UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANDRÉ PANISSON

Aplicação de Técnicas de Distribuição de Carga em Sistemas de Gerenciamento de Redes Baseados em P2P

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Profa. Dra. Maria Janilce B. Almeida
Orientadora

Porto Alegre, julho de 2007

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Panisson, André

Aplicação de Técnicas de Distribuição de Carga em Sistemas de Gerenciamento de Redes Baseados em P2P / André Panisson. – Porto Alegre: PPGC da UFRGS, 2007.

81 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2007. Orientadora: Maria Janilce B. Almeida.

1. Sistemas peer-to-peer. 2. Gerenciamento de redes de computadores. 3. Distribuição de carga. I. Almeida, Maria Janilce B. II. Título.

“Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution.”

— ALBERT EINSTEIN

AGRADECIMENTOS

Ao findar mais esta etapa, não posso deixar de prestar o meu sincero agradecimento a todas as pessoas que me acompanharam e apoiaram no decorrer destes últimos anos.

Em primeiro lugar, gostaria de agradecer à minha família. Aos meus pais Alfeu e Inês Panisson, pelo exemplo de vida e pela família maravilhosa que conseguiram construir, apesar de todas as dificuldades. Aos meus irmãos Renato, César, Jonas e Gelson, pelo apoio dado não só nesta etapa, mas em todas as outras já realizadas. Vocês são muito especiais para mim e agradeço a vocês de coração!

Agradeço a Deus por ter me acompanhado em mais uma etapa da minha vida, por ter colocado tantas pessoas importantes em meu caminho e por ter me dado força e determinação para seguir em frente.

Agradeço à minha orientadora, Profa. Maria Janilce Bosquiroli Almeida, por ter me auxiliado no desenvolvimento deste trabalho e pelo acompanhamento dado durante o mestrado. Ao Prof. Lisandro Zambenedetti Granville, o meu agradecimento e admiração, pelo desenvolvimento deste trabalho e pelo prestativo apoio dado durante o mestrado. Gostaria de agradecer também aos amigos e professores do grupo de Redes de Computadores, Juergen Rochol e a professora Liane Margarida Rockenbach Tarouco, pelos ensinamentos, auxílio e competência nas aulas ministradas.

Agradecimentos aos meus colegas de mestrado Ricardo Vianna, Rodrigo Sanger, Clarissa Marquezan, Diego Moreira da Rosa e Cristina Melchiors. Um agradecimento especial ao Diego e à Cristina pelo desenvolvimento dos artigos internacionais, e à Clarissa pelo auxílio dado no desenvolvimento deste trabalho.

Agradeço aos meus amigos colegas de apartamento, Alex e Roger, pelo incentivo e amizade, ao Gustavo Brauner, meu orientador em assuntos relacionados à língua inglesa, pelo auxílio na preparação das apresentações feitas em congresso internacional. Aos grandes amigos *Avengers* - Alex, Brauner, Gregory, Júlio, Luciano, Roger, Tomate, aos amigos de Porto Alegre, aos meus amigos de Vacaria, de São Lourenço do Sul, e tantos mais que já estão espalhados pelo mundo, obrigado pelo incentivo e amizade.

Gostaria de agradecer também aos demais colegas, professores e funcionários do Instituto de Informática pela prestatividade e ajuda dispensada.

Enfim, a todos que de forma direta ou indireta me apoiaram, auxiliaram e contribuíram não só na realização desta dissertação, mas em todas as atividades realizadas durante o mestrado, meus sinceros agradecimentos. Muito obrigado a todos!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
Axis	Apache eXtensible Interaction System
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DHT	Dynamic Hash Table
DOM	Document Object Model
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
JAX-RPC	Java API for XML-Based RPC
JMX	Java Management Extension
MbD	Management by Delegation
MIB	Management Information Base
MLM	Mid-Level Manager
NAT	Network Address Translation
OID	Object Identifier
P2P	Peer-to-Peer
QAME	QoS-Aware Management Environment
REST	Representational State Transfer
RFC	Request for Comments
RPC	Remote Procedure Call
SMI	Structure of Management Information

SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TLM	Top-Level Manager
UDDI	Universal Description, Discovery, and Integration
UDP	User Datagram Protocol
W3C	World Wide Web Consortium
WBEM	Web Based Enterprise Management
WS	Web Services
WSDL	Web Services Description Language
WSRF	Web Services Resource Framework
WSS	Web Services Security
XML	eXtensible Markup Language
XPath	XML Path Language
XQuery	XML Query Language

LISTA DE FIGURAS

RESUMO

Devido à evolução e à crescente complexidade dos sistemas computacionais, grandes mudanças ocorreram na área de gerenciamento de redes. Os modelos tradicionais centralizados se mostraram limitados, e novos modelos de gerenciamento de redes estão sendo propostos e investigados. Neste cenário, modelos de comunicação P2P permitem a construção de ambientes dinâmicos e versáteis capazes de resolver problemas de diferentes áreas da computação, entre elas o gerenciamento de redes. O modelo P2P torna-se atrativo porque se encaixa perfeitamente no gerenciamento de redes distribuído atualmente solicitado. A partir da proposta de gerenciamento de redes usando P2P e da apresentação de um novo modelo conceitual usando essa nova perspectiva, foi desenvolvido um projeto de arquiteturas para gerenciamento baseadas em P2P, focando-se em entidades tais como *peers* de gerenciamento *Top-Level* (TLM) e *Mid-Level* (MLM). A fim de efetivamente concretizar o gerenciamento baseado em P2P, foram introduzidas definições de blocos básicos para sistemas de gerenciamento baseados em P2P. A integração desses blocos básicos com soluções tradicionais de gerenciamento, tal como a estrutura SNMP, também foi apresentada. Um dos conceitos centrais da arquitetura apresentada são os Serviços de Gerenciamento, que definem as tarefas básicas de gerenciamento do sistema, e foram inspirados nos conceitos de *Web Services* e na concepção de serviços das Arquiteturas Orientadas a Serviço (SOA). Para a avaliação desse novo modelo de gerenciamento de redes e da arquitetura associada, surgiu a necessidade do desenvolvimento de um protótipo do sistema, com o intuito de servir como uma plataforma para desenvolvimento e implantação de Serviços de Gerenciamento. Esse protótipo é usado para avaliação e análise de testes e resultados associados. Para preservar as características de disponibilidade e escalabilidade que são inerentes dos sistemas P2P, definiu-se que os Serviços de Gerenciamento devem ser disponibilizados por Grupos de MLMs, organizados de forma a fazer a distribuição de tarefas de gerenciamento entre si. Diversos modelos de distribuição de carga foram investigados, foram realizadas avaliações e gráficos comparativos, e foram estabelecidas diretrizes sobre quando e como deve ser aplicado cada modelo, de modo a alcançar sempre o melhor rendimento e a maximização do aproveitamento dos recursos disponíveis ao sistema. A avaliação realizada consistiu na execução de diversas requisições com diferentes números de MLMs usando diferentes algoritmos de distribuição de carga, e na comparação entre a vazão, os tempos de resposta e o tráfego de rede gerados pelos diferentes casos de uso. A implementação desenvolvida neste trabalho demonstrou ser promissora, apresentando resultados bastante satisfatórios com relação aos parâmetros avaliados.

Palavras-chave: Sistemas peer-to-peer, gerenciamento de redes de computadores, distribuição de carga.

Load Distribution in a P2P-Based Network Management System

ABSTRACT

Due to the increasing complexity of computer systems, deep changes have occurred in network management in the past few years. Traditional centralized models have been proved limited, and new network management models are being proposed and investigated. In this scenery, P2P communication models allow for the creation of dynamic and versatile environments, able to solve problems in various computational areas, one of them being network management. The P2P model fits perfectly in the current demands of distributed network management. Starting from the proposed P2P-based network management model, a new P2P-based network management architecture has been developed, focusing on entities like Top-Level (TLM) and Mid-Level (MLM) management peers. With the purpose of making the P2P-based network management real, new building blocks for P2P-based network management systems have been defined. In the present work, an integration of these new building blocks with traditional network management solutions - like SNMP - is presented. One of the main concepts of the proposed architecture is on Management Services, inspired by Web Services and in the service definition of Service Oriented Architectures (SOA). To evaluate this new network management model and its associated architecture, it was necessary to develop a prototype system to be used as platform for development and deployment of Management Services. This prototype was used for evaluation, testing and result analysis. To hold the characteristics of availability and scalability that are inherent to the P2P systems, the Management Services had to be served by MLM groups, organized in a way to distribute the management tasks between them. Several load distribution models were investigated, with evaluation and comparative graphics fulfilled, and some guidelines were established as to when and how each model should be applied to reach the best performance and the maximization of the computational resources available to the system. The evaluation consisted in the execution of several requests in groups with distinct MLM numbers and using distinct load distribution algorithms. Then, the response time and network traffic generated by use cases were compared. We then showed that the developed implementation of our proposal is promising, and presents good results in relation with the evaluated parameters.

Keywords: Peer-to-Peer Systems, Computer Network Management, Load Distribution.

1 INTRODUÇÃO

Um dos objetivos da área de Gerenciamento de Redes é permitir o uso de mecanismos para se obter disponibilidade e desempenho nos serviços de comunicação das organizações, a fim de que a troca de informações entre os diversos dispositivos que participam do processo de comunicação seja feita de maneira confiável e rápida. O menosprezo ao gerenciamento de redes normalmente resulta em redes frágeis e perdas econômicas devido a falhas de comunicação.

Conforme os sistemas computacionais evoluíram, mudanças ocorreram na área de gerenciamento de redes. No princípio, os modelos centralizados do tipo gerente-agente (PARK; CHO, 1994) eram suficientes para abranger todo o processo de gerenciamento de uma organização. Porém, quando os sistemas passaram a ser interconectados e sua complexidade passou a aumentar muito, os modelos tradicionais se mostraram bastante limitados. Vários modelos de gerenciamento de redes foram propostos e investigados nos últimos anos, inclusive modelos mais sofisticados, usando agentes móveis (BIESZCZAD et al., 1998) e uso de inteligência artificial (MOUNTZIA; RODOSEK, 1999). Recentemente, com o uso de processos distribuídos através de organizações virtuais (OVs) (KATZY, 1998), o aumento do uso da Internet como rede de transporte e a adoção de modelos de computação distribuída como *grids*, cresceu sensivelmente a necessidade de um novo modelo de gerenciamento de redes fortemente distribuído. Neste cenário, modelos de gerenciamento tradicionais não são aplicáveis porque são concebidos para operar em ambientes restritos de um único domínio administrativo, enquanto no gerenciamento moderno é preciso que as operações sejam executadas entre domínios diferentes, onde as decisões de gerenciamento são tomadas de forma distribuída.

O modelo de comunicação *peer-to-peer* (P2P) (ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004), em contraste aos modelos tradicionais de gerenciamento, foi concebido sobre os protocolos Internet a fim de que opere como um sistema altamente distribuído. Embora tenham ganho popularidade através de ferramentas que permitem distribuição ilegal de arquivos, atualmente pode-se encontrar muitas aplicações P2P dedicadas a processamento distribuído, voz sobre IP (VoIP) e distribuição de conteúdo. Uma rede P2P é formada por um conjunto de *peers* e suas conexões lógicas. Apesar de as redes P2P operarem sobre redes físicas, elas formam redes sobrepostas que são independentes do endereçamento e roteamento próprios da Internet. Isso permite a construção de ambientes dinâmicos e versáteis capazes de resolver problemas de diferentes áreas da computação, entre elas gerenciamento de redes. O modelo P2P torna-se ainda mais interessante porque se encaixa perfeitamente no gerenciamento de redes distribuído atualmente solicitado.

A partir das experiências práticas na avaliação da proposta de gerenciamento de redes usando P2P e da apresentação de um novo modelo conceitual usando essa nova perspectiva (GRANVILLE et al., 2005), surgiu o projeto de arquiteturas para gerenciamento

baseadas em P2P. A fim de efetivamente concretizar o gerenciamento baseado em P2P, foram introduzidas definições de blocos básicos para sistemas de gerenciamento baseados em P2P. A integração de entidades de gerenciamento baseadas em P2P com soluções tradicionais de gerenciamento, tal como a estrutura SNMP (HARRINGTON; PRESUHN; WIJNEN, 2002), também foi investigada.

A partir desse novo modelo de gerenciamento de redes, foi definida uma nova arquitetura para gerenciamento distribuído de redes, que está sendo apresentada neste trabalho. Um dos conceitos centrais dessa arquitetura são os Serviços de Gerenciamento, que definem as tarefas básicas de gerenciamento do sistema, e foram inspirados nos *Web Services* (CURBERA et al., 2002) e na concepção de serviços das Arquiteturas Orientadas a Serviço (SOA) (MUKHI; KONURU; CURBERA, 2004), porém mantendo todas as características de sistemas P2P. Foram definidos também os passos de interação entre os elementos da arquitetura, desde o momento em que o TLM inicia a requisição de execução de um serviço até o momento em que o TLM recebe a resposta do MLM.

Para preservar as características de disponibilidade e escalabilidade que são inerentes dos sistemas P2P, constatou-se que os Serviços de Gerenciamento devem ser disponibilizados por Grupos de MLMs, organizados de forma a fazer a distribuição de tarefas de gerenciamento entre si. Diversos modelos de distribuição de carga foram investigados, desde modelos mais simples de compartilhamento de carga até algoritmos mais complexos de balanceamento de carga. Alguns algoritmos de distribuição de carga foram definidos, a fim de realizar avaliações e gráficos comparativos, além de estabelecer diretrizes sobre quando e como devem ser aplicados, de modo a alcançar sempre o melhor rendimento e a maximização do aproveitamento dos recursos disponíveis ao sistema.

Para a avaliação desse novo modelo de gerenciamento de redes e da arquitetura associada, surgiu a necessidade do desenvolvimento de um protótipo do sistema, com o intuito de servir como uma plataforma para desenvolvimento e implantação de Serviços de Gerenciamento. Esse protótipo é usado para avaliação e análise de testes e resultados associados. A implementação do protótipo, juntamente com seus detalhes, também são apresentados neste trabalho. Foi definida uma Interface de Programação (API) inspirada na API de *Web Services*, para que outros desenvolvedores possam implementar novos Serviços de Gerenciamento sobre esse protótipo usando conceitos já difundidos entre a comunidade de desenvolvedores. Foi definida também uma API para que os desenvolvedores dos TLMs possam acessar de modo padronizado os Serviços de Gerenciamento, acessando de modo transparente os recursos da rede P2P. Procurou-se tornar o padrão dessas APIs o mais próximo aos padrões atuais de desenvolvimento de serviços para aplicações, mantendo-se como foco as APIs de arquiteturas de *Web Services* e arquiteturas orientadas a serviços.

Esse protótipo foi desde o princípio de sua implementação orientado de forma que possa ser expansível, para que seja possível a adição de novas funcionalidades por desenvolvedores interessados em expandir a plataforma. As diretrizes de expansibilidade do sistema também foram levadas em consideração no momento da implementação dos algoritmos de distribuição de carga. A implementação dos algoritmos de distribuição foi feita tendo em vista dar liberdade ao desenvolvedor dos Serviços de Gerenciamento, para que escolha o algoritmo de distribuição que mais se adapte às suas necessidades, podendo inclusive implementar sua própria versão se for necessário.

O restante deste trabalho está organizado da seguinte forma: no capítulo 2, é feita uma revisão dos trabalhos relacionados em relação aos modelos de gerenciamento de redes, além de uma revisão sobre os conceitos de *Web Services*, SOA e *Peer-to-Peer*. No capítulo

3, são descritos os blocos básicos usados para construir arquiteturas de gerenciamento de redes baseadas em P2P, e é definida a arquitetura. No capítulo 4, é apresentado o protótipo do sistema e é feita a descrição da API a ser usada por desenvolvedores de software de gerenciamento. No capítulo 5, é apresentada a implementação da distribuição de carga entre os MLMs que participam de um grupo. No capítulo 6, é feita uma avaliação da proposta, através da apresentação de alguns resultados de testes de desempenho, e é feita uma comparação entre diferentes cenários usando diferentes algoritmos de distribuição de carga para uso de P2P com Gerenciamento de Redes. Por fim, o capítulo 7 encerra este trabalho apresentando algumas observações conclusivas e trabalhos futuros.

2 TRABALHOS RELACIONADOS

Um dos objetivos da área de Gerenciamento de Redes é manter a disponibilidade e o desempenho dos serviços de comunicação das organizações, a fim de que a troca de informações entre os computadores seja feita de maneira confiável e rápida. Muitas vezes, além da disponibilidade dos serviços, é desejável a redução dos custos operacionais envolvidos no uso de sistemas distribuídos.

Um grande número de protocolos existem para suportar o gerenciamento de redes. Entre eles destacam-se o SNMP, CMIP, WBEM, Transaction Language 1 e NETCONF. Os mecanismos usados para coleta de dados para gerenciamento incluem o uso de agentes instalados na infraestrutura, análise de *logs* de atividades, análise de tráfego e monitoramento de atividades.

Uma nova tecnologia que vem se tornando bastante promissora na área de gerenciamento de redes são os *Web Services* (WS). Embora os WS não tenham sido originalmente projetados para serem usados na tarefa de gerenciar dispositivos de rede, as suas características de funcionamento permitem a construção de sistemas altamente distribuídos. A utilização dessa tecnologia na área de gerenciamento vem sendo gradativamente introduzida através da utilização de gateways que convertem operações WS para operações SNMP e vice-versa.

Seguindo a evolução da arquitetura de Web Services, a Arquitetura Orientada a Serviços (*Service-Oriented Architecture* - SOA) (MUKHI; KONURU; CURBERA, 2004) pode suportar a integração de atividades de uma forma bastante complexa em sistemas altamente distribuídos. Esse modelo de arquitetura define o uso de serviços fracamente acoplados, independentes, possibilitando a utilização de recursos distribuídos que podem estar sob o controle de diferentes domínios.

Neste capítulo serão apresentados, inicialmente, conceitos sobre modelos de gerenciamento centralizado, distribuído e cooperativo. Em seguida, a tecnologia *Web Services* será mostrada, seguida por um estudo sobre os *gateways* existentes atualmente. A seguir, é feita uma discussão sobre a utilização de arquiteturas orientadas a serviços na construção de sistemas altamente distribuídos. Por fim, o capítulo encerra com uma revisão sobre os conceitos básicos de P2P.

2.1 Modelos de Gerenciamento de Redes

Vários modelos de gerenciamento de redes já foram propostos e investigados. O mais disseminado é o modelo centralizado do tipo gerente-agente (PARK; CHO, 1994), porém existem também modelos mais sofisticados, usando agentes móveis (BIESZCZAD et al., 1998) e também usando inteligência artificial (MOUNTZIA; RODOSEK, 1999). Os principais modelos podem ser classificados em três tipos, conforme seu grau de distribuição:

centralizado, distribuído e cooperativo (GOLDSZMIDT; YEMINI, 1995). Essa classificação se dispõe de forma progressiva, do modelo menos distribuído até o mais distribuído, conforme é descrito a seguir.

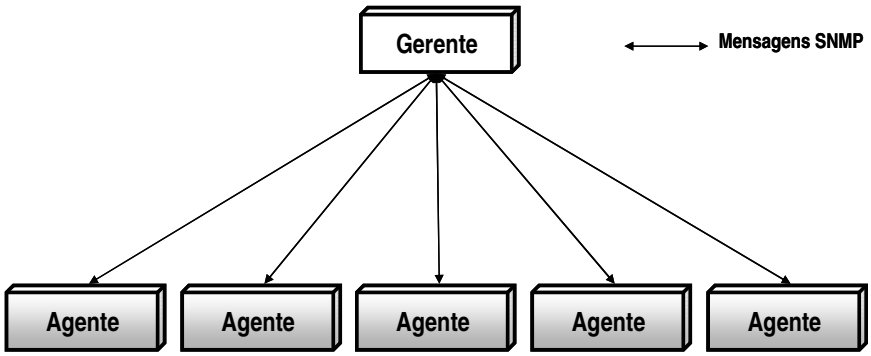

Figura 2.1: Gerenciamento centralizado

2.1.1 Gerenciamento Centralizado

No paradigma tradicional de gerenciamento centralizado de redes, o modelo dominante é o gerente-agente, em que existe uma única estação de gerenciamento que é responsável pela monitoração e controle dos agentes de gerenciamento distribuídos ao longo da rede. Os agentes são processos bastante simplificados, normalmente apenas um meio para coletar ou modificar os dados em dispositivos gerenciáveis, atuando de forma bastante simples no processo de gerenciamento. No gerenciamento centralizado, uma estação central (gerente) recupera os dados dos agentes e processa os dados coletados usando um protocolo de gerenciamento, tal como o SNMP (Figura 2.1).

Entretanto, esse paradigma já alcançou seus limites no caso de redes de grande escala (GOLDSZMIDT; YEMINI, 1995). Não é um modelo escalável, e dessa forma impõe um limite à gerenciabilidade da rede. Por exemplo, existe um limiar que deve ser levado em conta sobre quantas variáveis podem ser consultadas pela estação central e sobre a freqüência de consulta (BEN-ARTZI; CHADNA; WARRIER, 1990). Além do fator limitante dos recursos de rede usados para tal, com o crescimento da rede, o processamento de dados na estação central de gerenciamento pode se tornar inviável quando há uma elevada quantidade de dados a serem coletados e processados. O fato de o gerenciamento ser centralizado também afeta a disponibilidade do processo de gerenciamento, pois, como apenas uma única estação é responsável pela monitoração e consulta sobre os agentes da rede, essa estação se torna o ponto vulnerável do sistema, pois se falhar ou ficar sobrecarregada, todo o processo de gerenciamento é interrompido.

2.1.2 Gerenciamento Distribuído

Nos primeiros sistemas computacionais existentes, os limites impostos pelos sistemas centralizados de gerenciamento não eram um problema. Porém, passaram a se tornar um problema quando as dimensões, a complexidade e a conectividade dos sistemas computacionais evoluíram. A infra-estrutura de muitos dos grandes sistemas atuais é geograficamente e funcionalmente distribuída, e o gerenciamento de seus recursos nesses casos pode ser crítico.

O gerenciamento distribuído propôs a melhora na escalabilidade e flexibilidade dos sistemas de gerenciamento, contornando alguns dos problemas do paradigma centrali-

zado. O paradigma de gerenciamento distribuído por delegação (MbD) propõe a delegação de funções de gerenciamento para localizações distribuídas, que pode ser realizada pela transferência e controle remoto dos procedimentos de gerenciamento (YEMINI; GOLDSZMIDT; YEMINI, 1991). Isso acaba permitindo que funções de gerenciamento fiquem mais próximas das entidades gerenciadas, o que por sua vez diminui o tráfego de gerenciamento em um único ponto, distribuindo decisões de gerenciamento ao longo da rede gerenciada.

Seguindo a classificação de gerenciamento distribuído apresentada por Schönwälder et al. (SCHÖNWÄLDER; QUITTEK; KAPPLER, 2000), as entidades de um sistema de gerenciamento distribuído por delegação (MbD) podem ser classificadas como gerentes superiores (TLMs - *Top-Level Managers*), gerentes intermediários (MLMs - *Mid-Level Managers*) e Agentes. TLMs são responsáveis pelo monitoramento e controle de execução de procedimentos delegados para localizações remotas, onde os MLMs são encontrados. Portanto, MLMs são os receptores de procedimentos de gerenciamento e aqueles responsáveis pela execução de tais procedimentos como forma de realizar a tarefa de gerenciamento. Para fazer isso, MLMs também interagem com agentes de gerenciamento localizados dentro de dispositivos de rede. Por fim, agentes são responsáveis pela consulta ou alteração dos dados existentes nos dispositivos de rede aos quais eles encontram-se associados.

Seguindo ainda as definições apresentadas por Schönwälder et. al., os paradigmas de gerenciamento distribuído diferem entre si em relação a quantidade de TLMs, MLMs e agentes que fazem parte do processo de gerenciamento. Schönwälder et. al. definiram G como o número total de TLMs e MLMs, e N o número total de elementos no sistema de gerenciamento, isto é, a soma de TLMs e MLMs e do número de agentes. Com essas definições, o paradigma de gerenciamento pode ser dividido em gerenciamento: fracamente distribuído, fortemente distribuído e cooperativo.

O gerenciamento fracamente distribuído caracteriza-se pela quantidade de MLMs e TLMs ser próxima a 1 e muito menor que a quantidade total de elementos participantes do sistema de gerenciamento. De uma forma relacional, tem-se que: $1 < G \ll N$. Esse paradigma de gerenciamento caracteriza-se, também, por não existir interação direta entre os MLMs (Figura 2.2).

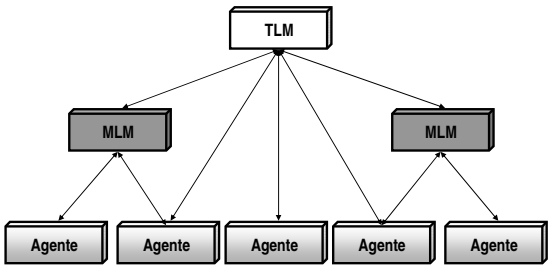

Figura 2.2: Gerenciamento fracamente distribuído

No gerenciamento fortemente distribuído, a quantidade de MLMs e TLMs é muito maior que 1 e é próxima da quantidade total de elementos participantes do sistema de gerenciamento, o que pode ser descrito por: $1 \ll G < N$. Além disso, diferentemente do gerenciamento fracamente distribuído, o gerenciamento fortemente distribuído caracteriza-se pela interação direta entre os MLMs para a realização das tarefas de gerenciamento (Figura 2.3).

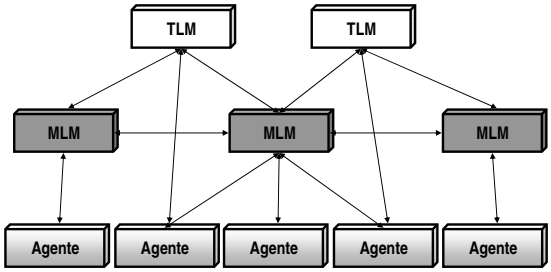

Figura 2.3: Gerenciamento fortemente distribuído

Por fim, o gerenciamento cooperativo caracteriza-se pela quantidade de gerentes (MLMs e TLMs) ser aproximadamente igual a quantidade total de elementos participantes do sistema de gerenciamento, podendo ser descrito por: $G \approx N$. Salienta-se que no gerenciamento cooperativo (Figura 2.4) não existe mais o papel de TLMs e MLMs: os gerentes cooperam entre si para gerenciar os agentes.

