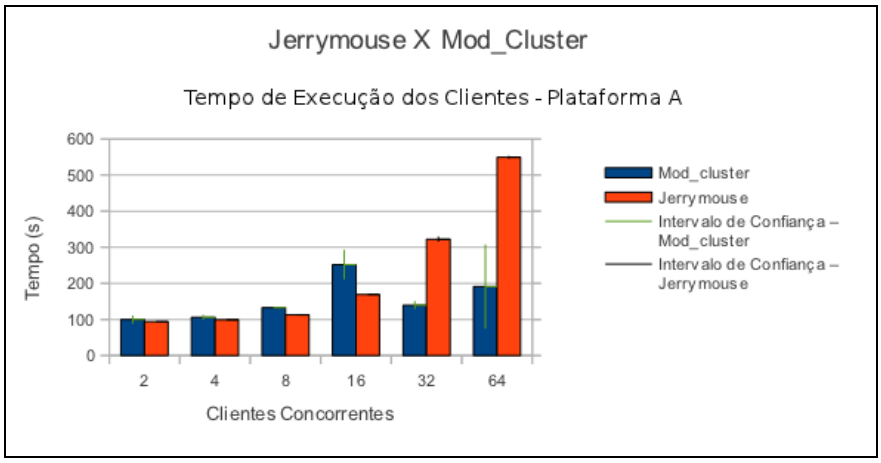

Figura 20: Tempo médio de execução das aplicações clientes

Devido à sobrecarga apresentada a partir do nível de 16 clientes concorrentes, o tempo de execução das aplicações clientes que realizam as requisições, apresenta instabilidade na sua execução nos últimos três níveis de concorrência (16, 32 e 64 clientes concorrentes), ver Figura 20. Essa instabilidade de execução é causada pela saturação na plataforma resultante da carga de trabalho aplicada. Apesar da instabilidade de execução no nível de 16 clientes, o tempo do *Mod_cluster* ainda é maior que o tempo obtido com a distribuição do *Jerrymouse*, pois ambas as políticas utilizadas atenderam todas as requisições que lhe foram submetidas (160.000 ao todo). Nos níveis 32 e 64 clientes o tempo de execução das aplicações clientes é bem inferior para a distribuição realizada pela política padrão do *Mod_cluster*, comparada com a distribuição realizada com a política *Performance*. Isso acontece porque a primeira política realiza descarte de requisições, enquanto a *Performance* atende todas as requisições. Com isso, o tempo total de execução das aplicações clientes (distribuídas pela política padrão do *Mod_cluster*) é inferior mesmo tendo o tempo médio de resposta superior. É importante esclarecer que o tempo médio de resposta é calculado usando como base o tempo das requisições completadas com sucesso. Enquanto que o tempo de execução da aplicação cliente se refere ao tempo total que a aplicação cliente permaneceu executando considerando as requisições descartas, no entanto se uma requisição é descartada o cliente recebe uma resposta de erro e não precisa aguardar o processamento da requisição em questão.

Nos experimentos realizados na plataforma B, que disponibiliza oito nós *back-ends*, a quantidade de nós *back-ends* foi variada durante os experimentos com objetivo de analisar o comportamento da distribuição com alteração na quantidade de nós dedicados a atenderem as

requisições. As quantidades de nós *back-ends* utilizados durante estes experimentos foram 2, 4 e 8.

Na Figura 21, são apresentados os tempos médios de resposta obtidos nos experimentos com 2, 4 e 8 nós *Back-ends*. Apesar da política *Performance* possuir melhor desempenho que a política padrão do *Mod_cluster,* em todos os casos; nos experimentos com 2 *Back-ends* todos os tempos médios de respostas estão próximos. Isso ocorre devido à sobrecarga dos nós *Back-ends* e o tempo de processamento das requisições ocultarem o desempenho da distribuição. No entanto, nos experimentos com 4 e 8 *Back-ends* os tempos médios de resposta apresentam diferença significativa em seus valores. A metodologia aplicada pela política *Performance* apresenta desempenho superior na distribuição das requisições, reduzindo o tempo médio de resposta e explorando melhor os recursos computacionais disponibilizados pela plataforma. Por outro lado, a distribuição realizada pela política padrão do *Mod_cluster* apresenta degradação em seu desempenho quando submetida à alta demanda contendo diversos nós *Back-end* disponíveis. Isso pode ser observado nos tempos médios de resposta obtidos com 16, 32 e 64 clientes concorrentes nos experimentos com 2, 4 e 8 nós *Back-ends*.

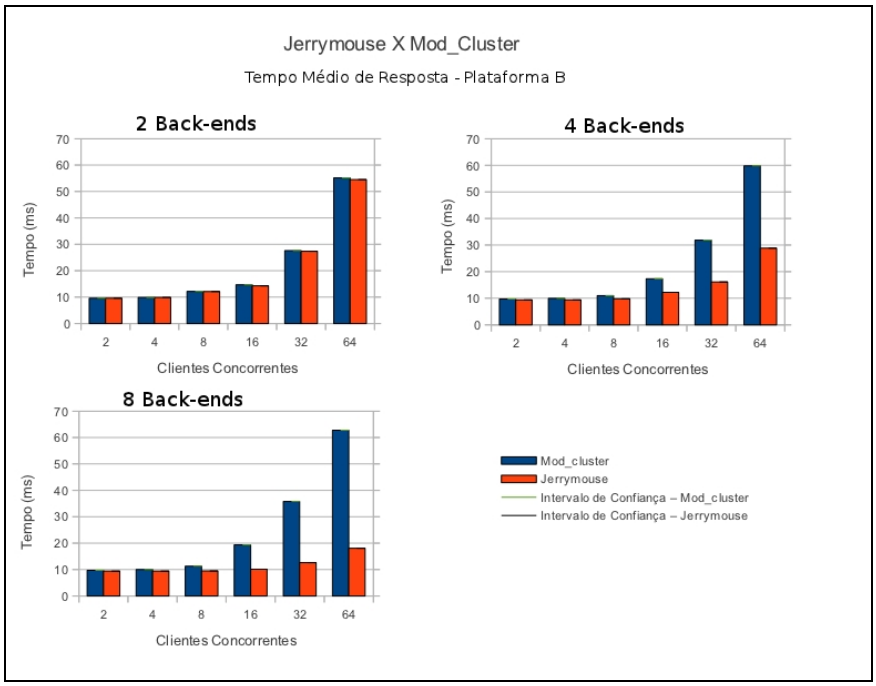

Figura 21: Tempos Médios de Resposta obtidos nos experimentos com 2, 4 e 8 nós *Back-ends*

Os valores apresentados na Figura 22 referem-se ao percentual da variação dos tempos médios de resposta obtidos com todas as configurações destes experimentos. Devidos os resultados obtidos nos experimentos com 2 *Back-ends* serem próximos, este gráfico não apresenta resultados significativos. Nos valores obtidos com os experimentos utilizando 4 e 8 *Back-ends* dedicados a atender as requisições, a política *Performance* apresenta, no caso mais extremo, percentual superior a 70% melhor quando comparado com os tempos obtidos com a distribuição realizada pela política padrão do *Mod_cluster*.

A quantidade de requisições atendidas por segundo nos experimentos realizados na plataforma B, ver Figura 23, correspondem aos resultados já apresentados nos gráficos referentes ao tempo médio de resposta das requisições (Figura 21). Os resultados obtidos com 2 *Back-ends* dedicados são próximos devido à sobrecarga causada pela demanda na plataforma. Enquanto que para os experimentos com 4 e 8 *Back-ends* dedicados, a distribuição realizada pela política *Performance* apresenta maior quantidade de requisições atendidas por segundo, proporcional aos seus respectivos tempos médios de resposta. Desta forma, quanto menor o tempo de resposta apresentado pelo atendimento da requisição maior a quantidade de requisições atendidas por segundo pelo *cluster*. A degradação no desempenho da política padrão do *Mod_cluster* também pode ser observado nas quantidades de requisições atendidas por segundo, considerando os valores obtidos para 16, 32 e 64 clientes concorrentes nos experimentos utilizando 2, 4 e 8 *Back-ends* dedicados ao atendimento das requisições.

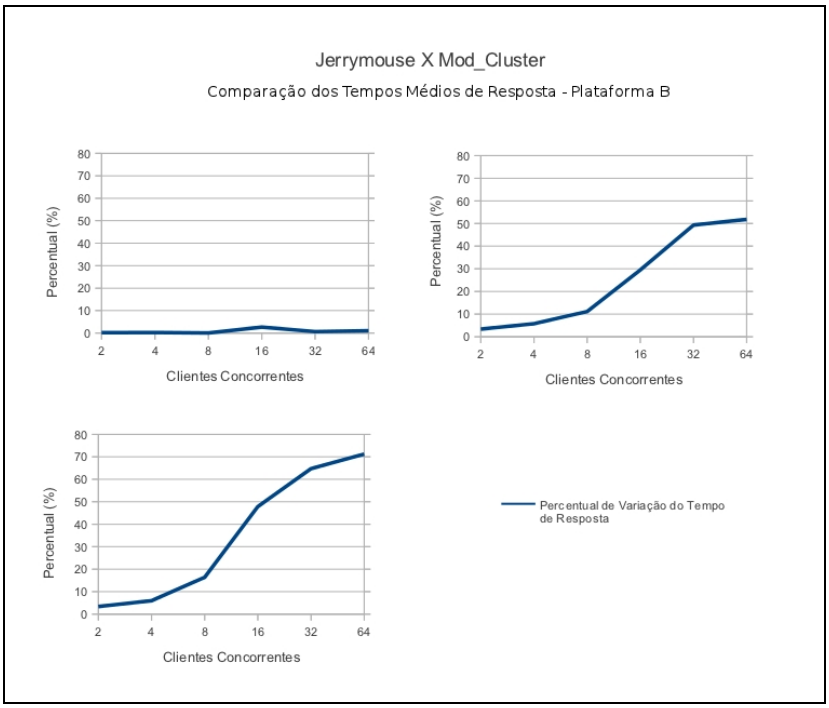

Figura 22: Comparação dos Tempos Médios de Resposta dos experimentos na plataforma B

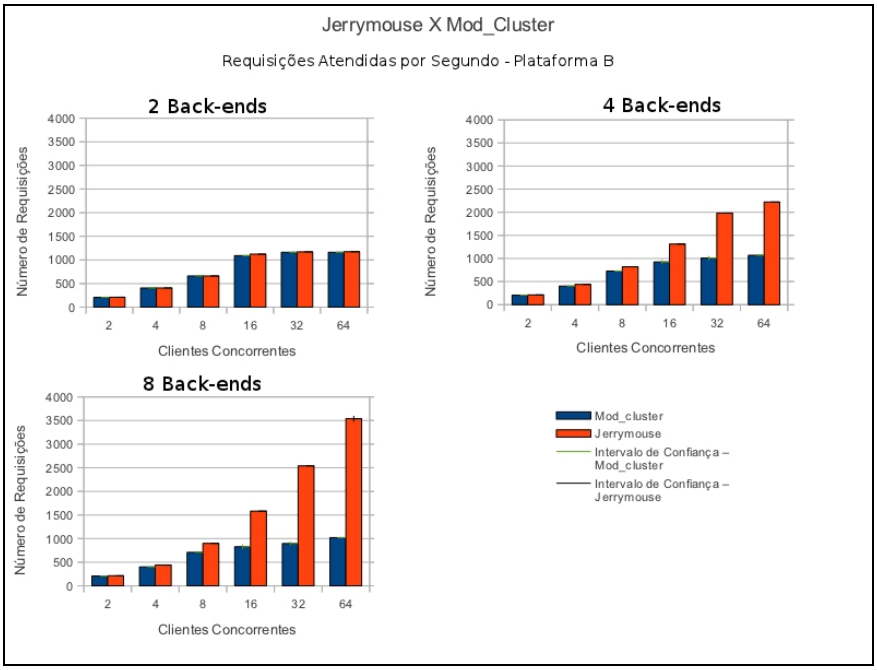

Figura 23: Quantidade de requisições atendidas por segundo na plataforma B

Assim como os gráficos apresentados na Figura 23, os resultados demonstrados pela Figura 24, tempos de execuções das aplicações clientes, são análogos aos resultados referentes ao tempo médio de resposta das requisições.

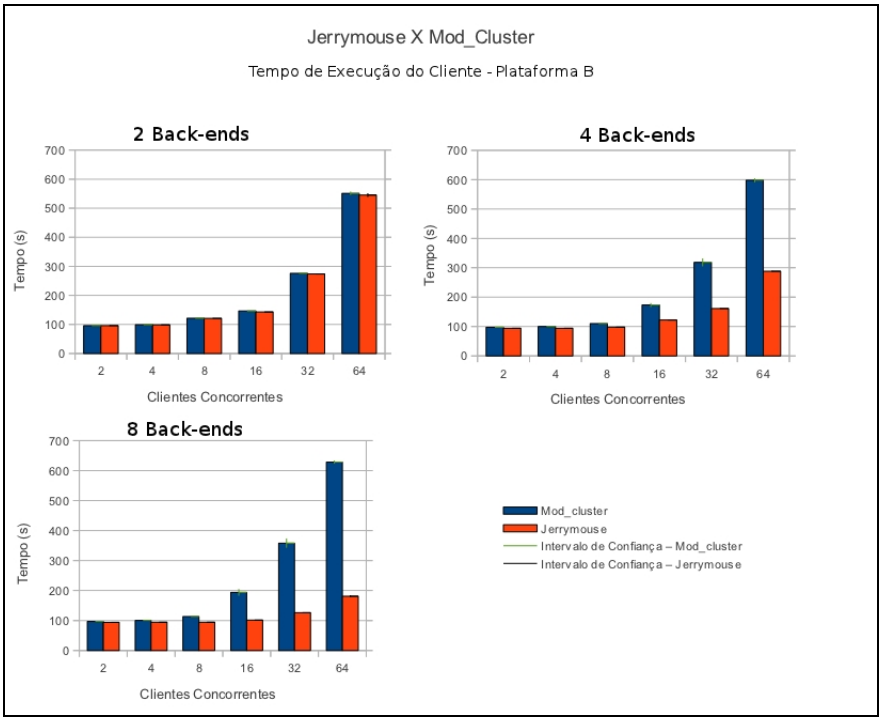

Figura 24: Tempo de execução das aplicações clientes durante os experimentos na plataforma B

A Figura 25 demonstra que todas as requisições realizadas nos experimentos na plataforma B foram atendidas por ambas as políticas de distribuição. Essa estabilidade no atendimento das requisições é o motivo pelo qual os resultados mantiveram um comportamento análogo nos gráficos co-relacionados (Figura 21, Figura 23 e Figura 24). Diferente dos resultados obtidos com a plataforma A, esta plataforma não apresentou descarte de requisições devido à utilização de Sistema Operacional, provedores e *hardware*s voltados para ambientes dedicados a atenderem requisições de *web services*.

A Figura 26 apresenta o calculo da influência dos fatores (Jain, 1991) considerando os resultados obtidos nos experimentos com 32 e 64 clientes concorrentes utilizando 4 e 8 *Back-ends*. Estes experimentos foram selecionados entre os demais por resultarem nos valores mais expressivos da distribuição de requisições sob alta demanda. O resultado da influência dos fatores aponta a política de distribuição aplicada como o fator que mais influência nos

respectivos resultados obtidos. Com isso, é possível observar a importância que a política de distribuição exerce no atendimento das requisições no *cluster* de *web services*.

