UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS - ICEX
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

ESTUDO DO MODELO DE COMPUTAÇÃO ORIENTADA A SERVIÇOS E SUA APLICAÇÃO A UM SISTEMA DE MINERAÇÃO DE DADOS

BRUNO ESTOLANO GROSSI

Belo Horizonte

30 de Junho de 2005

BRUNO ESTOLANO GROSSI

ESTUDO DO MODELO DE COMPUTAÇÃO ORIENTADA A SERVIÇOS E SUA APLICAÇÃO A UM SISTEMA DE MINERAÇÃO DE DADOS

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte

30 de Junho de 2005

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

Estudo do Modelo de Computação Orientada a Serviços e sua Aplicação a um Sistema de Mineração de Dados

BRUNO ESTOLANO GROSSI

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. Dr. Dorgival Olavo Guedes Neto - Orientador
Departamento de Ciência da Computação – ICEx – UFMG

Prof. Dr. Marcelo de Almeida Maia
Departamento de Computação – ICEB – UFOP

Prof. Dr. Osvaldo Sérgio Farhat de Carvalho
Departamento de Ciência da Computação – ICEx – UFMG

Prof. Dr. Wagner Meira Júnior
Departamento de Ciência da Computação – ICEx – UFMG

Belo Horizonte, 30 de Junho de 2005.

Resumo

Esse trabalho visa estudar a aplicabilidade e o desempenho de Web Services como uma materialização do modelo de arquitetura orientada a serviços (*Service Oriented Architecture, SOA*), com um estudo de caso em uma arquitetura orientada a serviços real. Nesse aspecto, a contribuição principal foi mostrar os passos necessários para a concepção e o desenvolvimento de aplicações baseadas em serviços, apontando dificuldades e possíveis soluções para questões que fazem parte desse novo modelo de programação.

Como aplicação real para o estudo de caso, optou-se por construir um sistema de mineração de dados interativa e multi-usuária, o Tamanduá. Aplicações de mineração de dados e descoberta de conhecimento vêm se tornando muito importantes. A construção de um ambiente distribuído para a mineração de grandes bases inclui diversos requisitos interessantes, como localização e uso dinâmico de serviços remotos, que podem ser mapeados no modelo de SOA. O resultado foi um sistema baseado em Web Services que atende aos requisitos necessários para a mineração de dados de distribuída. A extensibilidade e escalabilidade da plataforma foram medidas para demonstrar a viabilidade e as potencialidades desse tipo de sistema e a vantagem do modelo SOA.

A plataforma Tamanduá está em operação e os experimentos apresentados mostram que ela possui desempenho satisfatório e pode ser expandida para atender ao aumento da demanda de usuários. A análise de desempenho realizada serve também de apoio a novos trabalhos na área de Web Services.

Abstract

This work presents a study on the applicability and performance of Web Services as a materialization of Service Oriented Architectures (SOA), with a case study on a real service oriented system. In this concern, the major contribution was to show the steps necessary to design and build service oriented applications, discussing difficulties along the way and possible solutions for questions that are part of this new programming model.

For the real application that was to be the case study, we decided to build an interactive, multi-user data mining system, Tamanduá (Anteater). Data mining and knowledge discovery applications are becoming extremely important nowadays. The development of a distributed system for mining large data bases includes many interesting requirements, such as finding and using remote services dynamically, which can be directly mapped to the SOA model. The result was a system based on Web Services which meets the requirements for distributed data mining. The extensibility and scalability of the platform were measured to confirm the viability and the potential of this kind of system, as well as the advantages of SOA.

The Tamanduá platform is operational and the experiments presented show that its performance is satisfactory and it can be expanded to meet an increasing user demand. The performance analysis can also be useful as support for future work in the area of Web Services.

À Margarete, com amor...

Agradecimentos

Agradeço à Deus, por estar sempre ao meu lado, nos bons e maus momentos. Agradeço ao meu pai (*in memoriam*) e à minha mãe, por terem me ensidado desde cedo a importância dos estudos e terem me dado o suporte necessário para que eu fizesse o caminho até aqui. E também ao Sávio e ao Luciano, os melhores irmãos que eu poderia ter.