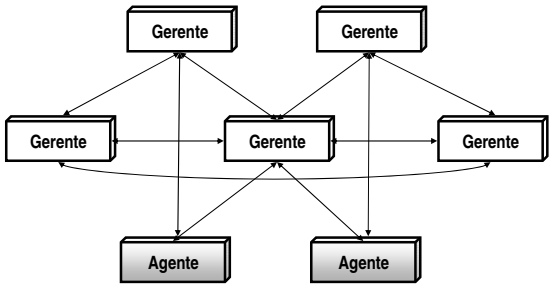

Figura 2.4: Gerenciamento cooperativo

Várias outras vantagens e características do modelo de gerenciamento distribuído por delegação são amplamente cobertos pela literatura sobre gerenciamento de redes (MARTIN-FLATIN, 2002) (KAHANI; BEADLE, 1997).

2.2 Web Services

A tecnologia de Web Services (WS) (CURBERA et al., 2002) tem se mostrado bastante promissora na área de gerenciamento de redes. Embora ela tenha sido desenvolvida originalmente para suportar processos de comércio eletrônico, ela pode também ser utilizada como uma ferramenta de integração em diversos sistemas, tais como redes de suporte a *grids* (GOTH, 2002), inteligência artificial (PREECE; DECKER, 2002) e, recentemente, gerenciamento de redes.

Os Web Services podem ser descritos como um conjunto de componentes independentes disponibilizados na Internet utilizando protocolos Web (e.g HTTP, SMTP e FTP) e que recebem invocações de serviços de clientes (ROY; RAMANUJAN, 2001). Os clientes de um WS podem ser aplicações de usuários finais ou mesmo outros WS. Nesse último caso, um WS pode requisitar operações de outro WS, permitindo construir uma hierarquia de requisições. Além disso, através de mecanismos conhecidos como orquestração e coreografia de WS, é possível a criação de WS bastantes complexos baseados na invocação de outros mais simples (CURBERA et al., 2003). A arquitetura geral de WS (Figura 2.5) é composta por, pelo menos, três elementos principais: Registro, Provedor de serviços e

Cliente.

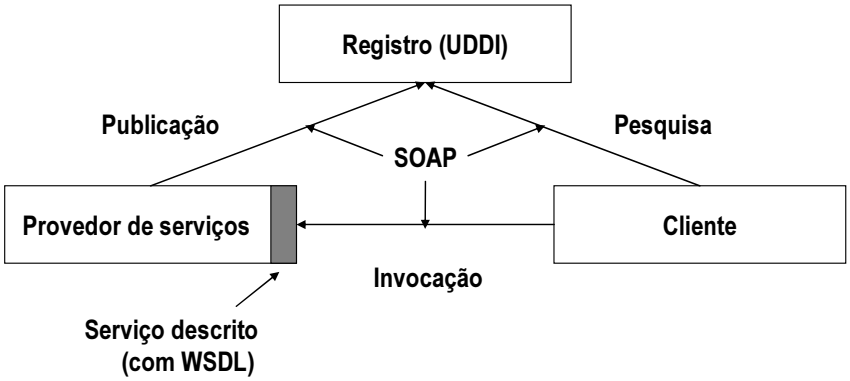

Figura 2.5: Arquitetura geral de Web Services

Para implementar os elementos dessa arquitetura, as principais tecnologias utilizadas, atualmente, são: UDDI (*Universal Description, Discovery, and Integration*) (BELLWOOD; CLÉMENT; RIEGEN, 2003), que tem a função de atuar como o registro dos WS; WSDL (*Web Services Description Language*) (CHRISTENSEN et al., 2001), um padrão para descrição dos WS; e SOAP (*Simple Object Access Protocol*) (GUDGIN et al., 2003), um protocolo baseado em XML utilizado para comunicação entre os elementos da arquitetura WS. Analisando mais detalhadamente, UDDI é um serviço de diretório emergente que trabalha como um repositório de dados para registrar e armazenar descrições de WS e informações de localização. O próprio registro UDDI é implementado como um WS, cujos serviços oferecidos são o cadastro e a pesquisa no repositório de dados. Dessa forma, clientes exploram o repositório UDDI, pesquisando por operações e WS disponíveis. Uma vez encontrado o WS apropriado, o cliente contata esse WS, invocando suas operações. Uma das informações que podem ser encontradas no registro UDDI é a localização do arquivo de descrição do WS. De posse desse arquivo, o cliente pode aprender os detalhes dos serviços oferecidos pelo WS e criar formas de acesso aos mesmos em tempo de execução. WSDL é usada para este propósito, ou seja, descrever as informações necessárias para a realização de acesso a um determinado serviço. Todas as comunicações entre cliente e UDDI, WS e UDDI, e cliente e WS são executadas através do protocolo SOAP.

Existem também trabalhos que propõem o uso de redundância de *Web Services* como uma alternativa para aumentar a confiabilidade e a disponibilidade dos WS. No caso do trabalho de Jiang et al (JIANG; WILLEY, 2005), um WS que precisa das características de confiabilidade e disponibilidade será disponibilizado por múltiplos servidores. A redundância da informação sobre os WS é mantida pelo servidor de registro. O cliente pode então escolher por invocar um WS a partir de qualquer um dos servidores que disponibilizam o WS.

A flexibilidade, a concepção para computação distribuída e a facilidade de utilização oferecida pelos WS parecem ser mais atrativas que soluções de gerenciamento mais antigas, tais como SNMP, CMIP e CORBA. As características dos WS demandam pesquisas para saber se os WS estariam aptos em melhorar ou substituir as atuais tecnologias e soluções de gerenciamento. Assim, várias investigações com respeito a utilização de WS na área do gerenciamento de redes estão em desenvolvimento (PAVLOU et al., 2004), (SLOTEN; PRAS; SINDEREN, 2004) e (KLIE; STRAUSS, 2004).

Embora a tecnologia WS mostre potencial à primeira vista, a introdução de WS no gerenciamento de redes deve ser cautelosa e bem analisada. Questões de desempenho nos elementos gerenciados e o consumo de banda da rede com o tráfego imposto pela utilização de WS poderiam impedir uma solução de gerenciamento efetiva. Também, não é adequado acreditar que o suporte a WS será encontrado em todos os elementos do processo de gerenciamento de rede. Por exemplo, dispositivos baseados em SNMP certamente ainda estarão presentes nas redes futuras. Dessa forma, é plausível acreditar que o cenário de gerenciamento baseado em WS mais comum será aquele onde as tecnologias estabelecidas de gerenciamento (como o SNMP) coexistirão com WS. Na seção a seguir será visto que isso é possível através da utilização de *gateways*.

2.2.1 *Gateways* WS para SNMP

Para usar a arquitetura de WS no gerenciamento de redes, processos de tradução devem ser introduzidos. Esses processos são necessários para traduzir as informações de gerenciamento obtidas através dos protocolos estabelecidos em informações oferecidas via WS. Uma maneira comum de implementar esses processos de tradução é utilizando *gateways* de protocolos nos sistemas de gerenciamento.

Yoon-Jung Oh et. al. (YOON-JUNG et al., 2002) definem *gateways* XML para SNMP e três métodos de tradução interativa: baseadas em DOM (*Document Object Model*) (APPARAO et al., 1998), em HTTP e em SOAP. Nas traduções baseadas em DOM, um gerente com suporte a XML chama uma interface DOM residente no *gateway*. Tais chamadas são traduzidas em operações SNMP entre o *gateway* e o dispositivo alvo. Na tradução baseada em HTTP, o *gateway* recebe expressões XPath (*XML Path Language*) e XQuery (*XML Query Language*) codificadas por um gerente com suporte a XML. Essas expressões são então traduzidas para requisições SNMP. Esse método de tradução permite que a filtragem de informação possa ser executada diretamente no *gateway*, reduzindo o conjunto de informações de gerenciamento entre o gerente com suporte a XML e o *gateway*, embora uma sobrecarga de processamento seja introduzida. Finalmente, na tradução baseada em SOAP, o *gateway* oferece serviços mais sofisticados, que são acessados pelo gerente com suporte a XML. Nesses serviços, o gerente pode pesquisar informações usando XPath ou prosseguir com consultas complexas através de expressões XQuery.

Strauss e Klie (KLIE; STRAUSS, 2004) propuseram um *gateway* XML para SNMP similar ao método de tradução de Yoon-Jung Oh et. al. O *gateway* aceita mensagens HTTP com expressões XPath na URL. As expressões são então verificadas e traduzidas para mensagens SNMP. DOM é usado para acessar os documentos XML dentro dos *gateways*, reduzindo os dados transferidos entre o gerente e o *gateway*. Em operações de escrita, mensagens POST são traduzidas para requisições SNMP `SetRequest`. *Traps* SNMP são suportadas dentro do *gateway* através de um buffer para *traps* que é acessado pelo gerente. Nesse processo, mensagens POST são enviadas pelo *gateway* para receptores (*listeners*) HTTP nos gerentes baseados em XML.

Em um trabalho desenvolvido no Grupo de Redes do Instituto de Informática da UFRGS, foi implementado um sistema (NEISSE et al., 2003) que, dado um arquivo SMI (*Structure of Management Information*) (ROSE; MCCLOGHRIE, 1990) de uma MIB, cria automaticamente *gateways* XML para SNMP. Os *gateways* criados consultam informações nos dispositivos e geram documentos XML, que são enviados de volta para o gerente, onde são analisados através de parsers. Como no trabalho anterior de Strauss e Klie, a tradução é executada com a ajuda da ferramenta `smidump` (KLIE; STRAUSS, 2004), a qual gera uma versão XML das MIBs.

Como verificado, *gateways* são criados para acessar dispositivos baseados em SNMP e exportar informações de gerenciamento em documentos XML (WS reais, baseados em SOAP, dificilmente são usados). Adicionalmente, a descrição de WS através de WSDL e seu registro em UDDI não são abordados nesses trabalhos. Na próxima seção, serão apresentadas duas abordagens para *gateways* SNMP, que melhor exploram as facilidades introduzidas pela arquitetura de WS.

2.2.1.1 Gateways *WS para SNMP em Nível de Protocolo*

O *gateway* WS para SNMP em nível de protocolo (SCHÖNWÄLDER; PRAS; MARTIN-FLATIN, 2003) fornece operações que são mapeamentos diretos das primitivas SNMP. Um gerente baseado em WS requisita informações de gerenciamento acessando o *gateway* WS para SNMP através de mensagens SOAP. Já os servidores que hospedam os *gateways*, recebem do gerente, a identificação da operação a ser acessada (e.g. Get ou Set) e uma lista dos parâmetros relacionados ao SNMP (o endereço do dispositivo alvo, uma comunidade SNMP válida e OIDs SNMP). Com essas informações, a operação apropriada é invocada dentro do *gateway* WS e o dispositivo alvo é acessado via SNMP.

Em um *gateway* desenvolvido pelo Grupo de Redes da UFRGS (NEISSE et al., 2004) foram implementadas as operações Get, GetNext e Set, que geram, para cada requisição do gerente, exatamente uma requisição SNMP do *gateway* para o dispositivo alvo, e exatamente uma resposta do dispositivo alvo para o *gateway*. Após as informações SNMP serem obtidas do dispositivo, o *gateway* monta uma mensagem SOAP com tais informações e envia essa mensagem de volta para o gerente.

Em um uso simples desse *gateway*, suas operações (Get, GetNext e Set) são descritas em WSDL e registradas em um repositório UDDI. Um gerente baseado em WS, procurando por WS de gerenciamento, pode pesquisar no UDDI e descobrir a localização dos *gateways* disponíveis. É importante notar que, com esta abordagem, o gerente baseado em WS ainda deve estar ciente dos OIDs SNMP para, corretamente, requisitar as instâncias dos objetos ao *gateway*. A vantagem dessa abordagem, entretanto, é o fato de que, toda vez que novos objetos passam a ser suportados por um agente SNMP, o *gateway* WS para SNMP não precisará ser alterado. A desvantagem, por outro lado, vem do fato do gerente ser obrigado a conhecer os objetos SNMP suportados em cada dispositivo gerenciado, mesmo na presença de UDDI. Além disso, o gerente ainda precisa lidar com OIDs SNMP, pois tais *gateways* não possibilitam operações de uso mais fácil do que aquelas encontradas atualmente usando SNMP.

2.2.1.2 Gateways *WS para SNMP em Nível de Objeto*

Um *gateway* WS para SNMP em nível de objeto (NEISSE et al., 2004), diferentemente do apresentado anteriormente, "conhece" os objetos de MIB suportados pelo dispositivo alvo, e apresenta tais objetos como operações de WS. Por exemplo, uma operação `GetIfTable` é uma operação que obtém a tabela completa de interfaces, enquanto que `SetAdminStatus` é uma operação que muda, no dispositivo alvo, o estado administrativo de uma das interfaces de rede disponíveis.

Uma vantagem do *gateway* em nível de objeto é que ele não apenas consulta e expõe as informações como um WS, mas também possui somente as operações permitidas para serem executadas sobre o dispositivo alvo. Uma outra vantagem está relacionada à forma que a base de informações de gerenciamento é consultada. No caso do *gateway* em nível de objeto, o próprio *gateway*, e não mais o gerente, é responsável por controlar as interações com o agente SNMP para obter todas as instâncias de uma tabela, construir as

mensagens de resposta SOAP e enviá-las para o gerente WS. Esse controle de interação, que é realizado pelo gerente no caso de utilização do *gateway* em nível de protocolo, é então movido para o *gateway* em nível de objeto, introduzindo com isso um certo nível de controle no *gateway*.

Em um *gateway* a nível de objeto, o número de mensagens trocadas entre o gerente WS e o *gateway* é menor do que quando se utiliza o *gateway* em nível de protocolo. Porém, a abordagem apresentada pelo *gateway* em nível de objeto perde flexibilidade quando o agente SNMP do dispositivo alvo é alterado (tanto para incluir como para remover objetos). Nesse caso, o WS associado precisa, de fato, ser refeito para refletir as mudanças do agente SNMP. Portanto, necessita-se de uma maneira eficiente de criação de *gateways* WS para SNMP em nível de objeto. No trabalho desenvolvido por Neisse et. al. (NEISSE et al., 2004), foi desenvolvido um sistema que, dado um arquivo de MIB SMI, cria um novo WS para SNMP em nível de objeto.

2.3 SOA - Arquitetura Orientada a Serviços

Seguindo a evolução da arquitetura de Web Services, a Arquitetura Orientada a Serviços (*Service-oriented architecture* - SOA) (MUKHI; KONURU; CURBERA, 2004) expressa arquitetura de software que define o uso de serviços fracamente acoplados, a fim de suportar os requisitos dos processos que envolvem diferentes aplicações. Os recursos das aplicações em um ambiente SOA são disponibilizados como serviços independentes que podem ser acessados sem o conhecimento de que tipo de plataforma está sendo usada para implementação.

Uma arquitetura orientada a serviços se resume a um conjunto de serviços disponibilizados. Esses serviços podem se comunicar com outros serviços. A comunicação pode envolver a simples passagem de dados ou pode também envolver a coordenação de atividades entre dois ou mais serviços. Arquiteturas orientadas a serviços oferecem uma maneira promissora para resolver problemas relacionados à integração de aplicações heterogêneas em um ambiente distribuído (MUKHI; KONURU; CURBERA, 2004).

Uma arquitetura orientada a serviços não está ligada a nenhuma tecnologia específica. Pode ser implementada com o uso de uma grande variedade de tecnologias, incluindo REST, RPC, DCOM, CORBA ou *Web Services*. SOA pode ser implementada usando qualquer um desses protocolos e, por exemplo, pode usar um mecanismo de sistema de arquivos para transporte de dados, conforme a especificação de interface que deve ser definida entre os processos. A idéia central é o uso de serviços independentes com interfaces bem definidas que podem ser invocados a fim de executar suas tarefas de um modo padronizado, sem que o serviço precise ter conhecimento prévio da aplicação que está o invocando, e sem que a aplicação tenha necessidade de conhecimento de como o serviço realmente executa suas tarefas.

SOA também pode ser considerado um estilo de arquitetura de sistemas de informação que possibilita a criação de aplicações que são construídas através da combinação de serviços interoperáveis e fracamente acoplados. Esses serviços comunicam-se usando como base uma definição formal (ou contrato, como uma definição WSDL) que é independente da plataforma base e de linguagem de programação. A definição da interface esconde a implementação do serviço e o uso de uma linguagem específica. Sistemas baseados em SOA são independentes de tecnologias de desenvolvimento e de plataformas (como Java, .NET, etc.). Serviços que são executados usando linguagem C-Sharp sobre uma plataforma .NET e serviços que são codificados em Java e executados em plataforma J2EE,

por exemplo, podem ambos ser consumidos por uma aplicação comum. Aplicações sendo executadas em ambas as plataformas podem também ser consumidoras umas das outras como *Web services*, o que facilita o reuso.

SOA pode suportar integração de atividades de uma forma bastante complexa em grandes sistemas, porém não define uma metodologia nem *framework* para documentação ou implementação dos serviços.

Linguagens de alto nível, como BPEL, e especificações, como WS-CDL e WS-Coordination, estendem o conceito de serviços, oferecendo um método para a definição e suporte de orquestração de serviços de menor granularidade em serviços de maior granularidade, que podem então ser incorporados em processos e *workflows*, implementados em grandes aplicações e portais.

Arquiteturas SOA são projetadas para a articulação de recursos computacionais (aplicações e dados) *on demand*, a fim de chegar aos resultados desejados para os consumidores dos serviços (que podem ser usuários finais ou outros serviços). Há atualmente diversas definições de SOA, porém somente o grupo OASIS (*Organization for the Advancement of Structured Information Standards*) (OASIS, 2006) possui uma definição formal com uma profundidade tal que pode ser aplicada tanto aos domínios tecnológico e de negócios. O grupo OASIS define SOA como:

> "Um paradigma para a organização e utilização de recursos distribuídos que podem estar sob o controle de diferentes domínios. Oferece um modo unificado para fornecimento, descoberta, interação e uso dos recursos a fim de produzir os efeitos desejados consistentes com expectativas e pré-condições mensuráveis."

No modelo SOA, provedores de serviços publicam descrições de seus serviços em um registro que pode ser acessado de forma pública. Processos que querem acessar os serviços descobrem as descrições desses serviços através de pesquisas nos registros, e se ligam dinamicamente ao serviço selecionado. Essa capacidade de ligar-se de forma dinâmica leva a um baixo acoplamento entre as aplicações e permite às aplicações adaptar-se de modo eficiente em um ambiente com mudanças constantes.

Serviços em um ambiente SOA podem interagir de uma grande variedade de maneiras, de forma a refletir a heterogeneidade das aplicações suportadas. Isso é uma conseqüência do extenso interesse que os *Web Services* criaram em praticamente todos os setores da indústria de software.

2.3.1 Princípios Arquiteturais de SOA

Esse estilo de arquitetura promove o reuso de software em nível macro (serviços) em contraste aos níveis mais baixos (objetos). Em alguns aspectos, pode ser considerada uma evolução na arquitetura *Web Services*, e não uma revolução, pois apresenta características de outras arquiteturas existentes. Possui basicamente os seguintes princípios para desenvolvimento, manutenção e uso:

- Reuso, granularidade, modularidade, componentização, interoperabilidade;
- Obediência a padrões, sejam eles comuns ou específicos de uma plataforma;
- Serviços identificados, categorizados, fornecidos, distribuídos, monitorados e rastreados.

Seguem os princípios arquiteturais para o projeto a definição de serviços em uma arquitetura SOA:

- Encapsulamento de serviço;
- Fraco acoplamento dos serviços: Os serviços devem manter um relacionamento que minimize as dependências entre si, sendo que um não deve precisar conhecer as especificidades de implementação de outro serviço;
- Contrato: os serviços aderem a um acordo sobre a comunicação entre si, definido de forma coletiva por um ou mais documentos de descrição de serviço;
- Abstração: os serviços escondem do mundo externo toda a lógica que não está definida no contrato;
- Reusabilidade: a lógica é toda dividida em serviços, de forma a promover o reuso;
- Possibilidade de composição: conjuntos de serviços podem ser coordenados e reunidos a fim de formar serviços compostos;
- Autonomia: os serviços possuem controle total sobre sua lógica encapsulada;
- Pouca informação de estado: os serviços minimizam a necessidade de guardar informações específicas sobre uma determinada atividade;
- Possibilidade de descoberta: serviços são projetados de forma a serem tão descritivos que possam ser encontrados e acessados através de um mecanismo de descoberta disponível.

2.4 O Modelo de Comunicação P2P

Desde os primórdios da Internet, os desenvolvedores da antiga ARPANET já sonhavam com o dia em que todos os computadores estariam conectados e oferecendo recursos à rede de modo colaborativo. Essa visão levou à criação do Programa de controle de Redes (NCP - Network Control Program), o primeiro protocolo de rede ponto a ponto, e o precursor do TCP/IP.

O conceito de ponto a ponto - que também é chamado *peer-to-peer* - foi crucial no desenvolvimento da Internet. Todos os computadores da rede eram iguais em termos de conectividade: todo computador poderia acessar os recursos de qualquer outro computador da rede e ao mesmo tempo deixar seus próprios recursos disponíveis. A comunicação entre eles também era considerada equivalente: não havia um computador visto como cliente ou como servidor, e todos os computadores possuíam mais ou menos os mesmos recursos de rede, que eram bastante escassos.

Muitos eventos levaram a Internet do conceito inicial de ponto a ponto para uma arquitetura cliente servidor, que agora é muito mais familiar. A Internet passou a se tornar muito mais comercial, corporações colocaram *firewalls* em torno de suas informações para controlar o acesso. Usários entram na Internet usando computadores que sequer chegam perto do poder dos servidores das grandes corporações. E muitas aplicações e serviços populares da Internet, incluindo páginas Web e transferências FTP, são baseadas em uma arquitetura cliente-servidor.

Nos últimos anos, a Internet cresceu de maneira a suportar uma infinidade de novas aplicações, ricas tanto em variedade quanto em tamanho. As tecnologias P2P mais uma vez passaram a ter um grande espaço Internet e na distribuição de informações e recursos.

Muito tem se falado sobre as tecnologias P2P, porém existe uma grande variedade de opiniões sobre a definição de P2P. Clay Shirky, consultor e escritor, definiu P2P como "uma classe de aplicações que aproveita recursos - armazenamento, processamento, conteúdo, presença humana - disponíveis às margens da Internet". Li Gong, da Sun Microsystems, escreve que "o termo Redes P2P é aplicado a uma grande variedade de tecnologias que aumentam em muito a utilização de informação, banda, e recursos computacionais da Internet. Freqüentemente, essas tecnologias P2P adotam uma arquitetura de rede que nem exclui nem depende de pontos de controle centralizados". Ed Dumbill, da XML.com, diz que "P2P é o que dizem que P2P é..."; de seu ponto de vista, qualquer situação em que informação do mesmo tipo é trocada torna-se essencialmente P2P. Essa é uma definição tomada a partir de uma perspectiva ideológica, excluindo e escondendo todo o ponto de vista tecnológico da definição de P2P. Porém, parece ser atualmente a definição mais popular e mais difundida.

Apesar de ter apenas poucos anos de uso, o compartilhamento de recursos através de sistemas P2P representa hoje uma grande fração de tráfego na Internet, em algumas situações até mesmo acima do tráfego Web.

2.4.1 Origem do P2P

A atual discussão sobre modelos de comunicação P2P começou em 1999 com o lançamento do programa Napster. No mês de janeiro, Shawn Fanning, um estudante de Ciência da Computação da Universidade Northeastern, criou esse programa, que permitia a usuários compartilharem arquivos de áudio (MP3). Em maio, foi criada a empresa Napster, Inc. Em agosto, o site Napster.com iniciou o seu serviço e se tornou a sensação momento. Em uma rede Napster, usuários (ou *peers*) guardavam seus arquivos MP3 em disco local, enquanto o sistema rodava um servidor central guardando apenas um índice contendo os arquivos disponíveis à comunidade de usuários. Para fazer o download um determinado arquivo, o usuário fazia uma pesquisa no servidor central baseada em palavras-chave, e obtinha assim o endereço IP de todos os *peers* que possuíam os arquivos com as palavras-chave usadas na pesquisa. O usuário podia então efetuar o download dos arquivos desejados de um destes *peers*.

A idéia por trás do programa Napster é muito simples - o armazenamento distribuído de arquivos nos usuários (ao invés de servidores) acompanhado por um índice centralizado a fim de facilitar a localização desses arquivos. Essa idéia simples provou ter grande sucesso. Apesar disso, provou ter um ponto vulnerável: a centralização do índice de arquivos. Vulnerabilidade provocada pelo fato de que, além de o sistema não ser totalmente escalável, possuir um único ponto de falha. Em 7 de dezembro de 1999, foi iniciada uma batalha judicial comandada pela Associação das Gravadoras dos EUA, a Recording Industry Association of America (RIAA), reclamando de violações de direitos autorais pela facilitação de compartilhamento de arquivos MP3, que só terminou em março de 2001, com o desligamento do servidor de indexação. Sendo o servidor de indexação crucial para o funcionamento do sistema, este servidor central foi obviamente o principal alvo. Acabando com o sistema de indexação central seria uma maneira fácil de terminar com todo o sistema.

No mês de junho de 1999, Ian Clark completou o estágio inicial da arquitetura conhecida hoje como FreeNet (CLARKE et al., 2000). Embora sendo, assim como o Napster, um programa feito para compartilhamento livre de arquivos, o FreeNet difere do Napster em alguns aspectos importantes, e pertence a uma classe de serviços conhecidos como Sistemas de Banco de Dados Distribuídos com Criptografia, ou C*ryptographically Distri-*

buted Databases (CDDs).

Seguindo o crescimento do Napster e FreeNet, sistemas similares como Gnutella (RIPEANU, 2001), Jungle Monkey (CLARKE et al., 2000), MojoNation e outros surgiram rapidamente. Esses sistemas seguiram o modelo de descentralização de armazenamento do sistema Napster, porém eram diferentes no modo de localização dos recursos. Ao invés de um índice central em um servidor, esses sistemas utilizam técnicas de busca descentralizadas, onde uma pesquisa é propagada entre os *peers* e qualquer *peer* que possuir um arquivo que corresponda à pesquisa responde diretamente ao *peer* requerente. Essa segunda geração de sistemas de compartilhamento de arquivos tornou-se completamente descentralizada tanto em termos de armazenamento quanto em termos de busca, uma mudança motivada principalmente pelos motivos legais e técnicos que levaram ao fim do Napster. Para escapar dessas limitações, esses sistemas foram projetados de forma que não houvesse um único ponto de centralização. Em março de 2001, o Napster foi terminado. Porém isto não significou o fim dos sistemas de compartilhamento de arquivos. Pelo contrário, os sistemas completamente descentralizados continuam, e novos sistemas descentralizados como Kazaa e Emule surgiram, fora o fato de que a população de usuários P2P continua crescendo dia a dia.