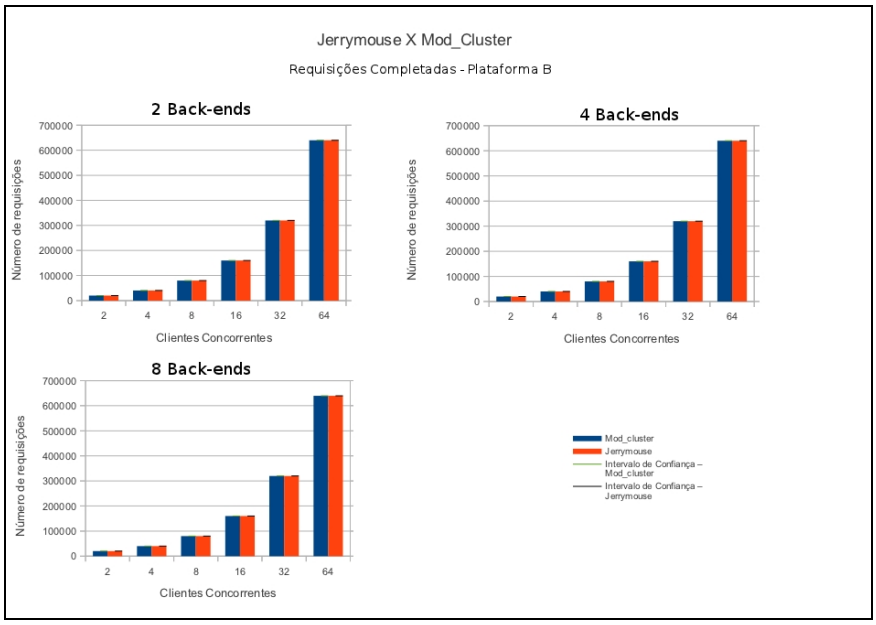

Figura 25: Total de requisições completadas com sucesso na plataforma B

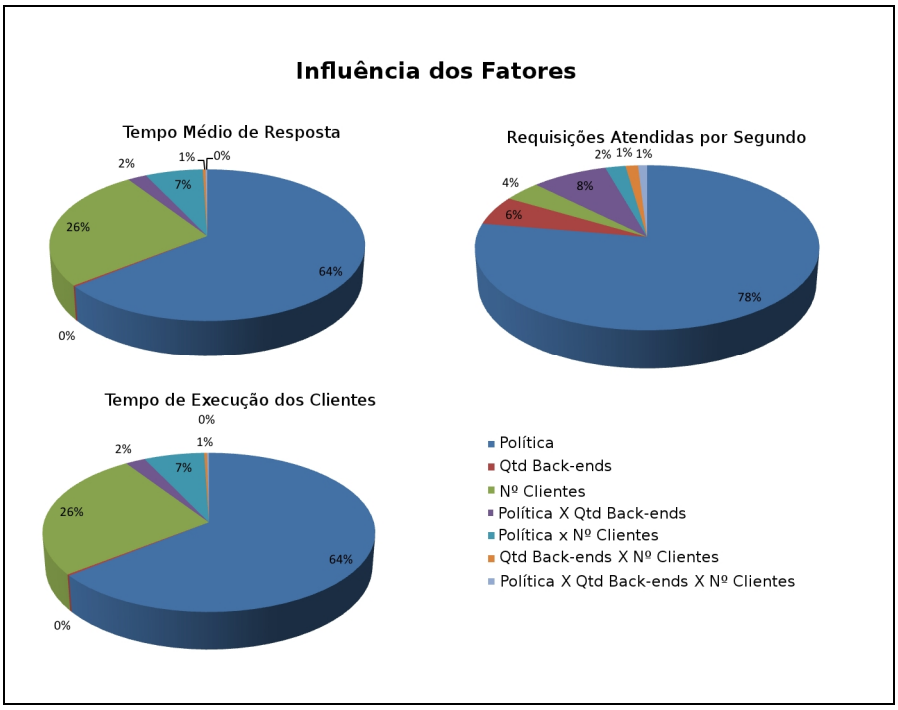

Figura 26: Cálculo da Influência dos Fatores considerando os resultados mais expressivos da plataforma B

Os estudos experimentais realizados com o Módulo Gerenciador de Energia têm como principal objetivo analisar a economia de energia alcançada mantendo o máximo de confiabilidade no atendimento das requisições. Estes experimentos foram realizados na plataforma B. Nos experimentos que o Módulo estiver em execução a plataforma inicia suas operações apenas com o nó *Front-end* ligado, para os experimentos sem a execução do Módulo Gerenciador de Energia os nós *back-ends* permanecem ligados durante toda a execução dos experimentos. O Módulo Gerenciador de Energia é configurado para utilizar o índice de carga *cpu_idle* (percentual de *CPU* disponível) e manter o valor médio do índice na plataforma entre 45 (mínimo) e 80 (máximo) porcento. O tempo de intervalo entre as monitorações realizadas pelo Módulo é definido em 10 segundos.

Embora o principal objetivo seja a economia de energia, esta análise confronta o tempo de resposta médio obtido com a distribuição das requisições utilizando a política *Performance* associado ao Módulo Gerenciador de Energia com o tempo médio de resposta obtido com a distribuição realizada apenas com a política *Performance*. Com isso, é possível verificar se o Módulo Gerenciador de Energia gerou aumento no tempo médio de resposta das requisições ao prover economia de energia, e se for o caso quantificar este aumento.

A Figura 27 apresenta o comportamento do índice de carga *cpu_idle* durante os experimentos do Módulo Gerenciador de Energia (MGE) na plataforma B. Estão representados os valores coletados referente ao índice de carga e a análise dos valores utilizando a suavização resultante da aplicação da Média Móvel Exponencial nos experimentos (Política *Performance*/MGE X Política *Performance*). Próximo as linhas referentes aos valores do índice de carga e da análise do MGE no experimento sem o gerenciamento dos nós *back-ends*, existem descrições referentes à quantidade de clientes concorrentes durante os períodos de variação do índice de carga. Estas descrições podem ser usadas como referencias da quantidade de clientes concorrentes para os valores do índice de carga obtidos no experimento com o gerenciamento dos nós *back-ends* pelo MGE.

Pode-se observar que a análise dos valores do índice de carga realizada pelo MGE espelha o comportamento do índice amortizando falsos picos de desempenho proporcionando estabilidade a suas representações em ambos os experimentos (Figura 27).

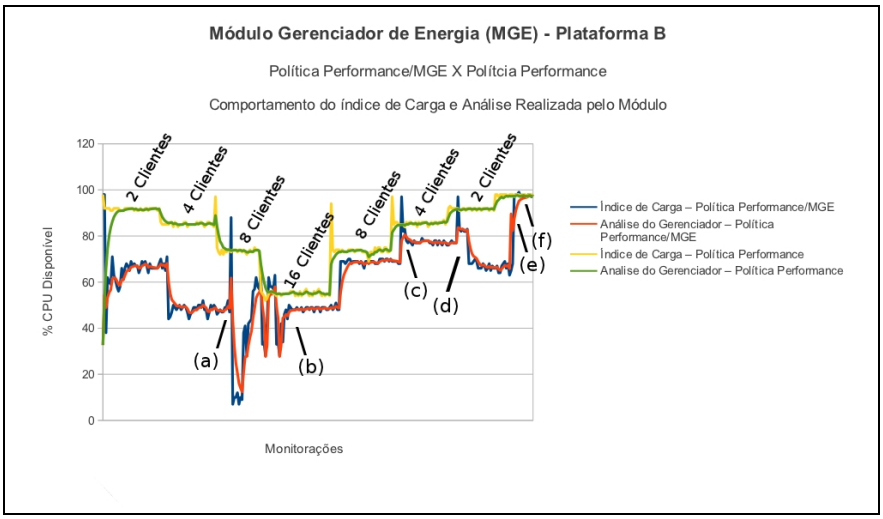

Figura 27: Representação do comportamento do índice de carga e na análise nos experimentos com o Módulo Gerenciador de Energia na plataforma B

Como já citado anteriormente, o experimento com execução do MGE é iniciado apenas com o nó *Front-end* ligado. O ponto *(a)* na Figura 27 representa o instante que 8 clientes concorrentes iniciam as suas requisições e a plataforma é saturada. O MGE identifica a saturação da plataforma e inicia o processo de *wake-up* dos nós *back-ends* desligados até que a saturação seja controlada. O processo de *wake-up* abrange todo o período de requisições de 8 clientes e o inicio das requisições realizadas por 16 clientes concorrentes. No ponto *(b)* a plataforma encontra-se com 5 nós em atividade (1 *Front-end* e 4 *Back-ends*) e a carga de trabalho na plataforma apresenta-se controlada[14]. O ponto *(c)* localizado no início das requisições realizadas por 4 clientes concorrentes indica alto nível de ociosidade da plataforma (carga de trabalho relativamente baixo) e o MGE inicia o processo de *shutdown* do último nó *back-end* que foi ligado, mantendo a carga de trabalho adequada para a plataforma. Quando inicia as requisições realizadas por 2 clientes simultâneos (ponto *(d)*), o MGE identifica nova situação de alta ociosidade da plataforma e inicia o processo de *shutdown* de outro nó *back-end*, permanecendo ligado 3 nós (1 *Front-end* e 2 *Back-ends*). No fim das requisições (pontos *(e)* e *(f)*), o MGE identifica situação de alta ociosidade da plataforma e desliga os últimos dois nós *back-ends* ligados, permanecendo ligado apenas o nó *front-end*. Com isso, os experimentos são concluídos e a monitoração é finalizada.

[14] O termo "controlado" é empregado considerando os limites máximos e mínimos definidos no arquivo de configuração.

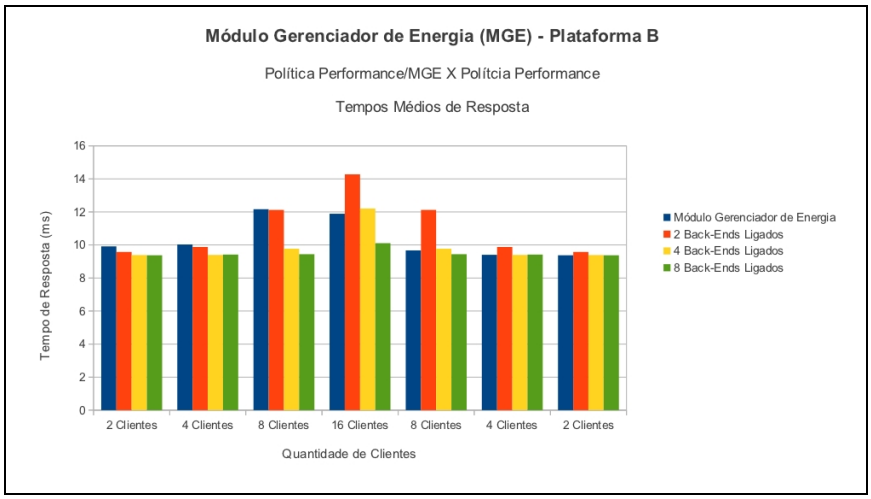

Figura 28: Tempos Médios de Respostas obtidos nos experimentos com o Módulo Gerenciador de Energia e com nós *back-ends* ligados permanentemente na plataforma B

A Figura 28 apresenta os tempos médios de resposta apresentados obtidos nos experimentos do Módulo Gerenciador de Energia na plataforma B. Os tempos médios de respostas obtidos com o Módulo Gerenciador de Energia apresentados para a primeira execução com 2 e 4 clientes são superiores porque durante estes períodos apenas o nó *Front-end* esta ativo na plataforma. Durante o primeiro momento das requisições de 8 clientes, o tempo médio passa ser próximo do obtido com dois *back-ends* ativos, porque nesse período o MGE identifica a saturação no sistema e inicia o processo de *wake-up* dos nós *back-ends* até que a saturação seja controlada. Para as requisições realizadas por 16 clientes concorrentes, o tempo médio de resposta obtido pelo MGE passa ser próximo do tempo obtido com 4 nós *back-ends* permanentemente ligados. Essa semelhança ocorre para os demais níveis de concorrência de clientes, mesmo com nós *back-ends* sendo desligados.

De acordo com a configuração definida, o Módulo Gerenciador de Energia ligou 4 nós *back-ends* para suportar a demanda gerada durante o período mais crítico dos experimentos, 16 clientes concorrentes. Por isso, as comparações realizadas referentes a economia de energia e tempos médios de resposta serão entre as informações obtidas na plataforma B pela Política *Performace*/MGE e Política *Performace* com 4 *back-ends* ligados.

Considerando todos os níveis de clientes concorrentes nos experimentos realizados com o Módulo Gerenciador de Energia na plataforma B, a Política *Performace*/MGE apresentou um tempo médio de resposta aproximadamente 5% superior ao apresentado pela Política *Performace*, onde os nós permaneceram ligados durante todo o experimento. Esse

percentual refere-se a 5,2 microssegundos no tempo médio de resposta, ver Tabela 3. Enquanto que o tempo total de duração do experimento com Política *Performace*/MGE teve um aumento de aproximadamente 10% em relação ao experimento com a Política *Performace*. Esse percentual refere-se a quase 5 minutos de acréscimo no tempo total de duração do experimento.

Tabela 3: Tempo Médio de Resposta, considerando todos os níveis de clientes concorrentes dos experimentos realizados com o Módulo Gerenciador de Energia na Plataforma B

	Política *Performance*/MGE	**Política *Performance***
Tempo Médio de Resposta (ms)	9,92	9,4
Tempo de duração dos experimentos (minutos)	00:48:15	00:43:38

Todos os computadores da plataforma B possuem uma fonte de energia com potência de 450 *Watts*. Para os cálculos de consumo de energia estimam que cada nó consume 450 *Watts* por hora durante o período em que permanece ligado. No experimento com a Política *Performace* foi consumido 1,63 *KWh*. Enquanto que no experimento com a Política *Performace*/MGE o consumo aproximado é de 1,11 *KWh*, considerando a Tabela 4 que demonstra o período em que os nós da plataforma B permaneceram ligados no experimento com a Política *Performace*/MGE. A Política *Performace*/MGE proveu uma economia de energia aproximada de 31% durante os experimentos realizados.

Tabela 4: Tempo dos nós da plataforma B que permaneceram ligados durante o experimento com a Política *Performace*/MGE

Nó	**Tempo de Funcionamento**
Front-end	00:48:15
Back-end 1	00:34:42
Back-end 2	00:27:15
Back-end 3	00:23:28
Back-end 4	00:15:28

Se a média do consumo obtido nos experimentos com a Política *Performace* e a Política *Performace*/MGE na plataforma B forem considerados cargas de trabalho normais para o período de um mês, a Política *Performace* consumirá cerca de 1670 *KWh* por mês enquanto que a Política *Performace*/MGE consumirá cerca de 1030 *KWh* por mês. Segundo CPFL (2012), o *KWh* custa R$ 0,31 reais, portanto os ambientes citados teriam um custo operacional durante um mês de atividade com consumo de energia de R$ 518,00 e R$ 320,00 reais respectivamente.

É importante salientar que o aumento no tempo médio de resposta pode ser reduzido diminuindo o valor do limite máximo da carga de trabalho aceita como adequada pelo Módulo Gerenciador de Energia, para que nós *back-ends* sejam ligados antes que apresentem acréscimos significativos em seu tempo médio de resposta. No entanto, tal alteração tende a aumentar o consumo de energia da plataforma, já que os nós *back-ends* serão mantidos ligados por maior período de tempo.

5.4 Considerações Finais

Este capítulo apresentou o planejamento, os resultados e a análise dos resultados obtidos com os estudos experimentais realizados. Os resultados obtidos com a política *Performance* apresentam desempenho superior na distribuição de requisições frente à política padrão do *Mod_cluster*. O Módulo Gerenciador de Energia por sua vez, alcançou seu objetivo de economizar o consumo de energia elétrica pela plataforma mantendo a alta confiabilidade no atendimento das requisições.

No próximo capítulo são detalhadas as conclusões, contribuições e trabalhos futuros deste projeto de mestrado.

6 Conclusões

6.1 Considerações Finais

Este trabalho é motivado pelo crescimento na utilização de *web services* em diversas áreas de aplicações, por exemplo, na integração de sistemas. A possibilidade de decompor sistemas e integrá-los de diversas formas, inclusive pela *web*, faz com que aumente de maneira significativa a quantidade de requisições realizadas a *clusters* que fornecem a estrutura necessária para a execução de *web services*. Devido ao alto fluxo de dados nestas estruturas, houve a necessidade de desenvolver políticas transparentes, flexíveis e dinâmicas que suportassem essa demanda e provessem uma distribuição eficiente das requisições sem sobrecarregar a plataforma.

Um dos componentes essenciais de uma política dinâmica de distribuição de requisições em *clusters* de *web services* são os índices de carga utilizados. Este trabalho aprofundou estudos em tais índices, analisando o comportamento de diversos índices de carga sob diferentes demandas específicas (homogênea) e sob demanda mista (heterogênea). Uma nova métrica foi proposta como resultado deste estudo, chamada *Average-JMX_Time*. Esta métrica é utilizada como índice de carga e apresentou estabilidade e fidelidade na representação dos desempenhos apresentados pelos diferentes *web services,* os quais geraram diferentes demandas na plataforma. O *Average-JMX_Time* foi capaz de encapsular a complexidade estrutural da plataforma e simplificar a interpretação do índice de carga.

Outro fator importante abordado neste mestrado é o consumo responsável de energia elétrica pela plataforma. Esta é uma preocupação que vem recebendo cada vez mais estudos para o desenvolvimento de tecnologias que colaborem com a conservação do meio ambiente (*Green Computing*).

Verificando a demanda por políticas de distribuição, este projeto de mestrado propôs uma política de distribuição de requisição denominada *Performance*. O principal objetivo desta política é proporcionar uma distribuição transparente, flexível e dinâmica das requisições em que é aplicada. A política *Performance* baseia-se em uma lista de elementos ordenada em ordem decrescente de desempenho de cada nó que constitui o *cluster*, representado pelo índice de carga definido, e uma metodologia de nós “ativos” define os nós

que podem receber as requisições. Um protótipo da política *Performance* foi implementado para ser usado no protótipo da arquitetura de distribuição proposta por Matos (2009), *Jerrymouse*. Estudos experimentais foram executados confrontando os resultados obtidos pela distribuição realizada pelo protótipo da política *Performance*, com os resultados obtidos pela distribuição realizada com a política padrão do distribuidor de requisições *Mod_cluster*.

Um novo Módulo Gerenciador de Energia (MGE) foi associado ao *Jerrymouse* com o objetivo de prover economia no consumo de energia do *cluster* de *web services*. Para isso, o MGE é capaz de ligar ou desligar nós do *cluster* conforme análise da carga de trabalho na plataforma. O limite máximo e mínimo de carga de trabalho adequados a plataforma são configuráveis conforme os objetivos da plataforma. O MGE possibilita que seja configurado um *cluster* heterogêneo, com nós gerenciado pelo MGE e nós que permanecem ligados durante toda a operação da plataforma. Este módulo possui integração com a política *Performance* para fornecer suporte a alta confiabilidade na distribuição das requisições, de maneira que não ocorram falhas na distribuição das requisições durante o gerenciamento de energia do *cluster*. Estudos experimentais foram executados confrontando os resultados obtidos pela distribuição realizada pelos protótipos, política *Performance* associada com o Módulo Gerenciador de Energia, com os resultados obtidos com a distribuição realizada apenas com o protótipo da política *Performance*.

Os estudos experimentais realizados demonstraram que a política *Performance* apresenta estabilidade e alta confiabilidade na distribuição das requisições independente das plataformas executadas em nossos experimentos, tendo todas as suas requisições atendidas. Esse comportamento não foi observado na política padrão do *Mod_cluster*, o qual descartou requisições nos experimentos realizados em plataforma com *softwares* e *hardwares* não ideais para *web services*. A política *Performance* também apresentou aumento de desempenho na distribuição das requisições, influenciando diretamente no tempo médio de resposta das requisições. É possível observar também que a distribuição realizada pela política *Performance* é capaz de suportar uma maior demanda sem sobrecarregar a plataforma em que é executada. Os resultados obtidos com a execução do Módulo Gerenciador de Energia (MGE) associado à política *Performance* apresentam comportamento satisfatório à configuração que o MGE foi submetido aos experimentos. Apesar dos experimentos com o MGE não serem considerados conclusivos, devido à limitação dos nossos experimentos, o MGE apresentou resultados favoráveis ao seu objetivo de prover economia no consumo de

energia da plataforma preservando a alta confiabilidade na distribuição das requisições. A política *Performance* manteve sua flexibilidade e dinamismo com o acréscimo de nós que estavam desligados, quando a plataforma foi considerada saturada pelo MGE, e na redução de nós ligados, quando o MGE considerou a plataforma ociosa.

Os resultados do experimento com a política *Performance* demonstram que o objetivo de prover distribuição transparente, flexível e dinâmica das requisições em *cluster* de *web services* foi alcançado e essa política também apresentou melhor desempenho na distribuição das requisições quando comparada a política padrão do *Mod_cluster*. Os resultados com o Módulo Gerenciador de Energia associado à política *Performance* apontam para um futuro promissor na realização do seu objetivo, fornecer economia no consumo de energia da plataforma.

6.2 Contribuições

A principal contribuição deste trabalho de mestrado é a melhoria da qualidade na distribuição de requisições a *clusters* de *web services*. Esta contribuição foi construída com esforços de investigação científica nas seguintes direções:

- Estudo sobre os índices de carga: Este estudo contribuiu com uma análise sistemática do comportamento de diferentes índices de cargas, comumente utilizados para balanceamento de carga, sob demandas variadas no contexto de *web services*. Os resultados deste estudo contribuem para demonstrar que os índices analisados não fornecem uma medida precisa do desempenho global dos *web services* que são executados na plataforma. Estes índices são altamente influenciados por diversos recursos particulares de *hardware* e a demanda pelo serviço global, e não podem ser aplicada com sucesso por si só, sem considerar esses outros fatores. Embora estes índices possam ser utilizados com sucesso para alguns tipos específicos de carga de trabalho e de *hardware*, qualquer tentativa de utilizá-los em um ambiente global apenas transfere para o utilizador final a responsabilidade de selecionar o índice correto para um serviço em particular.

- Proposta de uma nova métrica capaz de ser usada como índice de carga: A nova métrica proposta é capaz de representar fielmente o desempenho apresentado pelo *web service* e encapsular a complexidade estrutural da plataforma, simplificando a sua utilização. Esta

métrica apresenta baixo custo em sua obtenção e suporta a portabilidade entre plataformas heterogêneas, além de fornecer transparência às aplicações, provedores e serviços.

- Desenvolvimento de uma nova política de distribuição de requisições em *clusters* de *web services*: Esta política de distribuição apresenta estabilidade e alta confiabilidade na distribuição das requisições independente da plataforma executada. Esta política realiza uma distribuição transparente, flexível e dinâmica, além de prover aumento de desempenho na distribuição das requisições, influenciando diretamente no tempo médio de resposta das requisições em que é aplicada. A distribuição realizada por esta nova política fornece suporte à grande demanda de carga de trabalho sem sobrecarregar a plataforma em que é executada.

- Desenvolvimento de um módulo para a gerência da energia consumida no cluster: O módulo desenvolvido para gerenciar o consumo de energia é capaz de ligar ou desligar nós do *cluster* conforme análise, realizada pelo módulo, da carga de trabalho na plataforma. Este módulo possibilita que seja configurado um *cluster* heterogêneo, com nós gerenciados e nós que permanecem ligados durante toda a operação da plataforma. Este módulo pode ser executado independente da política aplicada. No entanto, fornece suporte a integração com a política de distribuição de requisições em *cluster* de *web service* (desenvolvida neste projeto de mestrado) para fornecer suporte à alta confiabilidade na distribuição das requisições, de maneira que não ocorram falhas na distribuição das requisições durante o gerenciamento de energia do *cluster*. Este módulo proporciona economia no consumo de energia da plataforma sem prejudicar a transparência, flexibilidade e dinamismo da distribuição de requisições no *cluster* de *web services*.

Os principais resultados obtidos com este trabalho foram descritos nos seguintes artigos:

- SOUZA, P. S. L., FAIÇAL, B. S., MATOS, Jonathan de, SANTANA, Marcos José, SANTANA, R. H. C., ZALUSKA, E. Jerrymouse: a tool for a flexible and dynamic distribution of web service requests. In: 8th IEEE 2011 International Conference on Services Computing (SCC 2011), 2011, Washington. Proceedings of SCC2011. Washington: IEEE, 2011. v.1. p.1 - 8

- FAIÇAL, B. S.; SOUZA, P. S. L.; MATOS, J.; SANTANA, M. J.; SANTANA, R. H. C. Jerrymouse - Manual de Instalação. Biblioteca Prof. Achille Bassi – ICMC/USP. ISSN-0103-2569, Nº 361. São Carlos, 2010.
- SOUZA, P. S. L.; FAIÇAL, B. S.; MATOS, J.; ZALUSKA, E.; SANTANA, R. H. C.; SANTANA, M. J. Load Index Metrics for Web Services: a Systematic Evaluation. In: Journal of Universal Computer Science (J-UCS 2011), 2011. (SUBMETIDO)

6.3 Trabalhos Futuros

A seguir são apresentados os trabalhos futuros relacionados a este projeto de mestrado:

- Implementar, no Módulo Gerenciador de Energia, uma função que analise o desempenho apresentado pelos nós do *cluster*, quando ligados, com objetivo de prover dinamismo na escolha do nó a ser ligado ou desligado. Com isso, pode-se definir se os primeiros nós a serem ligados seriam os que apresentassem melhor desempenho ou menor consumo de energia;
- Realizar estudos experimentais com a política *Performance* em plataforma heterogênea com objetivo de analisar a qualidade da distribuição e desempenho neste ambiente;
- Verificar o desempenho da Política *Performance* com diferentes índices de carga em diferentes plataformas homogêneas e heterogêneas.
- Realizar estudos experimentais com a política *Performance* utilizando carga de trabalho irregular.

Referências Bibliográficas

(Al-Masri; Mahmoud, 2007)
Al-Masri, E., Mahmoud, Q. H.: Discovering the Best Web Service. Proceedings of the 16th international conference on World Wide Web, 2007. doi: 10.1145/1242572.1242795

(AMD, 1995)
AMD. Magic Packet Technology. Publication# 20213, 1995. Disponível em <http://support.amd.com/us/Embedded_TechDocs/20213.pdf>, Acessado em 24 de fevereiro de 2012.

(Ardagna; Pernici, 2006)
Ardagna, D., Pernici, B.: Global and Local QoS Guarantee in Web Service Selection. Business Process Management Workshops, 2006. doi: 10.1007/11678564_4

(Binkert et. al., 2005)
Binkert, N. L., Hsu, L. R., Saidi, A. G., Dreslinski, R. G., Schultz, A. L., Reinhardt, S. K.: Performance Analysis of System Overheads in TCP/IP Workloads. 14th International Conference on Parallel Architectures and Compilation Techniques, 2005. doi:10.1109/PACT.2005.35

(Biolchini, 2005)
Biolchini, J.; Mian, P.; Natali, A..; Travassos, G. A Systematic Review Process for Software Engineering. Computer Science Dept., COPPE/UFRJ, Rio de Janeiro/RJ - Brazil, 2005.

(Brooks, 2008)
BROOKS, Christopher A. An Introduction to Web Services. Disponível em: <http:// www.cs.usask.ca/~cab938/An%20Introduction%20to%20Web%20Services.pdf>. Acesso em: 30 jan. 2008.

(Carter, 2007)
Carter, Sandy. The new language of business: SOA&Web2.0. Crawfordsville: Pearson Ed. Inc, 2007. 299 p.

(Chen et. al., 2008)
Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., Zhao, F.: Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services. Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, 337-350, 2008

(Conocchioli, 2006)
Conocchioli, Victor R. Load Balancing ArcIMS with an Apache/Tomcat Reverse Proxy. Proceedings of the Twenty-Sixth Annual ESRI User Conference, 2006.

(Copaciu et. al, 2007)
Copaciu, F.; Kluth, S.; Porzucek, T.; Zorn, W. Hierarchical modeling of the Axis2 web services framework with FMC-QE. 3rd IEEE/Create-Net International Conference on Communication System Software and Middleware, COMSWARE, p 74-81, 2007.

(CPFL, 2012)
CPFL. Taxas e Tarifas. Disponível em: <http://www.cpfl.com.br/Informaccedilotildees/TaxaseTarifas/tabid/206/Default.aspx>. Acessado em 25 de fevereiro de 2012.

(Dyachuk; Deters, 2007)
Dyachuk, D., Deters, R.: Optimizing Performance of Web Service Providers. Advanced Networking and Applications, 2007. doi: 10.1109/AINA.2007.107

(Engineering Village, 2009)
Engineering Village. *Printed_Documentation.* Disponível em <http://www.engineeringvillage.com/controller/servlet/Controller?CID=help&database=131073>, acessado em 16 de Set. 2009.

(Ferrari; Maldonado, 2007)
Ferrari, F. C.; Maldonado, J. C. Teste de Software Orientado a Aspectos: Uma Revisão Sistemática. Relatórios Técnicos do ICMC. ISSN 0103-2569. São Carlos-SP, 2007.

(Ferrari; Zhou, 1987)
Ferrari, D.; Zhou, S.: An Empirical Investigation of Load Indices for Load Balancing Applications. 12th Int'l Symposium on Computer Performance Modeling, Measurement, and Evaluation, 515-528, 1987.

(Ferris; Farrell, 2003)
Ferris, C.; Farrell, J. What are Web services? Communications of the ACM, v. 46, n. 6, p. 31–31, jun. 2003.

(Fleury et. al., 2005)
FLEURY, Mark; STARK, Scott; NORMAN, Richards. JBoss 4.0 The Official Guide. New York: Sams Publishin, 2005. 648 p.

(Jain, 1991)
Jain, Raj. The art of computer systems performance analysis : techniques for experimental design, measurement, simulation, and modeling. New York : Wiley, 685 p. 1991.

(Jayalath; Fernando, 2007)
Jayalath, C.M.; Fernando, R.U. A Modular Architecture for Secure and Reliable Distributed Communication. Second International Conference on Availability, Reliability and Security, ARES 2007, p 621-628, 2007.

(JBoss, 2010)
JBoss. The JBoss 4 Application Server Guide. Disponível em: <http://docs.jboss.org/jbossas/jboss4guide/r4/html/index.html>. Acessado em: 28 janeiro 2010.

(JBoss, 2012)
JBoss. The JBoss 5 Application Server Guide. Disponível em: < http://docs.jboss.org/jbossclustering/cluster_guide/5.1/html/cluster.concepts.chapt.html>. Acessado em: 11 janeiro 2012.

(Josuttis, 2007)
JOSUTTIS, NICOLAI M. SOA in Practice: Art of Distributed System Design. OREILLY & ASSOC. 1º Edição, ISBN: 0596529554, 2007, 307 p.

(Li; Jin; Han, 2006)
Li, Z., Jin, Y., Han, J.: A Runtime Monitoring and Validation Framework for Web Service Interactions. Australian Software Engineering Conference, 2006. doi:10.1109/ASWEC.2006.6

(Liu; Gorton; Zhu, 2007)
Liu, Y., Gorton, I., Zhu, L.: Performance Prediction of Service-Oriented Applications based on an Enterprise Service Bus. Computer Software and Applications Conference, 2007. doi: 10.1109/COMPSAC.2007.166

(Mahmoud, 2011)
Mahmoud, Q. H. Getting Started with Java Management Extensions (JMX): Developing Management and Monitoring Solutions. ORACLE, SDN Sun Developer Network, http://java.sun.com/developer/technicalArticles/ J2SE/jmx.html. 2004. Accessed 03 March 2011.

(Marzolla; Mirandola, 2007)
Marzolla, M., Mirandola, R.: Performance Prediction of Web Service Workflows. Quality of software architectures 3rd international conference on Software architectures, components, and applications, 127-144, 2007

(Massie et al., 2004)
MASSIE, Matthew L.; CHUN, Brent N.; CULLER, David E. The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing, North-holland, v. 30, n. 7, p.817-840, 01 jul. 2004.

(Matos, 2009)
MATOS, J. Distribuição de carga flexível e dinâmica para provedores de web services. São Carlos: ICMC/USP, 2009.

(Mod_cluster, 2011)
JBoss. Community Documentation: mod_cluster Documentation. Disponível em: <http://docs.jboss.org/mod_cluster/1.2.0/html/>. Acessado em: 20 novembro de 2011.

(Moodie, 2008)
MOODIE, Matthew. Pro Apache Tomcat 6. New York: Apress, 2007. 325 p.

(Newcomer, 2002)
Newcomer, Eric. Understanding Web services: XML, WSDL, SOAP, and UDDI. Pearson Education, Inc. ISBN: 0-201-75081-3. p. 302, 2002.

(Oracle, 2012)
Oracle. Java™ Management Extensions (JMX™) Specification, version 1.4. Final Release. Disponível em: < http://docs.oracle.com/javase/6/docs/technotes/guides/jmx/JMX_1_4_specification.pdf >. Acessado em: 07 fevereiro 2012.

(OSDB, 2010)
OSDB.: The Open Source Database Benchmark. Disponível em <http://osdb.sourceforge.net/>. Acessado em 10 de Novembro de 2010

(Papazoglou; Georgakopoulos, 2003)
Papazoglou, M. P.; Georgakopoulos, D. Service Oriented Computing. Communications of the ACM, New York, vol. 46, n. 10, out. 2003.

(Peng et. al., 2006)
Peng, L.; Koh, M.; Jie Song; See, S. Grid Service Monitoring for Grid Market Framework. 2006 IEEE International Conference on Networks, ICON 2006 - Networking-Challenges and Frontiers, v 1, p 84-89, 2006.

(Porter; Katz, 2006)
Porter, G., Katz, R. H.: Effective Web Service Loadbalancing through Statistical Monitoring. Commun. ACM, 2006. doi: 10.1145/1118178.1118201

(Red Hat, 2009)
RED HAT. JBoss.org – community driven: JBoss Web. Disponível em: <http://www.jboss.org/jbossweb/index.html>. Acesso em: 18 ago 2009.

(Red Hat, 2011)
Red Hat: JBoss AS6 Documentation. Disponível em: <http://www.jboss.org/jbossas/docs/6-x.html>. Acessado em: 20 Novembro de 2011.

(Red Hat Middleware, 2009)
RED HAT MIDDLEWARE. Hibernate.org – Hibernate. Disponível em: <https://www.hibernate.org/>. Acesso em: 16 ago 2009.

(Rosenberg; Platzer; Dustdar, 2006)
Rosenberg, F., Platzer, C., Dustdar, S.: Bootstrapping Performance and Dependability Attributes of Web Services. IEEE International Conference on Web Services, 2006. doi:10.1109/ICWS.2006.39

(Sabetta; Koziolek, 2008)
Sabetta, A., Koziolek, H.: Measuring Performance Metrics: Techniques and Tools (book chapter). Dependability Metrics, LNCS vol. 4009, p226-232, 2008. doi: 10.1007/978-3-540-68947-8_21

(Schulte et al. 2008)
Schulte, J.; Bopp, T.; Hinn, R.; Hampel, T. Wasabi Beans - SOA for collaborative learning and working systems. 2nd IEEE International Conference on Digital Ecosystems and Technologies, IEEE-DEST 2008, p 177-183, 2008.

(Shanmugam; Ponnayaikko, 2007)
Shanmugam, J.; Ponnavaikko, M. A solution to block Cross Site Scripting Vulnerabilities based on Service Oriented Architecture. 6th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2007; 1st IEEE/ACIS International Workshop on e-Activity, IWEA 2007, p 861-866, 2007

(Siddayatam; Gadge, 2008)
Siddavatam, I.; Gadge, J. Comprehensive test mechanism to detect attack on Web Services. 16th International Conference on Networks, ICON 2008, 2008.

(Souza et. al., 2011)
Souza, P. S. L. ; Faical, B.S. ; Matos, J. ; Santana, M. J. ; Santana, R. H. C. ; Zaluska, E. . Jerrymouse: a tool for a flexible and dynamic distribution of web service requests. In: 8th IEEE 2011 International Conference on Services Computing (SCC 2011), 2011, Washington. Proceedings of SCC2011. Washington : IEEE, 2011. v. 1. p. 520-527.

(Souza et. al., 2011b)
Souza, P. S. L.; Faiçal, B. S.; Matos, J.; Zaluska, E.; Santana, R. H. C.; Santana, M. J. Load Index Metrics for Web Services: a Systematic Evaluation. In: Journal of Universal Computer Science (J-UCS 2011), 2011. (SUBMETIDO)

(The Apache Software Foundation, 2009)
THE APACHE SOFTWARE FOUNDATION. Apache Tomcat. Disponível em: <http://tomcat.apache.org/>. Acesso em: 20 dezembro. 2011.

(The Apache Software Foundation, 2010)
THE APACHE SOFTWARE FOUNDATION. The Apache Tomcat Connector - Reference Guide. Disponível em: < http://tomcat.apache.org/connectors-doc/ >. Acessado em: 28 janeiro 2010.