2.4.2 Web X P2P

Apesar do desenvolvimento dessas aplicações remeter a um conjunto de questões legais, sociais e que interessam diretamente a toda comunidade da Internet, principalmente ao se falar em controle de tráfego, de uma perspectiva técnica o mais importante aspecto do P2P são suas vantagens tecnológicas. Para verificar estas vantagens, podemos fazer uma comparação, considerando se estas aplicações fossem desenvolvidas utilizando a tecnologia existente da Web.

Deste o início da década de 90, a Web serve como um meio efetivo de publicação e acesso a conteúdo na Internet. Com sua arquitetura cliente-servidor, os documentos são guardados em servidores. Para acessar um determinado documento, um usuário, ou cliente, localiza o servidor, envia uma requisição e obtém uma resposta. Tipicamente, o conteúdo está disponível em um número reduzido de servidores, e existe um número muito menor de servidores do que de clientes. Além disso, os servidores tendem a estar online por períodos relativamente longos de tempo (da ordem de dias) enquanto clientes tendem a estar menos tempo acessíveis. A duração de uma conexão discada, por exemplo, pode ser da ordem de minutos. O conteúdo na Web é acessado através de *Uniform Resource Locators* (URLs). O meio de busca dos servidores é o *Domain Name System* (DNS).

A arquitetura de sistemas P2P é bastante diferente da arquitetura cliente-servidor. Em sistemas P2P, um *peer* age como cliente e como servidor. Os *peers* buscam, guardam e servem conteúdo. Diferentemente de servidores Web, os *peers* são altamente transitórios, entrando e saindo do sistema em períodos curtos de tempo. Enfim, diferentemente da Web, um sistema P2P não possui uma única e definida maneira de nomear o conteúdo, como URLs. Os usuários são livres para nomear os arquivos guardados em sua máquina local da maneira que desejarem. As duas arquiteturas possuem maneiras diferentes de distribuição de conteúdo:

- A arquitetura P2P permite que o conteúdo seja publicado fácil e rapidamente. Na Web, é necessário primeiramente obter um domínio DNS e configurar um servidor DNS para que resolva este novo domínio. A propagação de informação pela adição

ou remoção de nomes de domínio através de sistemas DNS é um processo relativamente demorado, e normalmente requer configuração manual. Em sistemas P2P, a publicação de conteúdo resume-se à cópia dos arquivos desejados a um diretório em sua máquina local.

- A principal maneira de procurar conteúdo, tanto na Web quanto em sistemas P2P, é por busca através de palavras-chave. Esse sistema é efetivado na Web através de servidores de busca centralizados, como Google e Yahoo, que constroem índices centralizados de conteúdo através do processo de *crawling* na Web. Esse processo é inteiramente independente da vontade do usuário. Embora novos documentos na Web possam ser acessados conhecendo a exata URL, o mesmo documento não estará disponível em um *site* de procura enquanto este não fizer o seu processo de *crawling* sobre esse documento. Tal solução é apropriada quando o conteúdo fica no sistema por períodos relativamente longos de tempo, porém não é apropriada para documentos que são freqüentemente adicionados e removidos. Na arquitetura P2P, por outro lado, uma vez que um *peer* entra no sistema, os arquivos publicados estarão imediatamente disponíveis tanto para busca como para *download*.

- Diferentemente da Web, os sistemas P2P se baseiam inteiramente na participação das máquinas dos usuários, e não necessitam de uma infra-estrutura de servidores preexistente. Na Web, recursos de armazenamento e de rede são normalmente disponibilizados por servidores de grande capacidade. Além disso, seu funcionamento depende da infra-estrutura DNS. Isto torna seu funcionamento independente do comportamento do cliente. Os sistemas P2P, por outro lado, não fazem uso de infra-estrutura preexistente, e os recursos disponíveis aos usuários são a soma dos recursos disponíveis em cada máquina de usuário existente na rede.

Esses são alguns dos motivos pelos quais o fenômeno de utilização de aplicações P2P não poderia ter acontecido sobre uma infra-estrutura similar à Web. A atração dos sistemas P2P está em seu potencial em permitir o desenvolvimento e disponibilização rápida e de baixo custo de aplicações de grande escala.

2.4.3 Arquiteturas P2P

Na Internet existem milhões de computadores conectados à rede em qualquer momento, e todos estão teoricamente conectados um ao outro. Como um todo, a topologia na Internet é um grupo de computadores espalhado em diversos locais do mundo.

Dentro de cada subgrupo, ou subrede, computadores estarão visíveis a outros computadores da subrede, e algumas vezes para a rede externa, a Internet. Alguns desses computadores são servidores e fornecem conteúdo e serviços. Acessando um servidor a partir de um computador local torna esse computador um cliente. Enquanto um computador cliente poderia estar acessando um servidor, poderia também estar compartilhando pastas do disco local com membros de seu grupo local. Nessa situação, esse computador se tornaria um servidor para qualquer cliente que tentasse acessar arquivos em seu disco local. Na maioria dos sistemas P2P, a divisão entre ser um servidor e ser um cliente não é bem clara.

Na maioria dos casos, os *peer* estarão conectados um ao outro usando os protocolos TCP ou HTTP. O protocolo HTTP é muito popular em sistemas P2P porque muitas organizações mantém a porta 80 livre em seus *firewalls* a fim de permitir tráfego web.

Diversas topologias de rede podem ser usadas para conectar sistemas P2P. A seguir serão discutidas as principais topologias de rede a fim de explicar como a informação pode ser transmitida entre os *peers* de modo eficiente.

- Topologia Hierárquica

  Uma das topologias mais comuns é a hierárquica. Um dos exemplos mais conhecidos é o DNS - Domain Name Server. É um sistema construído de forma hierárquica, com servidores raiz nos níveis mais altos da topologia. Porém, nessa arquitetura, os servidores raiz representam um ponto central de falha.

- Topologia em Anel

  A topologia em anel, disseminada pelas redes *Token Ring*, é uma topologia de rede que usa o conceito da passagem de uma "ficha virtual"entre os computadores conectados em um padrão de anel. Quando uma máquina recebe a ficha, pode enviar informação para a rede. As redes *Token Ring* não são mais muito usadas em redes comuns, porém a topologia em anel é muito usada em arquiteturas de redes metropolitanas. Provê também um padrão interessante para balanceamento de carga de um sistema.

- Topologia Cliente-Servidor, ou Centralizada

  A topologia cliente-servidor, ou centralizada, é a mais comum. A terminologia cliente-servidor já existe há muitos anos; mais recentemente, o termo centralizado tem sido usado para descrever um sistema em que um único computador, o servidor, disponibiliza os serviços à rede. Os computadores clientes se conectam ao servidor quando serviços forem necessários. Obviamente, quanto mais clientes existirem na rede, mais poderoso o servidor deve ser. Em algum momento, o servidor poderá ser replicado a fim de tratar todo o volume de tráfego de todos os clientes.

- Topologia Descentralizada

  A topologia descentralizada é uma topologia de rede que mais se aproxima de ser verdadeiramente P2P. Não existe uma autoridade central, apenas computadores individuais que são capazes de se conectar e se comunicar entre si. Basicamente, todos os *peers* do sistema agem como clientes e como servidores, tratando requisições de busca e de acesso enquanto também fazem buscas e requisições de acesso eles próprios. As aplicações KaZaA e Gnutella usam essa topologia descentralizada em seus sistemas P2P.

- Topologia Híbrida

  Na topologia híbrida, os computadores são considerados clientes quando eles precisam de informação. O cliente que precisa de informação irá contatar um servidor central para obter uma referência a um outro cliente onde a informação necessária está guardada. O cliente que quer acessar a informação irá então contatar diretamente o computador que possui a informação. Dessa forma, um computador pode ser ao mesmo tempo cliente e servidor. Essa é a topologia usada pelo sistema Napster - *peers* contatam um servidor central para localizar informações e contatam diretamente outros *peers* para fazer acesso a informação.

2.4.4 Problemas relacionados a arquiteturas P2P

Apesar dos sistemas P2P oferecerem uma grande quantidade de benefícios em termos de distribuição de recursos, comunicação e soluções de problemas, é necessário ter em mente problemas que ainda não foram resolvidos no uso desse tipo de arquitetura.

- Políticas de Local de Trabalho

  Quando uma aplicação P2P é disponibilizada por computadores pertencentes a uma corporação, é preciso verificar quais são as políticas da empresa com aqueles que fazem e aplicam essas políticas. Algumas companhias têm políticas claras, e oferecem restrições ao uso de sistemas P2P em seus computadores.

- Propriedade Intelectual

  Um dos maiores problemas associados à Internet é que a mesma permite que os usuários distribuam material com direitos autorais de forma inapropriada, de forma rápida e em grande proporção. É por não ter o devido cuidado com a legislação vigente que muitos sistemas como Napster tiveram grandes problemas no meio jurídico.

- Custos de Banda de Rede

  Aplicações P2P normalmente consomem uma grande quantidade de banda de rede. Em muitos casos, aplicações P2P podem estar violando contratos com provedores de serviço, porque estariam agindo como servidores. Provedores de serviço usam esse tipo de política para preservar recursos de rede. Além dos custos de banda de rede, existe uma preocupação muito grande com sistemas P2P que utilizam mecanismos de broadcast para localizar outros *peers*. Enquanto mensagens são propagadas, muitos pacotes de broadcast poderiam influenciar na estabilidade da infraestrutura do provedor. Há casos de DoS (*Denial of Service*) que ocorrem devido ao grande volume de mensagens de broadcast na rede. Em muitos casos, é preferível que aplicações P2P operem em uma rede privada, com limitações em relação à conexão ao mundo externo.

- Segurança

  Aplicações Internet são bastante suscetíveis a falhas de segurança. Alguns exemplos de falhas de segurança que podem ocorrer nesse tipo de aplicações são os seguintes. Primeiro, os direitos de acesso que um *peer* remoto pode ter ao computador local quando está fazendo uma requisição ao software P2P que está rodando. É preciso saber que tipo de informação é colhida e enviada ao servidor remoto. Podem haver problemas também quanto à aplicação em si, se é segura, se possui *trojans* ou *Spywares*. Essas são questões e situações que precisam ser verificadas ao se usar uma aplicação P2P. E, como desenvolvedor, é preciso ter a responsabilidade de que as aplicações desenvolvidas sejam confiáveis e seguras.

3 O MODELO DE GERENCIAMENTO DE REDES BASEADO EM ESTRUTURAS P2P

O gerenciamento de redes tradicionalmente é feito usando o modelo gerente-agente (PARK; CHO, 1994). Nesse modelo, os gerentes de rede enviam requisições aos agentes (normalmente localizados nos dispositivos de rede) a fim de resgatar ou ajustar as informações do dispositivo gerenciável. Geralmente, os agentes também podem informar de modo assíncrono aos gerentes interessados sobre as alterações feitas no estado dos dispositivos gerenciáveis, através do envio de mensagens de evento (ou notificações). Além de ser extensamente usado, o modelo gerente-agente é também suportado por instituições de padronização. O modelo SNMP (HARRINGTON; PRESUHN; WIJNEN, 2002), por exemplo, definido pela IETF (*Internet Engineering Task Force*), é baseado no modelo gerente-agente.

De tempos em tempos, novos modelos de gerenciamento que extendem o modelo gerente-agente são definidos, resultando em soluções mais sofisticadas e introduzindo novas entidades de gerenciamento com responsabilidades especiais. No caso do modelo de gerenciamento por delegação (MbD) (GOLDSZMIDT; YEMINI, 1995), os gerentes podem delegar responsabilidades sobre tarefas de gerenciamento a outros gerentes. Assim, gerentes *Mid-Level* (MLMs) são entidades que atuam na função de agentes ao receber tarefas delegadas de gerentes de nível mais alto, mas também atuam como gerentes quando gerenciam agentes de nível mais baixo, que também podem ser outros MLMs.

Redes P2P (MILOJICIC et al., 2002) são redes sobrepostas compostas por nodos (*peers*) distribuídos na Internet, normalmente em computadores pessoais. Redes P2P são capazes de se auto-organizar, se adaptar a falhas e também suportam nodos com conectividade intermitente. Modeladas usando arquiteturas parcialmente ou totalmente descentralizadas, as redes P2P usam recursos distribuídos em diversos *peers* a fim de executar aplicações tais como compartilhamento de arquivos, processamento distribuído e trabalho cooperativo. Da mesma forma que no modelo MbD, o modelo de gerenciamento de redes baseado em estruturas P2P (GRANVILLE et al., 2005) extende o modelo gerente-agente incorporando a esse modelo características de redes P2P, tais como alta distribuição de entidades, auto-organização, tolerância a falhas, escalabilidade e alta conectividade. Essas características permitem o aperfeiçoamento de sistemas de gerenciamento atuais, por exemplo, permitindo a introdução de suporte a gerenciamento inter-domínios, troca confiável de mensagens de gerenciamento, replicação de serviços e distribuição de tarefas.

O modelo de gerenciamento de redes baseado em P2P redefine os papéis das três principais entidades de gerenciamento: gerente *Top-Level* (TLM), gerente *Mid-Level* (MLM), e agentes. A Figura 3.1 mostra um cenário hipotético onde o modelo de gerenciamento de redes baseado em P2P é usado.

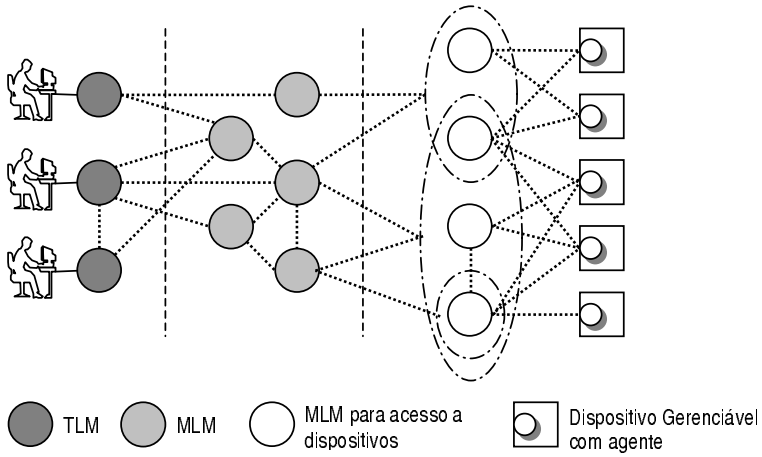

Figura 3.1: Modelo de gerenciamento de redes baseado em P2P

TLMs são definidos como *peers* de gerenciamento que possuem uma interface gráfica de usuário, usada por operadores humanos localizados remotamente, a fim de se comunicarem entre si para efetuarem tarefas de gerenciamento de modo cooperativo. Essencialmente, o TLM oferece a aplicação visual usando não apenas instrumentos tradicionais de gerenciamento (como mapas de topologia e listas de alarmes) mas também serviços de colaboração P2P (como serviços de mensagem instantânea, compartilhamento de arquivos e edição de documentos com múltiplos editores).

Os MLMs são *peers* que respondem a requisições dos TLMs ou de outros MLMs fornecendo serviços de comunicação confiáveis entre entidades de gerenciamento, através de roteamento em nível de aplicação. Alguns MLMs podem ser responsáveis pela coleta de informações diretamente dos dispositivos gerenciáveis. Esses MLMs funcionam como *gateways* de protocolo, que traduzem o tráfego P2P em tráfego de gerenciamento quando estão interagindo com os dispositivos gerenciáveis.

Um conceito importante no modelo de gerenciamento de redes baseado em P2P é a noção de Serviços de Gerenciamento. Um Serviço de Gerenciamento é um serviço oferecido por entidades de gerenciamento a outras entidades do mesmo tipo, e cujo resultado é a execução de uma tarefa de gerenciamento. Cada Serviço de Gerenciamento é identificado por um identificador de serviço e alcançado através da rede de gerenciamento P2P não importando a localização dos *peers* que oferecem o serviço. Um Serviço de Gerenciamento básico é o implementado pelos MLMs para fins de acesso a dispositivos: o serviço permite que *peers* remotos busquem ou modifiquem informações de dispositivos físicos através de um MLM para acesso a dispositivo.

De fato, os MLMs podem ser organizados em grupos de *peers* a fim de oferecer os Serviços de Gerenciamento. Nesse caso, um Serviço de Gerenciamento, ao invés de ser oferecido por apenas um MLM, é oferecido por um grupo de MLMs. Da perspectiva do usuário do serviço, entretanto, os detalhes de como o serviço é provido (como número de *peers* no grupo e localização de cada *peer*) são transparentes: o usuário está interessado na execução do Serviço de Gerenciamento e não precisa saber como este foi implementado.

A organização de MLMs em grupos de *peers* facilita a administração e aumenta a disponibilidade dos Serviços de Gerenciamento porque, enquanto houver um *peer* ativo no grupo, os Serviços de Gerenciamento de tal grupo estará disponível ao restante da rede. Em um grupo com mais de um *peer*, se um MLM sai do grupo os Serviços de Gerenciamento ainda ficam disponíveis porque outros MLMs implementam os mesmos

serviços. Mas é claro que se um grupo perder todos os seus membros os Serviços de Gerenciamento não estará mais disponível; por isso é necessário que exista um auto-gerenciamento pró-ativo entre os membros do grupo.

Finalmente, a última entidade de um sistema de gerenciamento baseado em P2P é o agente (da mesma forma que agentes SNMP), que normalmente é um processo (mas não um *peer*) localizado em dispositivos gerenciáveis que, pela requisição dos *peers* MLM, retornam ou modificam o estado dos dispositivos gerenciáveis e informam eventos relacionados aos dispositivos através de notificações.

3.1 Blocos Básicos para Arquiteturas de Gerenciamento de Redes Baseadas em P2P

Nesta seção serão apresentados os blocos básicos necessários para a criação de arquiteturas de gerenciamento baseadas em P2P que seguem o modelo de gerenciamento anteriormente apresentado. Para isso, serão detalhados os componentes de um *peer* de gerenciamento (Figura 3.2) e será apresentado como tais componentes interagem entre si a fim de fornecer uma infraestrutura de gerenciamento baseada em P2P.

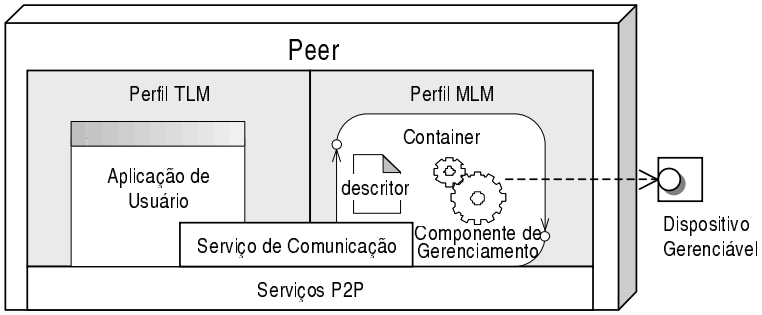

Figura 3.2: Arquitetura de um sistema de gerenciamento de redes baseado em estruturas P2P

Cada *peer* de gerenciamento pode apresentar dois perfis diferentes: o perfil de um TLM e o perfil de um MLM. Enquanto um *peer* TLM puro possui apenas o perfil TLM e um *peer* MLM puro possui apenas o perfil MLM, um sistema de gerenciamento pode possuir *peers* híbridos que implementam os dois perfis ao mesmo tempo.

3.1.1 Top-Level Managers

O perfil TLM oferece um conjunto de recursos visuais, que formam a interface de usuário da aplicação de gerenciamento. Cada *peer* com perfil TLM pode ter um conjunto distinto de recursos visuais, apresentando, assim, interfaces de usuário diferentes. Esses recursos, usados para interconectar operadores humanos (tais como interfaces de mensagem instantânea, compartilhamento de mapas de topologia), precisam estar presentes nas pontas de comunicação em que está o usuário, a fim de não comprometer a troca dessas informações.

3.1.2 Mid-Level Managers

No perfil MLM se situa o elemento fundamental de um MLM: o *Componente de Gerenciamento*. Componentes de Gerenciamento são responsáveis por implementar os Serviços de Gerenciamento oferecidos aos *peers* de gerenciamento remotos. Componentes

de Gerenciamento e seus Serviços de Gerenciamento associados podem variar de serviços muito simples (como um *gateway* para acesso a dispositivos gerenciáveis através de SNMP, SSH ou HTTP) a outros mais complexos (como suporte a execução de scripts ou medição e monitoramento de tráfego).

3.1.3 Agentes

Os agentes são entidades da camada de aplicação que possuem a função de receber e processar requisições, enviando respostas aos MLMs, e também de enviar notificações de eventos que ocorrem no sistema. A comunicação entre os MLMs e os agentes não ocorre dentro da rede P2P. O MLM se comunica com um agente através do protocolo específico para acesso ao agente. Dentre esses protocolos pode-se enumerar o SNMP, que é o mais usado, e também outros como HTTP, NetConf e WebServices.

3.1.4 Serviços de Gerenciamento

Os Serviços de Gerenciamento são similares aos *Web Services* (CURBERA et al., 2002), com a diferença de que possuem características de comunicação P2P, e que são aplicados especificamente ao Gerenciamento de Redes. Enquanto os Web Services puros são acessados através de um protocolo Internet como HTTP ou SNMP, os Serviços de Gerenciamento são acessados através da infraestrutura da rede P2P. Essa particularidade dá aos Serviços de Gerenciamento características que não existem nos Web Services: conectividade, heterogeneidade, disponibilidade e escalabilidade, características essas que são específicas de sistemas P2P.

Os Serviços de Gerenciamento são anunciados, pesquisados, descobertos e invocados em uma rede de gerenciamento P2P através dos serviços de comunicação disponíveis na plataforma usada. Serviços de Comunicação são meta-serviços P2P que todos os *peers* interessados em participar do processo de gerenciamento devem possuir. Entre esses serviços encontram-se os serviços de descoberta de recursos e os serviços de troca de mensagens. Ambos perfis TLM e MLM usam esses serviços de comunicação P2P; os TLMs instanciam os Serviços de Comunicação com o objetivo de descobrir e acessar os Serviços de Gerenciamento; os MLMs, por sua vez, instanciam os Serviços de Comunicação para anunciar os Serviços de Gerenciamento e deixá-los disponíveis para acesso a outros *peers*.

3.1.5 Componentes de Gerenciamento

Componentes de Gerenciamento são as instâncias dos Serviços de Gerenciamento que estão disponíveis nos *peers*. Um Serviço de Gerenciamento é formado por diversos Componentes de Gerenciamento distribuídos, cada qual sendo executado em um diferente *peer*, mas que possuem a mesma descrição, mesmo identificador na rede, e mesma interface para acesso. São descritos em um elemento do sistema chamado *descritor* (a ser detalhado na próxima seção) e controlados por elementos denominados de *Containers*. *Containers* intermediam a comunicação entre os Componentes de Gerenciamento e o restante do sistema de gerenciamento P2P usando um paradigma de Chamada de Procedimento Remoto (RPC) que pode ser implementado, por exemplo, através do uso de protocolos *Web services* (CURBERA et al., 2002). O uso desse paradigma também permite que os *Containers* possam controlar o acesso aos Componentes de Gerenciamento, impedindo acesso não autorizado de clientes a Serviços de Gerenciamento protegidos.

Os *Containers* também administram o ciclo de vida dos Componentes de Gerencia-

mento. São responsáveis pela instanciação dos Componentes de Gerenciamento, publicação de suas descrições e Serviços de Gerenciamento associados, e resposta a requisições de procura por serviços feitas por outros *peers* que estão procurando por Serviços de Gerenciamento específicos. Um *Container* pode, adicionalmente, impor restrições relacionadas aos recursos computacionais que podem ser usados por um Componente de Gerenciamento durante sua execução, limitando, por exemplo, a quantidade de memória que pode ser usada, a quantidade de processamento ou prioridade de execução.

3.1.6 Grupos de *Peers*

Diferentes instâncias de Componentes de Gerenciamento espalhadas pela rede P2P podem implementar um mesmo Serviço de Gerenciamento, com as mesmas descrições e interfaces para acesso. Para que essas diferentes instâncias sejam vistas pelos TLMs como um único Serviço de Gerenciamento, os *peers* que instanciam os Componentes de Gerenciamento se organizam em grupos, comunicando-se entre si a fim de manter a disponibilidade e a escalabilidade do serviço. Enquanto a disponibilização de um Serviço de Gerenciamento por um único *peer* não oferece ao sistema as vantagens de disponibilidade e escalabilidade decorrentes da arquitetura P2P, o uso de Grupos de *Peers* para oferecer os Serviço de Gerenciamento garante que o serviço sempre estará disponível enquanto houver pelo menos um *peer* participando do grupo, e a adição de mais *peers* ao grupo aumenta a escalabilidade do serviço.

3.2 Interação Entre os Elementos da Arquitetura

A fim de exemplificar como todos esses elementos interagem entre si a fim de efetuar as tarefas de gerenciamento, são apresentados os passos a seguir, descritos na Figura 3.3:

1. O TLM busca por um determinado Serviço de Gerenciamento, através de uma requisição ao serviço de descoberta de recursos da rede P2P. O parâmetro da busca normalmente é o nome do serviço.

2. O serviço de descoberta da rede P2P recebe a requisição de busca e responde com as informações referentes ao Serviço de Gerenciamento. Essas informações estão contidas em um documento WSDL, que usa o padrão Web Services para descrever as propriedades do serviço, como nome, descrição, documentação, além das operações disponíveis e dos parâmetros que devem ser usados.

3. O TLM recebe as informações do Serviço de Gerenciamento contidas no documento WSDL. A partir dessas informações, o TLM está apto a criar uma chamada a uma das operações definidas no serviço, usando o protocolo de chamada especificado pelo documento.

4. O TLM faz uma requisição de execução de uma das operações do Serviço de Gerenciamento, usando o protocolo de chamada definido no documento de especificação do serviço. Essa chamada é enviada à rede P2P em uma mensagem *multicast* a nível de aplicação, direcionada ao grupo de MLMs responsável pelo Serviço de Gerenciamento em questão.