(The Apache Software Foundation, 2012)
THE APACHE SOFTWARE FOUNDATION, Apache Software. The Apache Tomcat Connector - Reference Guide. Disponível em: <http://tomcat.apache.org/tomcat-7.0-doc/>. Acessado em: 20 janeiro 2012.

(Tseng; Huang, 2008)
Tseng, H.; Huang, C. A Service-Oriented Architecture Based Vendor Managed Inventory System. 4th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2008, p 190-195, 2008.

(Voorsluys; Santana, 2010)
Voorsluys, W., Santana, M. J.: Avaliação de Índices de Carga de Memória no Linux. http://pet.icmc.usp.br/std2008/artigos/639.Mestrado.pdf, 2006. Acessado em 1 de Novembro de 2010.

(ZhenChun; GuoQing, 2006)
ZhenChun, H.; GuoQing, L. An SOA based On-Demand Computation Framework for Spatial Information Processing. Grid and Cooperative Computing Workshops, 2006. GCCW '06. p. 487 – 490, 2006.

Anexo I – Resultados da Análise sobre Índices de Carga

Este anexo tem por objetivo apresentar os gráficos e tabelas dos resultados referentes ao estudo sobre a qualidade de diversos índices de carga no contexto de *web services* (seção 3.6 Resultados). Ressalta-se que a análise seguiu três critérios para avaliar a qualidade do índice de carga: a capacidade de representar a carga de trabalho, a relação da carga de trabalho com o desempenho do serviço e a estabilidade global do índice. Para os cálculos de estatística foram realizadas 1.000 iterações para cada serviço monitorando em cada nível de clientes concorrentes.

Tabela 5: Tempo de execução quando executado os quatro serviços separadamente.

Serviços	1 Cliente		2 Clientes			4 Clientes			8 Clientes		
	Média	D. P.	Média	D. P.	%	Média	D. P.	%	Média	D. P.	%
CPU-Bound	9.8	1.0	10.5	1.5	*8.0*	10.4	1.0	*-0.9*	19.7	2.1	*88.4*
Memory-Bound	10.6	0.1	20.7	0.1	*94.9*	43.1	0.3	*108.4*	100.8	14.2	*133.9*
Network-Bound	14.0	0.0	14.0	0.0	*0.1*	14.1	0.2	*0.4*	14.2	0.8	*0.5*
Banco de Dados	10.2	0.1	6.1	0.1	*-40.3*	6.5	0.2	*6.6*	7.8	0.5	*21.0*

Tabela 6: Índices de *CPU*

Serviços	Índices	1 Cliente		2 Clientes			4 Clientes			8 Clientes		
		Média	D. P.	Média	D. P.	%	Média	D. P.	%	Média	D. P.	%
CPU-bound	*CPU_Idle*	73.0	2.8	51.8	6.5	*-29.1*	11.2	15.1	*-78.3*	7.7	14.5	*-31.2*
	CPU_WIO	0.0	0.0	0.0	0.1	*2.8*	0.0	0.1	*4.6*	0.0	0.2	*52.9*
Memory-bound	*CPU_Idle*	72.4	3.1	46.8	4.8	*-35.4*	2.6	7.4	*-94.5*	12.6	22.5	*395.7*
	CPU_WIO	0.0	0.1	0.0	0.1	*0.0*	0.0	0.3	*0.0*	16.0	25.3	*159800.0*
Network-bound	*CPU_Idle*	92.8	2.9	88.7	4.2	*-4.5*	81.7	7.2	*-7.8*	70.8	12.5	*-13.4*
	CPU_WIO	0.0	0.0	0.0	0.1	*0.0*	0.0	0.1	*0.0*	0.0	0.2	*0.0*
Acesso a Banco de Dados	*CPU_Idle*	92.5	3.3	85.7	4.0	*-7.4*	76.0	5.6	*-11.3*	61.2	5.9	*-19.5*
	CPU_WIO	0.2	1.5	0.0	0.1	*-94.4*	0.0	0.1	*100.0*	0.2	0.4	*750.0*

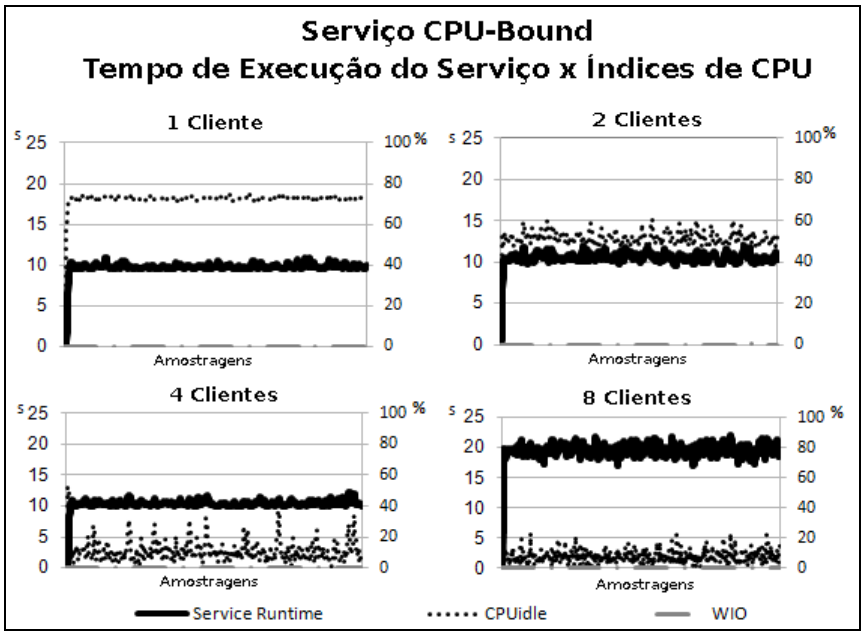

Figura 29: Resultados para os índices de *CPU*, considerando o serviço *CPU-bound* para 1, 2, 4 e 8 clientes.

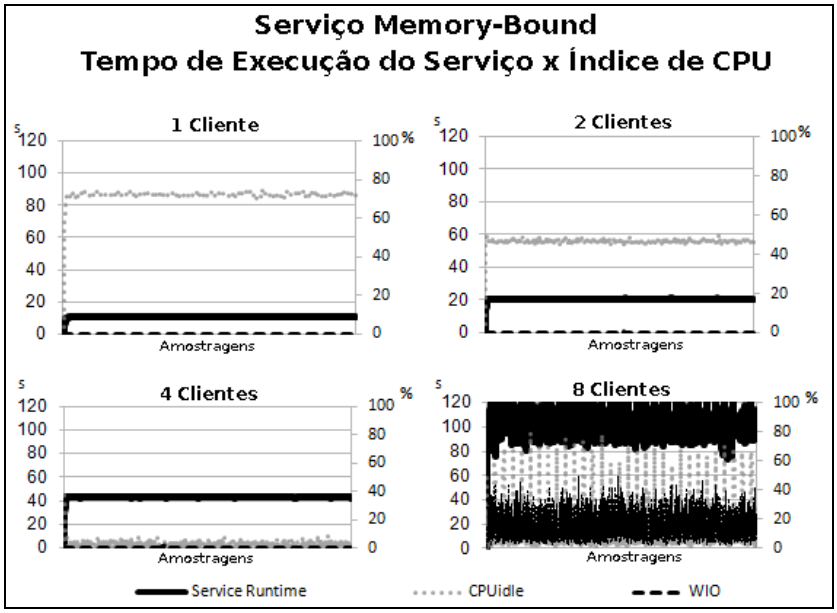

Figura 30: Resultados para os índices de *CPU*, considerando o serviço *memory-bound* para 1, 2, 4 e 8 clientes.

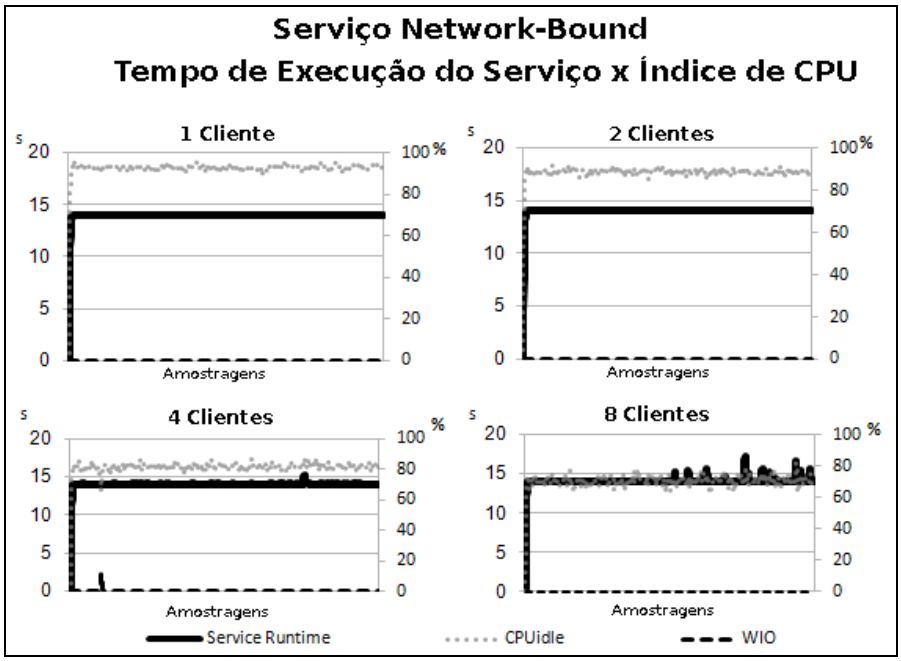

Figura 31: Resultados para os índices de *CPU*, considerando o serviço de *network-Bound* para 1, 2, 4 e 8 clientes.

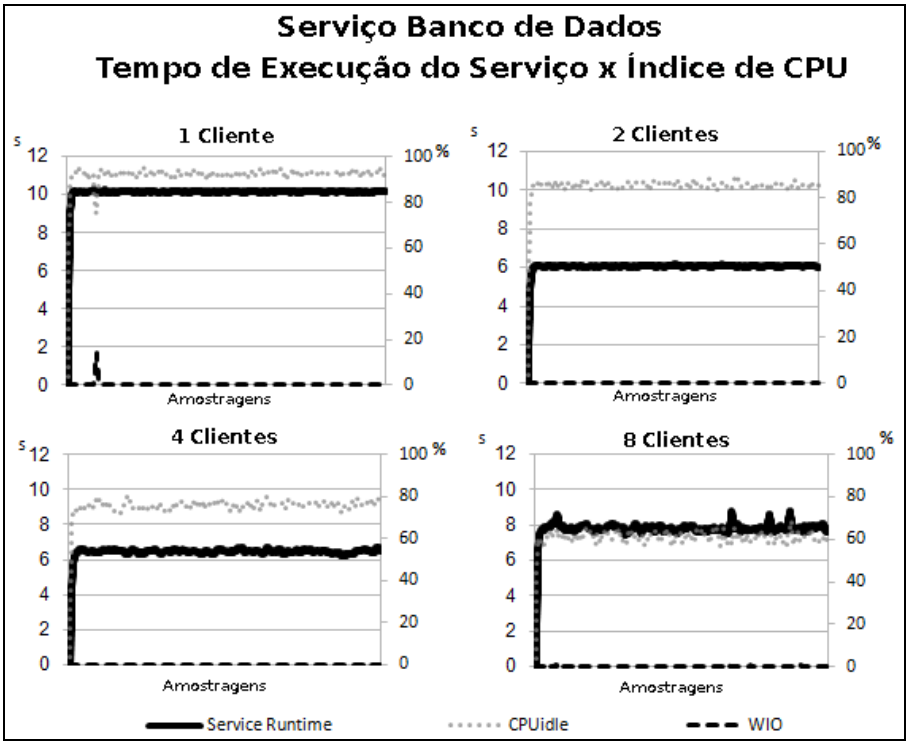

Figura 32: Resultados para os índices de *CPU*, considerando o serviço de Banco de Dados para 1, 2, 4 e 8 clientes.

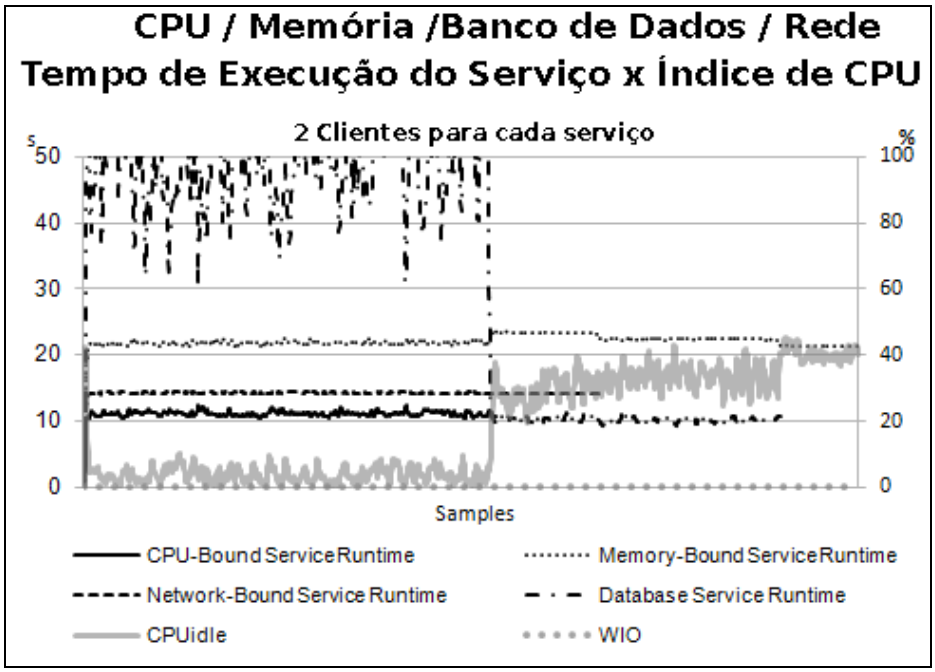

Figura 33: Resultados dos índices de *CPU*, considerando os quatro serviços executados simultaneamente.

Tabela 7: Índices de Memória

Serviços	Índices	1 Cliente		2 Clientes			4 Clientes			8 Clientes		
		Média	D.P.	Média	D.P.	%	Média	D.P.	%	Média	D.P.	%
CPU-Bound	*Free Mem*	2.8	0.0	2.7	0.0	*-1.4*	2.7	0.0	*0.5*	2.6	0.0	*-5.3*
	Swap Used	0.0	0.0	0.0	0.0	*0.0*	0.0	0.0	*0.0*	0.0	0.0	*0.0*
Memory-Bound	*Free Mem*	2.3	0.1	1.8	0.2	*-23.6*	0.7	0.3	*-59.4*	1.3	1.6	*81.7*
	Swap Used	0.0	0.0	0.0	0.0	*0.0*	0.0	0.0	*0.0*	0.7	0.0	*0.0*
Network-Bound	*Free Mem*	2.8	0.0	2.8	0.0	*0.4*	2.6	0.1	*-6.8*	2.4	0.0	*-8.4*
	Swap Used	0.0	0.0	0.0	0.0	*0.0*	0.0	0.0	*0.0*	0.0	0.0	*0.0*
Database-Access	*Free Mem*	2.5	0.1	2.8	0.0	*11.8*	2.7	0.0	*-1.5*	2.7	0.0	*1.1*
	Swap Used	0.0	0.0	0.0	0.0	*0.0*	0.0	0.0	*0.0*	0.0	0.0	*0.0*

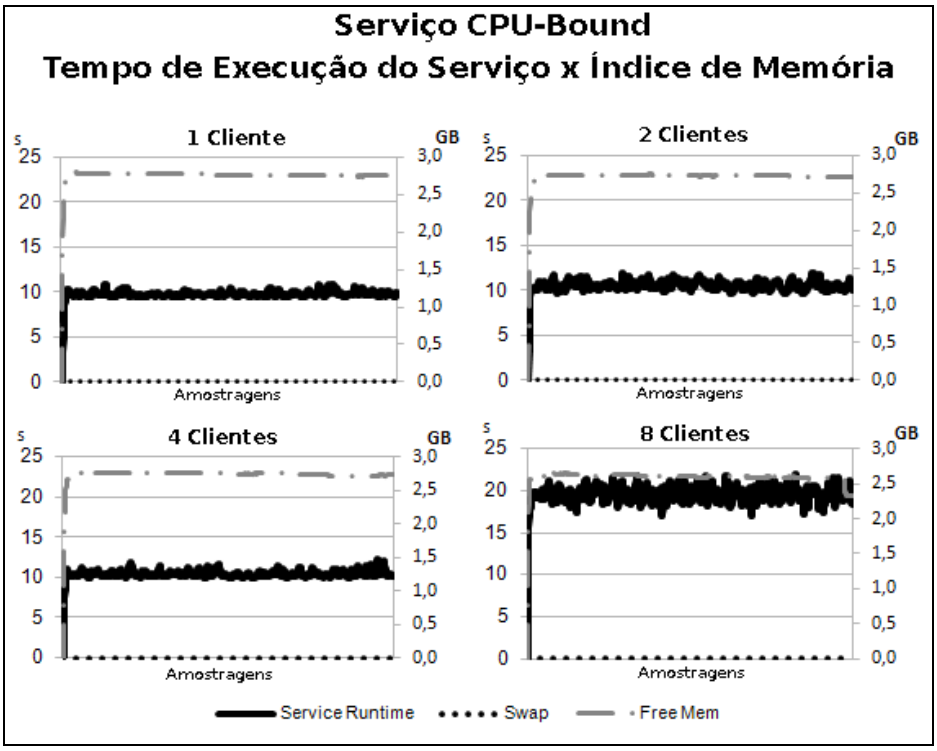

Figura 34: Resultados para os índices de memória, considerando o serviço CPU-Bound para 1, 2, 4 e 8 clientes.