5. O Grupo de MLMs responsável pelo Serviço de Gerenciamento recebe a requisição, executa o procedimento de distribuição de carga e o(s) MLM(s) escolhido(s) para responder à requisição passa(m) o fluxo de execução para o Container.

6. O Container executa a operação no Componente de Gerenciamento. Neste ponto, o Container é responsável pela manutenção do ciclo de vida do Componente de Gerenciamento, executando quando necessário as tarefas de instanciação e inicialização.

7. O Componente de Gerenciamento acessa o Agente. Neste passo é necessário o uso de protocolos específicos para acesso aos agentes distribuídos na rede, como SNMP ou HTTP. Este passo muitas vezes não é necessário, se todas as informações necessárias para execução da tarefa de gerenciamento estiverem imediatamente disponíveis ao Componente de Gerenciamento.

8. Depois que o Componente de Gerenciamento termina seu processamento, retorna os dados resultantes ao Container. Este, seguindo suas tarefas de manutenção do ciclo de vida do Componente de Gerenciamento, pode proceder à sua finalização se for o caso.

9. O Container envia a resposta do Componente de gerenciamento ao TLM, usando os serviços de transporte de mensagens da plataforma P2P.

10. O TLM recebe a resposta e executa as suas tarefas de gerenciamento adicionais.

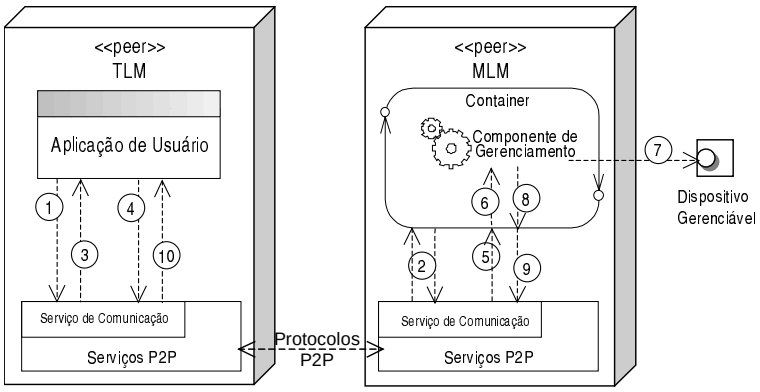

Figura 3.3: Passos da interação entre os elementos da arquitetura

3.3 Comparação da Arquitetura com Web Services e SOA

A fim de ter uma boa padronização e de facilitar a implementação dos Serviços de Gerenciamento pelos desenvolvedores, a arquitetura definida neste capítulo faz uso de diversos conceitos já bastante difundidos nas arquiteturas Web Servides e SOA. Como documentos de descrição dos Serviços de Gerenciamento, foram usados documentos WSDL, que possuem bibliotecas robustas para análise de conteúdo. Não foi usado o mesmo padrão UDDI dos Web Services para a busca dos documentos, pois o padrão UDDI é um padrão de busca centralizada de recursos, e a plataforma P2P deve possuir uma ferramenta de busca distribuída de recursos, que é muito mais útil nesse caso. Deve-se considerar que muitas das características presentes nessa arquitetura não estão disponíveis nas arquiteturas de Web Servides ou SOA. Pode-se dizer que a arquitetura definida possui todas as características de SOA, com as características adicionais de sistemas P2P, entre elas conectividade, heterogeneidade, disponibilidade e escalabilidade. Enquanto arquiteturas

SOA podem ser consideradas um avanço sobre arquiteturas Web Servides, a arquitetura de Serviços de Gerenciamento sobre estruturas P2P pode ser considerada em determinados aspectos um avanço sobre arquiteturas SOA, pois adiciona características úteis a sistemas de gerenciamento altamente distribuídos.

4 IMPLEMENTAÇÃO DO PROTÓTIPO DO SISTEMA DE GERENCIAMENTO BASEADO EM P2P

A implementação aqui apresentada do protótipo de sistema de gerenciamento baseado em P2P é codificada em Java, usando o Framework JXTA da Sun (GONG, 2001) como infraestrutura P2P. Essa implementação é referenciada posteriormente como a Plataforma ManP2P, abreviação para *Management using Peer-to-Peer*, e a API usada pelos programadores é referenciada como API ManP2P. Nas seções subseqüentes, serão detalhados o projeto do protótipo e sua implementação.

4.1 O Framework JXTA como Plataforma P2P

O projeto JXTA (GONG, 2001) - pronuncia-se `juxta`, de *juxtapose* ou justapôr - é um projeto de código aberto concebido inicialmente pela Sun Microsystens, Inc. e depois posto ao encargo de um grupo independente, sob a licença denominada *Apache Software License*. Foi desenvolvido com a participação crescente de instituições acadêmicas e da indústria de software. O projeto JXTA define um conjunto de protocolos para o desenvolvimento de aplicações P2P, a fim de resolver o problema de incompatibilidade entre os protocolos do grande número de sistemas P2P existentes. O principal objetivo do projeto JXTA é definir uma rede P2P genérica que possa ser usada para a implementação de uma grande variedade de aplicações e serviços P2P. Define também um conjunto de blocos a serem reutilizados e um conjunto padrão de regras a serem seguidas. Os desenvolvedores estão livres para substituir qualquer um destes blocos criando seus próprios blocos e criando suas próprias regras, porém respeitando as regras padrão.

4.1.1 Conceitos Importantes na Plataforma JXTA

A plataforma JXTA assume que a rede P2P seja totalmente adaptativa, de modo que as conexões podem ser feitas e desfeitas a qualquer momento e os *peers* possam entrar e sair da rede no momento que desejarem. Os caminhos físicos podem até mesmo ser unidirecionais (é o caso de *peers* dentro de redes com *firewalls* que só permitem conexões a partir de um endereço interno da rede), e os caminhos de roteamento podem mudar tão rapidamente quanto a topologia da rede.

4.1.1.1 Peers *e* PeerGroups

Os *peers* da rede JXTA se organizam em grupos denominados *peergroups*. Um *peergroup* representa um conjunto de *peers* que tem interesses comuns, e concordam com um conjunto comum de regras. Os protocolos JXTA descrevem como esses grupos serão criados, publicados e descobertos, e como o *peer* pode descobrir como deverá se compor-

tar. Esses grupos formam regiões lógicas na rede que não necessariamente refletem sua topologia física. Um *peer* pode fazer parte de quantos grupos desejar, e criar quantos grupos achar necessário. A plataforma JXTA foi projetada especialmente para atender *peers* que podem ser qualquer tipo de dispositivo. A especificação dos protocolos determina expressamente que os *peers* da rede devem ser assumidos como sendo qualquer tipo de dispositivo, desde os menores aparelhos com tecnologia embarcada até o maior *cluster* de supercomputadores. Desta forma tentam eliminar barreiras à participação de quaisquer sistemas operacionais, plataformas ou linguagens de programação. É assumido também que os *peers* e seus recursos podem aparecer e desaparecer espontaneamente da rede e a localização de um *peer* pode mudar espontaneamente ou até mesmo ser mascarada por equipamento de NAT ou *firewall*.

4.1.1.2 Advertisements

Todo e qualquer recurso da rede JXTA é representado por um anúncio, ou *advertisement*. Esses anúncios são representados como metadados independentes de linguagem, através de documentos XML. Através de anúncios, o problema de encontrar *peers* e todos os seus diferentes tipos de recursos se reduz ao problema de encontrar anúncios que descrevem estes recursos. Os *peers* guardam, publicam e trocam anúncios a fim de descobrir e alocar os recursos disponíveis. Os recursos só podem ser descobertos através de uma pesquisa de seus anúncios. Todos os anúncios são publicados com um tempo de vida e, após este tempo, o anúncio expira e se torna inválido.

4.1.2 Protocolos JXTA

Uma solução P2P completa oferece ao *peer* mecanismos para:

1. Descobrir outros *peers*, seus serviços e recursos;
2. Publicar seus serviços e recursos disponíveis;
3. Trocar informações com outro *peer*;
4. Rotear mensagens até outros *peers*;
5. Verificar as informações disponíveis em um determinado *peer*;
6. Agrupar os *peers* em grupos.

A plataforma JXTA define um conjunto de protocolos projetados para as funcionalidades comuns requeridas, a fim de que a rede P2P seja formada de forma independente do sistema operacional de cada *peer*, da linguagem de desenvolvimento usada e da rede de transporte de dados disponível.

Os protocolos JXTA estabelecem uma rede virtual sobre a rede física existente. Essa rede virtual fornece primitivas simples para esconder a complexidade da topologia da rede física, permitindo que qualquer *peer* se comunique com qualquer outro *peer* da rede. Cada recurso da rede é identificado unicamente através de um ID e endereçado, independentemente de sua localização, através de uma relação entre o seu endereço lógico e o endereço físico desse recurso. As mensagens são roteadas de forma transparente, e podem até mesmo atravessar *firewalls* e NATs, inclusive utilizando diferentes protocolos a fim de chegar a seu destino. É possível assim que se preveja roteamento de mensagens para os *peers* que não estiverem diretamente conectados ou acessíveis à rede (*firewall traversing*).

Os protocolos também padronizam a maneira como é feita a descoberta dos *peers*. Estes podem se organizar em grupos, publicar e descobrir recursos, comunicar-se e monitorar um ao outro. Os *peers* também guardam informações em cache a fim de reduzir o tráfego em rede.

Baseado nesses critérios de implementação e outros existentes na especificação dos protocolos (Protocols Specification (DUIGOU, 2007)), o grupo JXTA desenvolveu um conjunto de seis protocolos baseados em mensagens XML, como mostrado na figura 4.1.

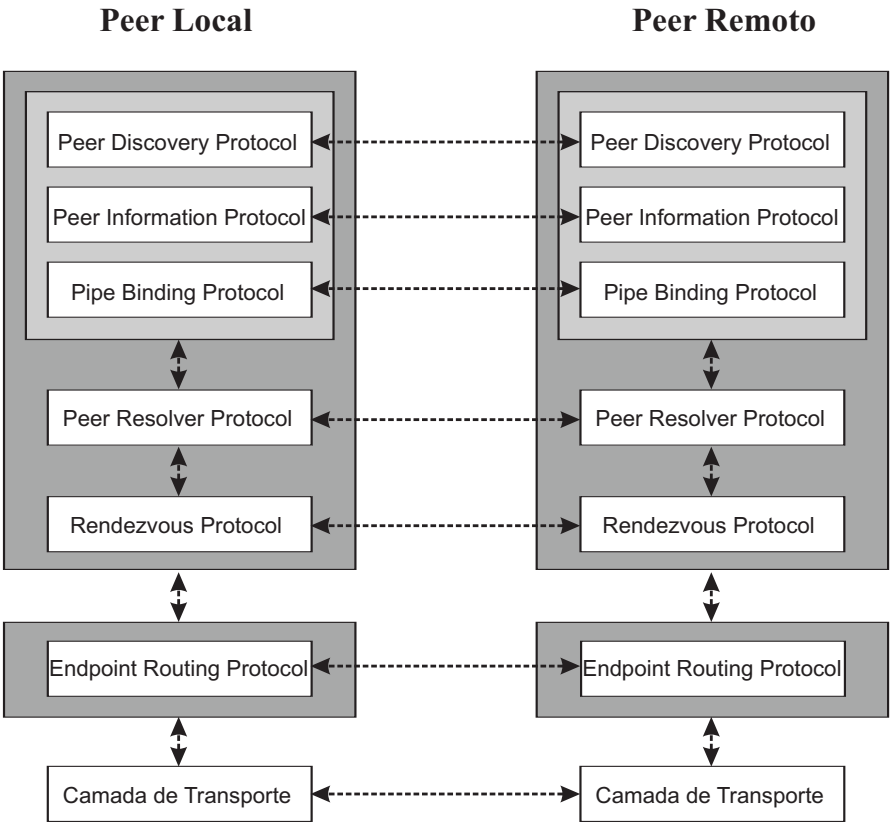

Figura 4.1: Comunicação entre o conjunto de Protocolos JXTA

O *Endpoint Routing Protocol* (ERP) fornece um mecanismo para determinar rotas até um determinado *peer*, permitindo que os *peers* se comuniquem utilizando camadas de transporte incompatíveis. Por exemplo, dois *peers* diferentes podem conectar-se à rede JXTA, um através de TCP, e outro através de HTTP (note-se que o protocolo HTTP está sendo usado como camada de transporte) e trocar mensagens de forma transparente. Permite também que mensagens sejam enviadas através de *firewalls* e NATs. Todas as camadas superiores ao ERP fazem uso de abstrações chamadas *endpoints*. Os *endpoints* mascaram o tipo de camada de transporte que está sendo usada, permitindo assim que um *peer* envie e receba dados independentemente da camada de transporte. A figura 4.2 ilustra a forma como é feita a comunicação entre dois *peers* protegidos por firewall através do ERP.

O *Peer Resolver Protocol* (PRP) trata do envio e recebimento de mensagens através de *queries*. Define um protocolo para o envio de uma determinada *query* a um destino localizado em outro *peer*, e para o processamento de uma *query* recebida.

Pipes são construções que enviam ou recebem dados de um *peer* remoto. Os serviços da plataforma JXTA utilizam tipicamente o PRP ou um *pipe* a fim de comunicar-se com

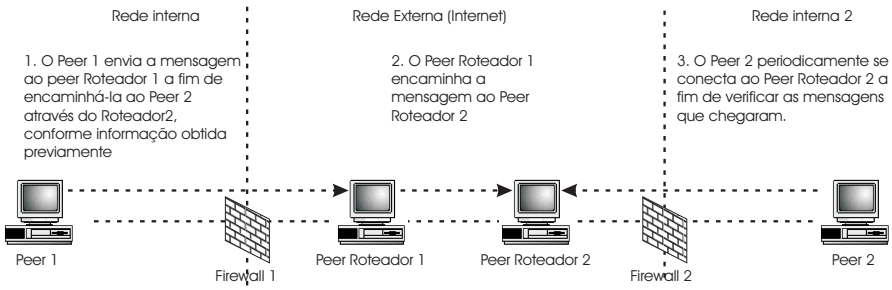

Figura 4.2: Mensagem ERP enviada do *Peer* 1 ao *Peer* 2, atravessando 2 *firewalls*

outro *peer*. Antes de um *pipe* ser usado, é necessário que ele esteja ligado a um *peer* remoto. Os *pipes*, como tudo na plataforma JXTA, também são representados por anúncios. O *Pipe Binding Protocol* (PBP) define um conjunto de mensagens - *queries* e *responses* - que um *peer* pode usar a fim de relacionar um *pipe* a um *peer* remoto.

O *RendezVous Protocol* (RVP) é responsável pela propagação de mensagens a outros *peers* via *peers* especiais chamados *rendezvous*. O RVP é responsável também pelo fornecimento do serviço de rendezvous a outros *peers* da rede. Antes de um *peer* poder usar um *peer rendezvous* para propagar as mensagens, é necessário conectar-se ao *rendezvous* e obter um *lease*, que é uma espécie de contrato de uso do serviço. Esse *lease* especifica por quanto tempo o *peer* está autorizado a usar os serviços do *rendezvous* antes de ter que renovar a conexão.

O *Peer Information Protocol* (PIP) permite que os *peers* possam obter informações sobre o estado de outros *peers*. Estas informações incluem o tempo de funcionamento, a quantidade de conexões, informações de tráfego e outras informações locais.

O *Peer Discovery Protocol* (PDP) define um protocolo cujo objetivo é definir como é feita a requisição de anúncios de outros *peers*, e como cada *peer* responde a uma requisição por um anúncio que foi recebida. Um *peer* descobre os recursos da rede através do envio de uma requisição a outro *peer*, usualmente um *peer Rendezvous*. Este *peer Rendezvous* é responsável por enviar a requisição adiante, e um *peer* que recebe uma requisição PDP e possui os anúncios procurados, envia a resposta diretamente ao *peer* requerente, através do ERP. O *peer* requerente recebe então um conjunto de respostas contendo anúncios que descrevem os recursos disponíveis na rede P2P.

Cada protocolo é semi-independente dos outros. A implementação em Java da plataforma JXTA disponibiliza a implementação de todos os seis protocolos especificados.

4.1.3 Algoritmos de Busca na Plataforma JXTA

Atualmente todo o algoritmo de busca definido pela plataforma JXTA está implementado no RVP, através do uso de DHTs, de forma que cada *rendezvous* possui uma lista interna de todos os outros *rendezvous* do grupo, ordenados por seu identificador. Esta lista é atualizada periodicamente através de troca de mensagens, de forma a manter uma certa consistência entre todas as listas existentes. A partir desta lista, é gerada então a tabela de *hash* distribuída (DHT), e cada *peer* é responsável por manter em *cache* um subconjunto das mensagens anunciadas.

4.2 A API JXTA-SOAP

A API JXTA-SOAP é uma biblioteca construída sobre a plataforma JXTA que possibilita a comunicação através do protocolo SOAP sobre a rede JXTA. Os objetivos principais são:

- O encapsulamento de *Web Services* como *JXTA Services*;
- O uso da plataforma JXTA para descoberta de *Web Services* e como plataforma para transporte de mensagens;
- Suporte a WSS e WSRF.

Atualmente, a API JXTA-SOAP é baseada na API Axis JAX-RPC, desenvolvida com o objetivo de permitir a invocação de *Web Services* entre plataformas heterogêneas. Um cliente JAX-RPC pode usar modelos de programação baseados invocação dinâmica a fim de invocar *Web Services*. Um dos requisitos do modelo JAX-RPC é que a invocação seja feita usando protocolo SOAP sobre HTTP. A API JXTA-SOAP se utiliza desses conceitos, porém sua implementação é feita usando o protocolo SOAP sobre a plataforma JXTA. É possível também que desenvolvedores construam extensões específicas para suportar segurança, anotação de atividades (*logging*) e várias outras facilidades. De modo particular, a biblioteca JXTA-SOAP utiliza *pipes* bidirecionais e o *multicast* a nível de aplicação da plataforma JXTA para transportar mensagens SOAP entre os TLMs e os MLMs.

A API JXTA-SOAP foi utilizada na implementação deste trabalho para a disponibilização dos Serviços de Gerenciamento. Os Serviços de Gerenciamento são implementados e disponibilizados à rede P2P como serviços JXTA, usando a API JXTA-SOAP. Um grupo de MLMs que fornece um Serviço de Gerenciamento é composto por diversos *peers* que disponibilizam o mesmo serviço JXTA-SOAP. Cada MLM possui uma instância de um serviço, e, ao ser feita uma requisição, todos os MLMs do grupo recebem a requisição através de um canal de comunicação *multicast* disponível a nível de aplicação pela plataforma JXTA. Cada instância de um serviço JXTA-SOAP disponível em um MLM é um Componente de Gerenciamento.

A biblioteca JXTA-SOAP usa a biblioteca Axis (YILDIZ; PALLICKARA, 2006) como *container* para os serviços. A biblioteca Axis é uma implementação do protocolo SOAP desenvolvida pela fundação Apache. O uso dessa API protege do programador os detalhes de ter que lidar diretamente com os protocolos SOAP e com a linguagem WSDL. O programador utiliza a biblioteca Axis no provedor do serviço a fim de implementar o *Web Service*, e usa a biblioteca Axis no consumidor dos serviços para facilitar a invocação dos serviços. A biblioteca Axis não possui apenas uma API para tratar *Web Services*, mas possui também um *container* completo que é responsável pelo gerenciamento do ciclo de vida de serviços em aplicações se baseiam na linguagem Java. É atualmente o *container* dominante entre as aplicações *Web Services*, e possui uma grande abundância de aplicações desenvolvidas em torno dele. Assim, com o uso da API JXTA-SOAP, disponibiliza-se ao programador a biblioteca Axis, e todo o ciclo de vida dos Componentes de Gerenciamento é gerenciado pelo *container* Axis, possibilitando assim um ciclo de vida idêntico ao dos *Web Services* tradicionais.

4.3 Implementação dos Serviços de Gerenciamento

Os Serviços de Gerenciamento podem ser acessados a partir de um TLM usando a API Axis JAX-RPC, e os Componentes de Gerenciamento podem ser disponibilizados

como *Web Services* nos containers de cada MLM. Porém, para facilitar ao programador o uso dos diversos componentes da plataforma, foi definida uma Interface de Programação (API) que os desenvolvedores de novos Componentes de Gerenciamento devem usar a fim de integrar seus Componentes de Gerenciamento com o Container de um *peer*. O uso dessa API permite que os desenvolvedores informem aos *Containers* quais Componentes de Gerenciamento devem ser instanciados em um MLM, para que dessa forma instanciem, inicializem e executem as operações dos Componentes de Gerenciamento que implementam os Serviços de Gerenciamento.

Para criar um Serviço de Gerenciamento, um desenvolvedor deve implementar um Componente de Gerenciamento. A implementação de um Componente de Gerenciamento começa pela codificação de uma ou mais classes Java, dentro dos padrões de Web Service definidos pela API JAX-RPC. Pela especificação, deve ser uma classe concreta que implementa a interface *ServiceLifecycle*. A interface *ServiceLifecycle* possui dois métodos que devem ser implementados: o método `init` e o método `destroy`. Esses dois métodos são usados pelo *container* para o gerenciamento do ciclo de vida do Componente de Gerenciamento. Os Componentes de Gerenciamento possuem um ciclo de vida bem definido, em que é especificado como o componente é carregado, instanciado, inicializado, como suas operações são chamadas e como é finalizado. As fases de carga e finalização são expressas na API através dos métodos `init` e and `destroy` da interface *ServiceLifecycle*.

O *Container* é responsável pela carga e instanciação dos Componentes de Gerenciamento. A carga e instanciação ocorrem quando o *Container* é iniciado. Primeiramente, o *Container* carrega a classe do Componente de Gerenciamento, usando os meios fornecidos pela linguagem Java para carga de classes. Depois de carregar cada classe dos Componentes de Gerenciamento, o *Container* instancia os objetos para serem usados. Após a instanciação, o *Container* precisa inicializar cada objeto de Componente de Gerenciamento antes de as operações serem chamadas. A inicialização ocorre a fim de que os Componentes de Gerenciamento possam localizar dados de configuração, inicializar recursos custosos e executar outras atividades de preparação.

O *Container* inicializa o objeto chamando o método `init` da interface *ServiceLifecycle*, passando como parâmetro um objeto de configuração que implementa a interface *ComponentConfig*. Esse objeto de configuração permite que o Componente de Gerenciamento acesse parâmetros de inicialização adquiridos a partir das informações de configuração do *peer*. Esse objeto de configuração também dá ao Componente de Gerenciamento acesso a um objeto que implementa a interface *ComponentContext* e que descreve o seu ambiente de execução. O relacionamento entre essas diversas interfaces, juntamente com um exemplo de implementação de um Componente de Gerenciamento, está detalhado na figura 4.3.

Durante a inicialização, pode ocorrer uma falha durante a execução de alguma tarefa de configuração do Componente de Gerenciamento, que poderá gerar uma exceção. Nesse caso, o Componente de Gerenciamento se torna indisponível, e é liberado pelo *Container*.

A fim de informar ao *container* que um determinado Componente de Gerenciamento deve ser instanciado em um MLM, o desenvolvedor deve informar as características do Componente a ser instanciado em um arquivo de configuração que é lido pelo MLM. O *descritor* é um documento XML que descreve quais Componentes de Gerenciamento devem ser instanciados, o nome e tipo de cada Componente de Gerenciamento e que classe Java deve ser usada. O *descritor* é organizado em elementos XML do tipo <management-component>, e cada elemento informa os parâmetros de inicialização que devem ser usa-

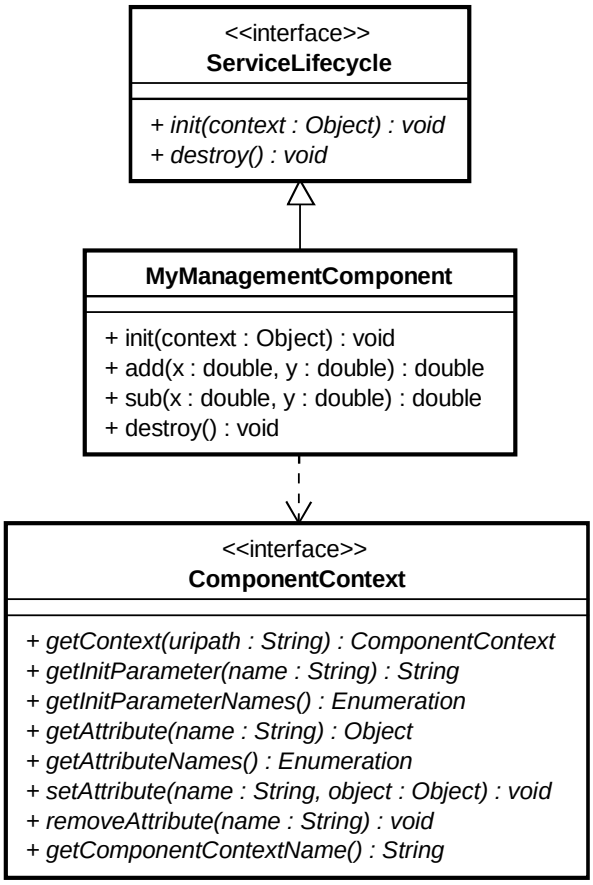

Figura 4.3: API de Componentes de Gerenciamento

dos para criar um objeto *ComponentConfig*, usado durante a inicialização do Componente de Gerenciamento. O *descritor* também contém um elemento que indica uma classe que é responsável por definir como o *Container* deve tratar o controle de acesso às operações dos Componentes de Gerenciamento. Essa classe confere as credenciais de usuário e verifica se o usuário possui acesso à execução das operações.

Outra forma de informar ao container as informações sobre um Componente de Gerenciamento que deve ser instanciado é através do uso direto da API ManP2P (4.4). Dessa forma, o desenvolvedor deve carregar os serviços programaticamente, através de chamadas ao método `loadComponent`, informando como parâmetro um objeto do tipo *ComponentDescriptor*, que é um objeto que possui em suas propriedades todas as informações sobre o Componente que deve ser disponibilizado.

O conjunto de operações oferecidas por um Componente de Gerenciamento é conhecido pelo *Container* depois que a classe do componente é carregada, e é gerado o arquivo WSDL correspondente. O *Container* usa a API *Java Reflection* para descobrir todos os métodos públicos da classe do componente, para que sejam usados como operações. A assinatura de cada método é usada como identificador da operação. Somente os métodos públicos que implementam a interface *ServiceLifecycle*, isto é, os métodos init e destroy, não são usados como operações.