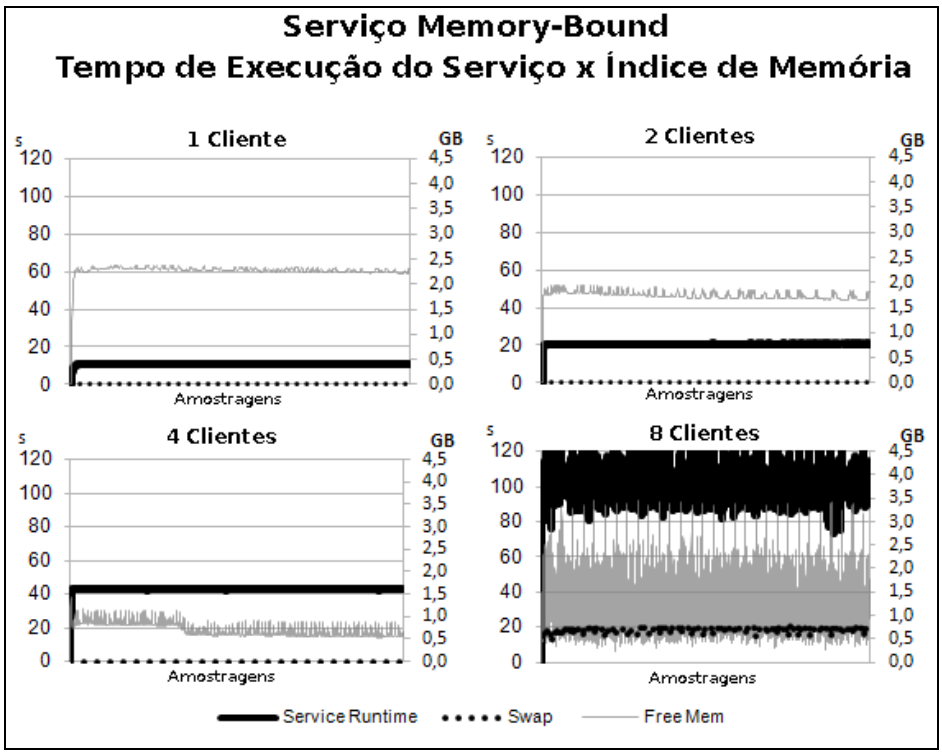

Figura 35: Resultados para os índices de memória, considerando o serviço *Memory-bound* para 1, 2, 4 e 8 clientes.

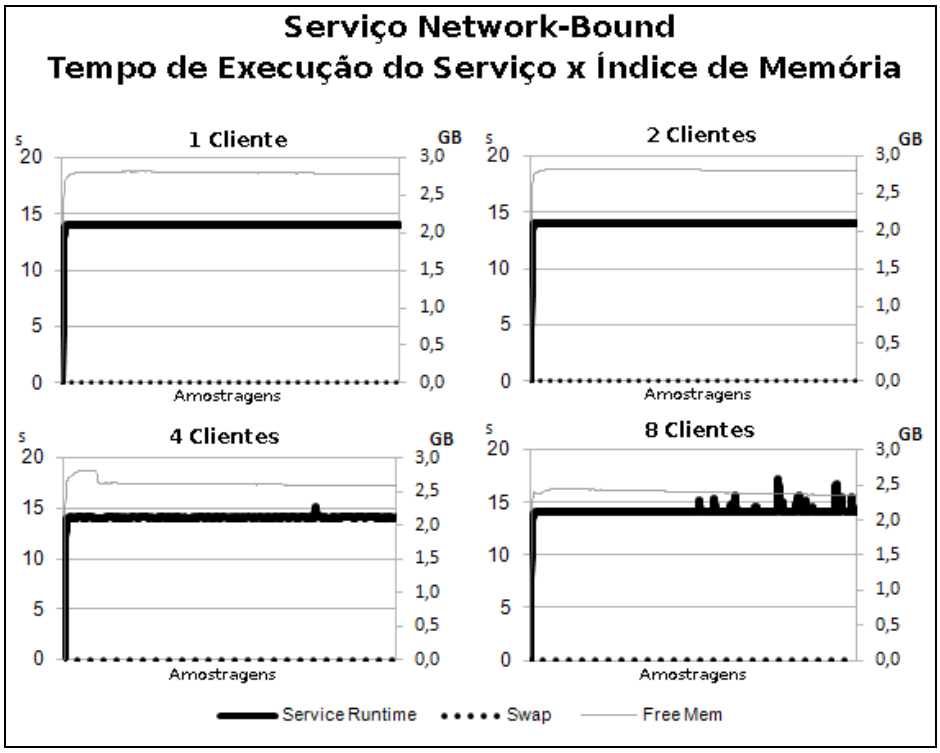

Figura 36: Resultados para os índices de memória, considerando o serviço *network-bound* para 1, 2, 4 e 8 clientes.

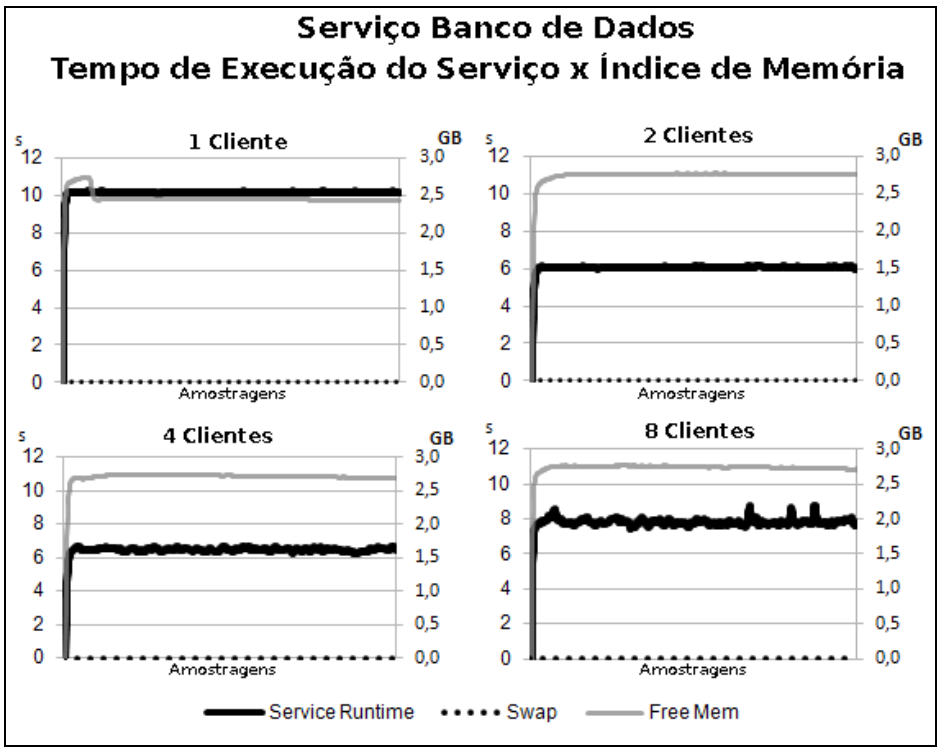

Figura 37: Resultados para os índices de memória, considerando o serviço de Banco de Dados para 1, 2, 4 e 8 clientes.

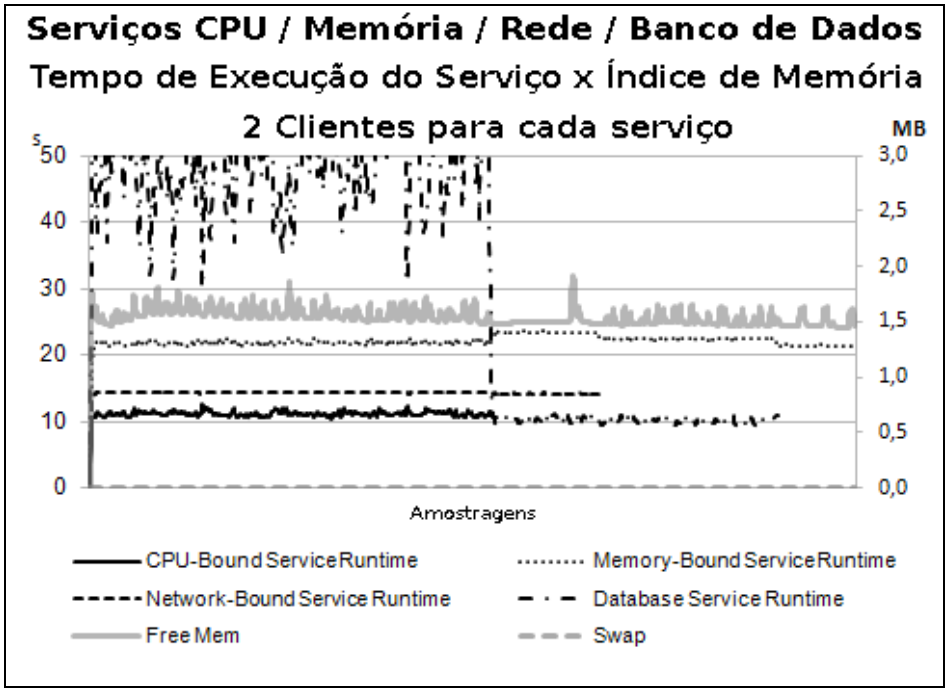

Figura 38: Resultados para os índices de memória considerando os quatro serviços em execução

Tabela 8: Índice *Saturation*

Serviços	1 Cliente		2 Clientes			4 Clientes			8 Clientes		
	Média	**D.P.**	**Média**	**D.P.**	*%*	**Média**	**D.P.**	*%*	**Média**	**D.P.**	*%*
CPU-Bound	2.2	0.0	2.3	0.1	*5.7*	2.3	0.1	*-1.9*	2.4	0.1	*4.2*
Mem-Bound	3.6	0.3	4.9	0.8	*37.0*	5.9	1.5	*20.1*	3.5	6.6	*-40.1*
Net-Bound	2.1	0.1	2.1	0.1	*-2.4*	1.7	0.2	*-16.6*	1.9	0.0	*12.9*
Bando de Dados	1.9	0.2	2.3	0.1	*19.6*	2.4	0.1	*5.3*	2.3	0.1	*-5.5*

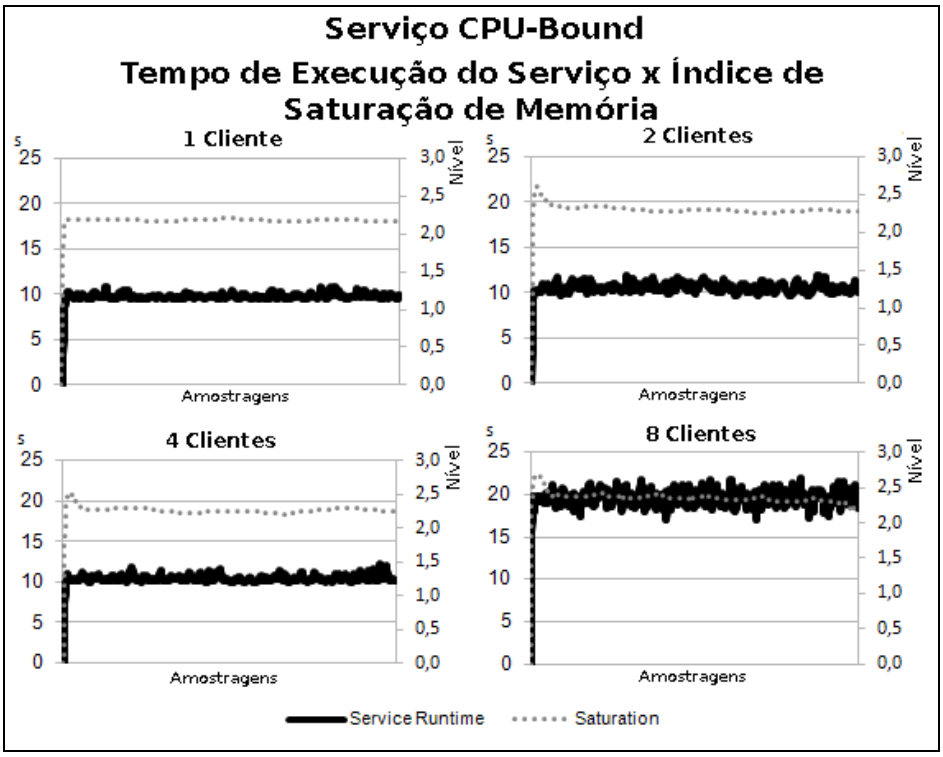

Figura 39: Resultados para o índice de saturação do sistema de memória, considerando o serviço *CPU-bound* para os 1, 2, 4 e 8 clientes.

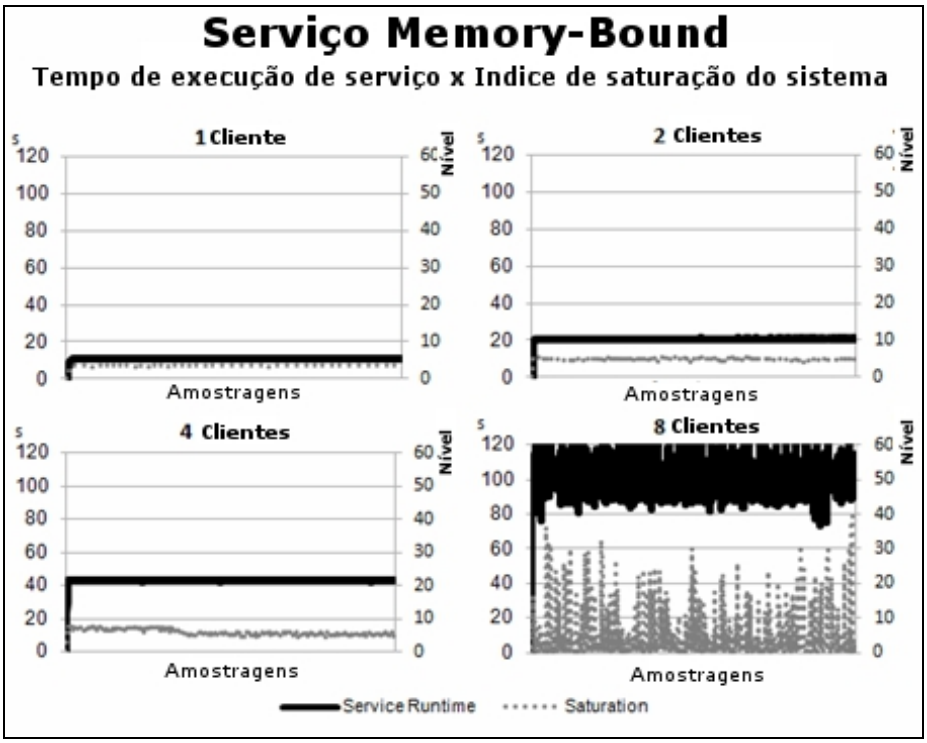

Figura 40: Resultados para o índice de saturação do sistema considerando o serviço de *memory-bound* para 1, 2, 4 e 8 clientes.

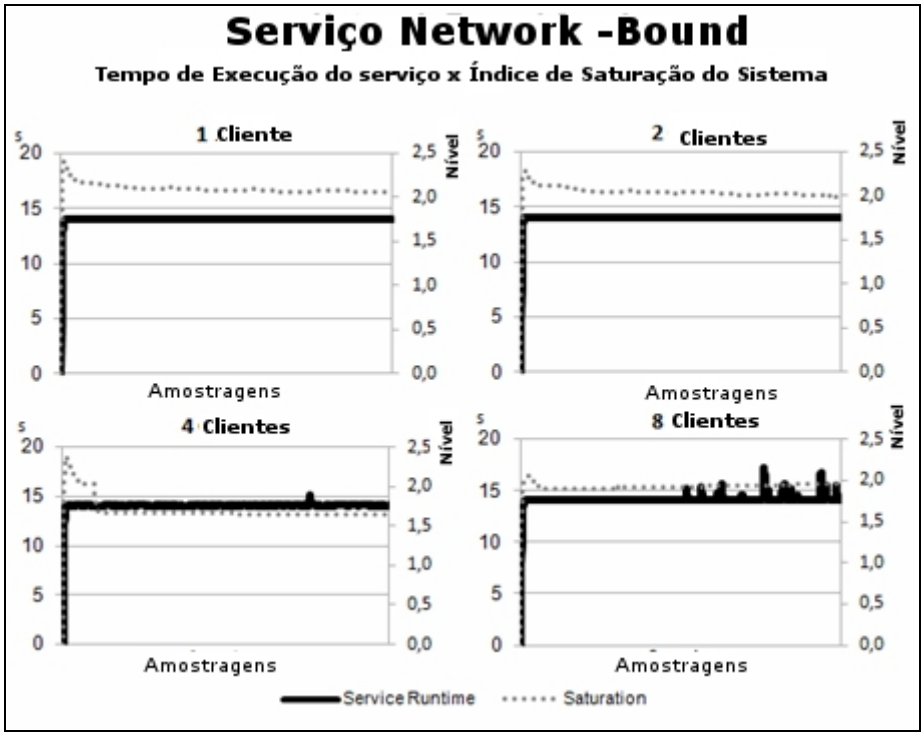

Figura 41: Resultados para o índice de saturação do sistema, considerando o serviço de *Network-Bound* para 1, 2, 4 e 8 clientes.