O *Container* usa o conjunto de operações, as informações e os parâmetros de inicialização que estão no *descritor* para construir o descritor WSDL do serviço. Esse descritor é um documento XML que descreve todas as informações e funcionalidades do Componente de Gerenciamento e que formam os Serviços de Gerenciamento, e é assim publicado

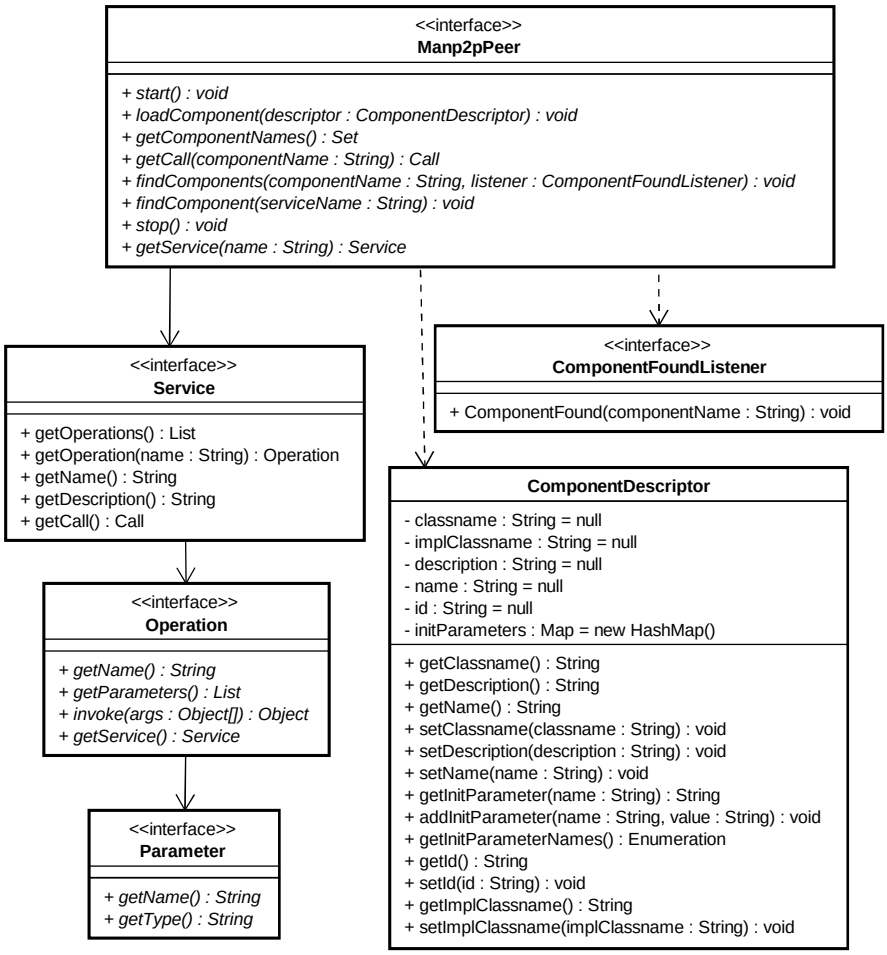

Figura 4.4: API ManP2P

na rede P2P usando a API da plataforma JXTA.

No *descritor*, existe também associado a cada Componente de Gerenciamento um identificador de serviço (também conhecido como identificador de grupo). Quando o Componente de Gerenciamento é instanciado, o *Container* verifica a existência de um grupo identificado pelo identificador de serviço acima mencionado, e insere o Componente de Gerenciamento nesse grupo, ou cria um novo grupo se não existir ainda. Todos os Componentes de Gerenciamento que entram em um mesmo grupo possuem as mesmas operações. São essas operações, expostas pelo *Container* e descritas pelo WSDL gerado, que formam os Serviços de Gerenciamento. Os Serviços de Gerenciamento são, dessa forma, fornecidos pelo grupo de todos os *peers* que possuem uma instância daquele Componente de Gerenciamento. Assim, grupos de *peers* aumentam a disponibilidade dos Serviços de Gerenciamento, pois os serviços de um grupo estão disponíveis ao sistema enquanto houver ao menos um *peer* ativo no grupo.

Para o programador fazer acesso aos Serviços de Gerenciamento em um MLM, devem ser acessados os métodos da API ManP2P. Através dessa API, os TLMs podem pesquisar, localizar e executar operações em Componentes de Gerenciamento remotos. É através dessa API também que um MLM pode ser instanciado e iniciado, através do método `start`, e parado, através do método `stop`.

4.4 Comunicação Dentro de um Grupo

Componentes de Gerenciamento são também capazes de se comunicarem com outros Componentes de Gerenciamento do mesmo grupo, a fim de fazer distribuição de carga, sincronização de estados e auto-monitoramento, verificando se o grupo possui um número razoável de membros ativos, de modo a assegurar que o grupo não ficará vazio e prevenindo a indisponibilidade dos serviços. Quando o número de membros cai a um número crítico, os TLMs são notificados, indicando que os Serviços de Gerenciamento de um grupo estão em perigo. O conjunto de *Containers* que possuem uma instância de um determinado Componente de Gerenciamento também formam um grupo, possibilitando, assim, não apenas comunicação direta entre Componentes de Gerenciamento, mas também comunicação entre os *Containers* que acomodam tais componentes. Nessa implementação, os grupos são controlados através dos *PeerGroup Services* da plataforma JXTA (GONG, 2001).

Através de grupos de Componentes de Gerenciamento, é possível fazer distribuição de carga entre as instâncias de Componentes de Gerenciamento de um mesmo grupo. Se apenas um *peer* possuir um *Container* que é responsável por instanciar um Componente de Gerenciamento a fim de fornecer seus Serviços de Gerenciamento, esse *peer* pode facilmente ter problemas de desempenho; contudo o uso de grupos possibilita a distribuição de tarefas. O *Container*, através de políticas de balanceamento de carga que são diferenciadas em cada grupo, verifica para que instância de Componente de Gerenciamento no grupo a requisição deve ser enviada, evitando que *peers* sobrecarregados recebam requisições que poderiam ser executadas por *peers* menos carregados. Dessa forma, os próprios *Containers* controlam a distribuição de tarefas, em contraste com os modelos de gerenciamento por delegação tradicionais, em que os TLMs são os responsáveis por essa distribuição. Algumas tarefas de gerenciamento como análise de tráfego, execução de scripts, e criação de relatórios fazem parte dessa categoria de tarefas que demandam uma grande quantidade de recursos computacionais.

Depois que um Componente de Gerenciamento foi corretamente inicializado e seus Serviços de Gerenciamento foram publicados, o *Container* usa suas operações para tratar as requisições dos TLMs. Os TLMs pesquisam pelas informações que foram publicadas pelos MLMs e chamam as operações dos Componentes de Gerenciamento através dos Serviços de Gerenciamento, através de chamadas RPC. O *Container* recebe as requisições e executa todas as tarefas relacionadas ao *Container*, como controle de acesso e balanceamento de carga. Depois disso, passa a chamada ao *container* AXIS, que é encarregado de gerenciar o ciclo de vida do Componente de Gerenciamento. O *container* AXIS passa os argumentos ao método adequado do Componente de Gerenciamento, fazendo uma chamada através da API *Java Reflection*. O objeto retornado pelo método é então enviado ao TLM como resposta à operação requisitada. Se uma exceção ocorre durante a execução do método, essa exceção é enviada como resultado ao TLM.

4.5 Outros Aspectos da API e Arquitetura de Gerenciamento Baseada em P2P

Funcionalmente, os Componentes de Gerenciamento funcionam como *gateways* de gerenciamento, mas possuem vantagens sobre outros mecanismos de *gateway*. A arquitetura e a API apresentadas possuem características que oferecem benefícios ao seu uso em gerenciamento de redes.

Um aspecto importante da arquitetura é que um Componente de Gerenciamento pode ser instanciado em diferentes *peers*, em diferentes *Containers*. Dessa forma, as múltiplas instâncias do mesmo Componente de Gerenciamento em diferentes *peers* oferecem tolerância a falhas e funcionalidades de distribuição de carga que são inerentes a todos os sistemas P2P.

Além disso, funções de controle de acesso são executadas através de implementações dentro dos *Containers*. Essas funções podem ser implementadas usando diversos métodos de validação, por exemplo, através de servidores LDAP. Tecnologias tradicionais, em contraste, são dependentes da implementação do protocolo, e em tecnologias como SNMP a restrição de acesso normalmente é feita usando a própria estrutura de rede, através de restrições em *firewalls* e roteadores.

Outros benefícios estão relacionados à linguagem de programação e as APIs usadas. A API de Componentes de Gerenciamento desenvolvida possui todas as vantagens da linguagem Java e das arquiteturas *Web Services*, incluindo facilidade de desenvolvimento e independência de plataforma. Usa também uma API padrão (JXTA) para utilização de serviços P2P, suportada por diversas plataformas P2P.

4.5.1 Conservação das Características de Sistemas P2P

Sistemas P2P possuem características bem definidas, e um sistema só pode ser considerado P2P se possui essas características. Pode-se usar uma plataforma P2P para comunicação, como a plataforma JXTA, mas o simples uso de uma plataforma P2P durante o desenvolvimento não garante que o sistema desenvolvido possui características P2P. Por isso, deve-se ter o cuidado de preservar todas as características que são inerentes a um sistema P2P durante o projeto.

Nesta seção, serão discutidas essas características e quais delas a implementação feita neste trabalho considera importantes e devem ser preservadas. As características P2P a serem discutidas são as seguintes:

Conectividade Em uma rede P2P, qualquer *peer* da rede pode se comunicar com qualquer outro, independentemente de localização ou de barreiras que possam existir na rede que possam dificultar essa comunicação. O simples uso do protocolo TCP em sistemas P2P torna-se insatisfatório devido à existência de *firewalls* e NATs. Em muitos sistemas, é usada uma rede sobreposta que se utiliza de roteadores de mensagens que podem garantidamente ser atingidos por qualquer nó da rede. A conectividade entre todos os nós da rede é então garantida através da garantia de conectividade com qualquer um desses nós roteadores existentes na rede sobreposta. Porém essa garantia só é válida se qualquer nó da rede conseguir efetuar uma conexão TCP com algum dos nós roteadores. Em algumas subredes, isso não é possível, devido ao uso de *proxies* HTTP. Nesses casos, os nós internos à rede não conseguem fazer diretamente nenhuma conexão TCP com qualquer nó externo, e a única maneira de acessar a rede Internet é através do uso de um servidor *proxy*. É pensando nisso que muitos sistemas usam roteadores que fazem roteamento de mensagens sobre o protocolo HTTP, ao invés de usar roteamento diretamente sobre o protocolo TCP. Essa abordagem permite que nós que estejam dentro de uma rede cujo acesso externo é feito através de *proxy* HTTP possam participar normalmente de um sistema P2P. No caso do sistema ManP2P, a conectividade é garantida através do uso da plataforma JXTA. A plataforma JXTA possibilita a configuração de roteadores que recebem conexões tanto em nível de protocolo TCP quanto em nível

de protocolo HTTP. A conectividade entre todos os nós é então garantida mesmo em redes com presença de *firewall*, NAT ou servidor *proxy* HTTP.

Heterogeneidade Em uma rede P2P, um *peer* pode ser desde um simples dispositivo de mão, como um PDA, quanto um servidor multiprocessado e com *GigaBytes* de memória RAM. Todos participam da rede do mesmo modo, consumindo serviços e também oferecendo serviços à rede. A quantidade de recursos de um determinado *peer* pode ser limitada, oferecendo apenas um limitado conjunto de serviços, porém não é por esse motivo que ele deixará de ser considerado um *peer*. As formas de acesso dos *peers* à rede também podem ser bem diferenciadas, desde *peers* que acessam a rede através de uma rede local cabeada até *peers* que acessam a rede através de uma conexão sem fio. Uma rede P2P é considerada heterogênea, tanto considerando a configuração dos *peers* quanto a configuração da rede em si.

Disponibilidade Os recursos que são disponibilizados a uma rede P2P são sempre redundantes. Tomando como exemplo um sistema de compartilhamento de arquivos, um *peer* que se conecta à rede e procede à requisição de um determinado arquivo, quando encontrá-lo na rede vai receber uma referência ao arquivo, juntamente com dezenas e até milhares de fontes que contém esse arquivo. O *peer* pode então proceder ao acesso ao recurso, em qualquer um dos *peers* que disponibilizam esse recurso, muitas vezes o fazendo de modo simultâneo em mais de um *peer*. Dessa forma, se qualquer um dos *peers* se tornar indisponível, isso não significa que o recurso se torna indisponível, pois outras fontes podem ser usadas.

Escalabilidade Em uma rede com recursos centralizados, como no modelo cliente-servidor, o tamanho da rede impacta significativamente sobre o modo de operação da mesma. Considerando que o servidor tem uma quantidade finita de recursos a serem disponibilizados aos clientes, quanto maior o número de clientes, maior deve ser essa quantidade de recursos, sob pena de que os clientes fiquem sem recursos disponíveis. Em um sistema muito grande, com um número de clientes da ordem de milhões, torna-se necessário o uso de *clusters* gigantescos e uma infra-estrutura muito grande, o que pode tornar muitos sistemas inviáveis. A quantidade de recursos disponíveis a cada cliente é inversamente proporcional ao número de clientes da rede - quanto maior o número de clientes, menor o número de recursos que cada cliente pode utilizar. No caso dos sistemas P2P, essa relação se inverte. Como cada *peer* que entra na rede traz recursos à mesma, os recursos são somados, e a entrada de novos *peers* aumenta a capacidade total de recursos do sistema. Considerando que, na grande maioria dos *peers* de um sistema P2P, a quantidade de recursos que são fornecidos à rede é bem maior que a quantidade de recursos consumidos, a quantidade de recursos disponíveis a cada *peer* é proporcional ao número de *peers* existentes - quanto maior o número de *peers*, maior o número de recursos que cada *peer* pode utilizar. Isso torna os sistemas P2P potencialmente escaláveis por natureza.

A característica de conectividade é mantida no sistema ManP2P, pois a plataforma P2P usada, a plataforma JXTA, é bem madura em relação a esse aspecto. Prevê a existência de *peers* protegidos por *firewalls*, NATs e até mesmo prevê a existência na rede de *peers* que só podem ser acessados através de outros protocolos de comunicação diferentes do TCP/IP, como por exemplo *BlueTooth*. Devido à possibilidade de qualquer TLM ser um dispositivo simples, como um PDA, pois a plataforma JXTA possui uma versão

J2ME para dispositivos portáteis, pode-se considerar que o sistema ManP2P é um sistema heterogêneo. Enquanto os TLMs podem estar em qualquer dispositivo, desde computadores portáteis até computadores de mesa ou servidores, os MLMs também podem estar disponíveis nesse tipo de dispositivo, devido à simplicidade com que um Serviço de Gerenciamento pode ser disponibilizado.

Porém, o uso de uma plataforma P2P como o JXTA não confere diretamente ao sistema as características de disponibilidade e escalabilidade. A plataforma fornece ferramentas para que o projeto tenha essas características, porém é tarefa do projetista garantir essas características através de mecanismos definidos no projeto do sistema. Por isso, as características de disponibilidade e escalabilidade somente podem ser adquiridas pelo sistema através do uso de distribuição de carga, que será detalhada no próximo capítulo.

5 DISTRIBUIÇÃO DE CARGA

Um dos grandes assuntos discutidos em sistemas distribuídos é o desenvolvimento de técnicas efetivas para distribuição de processos em vários computadores. O objetivo da distribuição de carga é aumentar o desempenho de um sistema distribuído, normalmente em termos de tempo de resposta ou em termos de disponibilidade, através da realocação de tarefas e da carga de trabalho entre um conjunto de nós que trabalham de modo cooperativo, minimizando atrasos e maximizando a utilização de recursos. Esta divisão de carga pode ser feita de modo estático ou de modo dinâmico.

Distribuição estática distribui nos nós as tarefas de modo probabilístico ou determinístico, sem considerar eventos que estão acontecendo durante a execução dessas tarefas. Essa proposta pode ser considerada simples e efetiva quando a carga de trabalho pode ser bem caracterizada e quando o ambiente de execução das tarefas é bem controlado ou bem caracterizado e consistente. Problemas poderão surgir se a carga no ambiente de execução tende a ter flutuações, ou se existirem processos fora do controle do distribuidor de carga.

Distribuição dinâmica (SHIVARATRI; KRUEGER; SINGHAL, 1992) é definida para superar os problemas de cargas desconhecidas ou não caracterizáveis e também problemas de variações no ambiente de execução (situações como indisponibilidade de nós, composição de cargas ou interação com usuário, que possam alterar os requerimentos do sistema ou a disponibilidade deste). Sistemas com distribuição dinâmica de carga normalmente monitoram sua carga e a de outros nós a fim de encontrar quaisquer fatores que possam afetar a escolha de qual nó seria mais apropriado para a melhor distribuição. Essa diferença entre as formas dinâmica e estática de distribuição de carga é a principal fonte de interesse na distribuição dinâmica de carga.

Deve-se considerar também que, além das soluções dinâmicas, que levam em consideração variáveis do ambiente em suas decisões, há também soluções adaptativas, que levam em conta os estímulos do ambiente a fim de modificar as próprias políticas de escalonamento em si (CASAVANT; KUHL, 1988).

5.1 Graus de distribuição de Carga

Para se fazer a divisão da carga de trabalho entre um grupo de nós que cooperam, a distribuição pode ser feita usando graus variados de granularidade, e a escolha exata de qual opção depende do ambiente e arquitetura envolvidos no sistema em questão.

Na literatura, distribuição de carga é normalmente dividida em balanceamento de carga ou compartilhamento de carga (CASAVANT; KUHL, 1988). Esses dois termos muitas vezes são usados sem muita diferenciação, mas podem também ter definições distintas. Um terceiro termo - nivelamento de carga - também pode ser encontrado.

Compartilhamento de carga é a forma mais grosseira e simples de distribuição de

carga. Um nó poderá ser visto apenas de duas formas, ou como ocupado ou como desocupado, e a carga será somente colocada em nós desocupados.

Balanceamento de carga é a forma com granularidade mais fina e também a forma mais complexa de distribuição de carga. O objetivo do balanceamento de carga é garantir uma equalização na carga existente em qualquer nó do sistema.

Nivelamento de carga ocupa o espaço entre os dois extremos - compartilhamento e balanceamento. Não se preocupa em garantir uma distribuição regular de carga entre todos os nós, mas também não se limita a simplesmente utilizar nós desocupados. O nivelamento de carga normalmente se preocupa apenas em evitar o congestionamento ou sobrecarga em algum dos nós.

Sistemas com compartilhamento de carga são comuns (DOUGLIS; OUSTERHOUT, 1991; NICHOLS, 1987), porém a distinção entre balanceamento de carga e nivelamento de carga é um pouco mais difícil. Em particular, para satisfazer a definição, um esquema de balanceamento de carga precisa continuar a distribuir a carga até que um critério de balanceamento seja atingido. Um exemplo de critério de balanceamento é considerar que o sistema esteja balanceado apenas se a diferença de carga entre os nós mais e menos carregados esteja dentro de um limite (KARA, 1995). Outros sistemas, embora possam ser considerados de balanceamento de carga, utilizam de fato nivelamento de carga, pois a fase de balanceamento ocorre apenas periodicamente (BARAK; SHILOH, 1985).

A arquitetura do sistema é importante, pois pode sugerir qual o grau mais apropriado de distribuição de carga. Por exemplo, em uma grande rede de estações, poderia ser proibitivamente caro o balanceamento, devido ao custo de se coletar informações de carga e estado, enquanto que a simples detecção e utilização de estações desocupadas para compartilhamento de carga seria muito mais fácil de ser feita (NICHOLS, 1987).

Dessa forma, compartilhamento, nivelamento e balanceamento de carga definem uma linha contínua desde a mais grosseira até a mais fina forma de distribuição de carga, e procuram distinguir entre as intenções não explícitas dos diferentes esquemas de distribuição.

5.2 Taxonomia da distribuição de carga

Políticas de distribuição de carga foram classificadas de diferentes formas, permitindo diferentes critérios de classificação (CASAVANT; KUHL, 1988). Existem inúmeras alternativas de combinações de políticas diferentes: balanceamento de carga vs. compartilhamento de carga, algoritmos centralizados vs. algoritmos distribuídos, algoritmos estáticos vs. algoritmos dinâmicos. Porém apenas parte desse conjunto de políticas é adequado para ser usado no caso de requisições para execução de tarefas de gerenciamento.

Existem diversas taxonomias disponíveis para a classificação da distribuição de carga, incluindo Wang e Morris (WANG; MORRIS, 1985), Casavant e Kuhl (CASAVANT; KUHL, 1988) e Jacqmot e Milgrom (JACQMOT; MILGROM, 1993). Wang e Morris propuseram uma classificação simples baseada no fato de a distribuição ser iniciada na fonte ou no destino, e no grau de informação necessária para as políticas. Essa classificação, porém, não oferece meios suficientes de comparação entre políticas de distribuição de carga que são similares nas duas classes. Casavant e Kuhl oferecem um esquema de classificação mais geral, que inclui escalonamento local e global, porém não facilita uma comparação detalhada entre os algoritmos de cada classe. Jacqmot e Milgrom possuem uma proposta totalmente diferente, onde a ordem em que as decisões de distribuição são feitas é significante. Essa taxonomia porém não encoraja a comparação ou identificação

dos componentes de um esquema de distribuição de carga. Também não classifica simetricamente políticas de inicialização, nem distingue entre transferência preemptiva e não-preemptiva de processos.

Todas as taxonomias mencionadas anteriormente carecem de meios para discutir de maneira clara e sucinta os esquemas de distribuição de carga. Porém, uma taxonomia apresentada por Bubendorfer (KRIS, 1996) se concentra nas decisões individuais feitas durante a distribuição de carga e também identifica os diferentes componentes de um esquema de distribuição de carga.

Esquemas de distribuição de carga normalmente se utilizam de políticas e mecanismos. Políticas são os conjuntos de escolhas feitas para a distribuição da carga. Mecanismos fazem a distribuição física da carga e fornecem as informações necessárias para o funcionamento das políticas.

As políticas podem se dividir em:

1. Política de participação. Chamada também de política de aceitação. Determina de que forma um nó participa da distribuição de carga. Pode ser simples, como a definição de um limite baseado na carga local, ou mais complexa, como uma política que define que processos provindos de nós não confiáveis sejam recusados.

2. Política de seleção da localização. Responsável por selecionar os nós participantes entre os quais a carga será distribuída.

3. Política de seleção de candidatos. É a política que define quais são as tarefas, processos, objetos ou unidades de trabalho que devem ser distribuídos. Pode ir de razões desde a quantidade insuficiente de recursos no nó corrente, até a redução de caminhos de comunicação.

Os mecanismos usados são mecanismos para representação da carga de trabalho, mecanismos de comunicação de carga e mecanismos de transferência de tarefas. A seguir, são discutidos com mais detalhes cada um desses mecanismos.

5.2.1 Representação da Carga de Trabalho

A informação de carga é tipicamente representada por um índice de carga, que é uma medida quantitativa da carga de cada processo. Yoshihara et al. propõem dois algoritmos (YOSHIHARA; ISOMURA; HORIUCHI, 2003) para adquirir esse índice. O primeiro é baseado na diferença absoluta entre a utilização da CPU para cada processo e a média de utilização de CPU em todos os processos. O segundo é baseado na diferença entre as utilizações máximas e mínimas de CPU em todos os processos. A atualização periódica dos índices de carga é necessária, porém o uso de períodos muito curtos de atualização pode gerar muito tráfego, diminuindo as vantagens de se ter índices atualizados. Da mesma forma, longos períodos de atualização podem tornar os índices de carga obsoletos.

5.2.2 Mecanismos de Comunicação de carga

Em um ambiente cujas informações estão distribuídas entre os processos, cada processo guarda localmente sua própria imagem da carga do sistema. Para que os outros processos conheçam as suas informações de carga, o processo passa as suas informações a outros processos em eventos específicos ou durante determinados intervalos de tempo. Pode-se também usar uma política de distribuição que não envia informação local a outros processos, como é o caso de escalonamento aleatório de tarefas (JOSHI; HOSSEINI;

VAIRAVAN, 1993), em que o processo a executar a tarefa é escolhido de forma aleatória entre o conjunto de processos disponíveis. Esse balanceamento aleatório de carga funciona bem quando a carga de todos os processos está relativamente alta, isto é, quando não faz muita diferença onde a tarefa está sendo executada. Porém pode não ser uma boa alternativa para o caso em que se quer obter uma boa distribuição com menor tempo de resposta enquanto os processos estão com uma baixa carga.

5.2.3 Transferência de Tarefas

A transferência preemptiva de tarefas envolve a transferência de uma tarefa parcialmente executada. Transferência não-preemptiva de tarefas, por outro lado, envolve apenas tarefas que ainda não começaram sua execução e dessa forma não necessitam de transferência de seu estado.

Thrashing ou instabilidade é um problema que precisa ser levado em consideração em esquemas de balanceamento de carga que usam transferência preemptiva de tarefas. Esse problema surge da possibilidade de muitos *peers* altamente carregados transferirem suas cargas para um *peer* com baixa carga, fazendo com que o último fique muito carregado e tenha que transferir novamente essas tarefas para outro *peer*, causando assim um efeito cascata.

5.3 Implementação da Distribuição de Carga

Na plataforma ManP2P, todos os MLMs que estão disponíveis em um determinado grupo e que oferecem um determinado Serviço de Gerenciamento participam da distribuição de carga para este Serviço de Gerenciamento. A política de participação usada se resume à aceitação de qualquer MLM que se junte ao grupo que disponibiliza um determinado Serviço de Gerenciamento. A seleção da localização, por sua vez, é bem mais complexa, e depende da implementação do algoritmo de distribuição de carga escolhido. Dentre os algoritmos implementados, cada um possui sua própria política de seleção da localização. Já a política de seleção de candidatos é simples, pelo fato de que todas as requisições que são feitas entre os MLMs são distribuídas, e a distribuição é iniciada no MLM e não no TLM - isto é, no destino, e não na fonte. Todas as tarefas que chegam ao destino são distribuídas, independente de que tipo de tarefa está sendo requisitada.

Em relação ao cálculo do índice de carga, os algoritmos estáticos de distribuição de carga usados na plataforma não necessitam desse cálculo, pois nenhuma informação de carga está envolvida no processo de seleção. É o caso dos algoritmos de escolha aleatória e de fila circular, onde a distribuição das tarefas de trabalho é independente da informação atual de carga do MLM. O algoritmo de escolha pelo menor número de conexões usa como informação de carga o número de requisições que estão sendo servidas pelo MLM. Já os algoritmos de escolha pela menor carga exigem um cálculo mais complexo do índice de carga, que pode envolver informações de uso de memória, tempo de processamento ou acesso a disco.

Quanto aos mecanismos de comunicação da informação de carga, existem eventos específicos em que informações precisam ser trocadas entre os MLMs. Entre esses eventos, como será explicado a seguir, estão o momento da passagem de *token* e o momento em que o MLM informa que o processamento de uma tarefa foi finalizado. Nesses eventos, a informação de carga pode ser transferida entre os MLMs. No caso de algoritmos em que não há informação de carga a transferir, nenhuma informação é adicionada aos eventos. Porém, quando há informação de carga a ser transferida, esses eventos são o momento

oportuno para a transferência dessa informação.

Na plataforma ManP2P, optou-se por não implementar a transferência preemptiva de tarefas, pois muitas das tarefas envolvem situações em que o MLM deve ficar esperando por respostas de dispositivos distribuídos na rede. Nesse caso, uma tarefa parcialmente executada - uma requisição que foi enviada a um dispositivo e está esperando por uma resposta, por exemplo - não pode ser executada por outro MLM, pois somente o MLM que iniciou a tarefa pode finalizar a mesma. Essa transferência preemptiva de tarefas só poderia ser realizada se todo o estado de execução da tarefa de gerenciamento estivesse disponível em um único lugar e pudesse ser recolhida pelo MLM, o que não é o caso.

A interface para implementação dos esquemas de distribuição de carga utilizados pela plataforma ManP2P foi escolhida de forma a não oferecer ao sistema uma única política de distribuição. A escolha de um único esquema de distribuição de carga para todo e qualquer componente de gerenciamento do sistema poderia dificultar a utilização de determinados componentes dependendo de suas características. Por isso foi determinado que o esquema de distribuição de carga pode ser definido pelo usuário, juntamente com a definição do Componente de Gerenciamento, no arquivo de descrição dos Componentes de Gerenciamento.

Na figura 5.1 é listada uma parte do arquivo de descrição dos Componentes de Gerenciamento, em que o usuário informa à plataforma qual algoritmo de distribuição de carga deve ser usado para um determinado componente. O usuário informa a classe de implementação do algoritmo através do parâmetro `LoadDistributer`. A interface JAVA também possui o nome *LoadDistributer*, e está representada na figura 5.2. Essa interface fornece os métodos para a implementação das primitivas necessárias para o envio e recebimento de mensagens e para a escolha do MLM que deverá processar a requisição.

```
<management-component>
    <name>
        ProcessingService LeastConnectionsFirst
    </name>
    <description>
        Carga de processamento - LeastConnectionsFirst
    </description>
    <id>uuid-A-UNIQUE-ID</id>
    <classname>package.ProcessingComponent</classname>
    <init-param>
        <param-name>LoadDistributer</param-name>
        <param-value>
            package.LeastConnectionsFirst
        </param-value>
    </init-param>
</management-component>
```

Figura 5.1: Extrato do arquivo de descrição dos Componentes de Gerenciamento

Várias políticas podem ser definidas através do uso de algoritmos de distribuição, e o usuário poderá definir para cada Componente de Gerenciamento as suas políticas e o seu algoritmo diferenciado, de forma a possibilitar o uso das políticas que se adaptarem melhor às necessidades do componente de gerenciamento em questão, dependendo de seu tipo, seu comportamento, e número de nós em que estará disponível.

Alguns esquemas de distribuição estão disponíveis para uso na implementação da plataforma, porém se nenhum dos esquemas for o mais otimizado para o componente em

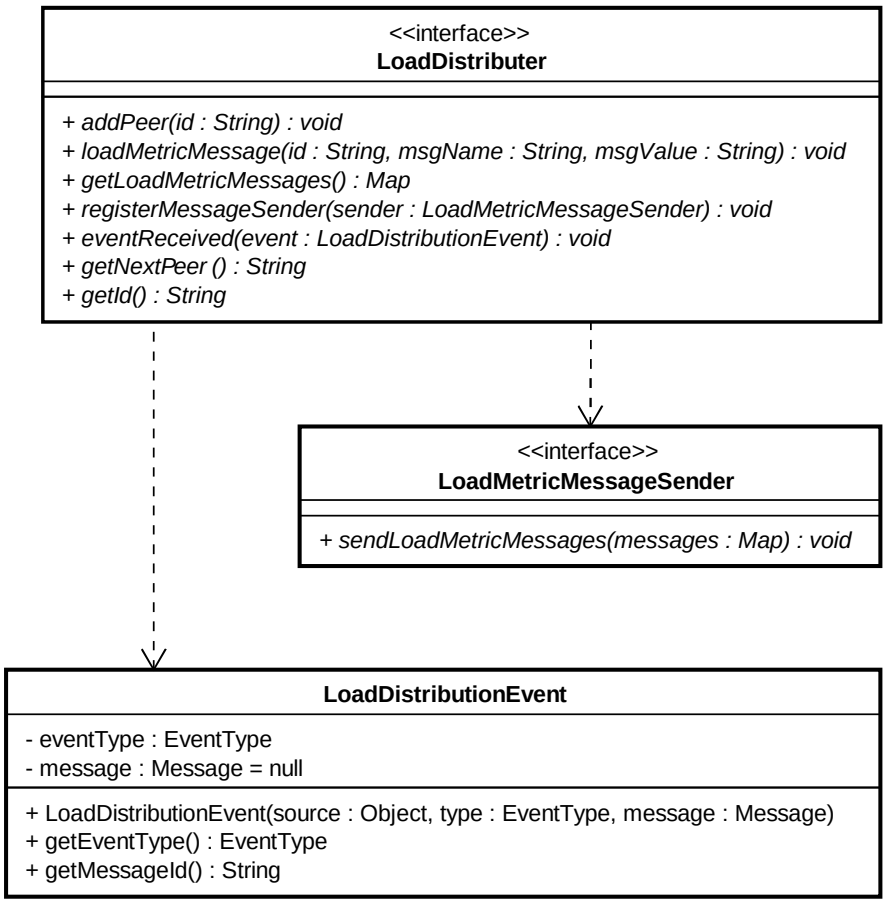

Figura 5.2: Interface LoadDistributer

questão, o usuário poderá disponibilizar sua própria implementação de um esquema de distribuição, através da implementação da interface *LoadDistributer*.

5.3.1 Especificação do Protocolo de Distribuição de Carga

Neste trabalho foi usado um protocolo de distribuição de carga que usa o conceito de passagem de *token*, que é especificado nesta seção. Para isso, são assumidas as seguintes características do sistema:

- Cada *MLM* possui um identificador único, na forma de um *Uniform Resource Name* (URN). O objetivo de uma URN é servir como um identificador independente de localização.

- Cada Serviço de Gerenciamento também possui um identificador único, na forma de um URN.

- Os MLMs que oferecem um determinado Serviço de Gerenciamento participam de um grupo. MLMs que entram na rede e querem participar do grupo, oferecendo o Serviço de Gerenciamento que identifica o grupo, devem entrar no grupo através da primitiva `JoinGroup`. Se um MLM entra na rede oferecendo um serviço que ainda não está disponível, esse MLM será o primeiro a participar do grupo, fazendo assim a inicialização desse grupo.

- Mensagens são trocadas entre os MLMs do grupo através das primitivas `Send` e `SendToGroup`. A primeira envia uma mensagem a um MLM específico, enquanto a segunda envia uma mensagem a todos os MLMs do grupo.

- As requisições para execução de tarefas de gerenciamento são enviadas aos MLMs do grupo através de *multicast* a nível de aplicação.

O protocolo usado para distribuição de carga usa passagem de *tokens* entre os MLMs. Para o protocolo de passagem de *token*, considera-se que no momento da criação de um grupo, é criado um único *token*, que estará sempre sob a propriedade de apenas um MLM do grupo.

Quando um MLM entra em um grupo, usa a primitiva `JoinGroup`. Se já existem MLMs no grupo, significa que o *token* já foi criado e já está com algum outro MLM do grupo. Se ainda não existe nenhum MLM no grupo e este precisa ser inicializado, o *token* é criado e fica sob a responsabilidade do MLM que fez a inicialização do grupo.

No caso de dois MLMs entrarem na rede ao mesmo tempo, sem que o grupo ainda exista, e a inicialização do grupo for feita pelos dois, pode haver um problema de duplicação de *tokens* e de disjunção do grupo, que deve ser resolvido através de mensagens posteriores de descoberta de MLMs que estão tentando responder pelo mesmo grupo. Porém a implementação dessas mensagens não foi considerada nesse trabalho.

No momento da chegada de uma requisição ao grupo, o MLM que possui o *token* passa imediatamente à execução da tarefa. Dessa forma, o tempo de resposta para a requisição não é prejudicado pelo uso de distribuição de tarefas. Ao mesmo tempo, uma segunda *Thread* inicia o processo de escolha do novo MLM que deverá atender à requisição seguinte. Esse processo de escolha é feito no MLM que possui o *token*, e por isso depende exclusivamente das informações locais ao MLM, que foram enviadas anteriormente a ele pelos outros MLMs. Escolhido o novo MLM, o *token* é passado adiante, e um evento de passagem de *token* é gerado, de forma que as informações de carga do MLM que está enviando o *token* possam ser compartilhadas com os outros MLMs.

A escolha de um protocolo que usa passagem de *tokens* entre os MLMs exige uma implementação que envolve a solução de diversos problemas relacionados à perda ou duplicação do *token*. Porém, como essa implementação foi usada apenas para testes de desempenho dos algoritmos de distribuição, os casos de uso levaram em conta uma perspectiva otimista em relação a esses problemas, e a implementação da solução desses problemas não foi considerada nesse trabalho.

5.3.2 Problemas Relacionados à Distribuição de Carga

Há muitos problemas que podem ocorrer em esquemas de distribuição de carga. Algumas das características críticas que devem ser levadas em conta são as seguintes:

- O *overhead* causado pela transferência de informações entre *peers* quando se usa um esquema preemptivo pode tornar o desempenho inaceitável se for muito grande, ou se as tarefas tiverem tempo de execução muito rápido se comparado ao tempo de transferência de tarefas de um *peer* ao outro.

- O processo de decisão para a seleção do nó ao qual a tarefa será transferida pode ser de uma complexidade computacional muito grande.

- Os atrasos de comunicação devido à realocação de tarefas.

- A intensidade de tráfego gerada pelo protocolo escolhido e pela troca de mensagens entre os processos envolvidos.

Alguns problemas, como os atrasos de comunicação devido à realocação de tarefas, não se aplicam à plataforma ManP2P. O atraso devido à realocação de tarefas ocorre quando uma tarefa que foi recebida por um determinado nó deve ser realocada e transferida a um outro nó, com menos carga. Esse problema não ocorre nessa plataforma, porque quando um TLM faz uma requisição, é usado um mecanismo de *multicast* a nível de aplicação da plataforma P2P, que faz com que todos os nós envolvidos no processo de distribuição recebam a solicitação da execução da tarefa. Nesse caso, quando é terminado qual MLM deverá executar a tarefa, a requisição para a tarefa já se encontra no MLM escolhido, e este passa diretamente à fase de execução da tarefa.

O processo de decisão para a seleção do MLM ao qual a tarefa será transferida depende muito do algoritmo implementado, mas normalmente depende de informações que foram anteriormente trocadas entre MLMs do grupo, tornando-se assim informações locais ao MLM no momento da decisão.

A intensidade de tráfego gerada pelo protocolo escolhido e pela troca de mensagens entre os processos envolvidos ainda é um problema, pelo fato de que as mensagens no protocolo JXTA são documentos XML com diversas informações. O *overhead* causado pelo uso da plataforma JXTA é bastante grande, e torna-se menor quando a quantidade de dados a serem transferidos é maior. Porém, para mensagens curtas de transferência de informação de carga, é um *overhead* muito grande. É por esse motivo que essas mensagens são enviadas em eventos específicos, no momento em que outras informações também devem ser enviadas, a fim de agrupá-las e minimizar o *overhead* causado pelo uso do protocolo JXTA.

5.4 Algoritmos de Distribuição de Carga

A implementação realizada neste trabalho considera que o algoritmo de cada Serviço de Gerenciamento pode ser definido pelo usuário. Uma implementação de algoritmo de distribuição de carga é uma classe Java concreta que deve implementar a interface `LoadDistributer`, representada na figura 5.2. A implementação dessa interface obriga o programador a implementar a primitiva `GetNextPeer`, que retorna qual deverá ser o próximo MLM a receber uma tarefa, conforme o algoritmo implementado. Numa distribuição aleatória, por exemplo, a primitiva `GetNextPeer` retornará aleatoriamente um MLM do grupo. Já em uma distribuição dinâmica que leva em conta a carga de cada MLM, a primitiva `GetNextPeer` retornará o MLM com a menor carga no momento da execução da primitiva.

Nesta seção serão descritos os algoritmos de distribuição de carga que foram implementados para uso na plataforma ManP2P. Para a implementação das primitivas, cada classe é uma implementação da interface `LoadDistributer`, representada na figura 5.2. Na instância de cada classe existe um objeto do tipo *SortedSet*, que possui um conjunto ordenado de *Strings* que representam os identificadores dos MLMs que participam do grupo. Esse objeto é preenchido inicialmente apenas com o identificador local do MLM, mas à medida que mensagens são trocadas e outros MLMs entram no grupo, o conjunto vai sendo preenchido, juntamente com outras listas de informações referentes a cada MLM.

5.4.1 Algoritmo de Escolha Aleatória

O algoritmo de escolha aleatória é um algoritmo que pode ser classificado como um algoritmo estático, pois não leva em conta a carga atual de cada MLM no momento do recebimento da requisição. Apesar disso, resulta em uma boa distribuição das tarefas, quando a escolha é feita de maneira homogênea entre todos os MLMs do grupo. É importante notar também que pode ter um mau resultado no caso em que o grupo de MLMs é bastante heterogêneo.

A implementação do algoritmo de escolha aleatória é bastante simples pois não leva em consideração variáveis do ambiente de execução. Pode ter um bom desempenho em relação à troca de mensagens entre os MLMs, pois há troca de um número menor de mensagens em relação a outros algoritmos. Na inicialização do grupo, os MLMs inicialmente concordam com uma lista que possui uma seqüencia aleatória de MLMs que participam do grupo. Ao receber uma nova requisição a ser processada, cada MLM verifica localmente qual é o próximo MLM da lista. Essa lista é criada na inicialização do grupo, e é atualizada sempre no momento da entrada de um novo MLM no grupo e no momento da saída de um MLM do grupo. É uma lista de tamanho finito, portanto também precisa ser atualizada quando o último elemento da lista for usado em uma requisição.

Para facilitar e tornar mais rápida a concordância de todos os MLMs com uma única seqüencia aleatória, essa seqüencia é sempre gerada por um único MLM: aquele que está com o *token* de execução. A seqüencia é gerada - ou atualizada, em caso de entrada de novo MLM, saída de um ou de término da seqüencia - e enviada a todos os MLMs do grupo, de forma que a seqüência criada imediatamente passa a ser a nova seqüência válida para o grupo.

A implementação em Java da primitiva `GetNextPeer` apenas se resume à obtenção do MLM seguinte na lista que foi previamente concordada, guardada localmente no objeto `peerSequence`, do tipo *Queue*, como segue na listagem da figura 5.3.

```
private SortedSet<String> peers = new TreeSet<String> ();
private Queue<Integer> peerSequence;

public String getNextPeer() {
    Integer nextPeer = peerSequence.poll(); //acessa o próximo elemento da fila
    return peers.toArray(new String[peers.size()])[nextPeer.intValue()];
}
```

Figura 5.3: Listagem da implementação do Algoritmo de Escolha Aleatória

A geração da seqüencia é simples, conforme na listagem da figura 5.4.

```
private Queue<Integer> getSequence() {
    Queue<Integer> sequence = new LinkedList<Integer>();
    for (int i=0; i<100; i++) {
        Double next = (Math.random() * peers.size());
        sequence.offer(new Integer(next.intValue()));
    }
    return sequence;
}
```

Figura 5.4: Listagem da implementação da fila para Escolha Aleatória

Para resolver o problema de grupos bastante heterogêneos de MLMs, é possível fazer uma modificação no algoritmo, de forma que os MLMs que estiverem menos sobrecarregados recebam mais requisições do que os outros. Essa modificação implica que periodicamente haja troca de mensagens entre os MLMs informando sua carga atual, para que essa informação possa ser usada posteriormente, no momento da geração da seqüencia.

5.4.2 Algoritmo de Fila Circular

O algoritmo de fila circular também é um algoritmo que pode ser classificado como um algoritmo estático, não levando em consideração a carga atual de cada MLM no momento do recebimento da requisição.

Nessa implementação, considera-se que cada MLM, ao entrar no grupo, participa de um anel lógico, e o próximo MLM a executar cada requisição é o MLM seguinte na ordem definida por esse anel lógico. Da mesma forma que no algoritmo de escolha aleatória, esse algoritmo não envolve troca de mensagens entre os MLMs a cada recebimento de requisição para concordar sobre qual é o próximo MLM a executar a requisição. Por isso, pode também ter um bom desempenho em relação à troca de mensagens entre os MLMs, pois possui um número reduzido de mensagens em relação a outros algoritmos.

A implementação em Java da primitiva `GetNextPeer` se resume à obtenção do MLM seguinte na ordem dos MLMs. Porém, para cada requisição recebida, o MLM precisa guardar qual foi o MLM que respondeu à requisição anterior, como mostra listagem da figura 5.8.

```
private SortedSet<String> peers = new TreeSet<String> ();
private String actual;

public String getNextPeer() {

    String next = null;

    try {
        // retorna o peer posterior ao ultimo acessado
        next = peers.tailSet(actual+"\0").first();
    } catch (NoSuchElementException e) {
        next = peers.first();
    }

    return next;
}
```

Figura 5.5: Listagem da implementação do Algoritmo de Fila Circular

5.4.3 Algoritmo de Escolha pelo Menor Número de Conexões

O algoritmo de escolha pelo menor número de conexões leva em conta o número de conexões, ou requisições, que estão sendo atendidas pelo MLM. Para a implementação de um algoritmo desse tipo, a única informação que precisa ser trocada entre os MLMs é o número de requisições que estão sendo atendidas por cada MLM em um determinado momento.

O número de requisições que estão sendo atendidas por cada MLM é conhecido por todos os outros MLMs, e é guardado internamente em uma lista. Na chegada de uma nova

conexão, o MLM que possui o *token* consulta a lista, para saber qual é o MLM que possui o menor número de conexões sendo atendidas. Caso haja um empate, será escolhido o MLM com precedência em um determinado ordenamento. No caso dessa implementação, esse ordenamento é dado pela ordem lexicográfica dos identificadores dos MLMs.

A primitiva `GetNextPeer` deverá então retornar o *peer* que estiver atendendo ao menor número de requisições. Dessa forma, todos os *peers* passam a atender em média o mesmo número de conexões, o que pode ser bom em sistemas bastante homogêneos, em que os *peers* possuem aproximadamente a mesma capacidade de processamento.

```
private SortedSet<String> peers = new TreeSet<String> ();
private Map<String, Integer> connections = new HashMap<String,Integer>();

public String getNextPeer() {

    String nextPeerId = peers.first();
    Integer nextPeerConnections = connections.get(peers.first());

    for(String peerId: peers) {

        Integer peerConnections = connections.get(peerId);

        if( peerConnections <= nextPeerConnections) {

            if(peerConnections < nextPeerConnections
                    || peerId.compareTo(nextPeerId) < 0) {
                nextPeerId = peerId;
                nextPeerConnections = connections.get(peerId);
            }

        }
    }

    return nextPeerId;
}
```

Figura 5.6: Listagem da implementação do Algoritmo de Escolha pelo Menor Número de Conexões

Esse algoritmo pode não apresentar um bom desempenho quando os nós envolvidos possuem capacidade de processamento das requisições bastante diferenciados, como é o caso de redes heterogêneas. De fato, todos os MLMs passam a atender um mesmo número de requisições, sem qualquer distinção, o que torna desaconselhável o uso desse algoritmo. Mas este problema pode ser resolvido através do uso de pesos, conforme o exemplo do algoritmo a seguir.

5.4.4 Algoritmo de Escolha pelo Menor Número de Conexões com Peso

Este algoritmo é bastante similar ao algoritmo descrito anteriormente, porém leva em conta as características específicas da capacidade total da máquina em que o MLM está sendo executado.

Nesse algoritmo, a primitiva `GetNextPeer` retorna a razão entre o número total de conexões que estão sendo atendidas pelo MLM e um peso que identifica a capacidade total do ambiente que está executando o MLM. Esse peso é calculado a partir de uma

configuração padrão, definida como sendo peso 1. Se o MLM estiver rodando em um computador com menor capacidade que o definido pelo padrão, então o seu peso será menor que 1. Se o MLM estiver rodando em um computador com maior capacidade que o computador definido como padrão, então o peso será maior que 1. Um MLM que roda em um computador com capacidade que é metade do padrão terá peso 0.5, enquanto um MLM que roda em um computador com o dobro da capacidade padrão terá peso 2.

```
    private SortedSet<String> peers = new TreeSet<String> ();
    private Map<String, Integer> connections = new HashMap<String,Integer>();
    private Map<String, Double> peersCapacity = new HashMap<String,Double>();

    public String getNextPeer() {

        String nextPeerId = peers.first();
        Double nextPeerCapacity = peersCapacity.get(peers.first());
        Integer nextPeerConnections = connections.get(peers.first());
        Double nextPeerConnectionsPerCapacity = nextPeerConnections/nextPeerCapacity;

        for(String peerId: peers) {

            Integer peerConnections = connections.get(peerId);
            Double peerCapacity = peersCapacity.get(peerId);
            Double connectionsPerCapacity = peerConnections/peerCapacity;

            if( connectionsPerCapacity <= nextPeerConnectionsPerCapacity) {

                if(connectionsPerCapacity < nextPeerConnectionsPerCapacity
                        || peerId.compareTo(nextPeerId) < 0) {
                    nextPeerId = peerId;
                    nextPeerConnectionsPerCapacity = connectionsPerCapacity;
                }

            }
        }

        return nextPeerId;
    }
```

Figura 5.7: Listagem da implementação do Algoritmo de Escolha pelo Menor Número de Conexões com Peso

O resultado esperado é que MLMs que rodam em computadores com maior capacidade de carga acabarão por receber um maior número de requisições em comparação aos outros com menor capacidade. A distribuição das requisições torna-se função da capacidade de cada MLM, buscando assim um menor tempo de resposta. Pode ser considerada uma boa solução para sistemas bastante heterogêneos.

5.4.5 Algoritmo de Escolha pela Menor Carga

A definição do índice de carga total de um determinado processo pode envolver muitas variáveis, dependendo do tipo de tarefa a ser executada. Isso pode tornar o cálculo do índice de carga total do *peer* ineficiente para a execução de determinada tarefa.

Pode-se definir, por exemplo, que o índice de carga seja a percentagem de CPU que está sendo usada. Nesse caso, a primitiva `GetNextPeer` irá retornar o *peer* com o

menor uso de CPU. Isso será bom para tarefas cujo gargalo é a utilização de CPU, porém pode ser prejudicial para tarefas cujo gargalo seja, por exemplo, o acesso a disco ou o uso de memória. Para outros tipos de tarefa, o gargalo poderá ser o uso de recursos de rede.

```
private SortedSet<String> peers = new TreeSet<String> ();
private Map<String, Double> loadIndexes = new HashMap<String,Double>();

public String getNextPeer() {

    String nextPeerId = peers.first();
    Double nextPeerLoadIndex = loadIndexes.get(peers.first());

    for(String peerId: peers) {

        Double loadIndex = loadIndexes.get(peerId);

        if( loadIndex <= nextPeerLoadIndex) {

            if(loadIndex < nextPeerLoadIndex
                    || peerId.compareTo(nextPeerId) < 0) {
                nextPeerId = peerId;
                nextPeerLoadIndex = loadIndex;
            }

        }
    }

    return nextPeerId;
}
```

Figura 5.8: Listagem da implementação do Algoritmo de Escolha pela Menor Carga

Uma forma de fazer o cálculo do índice total de carga é levar em conta todos os tipos recursos que as tarefas poderão utilizar e, através de um cálculo com médias e pesos, determinar esse índice. Outra forma é determinar dinamicamente qual o tipo de recurso que está sendo mais usado pela tarefa em questão, e colocar um peso maior para esse tipo de recurso no momento de calcular o índice. Essa maneira porém pode implicar que o comportamento da tarefa em relação ao uso de recursos computacionais seja previamente conhecido, o que muitas vezes não é possível.

6 AVALIAÇÃO

Embora o uso de tecnologias P2P para gerenciamento de redes traga vantagens interessantes em relação a modelos tradicionais de gerenciamento, o uso dessas tecnologias pode impactar de forma positiva ou negativa no desempenho das tarefas de gerenciamento. A fim de avaliar esse impacto, foram feitos testes no protótipo implementado, usando Componentes de Gerenciamento que se comportam como *gateways* para acesso a recursos disponibilizados nos MLMs.

Nos capítulos anteriores, foi apresentado o protótipo desenvolvido neste trabalho. Foi apresentada a forma que os TLMs e MLMs interagem com os Agentes para realizarem uma ação de gerenciamento. Foram apresentadas também as implementações dos algoritmos de distribuição de carga a serem utilizados. Uma vez encerrada a etapa de desenvolvimento do protótipo e dos algoritmos de distribuição, faz-se necessária uma avaliação dos mesmos para verificar questões de desempenho.

Neste capítulo, as implementações de cada um dos algoritmos de gerenciamento são comparadas entre si. A comparação foi feita no âmbito do gerenciamento distribuído e teve como parâmetros de avaliação a vazão do sistema, o tráfego total gerado e o tempo de resposta percebido.

Inicialmente, a metodologia de avaliação das soluções de gerenciamento é apresentada e os cenários de avaliação utilizados para a análise dos protótipos serão apresentados. Em seguida, os requisitos de hardware e software das máquinas utilizadas nos cenários de avaliação são descritos. Por fim, o capítulo é encerrado com a apresentação dos resultados obtidos.