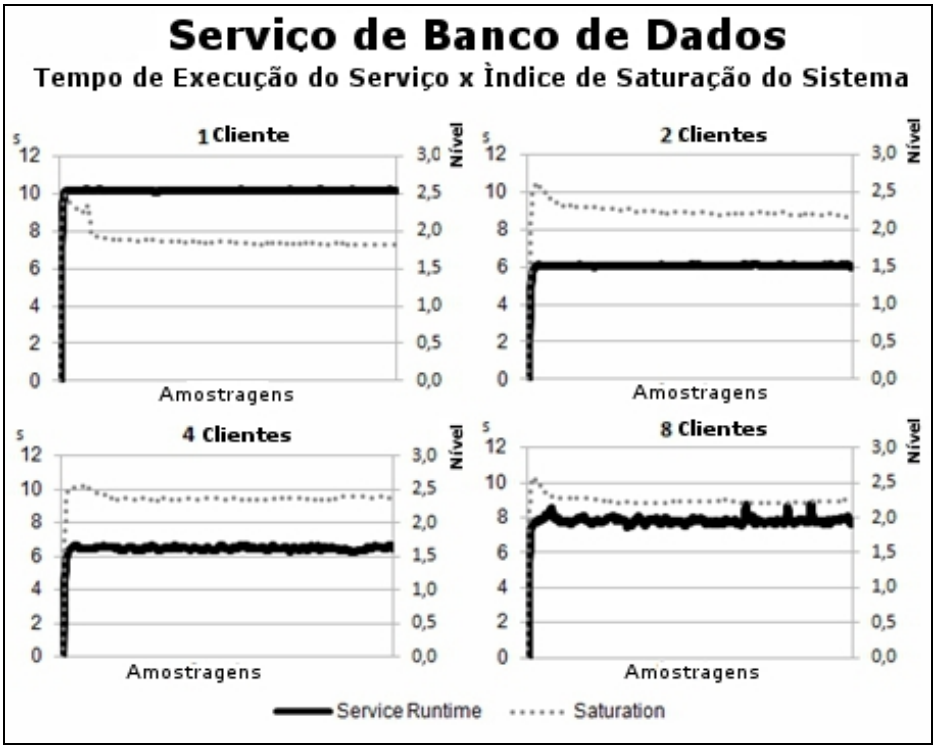

Figura 42: Resultados para o índice de saturação do sistema considerando o serviço de banco de dados para 1, 2, 4 e 8 clientes.

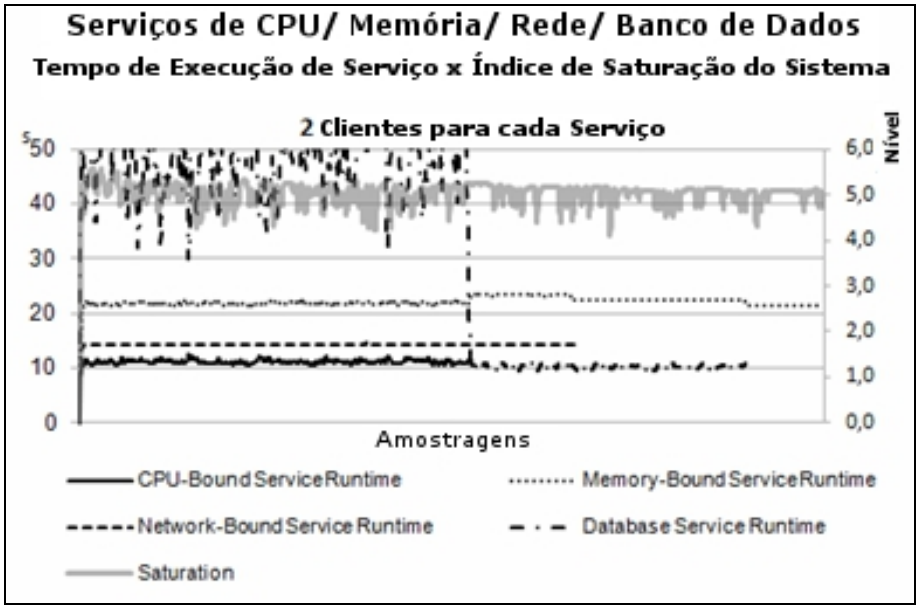

Figura 43: Resultados para o índice de Saturação do Sistema considerando todos os serviços para 1, 2, 4 e 8 clientes.

Tabela 9: Índices de Rede

Serviço	Índices	1 Cliente		2 Clientes			4 Clientes			8 Clientes		
		Média	D.P.	Média	D.P.	%	Média	D.P.	%	Média	D.P.	%
CPU-Bound	*Bytes In*	0.2	0.2	0.3	0.2	21.1	1.8	0.5	556.9	0.7	0.7	-59.6
	Bytes Out	0.0016	0.0	1.3	0.6	82479.5	1.3	0.6	-3.3	5.0	1.6	285.9
Memory-Bound	*Bytes In*	0.2	0.2	0.2	0.2	0.0	0.2	0.3	0.0	0.2	0.4	-5.6
	Bytes Out	1.3	0.6	1.2	0.6	-4.7	1.0	0.6	-18.9	0.6	0.6	-37.4
Network-Bound	*Bytes In*	0.9	0.5	1.6	0.8	81.1	3.1	1.6	90.2	5.0	2.9	61.0
	Bytes Out	0.9	0.5	1.6	0.8	81.1	3.1	1.6	90.2	5.0	2.9	61.0
Banco de Dados	*Bytes In*	1.1	1.0	3.6	1.9	221.2	6.7	3.7	84.3	10.6	6.6	58.0
	Bytes Out	1.4	0.3	4.4	1.2	219.6	8.2	2.9	86.8	13.1	7.8	58.6

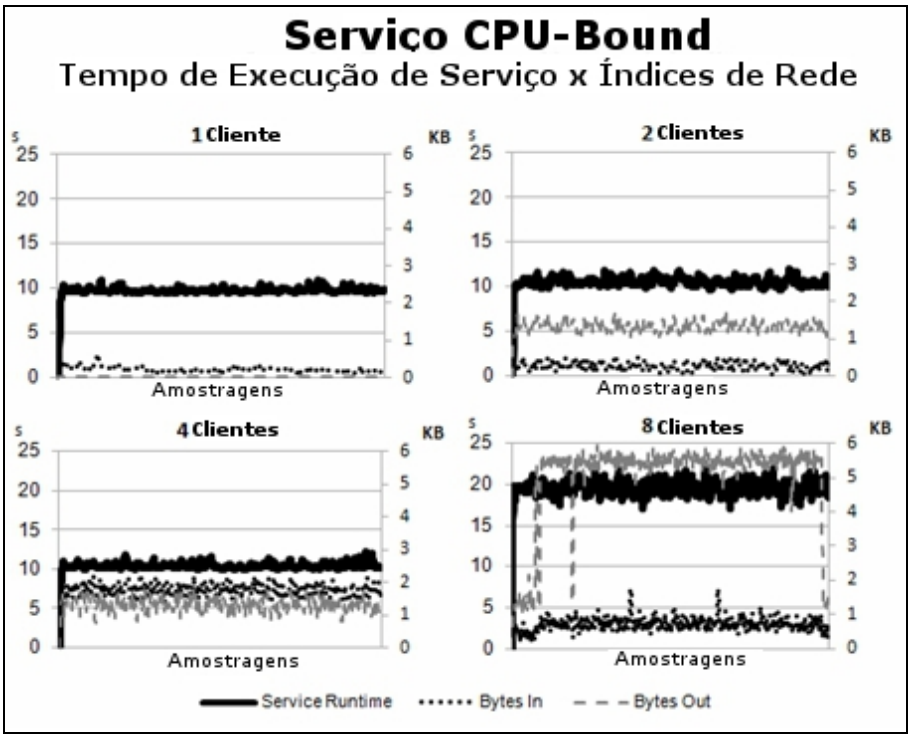

Figura 44: Resultados para os índices de rede considerando o serviço *CPU-bound* para 1, 2, 4 e 8 clientes.

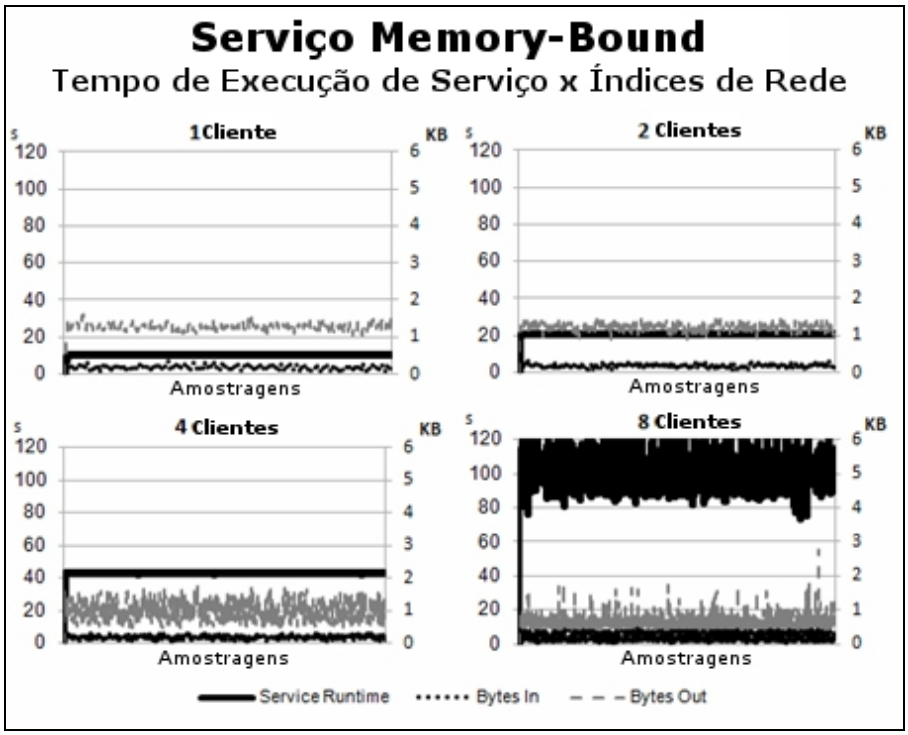

Figura 45: Resultados para os índices de rede considerando o serviço de *memory-bound* para 1, 2, 4 e 8 clientes.

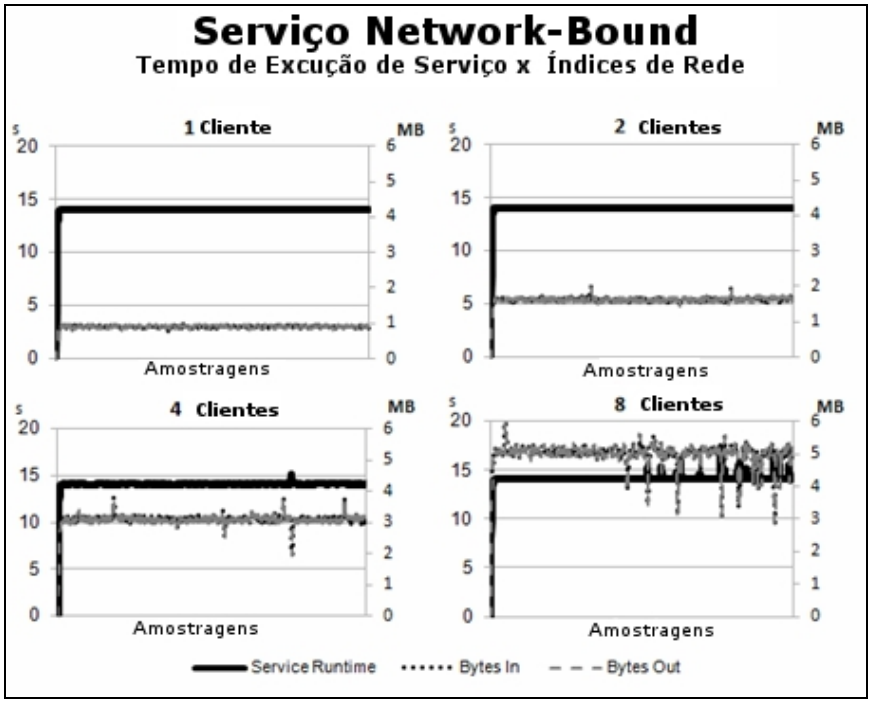

Figura 46: Resultados para os índices de rede considerando o serviço de *Network-bound* para 1, 2, 4 e 8 clientes.

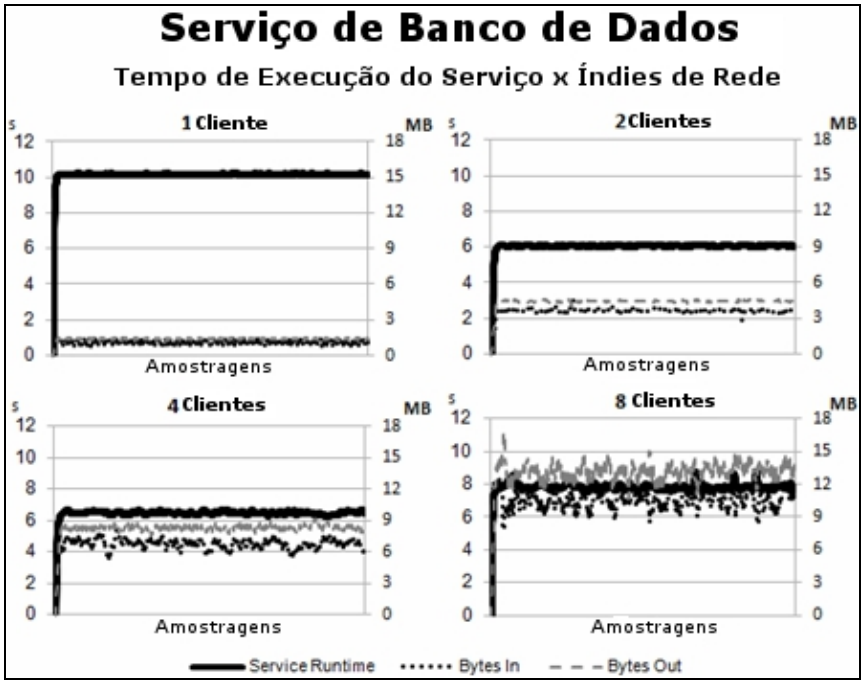

Figura 47: Resultados para os índices de rede considerando o serviço de banco de dados para 1, 2, 4 e 8 clientes.

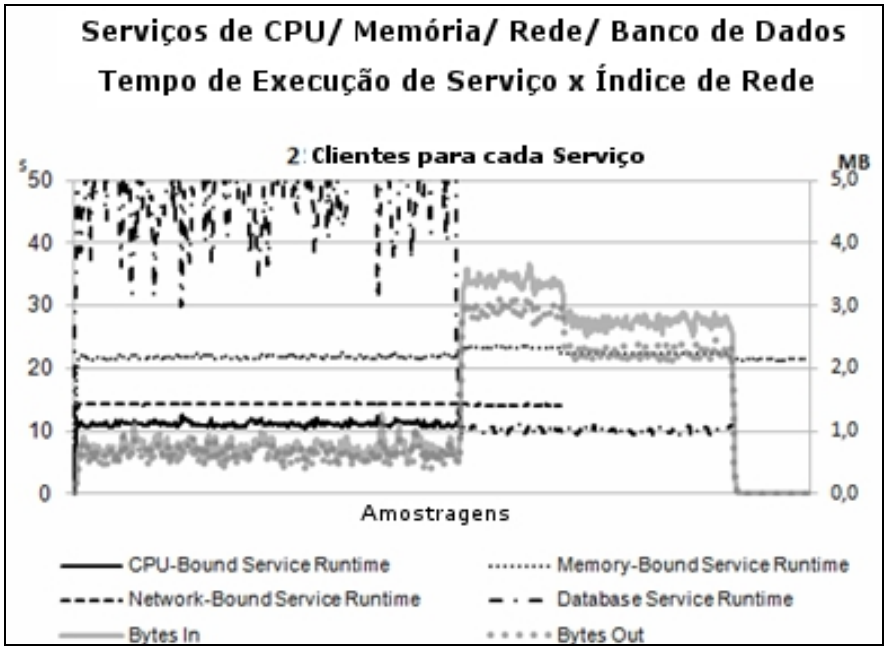

Figura 48: Resultados para os índices de rede, considerando todos os serviços para 1, 2, 4 e 8 clientes.