6.1 Metodologia de Avaliação

A metodologia usada para avaliação do protótipo envolve a enumeração dos aspectos que devem ser avaliados, a caracterização da carga de trabalho a ser executada e a descrição dos cenários de avaliação. Nesta seção é feita a discussão desses assuntos.

6.1.1 Aspectos Avaliados

A fim de que todas as características do sistema sejam levadas em consideração durante a avaliação, é preciso inicialmente enumerar quais as variáveis e aspectos que podem influenciar no desempenho do sistema. Os aspectos avaliados são os seguintes:

- O algoritmo usado para distribuição de carga;
- O número de MLMs que oferecem um determinado Serviço de Gerenciamento;
- A quantidade de requisições feitas pelos TLMs ao serviço;

- A carga aplicada aos TLMs pelas tarefas de gerenciamento;

- A carga atual de cada um dos MLMs no momento em que é recebida uma nova requisição.

O primeiro aspecto a ser levado em consideração é o uso de diferentes algoritmos de distribuição de carga. Três algoritmos foram escolhidos, devido à sua simplicidade: dois algoritmos estáticos, e um algoritmo dinâmico. Os dois algoritmos estáticos são o algoritmo de distribuição aleatória, que é o mais simples de todos, e o algoritmo de distribuição em fila circular. O algoritmo dinâmico usado considera o número de requisições ativas em cada MLM, sendo que o MLM escolhido para processamento da requisição seguinte é sempre o MLM que possui o menor número de conexões.

Outro aspecto a ser considerado é o número de MLMs que oferecem um determinado Serviço de Gerenciamento. O Serviço de Gerenciamento é fornecido ao sistema por um grupo de MLMs, que se organizam e se comunicam a fim de distribuir as tarefas e requisições entre si. Para um serviço pouco requisitado, um grupo com alguns MLMs pode ser suficiente para manter o serviço disponível. Porém esse número deve aumentar quando há um aumento no número de requisições, ou quando há aumento no tempo necessário para execução de cada requisição. Além disso, deve ser um número suficientemente grande quando se deseja uma boa disponibilidade do serviço, de forma que a falha em alguns dos membros do grupo não afete sua disponibilidade. Esse aspecto será representado nessa avaliação através da execução dos testes em diversos grupos com diferente número de MLMs por grupo.

O terceiro aspecto é a quantidade de requisições feitas pelos TLMs ao serviço. Um serviço pouco requisitado oferece pouca carga ao grupo de MLMs que disponibiliza o serviço, enquanto um serviço muito requisitado oferece muita carga. É a medida do número de requisições feitas ao serviço em um determinado período de tempo. Esse aspecto será representado na avaliação como o número de requisições por segundo.

Outro aspecto é a carga aplicada aos TLMs pelas tarefas de gerenciamento. Essa carga pode ser variada pela quantidade de requisições feitas pelos TLMs em um determinado tempo, ou mesmo pelo tempo necessário para a execução de cada tarefa requisitada. Além disso, determinadas tarefas de gerenciamento podem impor uma carga variável sobre cada um dos TLMs durante o processo de execução da tarefa. Os recursos a serem usados nos TLMs podem ser também variados, pois determinadas tarefas podem precisar de espaço em memória, ou acesso a I/O, ou tempo de processamento. A fim de manter uma homogeneidade sobre os resultados comparados, optou-se pela escolha de tarefas que executam nos MLMs e que acessam um recurso qualquer do MLM, porém que só podem ser acessadas por um processo de cada vez. Esse comportamento será simulado no MLM através da característica de sincronização de recursos da linguagem Java.

Por fim, o último aspecto a ser levado em consideração é a carga atual de cada um dos MLMs no momento em que é recebida uma nova requisição. Um serviço que está recém sendo disponibilizado reage de forma bastante rápida quando recebe uma requisição, pois todos os TLMs estão disponíves para execução da tarefa. Porém, após um determinado tempo em que o serviço está disponiblizado, novas requisições que chegam podem ter a execução bloqueada se os recursos necessário para execução da tarefa já estiverem sendo ocupados. Nesse caso, o grupo começa a se tornar sobrecarregado, e o tempo de resposta passa a aumentar se comparado à situação inicial de disponibilização do serviço. Esse aspecto foi levado em conta nessa avaliação através do envio sucessivo de um determinado

número de requisições, de forma que o efeito da execução das primeiras requisições seja verificado nas requisições subseqüentes.

6.1.2 Caracterização da Carga de Trabalho

A carga de trabalho do sistema é toda executada nos MLMs. Em um exemplo real, para que os MLMs executem uma tarefa de gerenciamento, são utilizados recursos do *peer*, como acesso a disco, tempo de processamento, espaço em memória, acesso a rede e principalmente acesso a dispositivos de gerenciamento. Normalmente esses recursos só podem ser acessados por um processo por vez em um determinado momento, e bloqueiam a execução de outros processos que estão querendo utilizá-los enquanto estão sendo acessados.

A fim de emular o acesso a recursos bloqueantes que só podem ser usados por um processo por vez em um determinado momento, foi utilizada a característica de sincronização de recursos da linguagem Java. O trecho de código que representa o acesso ao recurso é um trecho sincronizado, que só pode ser acessado por uma *Thread* por vez, e é representado pelo seguinte algoritmo:

```
synchronized (LOCK) {
    Thread.sleep(loadTime);
}
```

Sendo que o objeto `LOCK` é compartilhado por todas as *Threads*. Dessa forma, a execução do `sleep` só pode ser executada por uma *Thread* por vez, emulando assim o acesso a um recurso qualquer do *peer*.

A carga de trabalho de cada requisição é então controlada pelo `loadTime`, ou tempo de carga, que representa o tempo total que o recurso deve ser usado pelo processo durante essa requisição.

O tempo que um MLM utiliza para acesso a recursos normalmente é um tempo variável, pois é um tempo que normalmente envolve acesso a dispositivos de rede. Por isso optou-se pela geração de um tempo variável, utilizando uma distribuição probabilística. A distribuição normalmente usada para modelagem de tempos de serviço de dispositivos em modelos de rede é a Distribuição Gamma (JAIN, 1991).

Para a geração de tempos segundo uma distribuição Gamma, escolhe-se dois parâmetros, α e β, que melhor representem a média e o desvio padrão dos tempos de serviço. Por exemplo, para que a carga de trabalho média das requisições seja de 1 segundo e o desvio padrão de 0,5, o tempo de carga de cada requisição deve obedecer à distribuição estatística representada pela figura 6.1, que é uma distribuição *Gamma* com os parâmetros α=2 e β=0,5.

O número máximo de requisições por segundo que um grupo de MLMs consegue atender depende principalmente do número de MLMs que participam do grupo, além da carga exigida por cada uma das requisições. Teoricamente, considerando que o tempo médio para a execução da carga de trabalho das requisições seja de 1 segundo, a taxa máxima que pode ser atendida por um grupo é de *n* requisições por segundo, sendo *n* igual ao número de MLMs do grupo. Se a taxa de requisições por segundo ultrapassar esse valor teórico, ou se o tempo médio para a execução da carga de trabalho das requisições aumentar, os MLMs que respondem pelo grupo se tornarão sobrecarregados, e as requisições que não podem ser atendidas passam a entrar em uma fila, que tende a crescer até que o *buffer* de requisições não comporte novas requisições.

A sobrecarga do grupo normalmente acontece antes de alcançada a carga máxima teórica do grupo. Neste ponto, o algoritmo de distribuição escolhido pode fazer muita

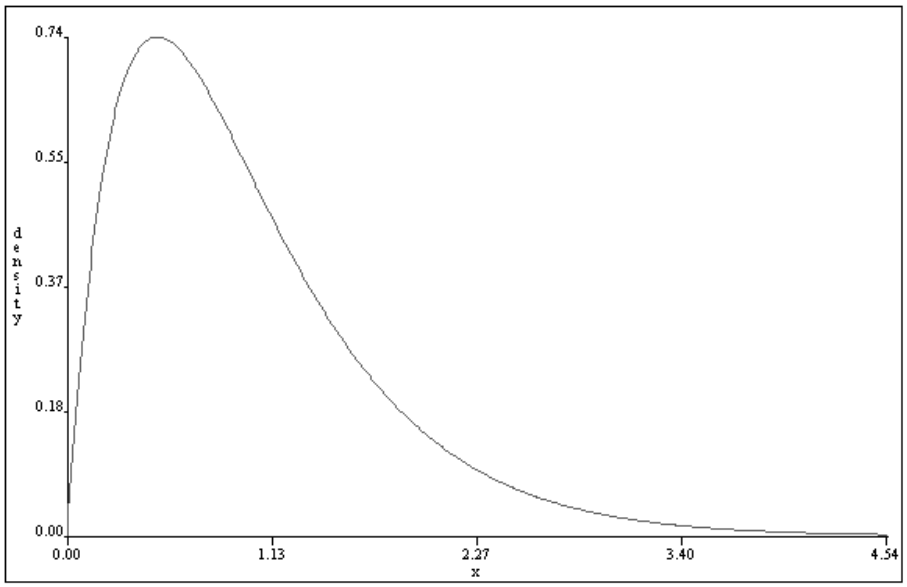

Figura 6.1: Distribuição *Gamma* com os parâmetros α=2 e β=0,5

diferença. Em um algoritmo ineficiente, a sobrecarga pode acontecer perto dos 50% desse valor. Já um algoritmo eficiente só permitirá que o grupo fique sobrecarregado quando a carga aplicada se aproximar do total da carga teórica.

É por esse motivo que as cargas escolhidas para envio de requisições foram de 50%, 90% e 100% da carga máxima teórica. A taxa de 50% representa o comportamento do sistema quando este está em baixa carga; a taxa de 90% representa o comportamento do sistema quando este está próximo à sobrecarga, e a taxa de 100% representa o comportamento do sistema quando este está sobrecarregado.

O aumento ou diminuição da carga pode ser adquirido de duas formas: ou através do aumento ou diminuição da taxa de mensagens por segundo, ou através do aumento ou diminuição do tempo médio de execução das tarefas. Neste trabalho optou-se por variar o tempo médio de cada requisição. Assim, para que a carga do sistema seja de 50%, o tempo médio para a execução das requisições é de 0,5 segundos. Da mesma forma para atingir as cargas de 90% e 100%, os tempos médios são de 0,9 e 1 segundo, respectivamente. Já a taxa de mensagens por segundo só depende do número de MLMs, e é sempre igual a *n* requisições por segundo, sendo *n* igual ao número de MLMs do grupo.

A média dos tempos de execução das tarefas é exemplificada para alguns casos na tabela da figura 6.2. Nessa tabela, estão listados alguns números de MLMs que compõem os diferentes grupos, e para cada número de MLMs, conforme a carga total a ser aplicada ao sistema - 50%, 90% ou 100% - estão indicadas duas variáveis: primeiro, a carga de cada requisição, em que **c** representa a carga a ser utilizada nas requisições, e é calculada tendo como base **C**, que representa a carga adquirida pelo cálculo usando a distribuição estatística, e segundo, a taxa de requisições por segundo a ser aplicada pelo TLM.

6.1.3 Cenários Avaliados

Os cenários de avaliação foram construídos com o fim de medir os aspectos que possam interferir no tempo de resposta, no tráfego gerado e na vazão do sistema. A metodologia de avaliação utilizada para os diversos cenários escolhidos segue os passos a seguir:

	Carga 50%	**Carga 90%**	**Carga 100%**
1 MLM	c = 0,5C, 1req/s	c = 0,9C, 1req/s	c = C, 1req/s
2 MLMs	c = 0,5C, 2req/s	c = 0,9C, 2req/s	c = C, 2req/s
5 MLMs	c = 0,5C, 5req/s	c = 0,9C, 5req/s	c = C, 5req/s
10 MLMs	c = 0,5C, 10req/s	c = 0,9C, 10req/s	c = C, 10req/s
15 MLMs	c = 0,5C, 15req/s	c = 0,9C, 15req/s	c = C, 15req/s

Figura 6.2: Exemplos de cargas e taxas utilizadas

- Escolhe-se o número de MLMs que compõem o grupo que disponibiliza o Serviço de Gerenciamento. Esse número varia de 1 a 15.
- Escolhe-se o algoritmo de distribuição a ser usado. São três os algoritmos avaliados: Distribuição por Fila Circular, Distribuição Aleatória e Distribuição pelo Menor Número de Conexões.
- Escolhe-se a carga de trabalho a ser aplicada ao grupo. A carga escolhida corresponde a uma porcentagem relativa à carga total teórica que o grupo é capaz de atender. As cargas avaliadas correspondem a 50% da carga total, 90% da carga total e 100% da carga total.
- Um TLM envia um conjunto de requisições ao serviço, de forma que a taxa de requisições enviadas por segundo obedeça à carga de trabalho escolhida no item anterior. O número de requisições enviadas depende no número de MLMs envolvidos, de forma que o tempo total do teste dure um tempo de aproximadamente 5 segundos.

Esse procedimento foi repetido 30 vezes, o que totalizou 30 amostras para cada solução avaliada, cada uma delas sendo uma execução com duração de aproximadamente 5 segundos cada. Chega-se ao total de cenários avaliados através da combinação de todas as variáveis que são usadas no sistema: 3 algoritmos diferentes, 15 grupos diferentes de MLMs e 3 taxas diferentes de envio de requisições, totalizando assim 135 casos de uso. O tempo total para execução de todos os testes dura aproximadamente 6 horas.

Nas amostras coletadas, foram avaliados os parâmetros de vazão do sistema, tráfego total gerado e tempo de resposta percebido. O tráfego total gerado se refere a quantidade total de bytes gerada pela comunicação entre o TLM e o MLM. Já o tempo de resposta percebido se refere ao tempo que o TLM leva para perceber que a execução de uma tarefa de gerenciamento encerrou-se no MLM. O tempo de resposta percebido é calculado através do registro do tempo em que foi enviada a mensagem e pelo registro do tempo em que o MLM responde ao TLM com o resultado da operação. Neste trabalho foram considerados os parâmetros tráfego gerado e tempo de resposta percebido porque tais parâmetros são comumente utilizados nas pesquisas genéricas sobre desempenho de sistemas de gerenciamento. A vazão do sistema corresponde ao número de requisições por segundo que o grupo é capaz de atender. Essa medida foi feita a fim de demonstrar se existe realmente vantagem no uso de um grande grupo de MLMs para execução de tarefas de gerenciamento, e qual é o benefício de um grande número de MLMs em um grupo para a escalabilidade do sistema.

Cada caso de uso é disponibilizado como um diferente serviço nos MLMs do sistema. Assim, os serviços possuem um nome que representa o caso de uso com seus respecti-

vos parâmetros. A nomenclatura escolhida para os Serviços de Gerenciamento segue o seguinte padrão: `SVC-[nn]-[aa]`, onde `nn` representa o número de MLMs que disponibilizam o serviço, e `aa` representa o algoritmo que está sendo usado para distribuição de carga do serviço. Os algoritmos são `RD` para *RandomicDistribution*, `RR` para *RoundRobin* e `LC` para *LeastConnectionsFirst*. Por exemplo, o serviço cujo nome é `SVC-07-LC` é um serviço disponibilizado por 7 MLMs e cujo algoritmo de distribuição é o *LeastConnectionsFirst*.

6.2 Configuração das Máquinas

Para a realização das avaliações do trabalho, foram utilizadas as máquinas do Cluster LabTec do Grupo de Processamento Paralelo e Distribuído do Instituto de Informática - UFRGS.

A seguir estão resumidas as características de hardware e software de cada máquina utilizada nas avaliações.

- **Quantidade de Nós:** 20
- **Processadores por Nó:** 2 Pentium III 1.1 GHz
- **Memória por Nó:** 1 Gb
- **Rede de Comunicação:** Gigabit Ethernet
- **Sistema Operacional:** Linux Kernel 2.6.18-3-686
- Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0-b64)

Os requisitos de software são comuns a todos os cenários de avaliação apresentados. O ambiente apenas requer Java JDK 1.5 para funcionamento da plataforma ManP2P.

Foram utilizadas 15 máquinas como MLMs para disponibilização dos Serviços de Gerenciamento, uma máquina foi usada como *rendezvous* para a plataforma JXTA e uma outra máquina foi usada como TLM. Os Serviços de Gerenciamento foram distribuídos entre os MLMs uniformemente, de maneira que todos ficassem com o mesmo número de Componentes de Gerenciamento. A distribuição de serviços da tabela da figura 6.3 representa a distribuição dos componentes entre os MLMs com o algoritmo *RandomicDistribution* (RD). A nomenclatura dos serviços segue o mesmo padrão anteriormente mencionado, de forma que a mesma distribuição da tabela 6.3 ocorre para os algoritmos *RoundRobin* (RR) e *LeastConnectionsFirst* (LC).

6.3 Vazão do Sistema

O gráfico da figura 6.4 mostra o número de conexões por segundo que o sistema é capaz de atender, sendo que as conexões oferecem ao sistema uma carga média de 1 segundo, e o TLM fornece ao grupo uma taxa de requisições igual a uma requisição por segundo para cada MLM.

O gráfico mostra que a vazão do sistema aumenta de forma praticamente linear conforme o número de MLMs do grupo, para os casos analisados. Seria interessante se fosse possível analisar qual é o comportamento do sistema em casos com um número maior de MLMs, porém isso não foi possível devido ao número limitado de nós do *cluster*. Mesmo

MLM1	SVC-15-RD	SVC-01-RD	SVC-02-RD	SVC-03-RD	SVC-04-RD	SVC-05-RD	SVC-06-RD	SVC-07-RD
MLM2	SVC-15-RD	SVC-14-RD	SVC-02-RD	SVC-03-RD	SVC-04-RD	SVC-05-RD	SVC-06-RD	SVC-07-RD
MLM3	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-03-RD	SVC-04-RD	SVC-05-RD	SVC-06-RD	SVC-07-RD
MLM4	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-04-RD	SVC-05-RD	SVC-06-RD	SVC-07-RD
MLM5	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-05-RD	SVC-06-RD	SVC-07-RD
MLM6	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-10-RD	SVC-06-RD	SVC-07-RD
MLM7	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-10-RD	SVC-09-RD	SVC-07-RD
MLM8	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-10-RD	SVC-09-RD	SVC-08-RD
MLM9	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-10-RD	SVC-09-RD	SVC-08-RD
MLM10	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-10-RD	SVC-09-RD	SVC-08-RD
MLM11	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-10-RD	SVC-09-RD	SVC-08-RD
MLM12	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-10-RD	SVC-09-RD	SVC-08-RD
MLM13	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-10-RD	SVC-09-RD	SVC-08-RD
MLM14	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-10-RD	SVC-09-RD	SVC-08-RD
MLM15	SVC-15-RD	SVC-14-RD	SVC-13-RD	SVC-12-RD	SVC-11-RD	SVC-10-RD	SVC-09-RD	SVC-08-RD

Figura 6.3: Distribuição dos Componentes de Gerenciamento nos MLMs

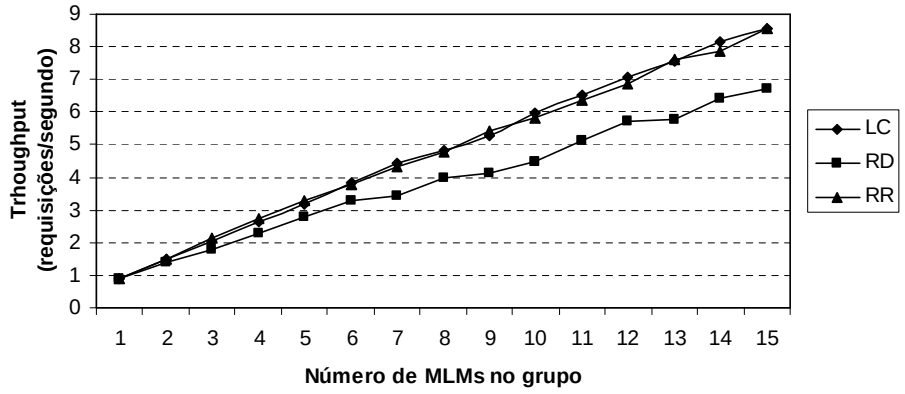

Figura 6.4: Número de conexões por segundo que o sistema é capaz de atender

assim, o gráfico não mostra sinais de diminuição da linearidade, demonstrando assim a boa escalabilidade do sistema. Dessa forma, esse gráfico mostra claramente a vantagem de se disponibilizar os serviços de gerenciamento usando um grupo de MLMs.

Através de uma análise do gráfico, é possível verificar que os resultados para os algoritmos RR e LC se aproximam de uma reta que possui uma inclinação aproximada de 0,543. Já para o algoritmo de distribuição aleatória, a inclinação da reta é menor, de aproximadamente 0,416. Quanto maior a inclinação, e quanto mais próxima de 1, maior é o aproveitamento dos nós disponíveis. Assim, as implementações que oferecem uma maior vazão ao sistema são RR e LC.

6.4 Tempo de Resposta Percebido

A medida do tempo de resposta percebido pelo TLM pode ser feita através da medida do tempo decorrido entre o envio da requisição e a chegada da resposta fornecida pelo MLM. Porém, deve-se considerar que parte desse tempo é usado para a execução da tarefa. O que se pretende neste trabalho é medir o tempo decorrido no atendimento à requisição que não faz parte do processamento útil da execução da tarefa.

A fim de medir o tempo de resposta que não faz parte do processamento útil da requisição, o atendimento à requisição é divido em diversas etapas, conforme a figura 6.5. Dessa forma, o tempo de resposta que não faz parte do processamento útil da requisição é igual ao tempo de resposta percebido pelo TLM subtraído do tempo de processamento

útil. O tempo de resposta percebido pelo TLM é a medida do tempo decorrido desde o envio da requisição até o recebimento da resposta, e é medido no TLM, enquanto o tempo de processamento útil é informado como parâmetro da requisição, e pode ser resgatado pelo TLM para ser subtraído do tempo percebido.

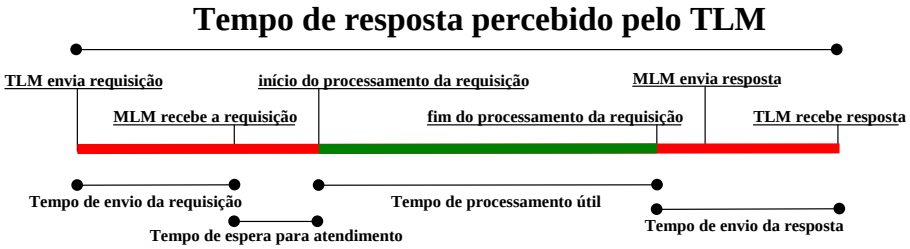

Figura 6.5: Etapas do atendimento à requisição

O gráfico da figura 6.6 mostra a evolução dos tempos de resposta percebidos pelo TLM e subtraídos do tempo de processamento útil de cada um dos três tipos de políticas de distribuição de carga, conforme a quantidade de MLMs no grupo, com o grupo sendo submetido a uma carga de 50% da carga total.

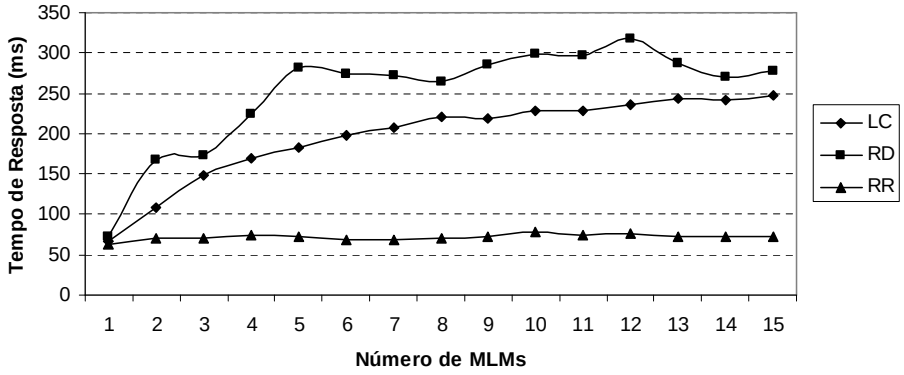

Figura 6.6: Evolução dos tempos de resposta com carga a 50%, descontado o tempo de processamento útil

O gráfico da figura 6.7 mostra a evolução dos tempos de resposta percebidos pelo TLM e subtraídos do tempo de processamento útil de cada um dos três tipos de políticas de distribuição de carga, conforme a quantidade de MLMs no grupo, com o grupo sendo submetido a uma carga de 90% da carga total.

O gráfico da figura 6.8 mostra a evolução dos tempos de resposta percebidos pelo TLM e subtraídos do tempo de processamento útil de cada um dos três tipos de políticas de distribuição de carga, conforme a quantidade de MLMs no grupo, com o grupo sendo submetido a uma carga de 100% da carga total.

Pelas medidas apresentadas, a primeira conclusão é a de que a distribuição aleatória, apesar de ser de baixo custo computacional, não traz benefícios nem em relação à vazão do sistema, nem em relação ao tempo de resposta das requisições.

Conclui-se também que, apesar de o algoritmo de distribuição pelo menor número de conexões ser um algoritmo dinâmico e o algoritmo de distribuição por fila circular ser um algoritmo estático, o fato de ser um algoritmo dinâmico não traz benefícios em relação ao tempo de resposta nos casos de uso escolhidos. O throughput do sistema nos dois casos é bem próximo, porém o tempo de resposta do algoritmo dinâmico é maior, devido ao

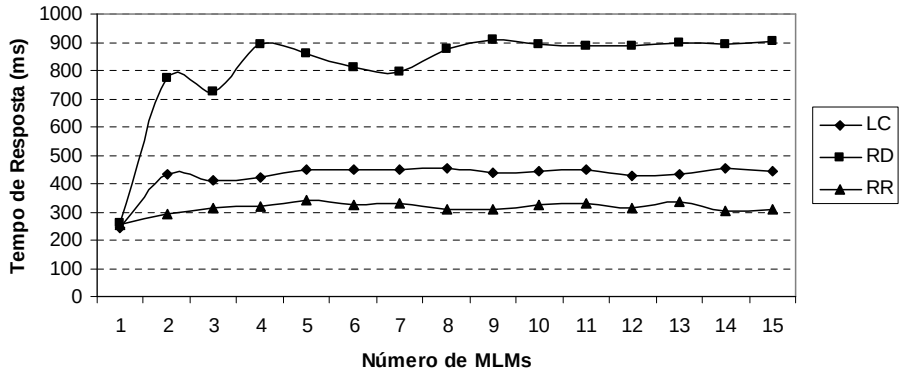

Figura 6.7: Evolução dos tempos de resposta com carga a 90%, descontado o tempo de processamento útil

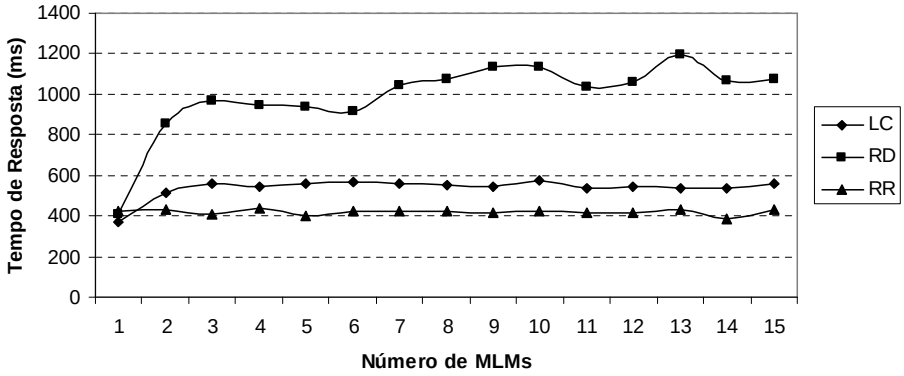

Figura 6.8: Evolução dos tempos de resposta com carga a 100%, descontado o tempo de processamento útil

fato de que são necessárias muitas trocas de mensagens entre os participantes do grupo, ao contrário do algoritmo estático, com complexidade computacional reduzida.

Porém, o mau desempenho do algoritmo de distribuição pelo menor número de conexões em relação ao algoritmo de distribuição por fila circular depende muito da amostra que foi escolhida. É bem provável que, se fosse escolhida uma amostra com uma variabilidade maior no tempo de execução das requisições, o algoritmo dinâmico tivesse um desempenho melhor. Mas essa afirmação não passa de uma especulação com base em observações empíricas, e seria necessário um novo teste com amostras diferentes de tempos de processamento de requisições para se ter uma evidência mais conclusiva.

Muitas das medidas apresentadas nesta seção dependem profundamente da plataforma P2P adotada. No caso desse protótipo, usando a plataforma JXTA, boa parte do tempo é gasta em conexões com roteadores de nível de aplicação e *rendezvous* (GONG, 2001).

A independência de protocolo alcançada com o uso desse modelo permite o desenvolvimento de níveis de comunicação entre TLMs e MLMs mais otimizados e sofisticados. Comparando com o protocolo SNMP, por exemplo, tais vantagens podem ser observadas quando há a necessidade de execução de tarefas que exigem a troca de uma grande quantidade de mensagens entre agentes e gerentes SNMP. A Figura 6.9 mostra o tempo de *download* da tabela de roteamento de um dispositivo de rede localizado em um domínio diferente. O tempo de *download* usando protocolo SNMP é proporcional ao tempo de resposta existente entre o agente e o gerente, multiplicado pelo número de entradas da tabela de roteamento. O uso da arquitetura P2P reduz o tempo de *download* a um tempo

proporcional ao número de bytes que devem ser enviados, pelo fato de que apenas uma resposta precisa ser enviada ao TLM.

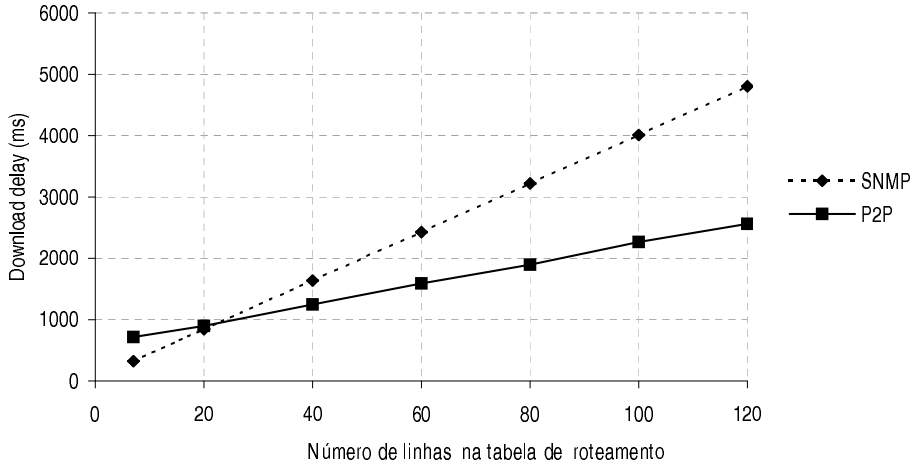

Figura 6.9: Tempo necessário para *download* da tabela de roteamento de um dispositivo de rede

Esses resultados mostram que, ao mesmo tempo em que oferece vantagens interessantes, o desempenho de um sistema de gerenciamento baseado em P2P não se torna ruim, ao menos se comparado com o desempenho apresentado pelo protocolo SNMP. Isso é importante porque o desempenho de um sistema de gerenciamento precisa ser adequado à rede a ser gerenciada.

6.5 Tráfego Gerado

O tráfego gerado por cada requisição no TLM está representado no gráfico da figura 6.10, que mostra o número de bytes transferidos por mensagem conforme a evolução do número de MLMs nos grupos.

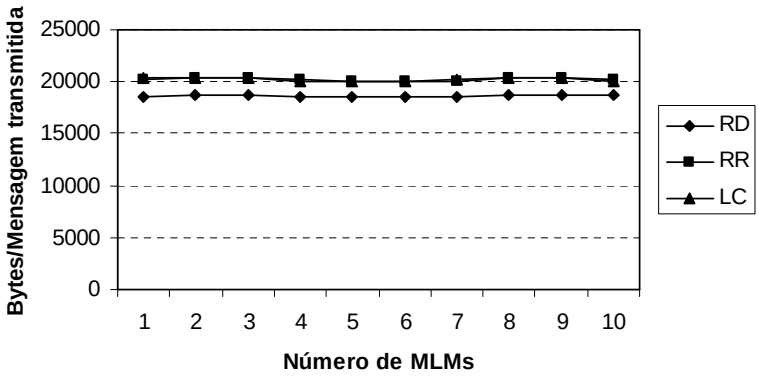

Figura 6.10: Número de bytes transferidos por mensagem no TLM

Conforme observado, o número de bytes transferidos por mensagem não aumenta com o aumento de MLMs no grupo. A diferença entre o número de bytes transferidos está entre cada um dos algoritmos usados. O algoritmo de distribuição aleatória (RD) possui uma quantidade menor de bytes transferidos, devido ao fato de que não é necessária a troca de mensagens para informação de estado entre os MLMs. A única mensagem que os MLMs precisam enviar aos outros é um aviso do início do processamento de

uma requisição, e a passagem de *token* para o próximo MLM que deve processar uma requisição. Já no algoritmo de escolha por menor número de conexões, por exemplo, cada MLM deve enviar uma mensagem a mais no final do processamento da requisição, avisando aos outros MLMs que a requisição foi processada e que o número de requisições sendo atendias mudou.

Porém, a quantidade superior de *bytes* transferidos por mensagem nos algoritmos de fila circular e de escolha pelo menor número de conexões não torna proibitivo o seu uso. Pelo contrário, a vazão muito superior e o baixo tempo de espera torna esses dois algoritmos mais atrativos do que o algoritmo por distribuição aleatória.

7 CONCLUSÕES E TRABALHOS FUTUROS

Conforme a complexidade dos sistemas computacionais aumentou, passando a ser altamente distribuídos e interconectados, os modelos tradicionais de gerenciamento de redes se mostraram limitados. Por isso, novas tecnologias vêm sendo utilizadas no gerenciamento de redes como soluções alternativas ao protocolo padrão de gerenciamento de redes TCP/IP, o SNMP. Modelos de gerenciamento tradicionais não são aplicáveis nestes cenários porque são concebidos para operar em ambientes restritos de um único domínio administrativo, enquanto no gerenciamento moderno é preciso que as operações sejam executadas entre domínios diferentes, de forma distribuída.

Os modelos de comunicação P2P foram construídos sobre os protocolos Internet a fim de que operem como sistemas altamente distribuídos. Apesar de as redes P2P operarem sobre redes físicas, elas formam redes sobrepostas que são independentes do endereçamento e roteamento próprio da Internet. Isso permite a construção de ambientes dinâmicos e versáteis capazes de resolver problemas encontrados no gerenciamento de redes. O modelo P2P torna-se ainda mais interessante porque se encaixa perfeitamente no gerenciamento de redes distribuído atualmente solicitado.

A partir das experiências práticas na avaliação da proposta de gerenciamento de redes usando P2P e da apresentação de um novo modelo conceitual usando essa nova perspectiva, foram definidos os blocos básicos necessários para o desenvolvimento de arquiteturas que seguem o modelo de gerenciamento de redes baseado em estruturas P2P. De acordo com os elementos usados em um *peer* de gerenciamento, esse *peer* pode se comportar como um gerenciador *Top-Level*, um gerenciador *Mid-Level* ou como um gerenciador híbrido. A integração de entidades de gerenciamento baseadas em P2P com soluções tradicionais de gerenciamento, tal como a estrutura SNMP, também foi apresentada.

Uma arquitetura foi definida a partir desse novo modelo de gerenciamento de redes, juntamente com os papéis das entidades do modelo de gerenciamento de redes baseado em P2P. Um dos conceitos centrais dessa arquitetura são os Serviços de Gerenciamento, que definem as tarefas básicas de gerenciamento do sistema, e foram inspirados nos *Web Services* e na concepção de serviços das Arquiteturas Orientadas a Serviço (SOA), porém mantendo todas as características de sistemas P2P. Foram definidos também os passos de interação entre os elementos da arquitetura, desde o momento em que o TLM inicia a requisição de execução de um serviço até o momento em que o TLM recebe a resposta do MLM. Para preservar as características de disponibilidade e escalabilidade que são inerentes dos sistemas P2P, definiu-se também que os Serviços de Gerenciamento devem ser disponibilizados por Grupos de MLMs, organizados de forma a fazer a distribuição de tarefas de gerenciamento entre si.

Foram apresentados o projeto e implementação do protótipo de uma plataforma de sistema de gerenciamento de redes baseado em estrutura P2P - a plataforma ManP2P,

usada para avaliar o modelo de gerenciamento apresentado . Esse protótipo é usado para avaliação e análise de testes e resultados associados. A implementação apresentada do protótipo é codificada em Java, usando o Framework JXTA da Sun como infraestrutura P2P. Esse protótipo foi desde o princípio de sua implementação orientado de forma que possa ser expansível, para que seja possível a adição de novas funcionalidades por desenvolvedores interessados em expandir a plataforma. Foi definida uma Interface de Programação (API) inspirada na API de *Web Services*, para que outros desenvolvedores possam implementar novos Serviços de Gerenciamento sobre esse protótipo usando conceitos já difundidos entre a comunidade de desenvolvedores. Foi definida também uma API para que os desenvolvedores dos TLMs possam acessar de modo padronizado os Serviços de Gerenciamento, acessando de modo transparente os recursos da rede P2P. Procurou-se tornar o padrão dessas APIs o mais próximo aos padrões atuais de desenvolvimento de serviços para aplicações, mantendo-se como foco as APIs de arquiteturas de *Web Services* e arquiteturas orientadas a serviços.

Uma das principais características de uma arquitetura de gerenciamento baseada em P2P é o uso de grupos de *peers*, onde um conjunto de Componentes de Gerenciamento que oferecem os mesmos Serviços de Gerenciamento são agrupados a fim de oferecer serviços mais confiáveis em termos de disponibilidade. As vantagens de se disponibilizar os serviços de gerenciamento usando grupos de MLMs torna-se clara na avaliação da vazão do sistema. O aumento da capacidade do sistema aumenta de forma linear em relação ao número de MLMs do grupo, demonstrando a boa escalabilidade do sistema.

As diretrizes de expansibilidade do sistema também foram levadas em consideração no momento da implementação dos algoritmos de distribuição de carga. Diversos algoritmos de distribuição, desde os mais simples até alguns algoritmos mais complexos de balanceamento de carga foram definidos e implementados, sempre tendo em vista dar liberdade ao desenvolvedor dos Serviços de Gerenciamento, para que escolha o algoritmo de distribuição que mais se adapte às suas necessidades, podendo inclusive implementar sua própria versão se for necessário. A partir dessas implementações, puderam então ser realizadas avaliações e gráficos comparativos, e puderam ser estabelecidas diretrizes sobre quando e como usar determinados algoritmos de distribuição de carga, de modo a alcançar sempre o melhor rendimento e a maximização do aproveitamento dos recursos disponíveis ao sistema.

Entre os algoritmos de distribuição apresentados, o algoritmo que apresentou mais benefícios em relação ao tempo de resposta percebido foi o algoritmo de fila circular. O algoritmo de distribuição pelo menor número de conexões apresentou um desempenho bom também, e a vazão do sistema nos dois casos tem um resultado parecido, porém o tempo de resposta do segundo algoritmo é maior, devido ao fato de que são necessárias muitas trocas de mensagens entre os participantes do grupo, ao contrário do primeiro algoritmo, com complexidade computacional reduzida. O algoritmo de distribuição aleatória também possui complexidade computacional reduzida, porém não traz benefícios em relação ao tempo de resposta, e além disso possui vazão inferior aos outros dois algoritmos apresentados.

Os resultados da avaliação mostram que, ao mesmo tempo em que o gerenciamento baseado em P2P oferece vantagens interessantes, o desempenho de um sistema de gerenciamento baseado em P2P é aceitável, ao menos se comparado com o desempenho apresentado pelo protocolo SNMP. Isso é importante porque o desempenho de um sistema de gerenciamento precisa ser adequado à rede a ser gerenciada. Os resultados da avaliação também mostraram que, com o devido cuidado, o desempenho de um sistema

de gerenciamento baseado em P2P pode ser melhor do que o desempenho de modelos de gerenciamento tradicionais, tal como a arquitetura SNMP.

Entre os trabalhos a serem realizados, existem problemas que devem ser resolvidos, como a implementação do tratamento aos casos de falha de um MLM do grupo. Deve ser estudado também um mecanismo de controle de acesso mais robusto para os Serviços de Gerenciamento. A persistência e replicação distribuída de informações de gerenciamento também pode ser implementada, utilizando os mesmos serviços de gerenciamento usados para a execução das tarefas de gerenciamento. A implementação de alertas de rede também precisa ser estudada, através do uso de componentes de gerenciamento que geram alertas nos MLMs, e outros componentes de gerenciamento que recebem esses alertas nos TLMs. Outro trabalho a ser realizado é a avaliação das características do sistema quando os *peers* são inseridos em um contexto de grandes redes (com milhares de dispositivos sendo gerenciados).

REFERÊNCIAS

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. A Survey of Peer-to-Peer Content Distribution Technologies. **ACM Computing Surveys**, New York, NY, USA, v.36, n.4, p.335–371, 2004.

APPARAO, V. et al. **Document Object Model (DOM) Level 1 Specification**. [S.l.], 1998. Disponível em: <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/>. Acesso em: 21 mar. 2005.

BARAK, A.; SHILOH, A. A distributed load-balancing policy for a multicomputer. **Softw. Pract. Exper.**, New York, NY, USA, v.15, n.9, p.901–913, 1985.

BELLWOOD, T.; CLÉMENT, L.; RIEGEN, C. **UDDI Version 3.0.1**. [S.l.], 2003. Disponível em: <http://uddi.org/pubs/uddi-v3.0.1-20031014.htm>. Acesso em: 21 mar. 2005.

BEN-ARTZI, A.; CHADNA, A.; WARRIER, U. Network Management of TCP/IP Networks: present and future. **IEEE Network Magazine**, New York, NY, USA, v.4, n.4, p.35–43, July 1990.

BIESZCZAD, A.; PAGUREK, B.; ; WHITE, T. Mobile Agents for Network Management. **IEEE Commun. Surveys Tuts.**, [S.l.], v.1, n.1, p.2–9, September 1998.

CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. **IEEE Trans. Softw. Eng.**, Piscataway, NJ, USA, v.14, n.2, p.141–154, 1988.

CHRISTENSEN, E. et al. **Web Services Description Language (WSDL) 1.1**. [S.l.], 2001. Disponível em: <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>. Acesso em: 21 mar. 2005.

CLARKE, I. et al. Freenet: a distributed anonymous information storage and retrieval system. In: INTERNATIONAL WORKSHOP ON DESIGN ISSUES IN ANONYMITY AND UNOBSERVABILITY, 2000, Berkeley, California. **Proceedings...** Berlin: Springer, 2000. p.46–66. (Lecture Notes in Computer Science, v. 2009).

CURBERA, F. et al. Unraveling the Web services Web: an introduction to SOAP, WSDL, and UDDI. **IEEE Internet Computing**, New York, NY, USA, v.6, n.2, p.86–93, Mar./Apr. 2002.

CURBERA, F. et al. The Next Step in Web Services. **Communications of the ACM**, New York, NY, USA, v.46, n.10, p.29–34, Oct. 2003.

DOUGLIS, F.; OUSTERHOUT, J. K. Transparent Process Migration: design alternatives and the sprite implementation. **Software - Practice and Experience**, [S.l.], v.21, n.8, p.757–785, 1991.

DUIGOU, M. **JXTA v2.0 Protocols Specification**. Disponível em: <http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html/>. Acesso em: jan. 2007.

GOLDSZMIDT, G.; YEMINI, Y. Distributed management by delegation. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 15., 1995, Washington, DC, USA. **Proceedings...** [S.l.]: IEEE Computer Society, 1995. p.333–340.

GONG, L. JXTA: a network programming environment. **IEEE Internet Computing**, Piscataway, NJ, USA, v.5, n.3, p.88–95, 2001.

GOTH, G. Grid Services Architecture Plan Gaining Momentum. **IEEE Internet Computing**, New York, NY, USA, v.6, n.4, p.11– 12, July/Aug. 2002.

GRANVILLE, L.; ROSA, D.; PANISSON, A.; MELCHIORS, C.; ALMEIDA, M.; TAROUCO, L. Managing Computer Networks Using Peer-to-Peer Technologies. **IEEE Communications Magazine**, [S.l.], v.43, n.10, p.62–68, Oct. 2005.

GUDGIN, M. et al. **SOAP Version 1.2 Part 1 - Messaging Framework**. [S.l.: s.n.], 2003. Disponível em: <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>. Acesso em: 21 mar. 2005.

HARRINGTON, D.; PRESUHN, R.; WIJNEN, B. **An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks**: RFC 3411. [S.l.]: Internet Engineering Task Force, Network Working Group, 2002.

JACQMOT, C.; MILGROM, E. A Systematic Approach to Load Distribution Strategies for Distributed Systems. In: IFIP WG10.3 INTERNATIONAL CONFERENCE ON DECENTRALIZED AND DISTRIBUTED SYSTEMS, 1993, Amsterdam, The Netherlands. **Proceedings...** Amsterdam: North-Holland, 1993. p.291–303.

JAIN, R. **The Art of Computer Systems Performance Analysis**: techniques for experimental design, measurement, simulation, and modeling. [S.l.]: John Wiley and Sons, 1991. 685p.

JIANG, M.; WILLEY, A. Service-Oriented Architecture for Deploying and Integrating Enterprise Applications. In: WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, WICSA, 5., 2005, Pittsburgh, Pennsylvania. **Proceedings...** [S.l.]: IEEE, 2005. p.272–273.

JOSHI, B.; HOSSEINI, S.; VAIRAVAN, K. A methodology for evaluating load balancing algorithms. In: INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 2., 1993, Spokane, WA, USA. **Proceedings...** Los Alamitos, CA: IEEE Computer Society Press, 1993. p.216–222.

KAHANI, M.; BEADLE, H. Decentralized Approaches for Network Management. **Computer Communications Review**, New York, NY, USA, v.27, n.3, p.36–47, July 1997.

KARA, M. A Global Plan Policy for Coherent Cooperation in Distributed Dynamic Load Balancing Algorithms. **Distributed Systems Engineering Journal**, [S.l.], p.212–223, Dec. 1995.

KATZY, B. R. Design and implementation of virtual organizations. In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 31., 1998, Kohala Coast, HI, USA. **Proceedings...** [S.l.]: IEEE, 1998. v.4, p.142–151.

KLIE, T.; STRAUSS, F. Integrating SNMP agents with XML-based management systems. **IEEE Communications Magazine**, New York, NY, USA, v.42, n.7, p.76–83, July 2004.

KRIS, B. **Resource Based Policies for Load Distribution**. 1996. MSc Thesis, Department of Computer Science, Victoria University of Wellington, Wellington, New Zealand.

MARTIN-FLATIN, J. **Web-Based Management of IP Networks and Systems**. Hoboken, New Jersey: John Wiley & Sons, 2002. v.1.

MILOJICIC, D. S. et al. **Peer-to-Peer Computing**. Palo Alto: HP Laboratories, 2002. Disponível em: <http://www.hpl.hp.com/techreports/2002/HPL-2002-57.html>. Acesso em: jan. 2007.

MOUNTZIA, M.-A.; RODOSEK, G. D. Using the Concept of Intelligent Agents in Fault-Management of Distributed Services. **Journal of Network and Systems Management**, New York, NY, USA, v.7, n.4, p.425–446, 1999.

MUKHI, N. K.; KONURU, R.; CURBERA, F. Cooperative middleware specialization for service oriented architectures. In: INTERNATIONAL WORLD WIDE WEB CONFERENCE ON ALTERNATE TRACK PAPERS & POSTERS, WWW ALT., 13., 2004, New York, NY, USA. **Proceedings...** New York: ACM Press, 2004. p.206–215.

NEISSE, R.; GRANVILLE, L. Z.; BALLVÉ, D. O.; ALMEIDA, M. J. B.; TAROUCO, L. M. R. A Dynamic SNMP to XML Proxy Solution. In: IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, IM, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.481–484.

NEISSE, R.; VIANNA, R. L.; GRANVILLE, L. Z.; ALMEIDA, M. J. B.; TAROUCO, L. M. R. Implementation and Bandwidth Consumption Evaluation of SNMP to Web Services Gateways. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM, NOMS, 2004, Seoul, South Korea. **Proceedings...** [S.l.: s.n.], 2004. p.715–728.

NICHOLS, D. Using idle workstations in a shared computing environment. **ACM SIGOPS Operating Systems Review**, New York, NY, USA, v.21, n.5, p.5–12, 1987. Trabalho apresentado no 11. ACM Symposium on Operating systems principles, 1987.

OASIS. **OASIS Service Oriented Architecture Reference Model Technical Committee (OASIS SOA-RM TC)**. [S.l.], 2006. Disponível em: <http://www.oasis-open.org/committees/soa-rm>. Acesso em: nov. 2006.

PARK, J. T.; CHO, Y. H. A generic manager/agent architecture for TMN applications. In: SINGAPORE IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS SYSTEMS, 1994, Singapore. **Proceedings...** New York: IEEE, 1994. v.2, p.794–798.

PAVLOU, G. et al. On Management Technologies and the Potential of Web Services. **IEEE Communications Magazine**, New York, NY, USA, v.42, n.7, p.58–66, July 2004.

PREECE, A.; DECKER, S. Intelligent Web Services. **IEEE Intelligent Systems**, New York, NY, USA, v.17, n.1, p.15–17, Jan./Feb. 2002.

RIPEANU, M. Peer-to-Peer Architecture Case Study: Gnutella Network. In: INTERNATIONAL CONFERENCE ON PEER-TO-PEER COMPUTING, 1., 2001. **Proceedings...** [S.l.]: IEEE Computer Society, 2001. p.99.

ROSE, M.; MCCLOGHRIE, K. **Structure and Identification of Management Information for TCP/IP-based Internets**: RFC 1155. [S.l.]: Internet Engineering Task Force, Network Working Group, 1990.

ROY, J.; RAMANUJAN, A. Understanding Web Services. **IT Professional**, New York, NY, USA, v.3, n.6, Dec. 2001.

SCHÖNWÄLDER, J.; PRAS, A.; MARTIN-FLATIN, J. P. On the Future of Internet Management Technologies. **IEEE Communications Magazine**, New York, NY, USA, v.41, n.10, p.90–97, Oct. 2003.

SCHÖNWÄLDER, J.; QUITTEK, J.; KAPPLER, C. Building Distributed Management Applications with the IETF Script MIB. **IEEE Journal on Selected Areas in Communications**, New York, NY, USA, v.18, n.5, p.702–714, May 2000.

SHIVARATRI, N. G.; KRUEGER, P.; SINGHAL, M. Load Distributing for Locally Distributed Systems. **Computer**, Los Alamitos, CA, USA, v.25, n.12, p.33–44, Dec. 1992.

SLOTEN, J. van; PRAS, A.; SINDEREN, M. J. van. On the standardisation of Web Services management operations. In: OPEN EUROPEAN SUMMER SCHOOL AND IFIP WG6.3 WORKSHOP, EUNICE, 10., 2004, Tampere, Finland. **Proceedings...** [S.l.: s.n.], 2004. p.143–150.

WANG, Y.-T.; MORRIS, R. J. T. Load Sharing in Distributed Systems. **IEEE Trans. Computers**, [S.l.], v.34, n.3, p.204–217, 1985.

YEMINI, Y.; GOLDSZMIDT, G.; YEMINI, S. Network Management by Delegation. In: IFIP INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT, 2., 1991, Washington, D.C. **Proceedings...** [S.l.: s.n.], 1991. p.95–107.

YILDIZ, B.; PALLICKARA, S. Experiences in Deploying Services within the Axis Container. In: INTERNATIONAL CONFERENCE ON INTERNET AND WEB APPLICATIONS AND SERVICES/ADVANCED INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS, AICT-ICIW, 2006, Guadeloupe, France. **Proceedings...** [S.l.]: IEEE Computer Society, 2006.

YOON-JUNG, O. et al. Interaction Translation Methods for XML/SNMP Gateway. In: IFIP/IEEE INTERNATIONAL WORKSHOP ON DISTRIBUTED SYSTEMS: OPERATIONS AND MANAGEMENT, DSOM, 13., 2002, London, UK. **Proceedings...** [S.l.]: Springer-Verlag, 2002. p.54–65.

YOSHIHARA, K.; ISOMURA, M.; HORIUCHI, H. Dynamic Load Balancing for Distributed Network Management. In: INTEGRATED NETWORK MANAGEMENT, 2003, Colorado Springs, Colorado, US. **Proceedings...** [S.l.]: Kluwer, 2003. p.277–290. (IFIP Conference Proceedings, v.246).