Tabela 10: Índices de *Ready-process*

Serviço	1 Cliente		2 Clientes			4 Clientes			8 Clientes		
	Média	D.P.	Média	D.P.	%	Média	D.P.	%	Média	D.P.	%
CPU-Bound	1,1103	0,1626	2,0937	0,2257	*88,5675*	5,6196	0,5651	*168,3990*	9,2855	0,7302	*65,2352*
Memory-Bound	1,1000	0,1200	2,3000	0,1900	*109,0909*	6,1700	0,3900	*168,2609*	11,0500	1,6100	*79,0924*
Network-Bound	0,0500	0,0600	0,0600	0,0800	*20,0000*	0,1131	0,1064	*83,3333*	0,1500	0,1100	*36,3636*
Banco de Dados	0,1400	0,1100	0,2200	0,1200	*57,1429*	0,5200	0,2300	*136,3636*	0,9100	0,3000	*75,0000*

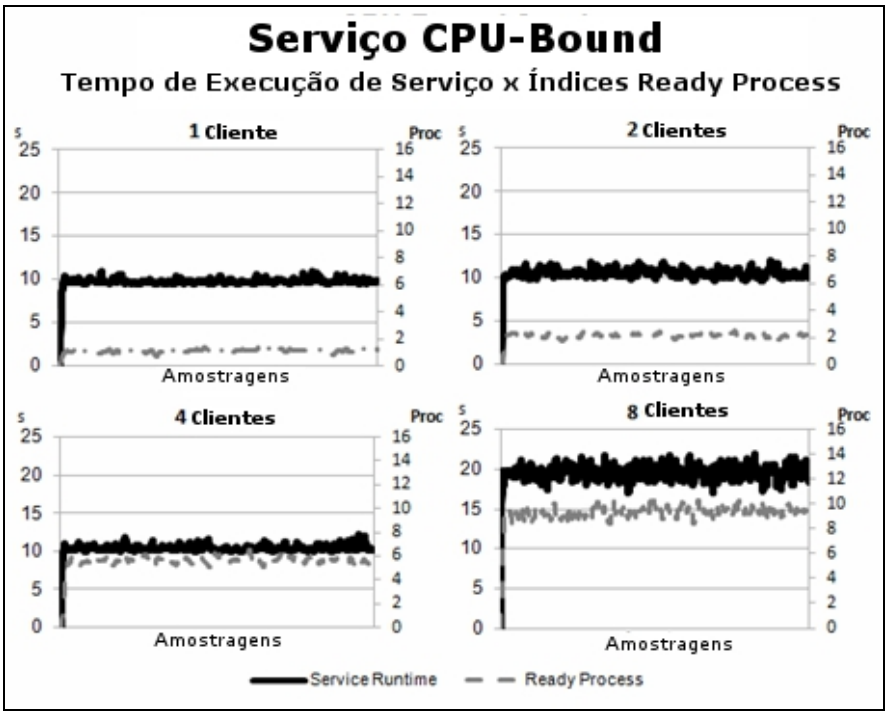

Figura 49: Resultados para o índice de *Ready Process* considerando o serviço *CPU-Bound* para 1, 2, 4 e 8 clientes.

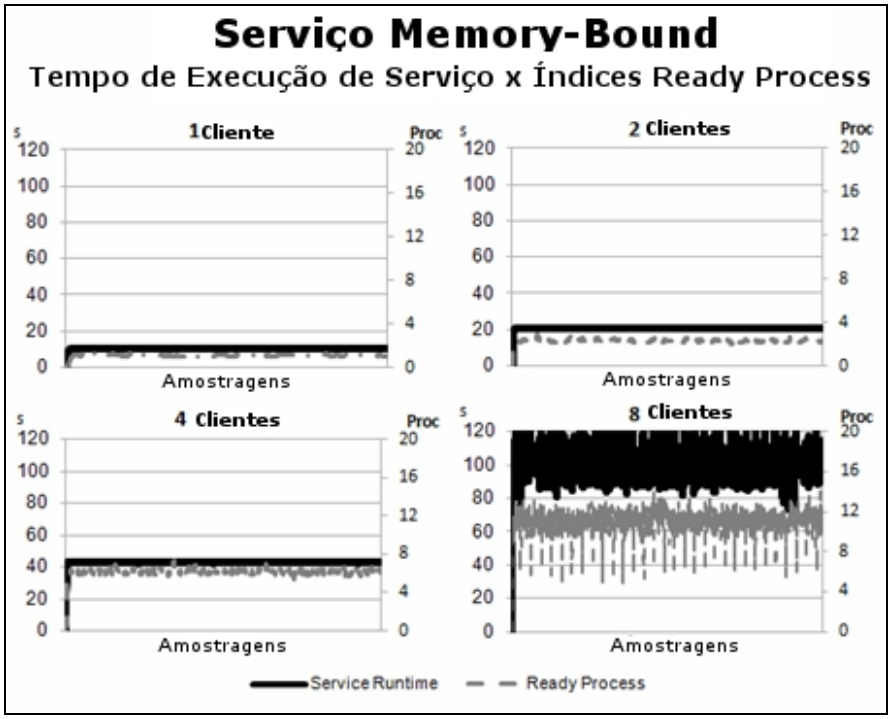

Figura 50: Resultados para o índice de *Ready Process* considerando o serviço *memory-bound* para 1, 2, 4 e 8 clientes.

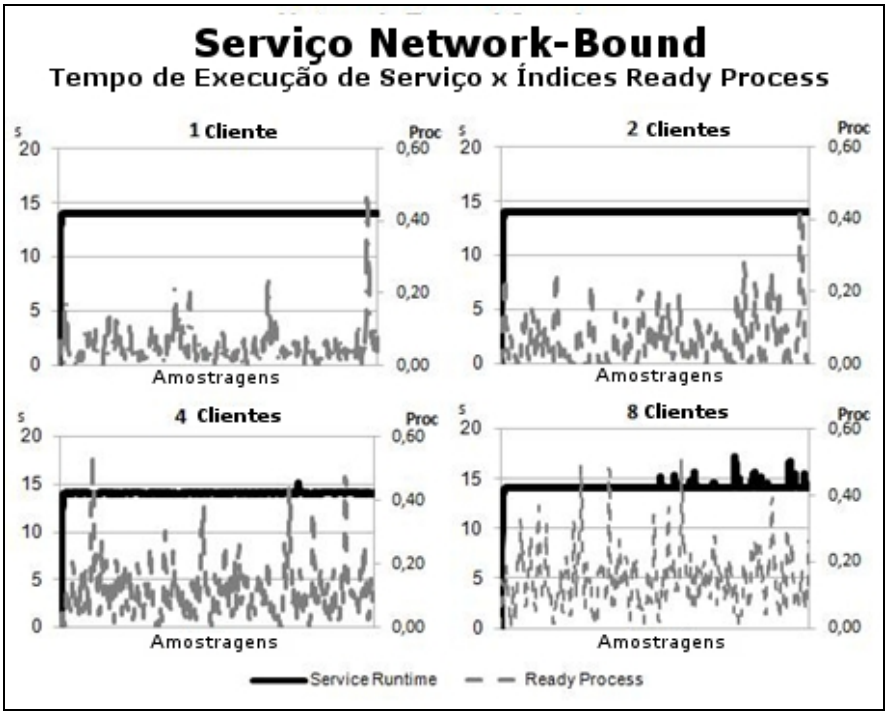

Figura 51: Resultados para o índice de *Ready Process* considerando o Serviço *Network-Bound* para 1, 2, 4 e 8 clientes.

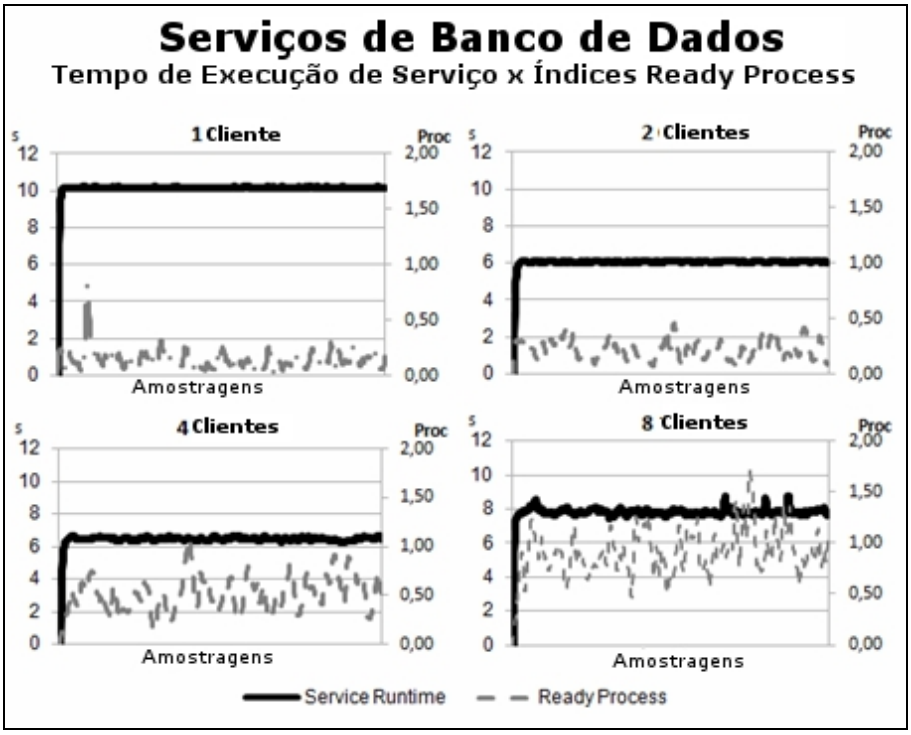

Figura 52: Resultados para o índice de *Ready Process* considerando o serviço de banco de dados para 1, 2, 4 e 8 clientes.

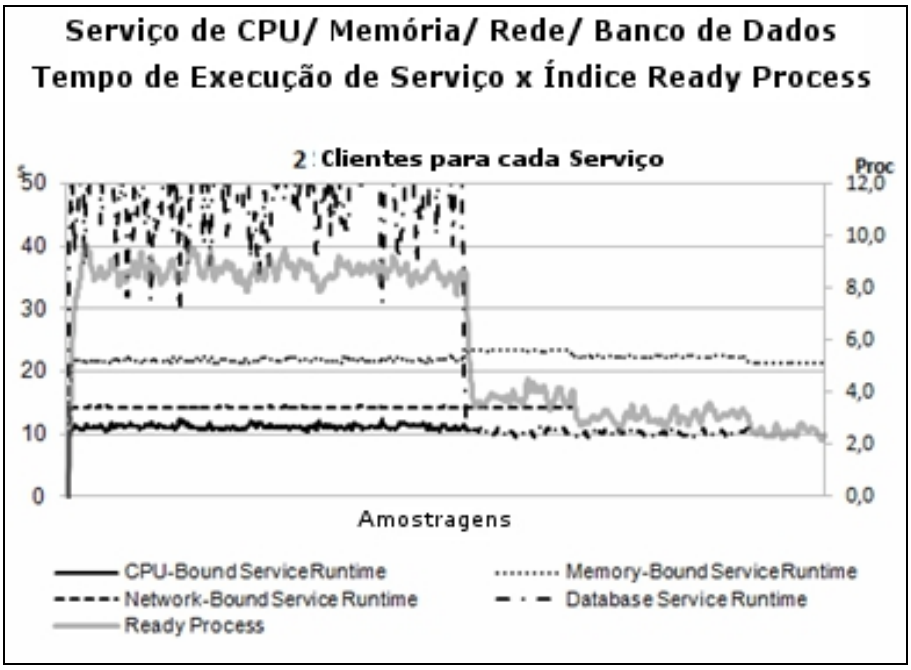

Figura 53: Resultados para o Índice de *Ready Process* considerando todos os serviços executados

Tabela 11: Índice Amount Submitted-requests

Serviços	1 Cliente		2 Clientes			4 Clientes			8 Clientes		
	Média	D.P.	Média	D.P.	*%*	Média	D.P.	*%*	Média	D.P.	*%*
CPU-Bound	1.2	0.2	2.1	0.3	*83.2*	3.9	0.6	*85.1*	4.3	2.1	*10.2*
Memory-Bound	1.1	0.2	1.1	0.2	*4.6*	1.0	0.4	*0.0*	0.9	1.1	*-11.3*
Network-Bound	0.8	0.1	1.6	0.3	*100.0*	2.9	0.5	*0.0*	6.1	1.0	*112.8*
Banco de Dados	1.1	0.2	3.6	0.5	*217.7*	6.0	0.8	*0.1*	10.5	1.4	*76.2*

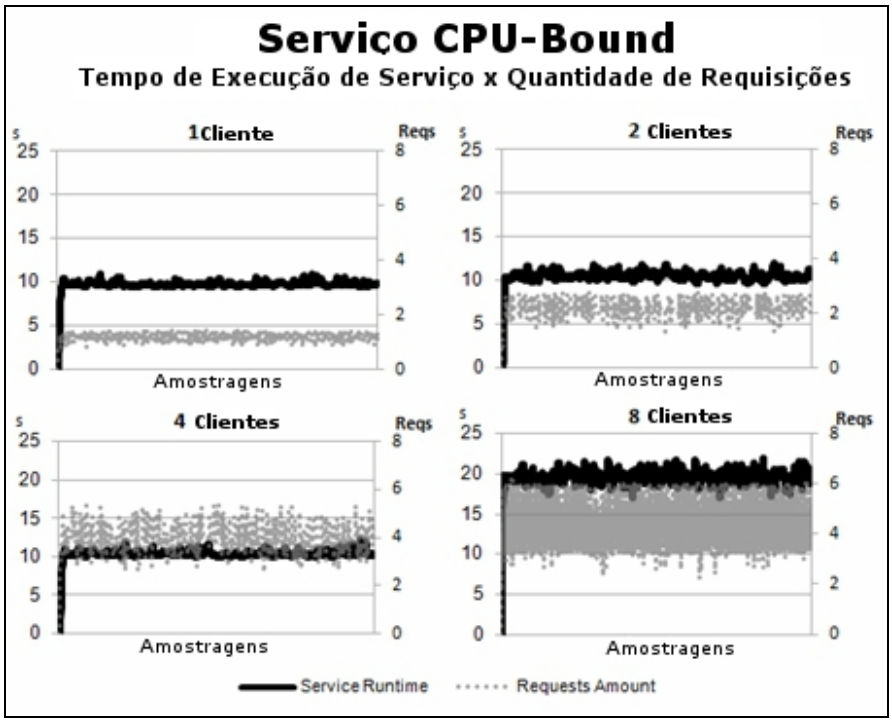

Figura 54: Resultados para o índice de quantidade de requisições considerando o serviço *CPU-Bound* para 1, 2, 4 e 8 clientes.

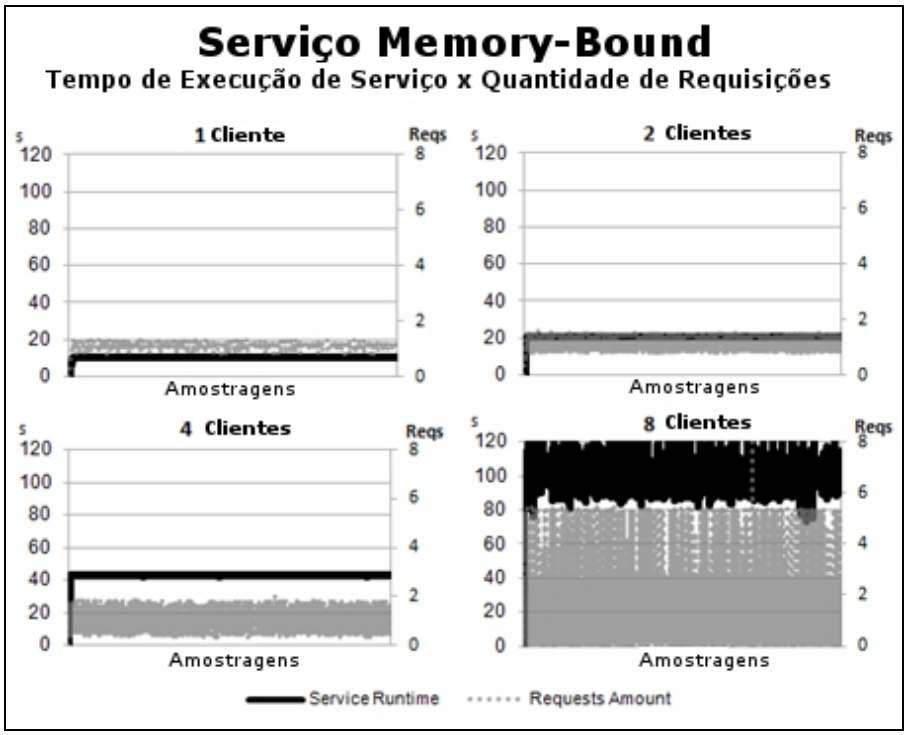

Figura 55: Resultados para o índice quantidade de requisições considerando o serviço de *memory-bound* para 1, 2, 4 e 8 clientes.

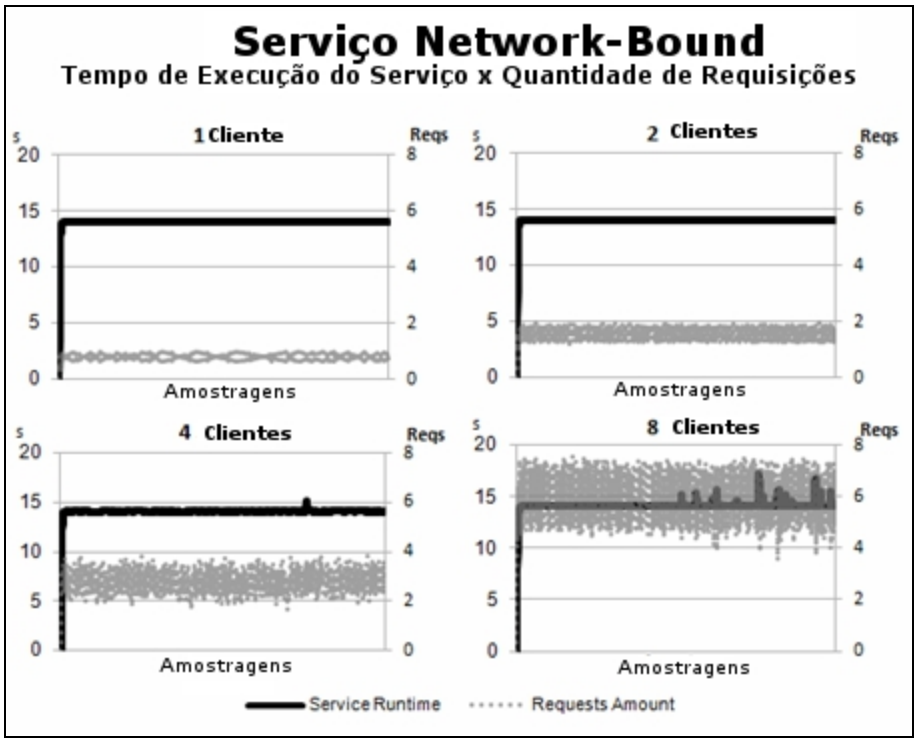

Figura 56: Resultados para o índice quantidade de requisições considerando o serviço de *Network-Bound* para 1, 2, 4 e 8 clientes.

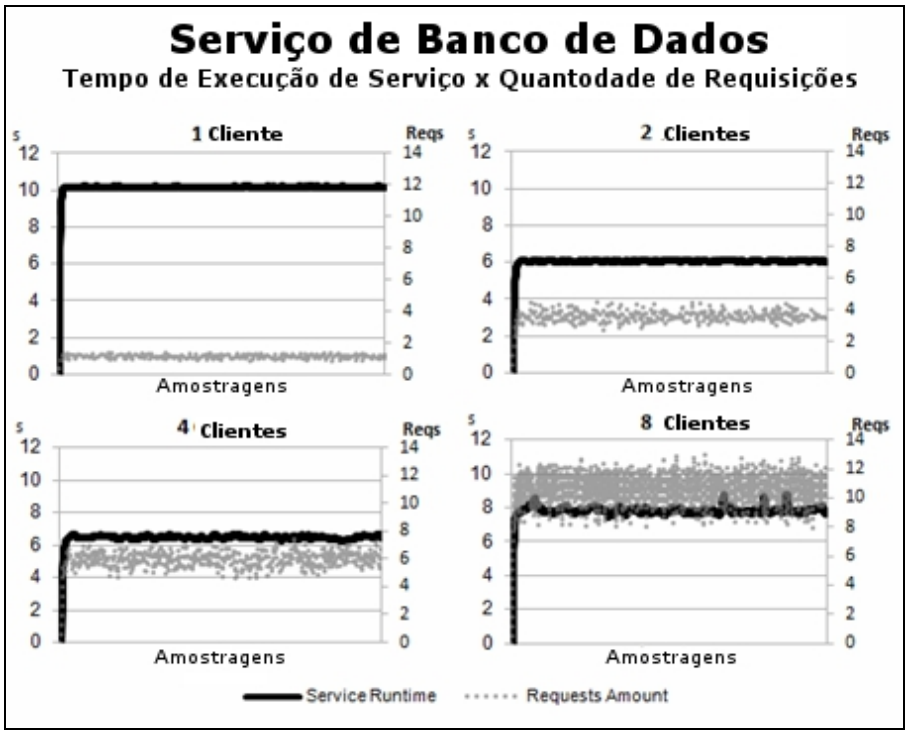

Figura 57: Resultados para o índice quantidade de requisições considerando o serviço de banco de dados para 1, 2, 4 e 8 clientes.

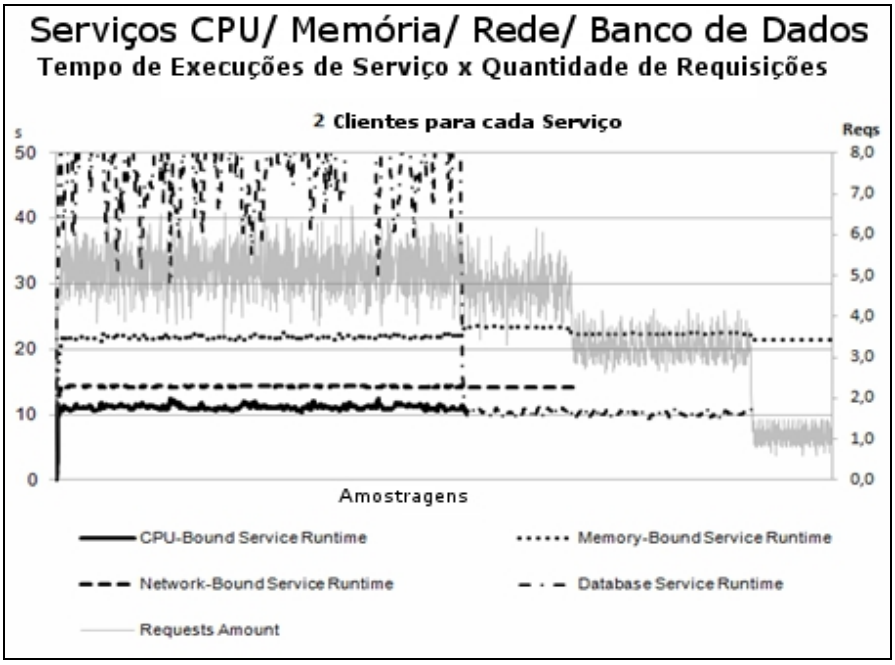

Figura 58: Resultados para o índice quantidade de requisições considerando todos os serviços em execução.

Tabela 12: Índices baseados em tempo de execução do serviço

Serviços	Índices	1 Cliente		2 Clientes			4 Clientes			8 Clientes		
		Média	D.P.	Média	D.P.	%	Média	D.P.	%	Média	D.P.	%
CPU-Bound	*Average-JMX_Time*	9.8	0.6	10.2	0.6	*3.9*	10.8	0.8	*5.8*	19.0	0.1	*76.6*
	Client Resp-Time	10.1	1.0	11.0	1.5	*8.6*	10.9	1.0	*-0.4*	20.3	2.1	*85.9*
Memory-Bound	*Average-JMX_Time*	10.5	0.7	20.5	1.0	*94.3*	47.3	2.3	*131.2*	104.1	5.6	*120.1*
	Client Resp-Time	11.0	0.1	21.1	0.1	*93.0*	43.6	0.3	*106.4*	112.6	16.6	*158.2*
Network-Bound	*Average-JMX_Time*	14.0	0.8	13.8	0.8	*-1.2*	15.2	0.9	*10.1*	14.2	0.8	*-6.9*
	Client Resp-Time	14.4	0.1	14.5	0.1	*0.8*	14.6	0.2	*0.8*	14.7	0.9	*0.9*
Banco de Dados	*Average-JMX_Time*	10.2	0.8	6.0	0.5	*-40.6*	7.0	0.6	*16.6*	7.5	0.6	*5.8*
	Client Resp-Time	10.5	0.1	6.5	0.1	*-38.0*	7.0	0.2	*7.2*	8.5	0.5	*21.4*

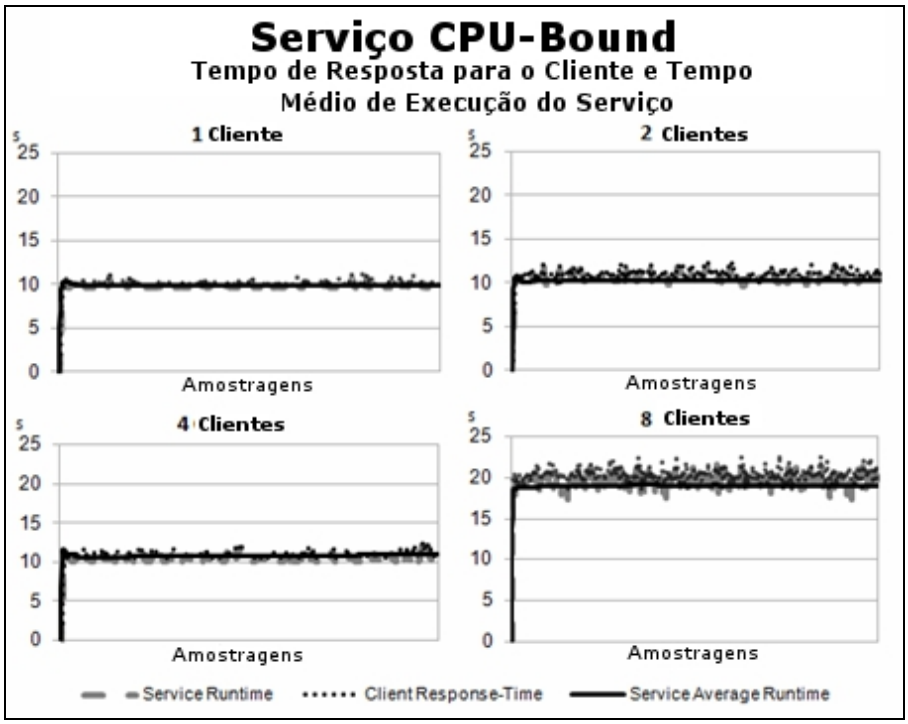

Figura 59: Resultados para os índices baseados em tempo considerando o serviço *CPU-Bound* para 1, 2, 4 e 8 clientes.

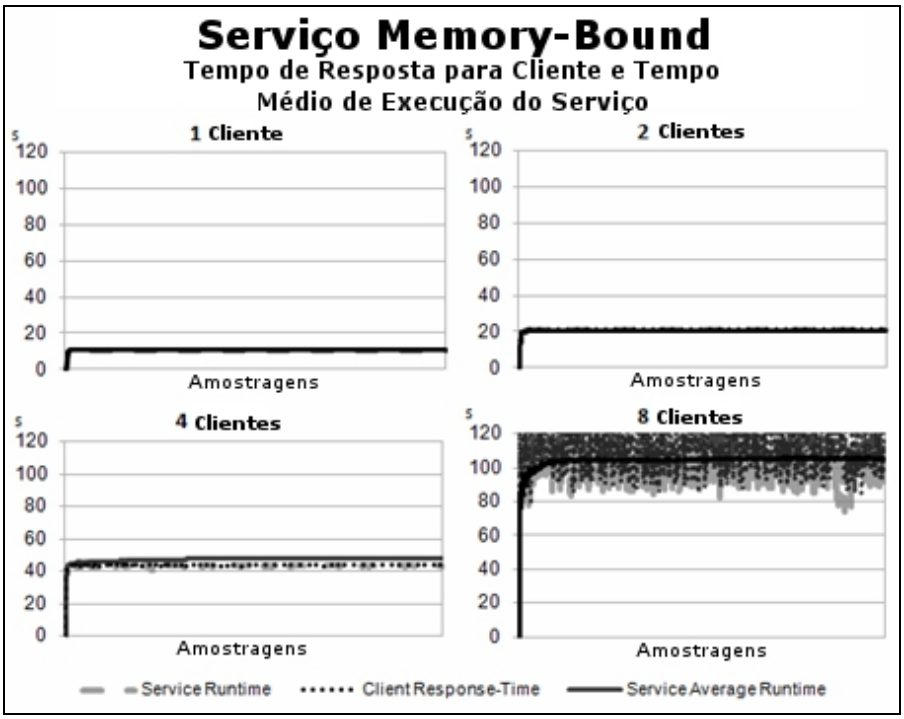

Figura 60: Resultados para os índices baseados em tempo considerando o serviço de *memory-bound* para 1, 2, 4 e 8 clientes.

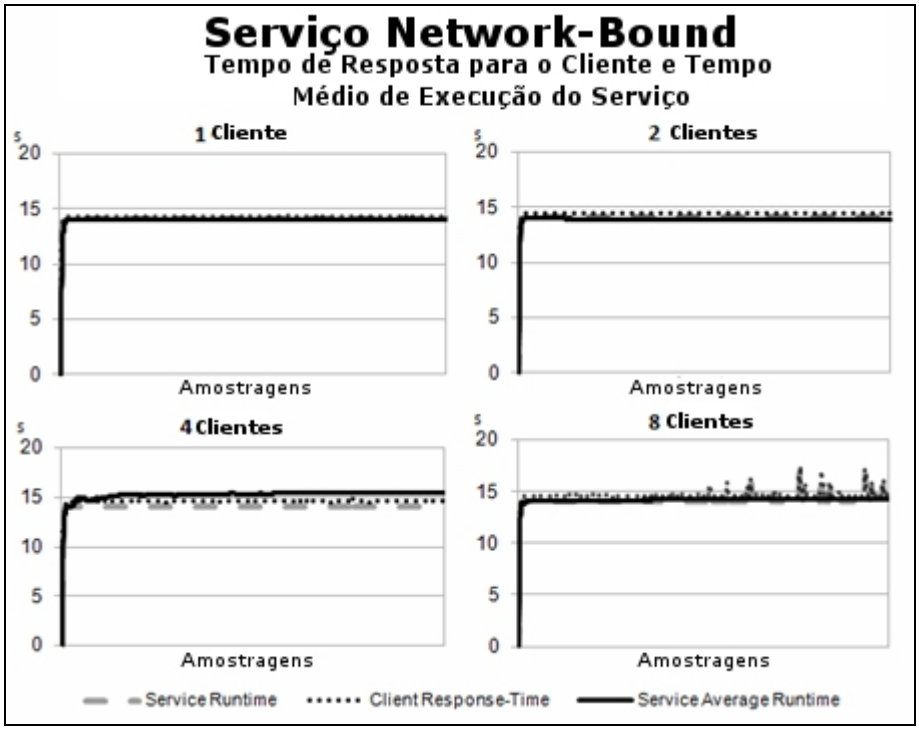

Figura 61: Resultados para os índices baseados em tempo considerando o serviço *Network-Bound* para 1, 2, 4 e 8 clientes.

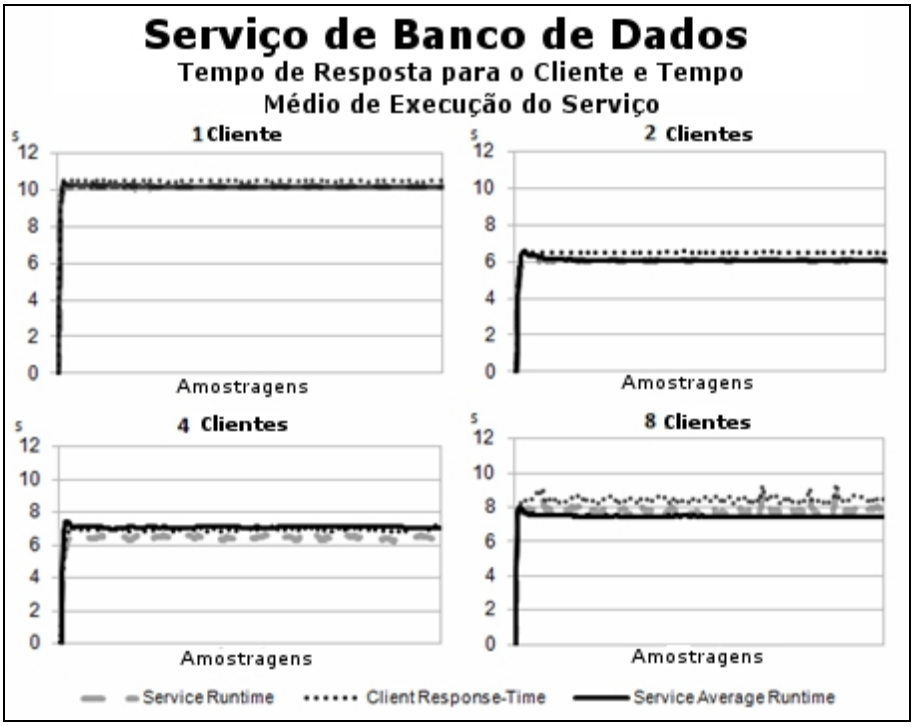

Figura 62: Resultados para os índices baseados em tempo considerando o serviço de banco de dados para 1, 2, 4 e 8 clientes.

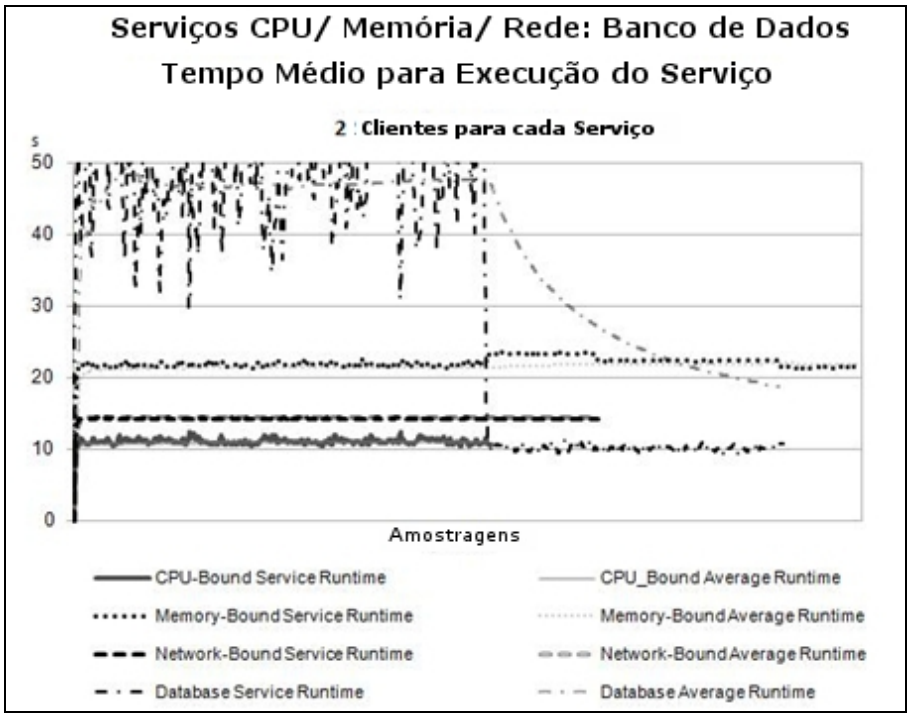

Figura 63: Resultados para os índices baseados em tempo considerando todos os serviços em execução.