Agradeço imensamente à Margarete, que eu tenho a sorte e alegria de ter ao meu lado nessa caminhada, por suportar minhas ausências, compartilhar sofrimentos e alegrias e me incentivar sempre a continuar em frente; e ao Lucas, esse anjo que Deus nos deu, para aprendermos que a vida é feita de desafios e que a felicidade pode estar em pequenas coisas, como o seu sorriso.

Ao professor e orientador Dorgival Olavo Guedes Neto, que confiou em mim, me orientando e aconselhando sempre que precisei, agradeço por tornar realidade esse trabalho. Foram muitos os conflitos e atribulações, mas poder contar com você me deu forças para ir até o fim. Agradeço ao professor Wagner Meira Jr., pela confiança, pelas oportunidades e pelos valiosos ensinamentos. À Lucília Figueiredo e outros professores da UFOP, pelo empurrão inicial.

Aos meus tios e tias, pela força, em especial à Tia Aparecida, pelo bom exemplo e pela presença sempre nas horas certas, e à Tia Fátima e Tia Ana, que mesmo distantes, sempre torceram muito. Aos meus sogros, Antônio e Renilda, e à toda a nova familia que ganhei, obrigado pelo carinho e amizade, que me deu a tranquilidade necessária.

Aos novos amigos que fiz no DCC, como Nestor Volpini, Fernando Almir, Robert Pinto, Tiago Macambira, e tantos outros do SPEED, que me acompanharam, me ajudaram e tornaram possível esse caminho. Aos bons amigos Allan Hosken e Ademir Alvarenga, pelo companherismo. A muitos outros que eu não citei aqui, mas que com certeza contribuiram com essa jornada, meu muito obrigado.

Sumário

Lista de Figuras

Lista de Tabelas

Esse trabalho visa estudar a aplicabilidade e o desempenho de Web Services como uma materialização do modelo de arquitetura orientada a serviços (*Service Oriented Architecture, SOA*), com um estudo de caso em uma arquitetura orientada a serviços real. Nesse aspecto, a contribuição principal foi mostrar os passos necessários para a concepção e o desenvolvimento de aplicações baseadas em serviços, apontando dificuldades e possíveis soluções para questões que fazem parte desse novo modelo de programação.

Como aplicação real para o estudo de caso, optou-se por construir um sistema de mineração de dados interativa e multi-usuária, o Tamanduá. Aplicações de mineração de dados e descoberta de conhecimento vêm se tornando muito importantes. A construção de um ambiente distribuído para a mineração de grandes bases inclui diversos requisitos interessantes, como localização e uso dinâmico de serviços remotos, que podem ser mapeados no modelo de SOA. O resultado foi um sistema baseado em Web Services que atende aos requisitos necessários para a mineração de dados de distribuída. A extensibilidade e escalabilidade da plataforma foram medidas para demonstrar a viabilidade e as potencialidades desse tipo de sistema e a vantagem do modelo SOA.

A plataforma Tamanduá está em operação e os experimentos apresentados mostram que ela possui desempenho satisfatório e pode ser expandida para atender ao aumento da demanda de usuários. A análise de desempenho realizada serve também de apoio a novos trabalhos na área de Web Services.

This work presents a study on the applicability and performance of Web Services as a materialization of Service Oriented Architectures (SOA), with a case study on a real service oriented system. In this concern, the major contribution was to show the steps necessary to design and build service oriented applications, discussing difficulties along the way and possible solutions for questions that are part of this new programming model.

For the real application that was to be the case study, we decided to build an interactive, multi-user data mining system, Tamanduá (Anteater). Data mining and knowledge discovery applications are becoming extremely important nowadays. The development of a distributed system for mining large data bases includes many interesting requirements, such as finding and using remote services dynamically, which can be directly mapped to the SOA model. The result was a system based on Web Services which meets the requirements for distributed data mining. The extensibility and scalability of the platform were measured to confirm the viability and the potential of this kind of system, as well as the advantages of SOA.

The Tamanduá platform is operational and the experiments presented show that its performance is satisfactory and it can be expanded to meet an increasing user demand. The performance analysis can also be useful as support for future work in the area of Web Services.

Capítulo 1

Introdução

1.1 Apresentação

O conceito de Serviços Internet é muito amplo e tem origem nos sistemas cliente/servidor das décadas de 70 e 80. Esse modelo emergiu da convergência entre computadores e redes de comunicação que permitiu o uso de computadores simples — apenas com interface gráfica (GUI) e menor poder de processamento — e portanto mais disponíveis para a realização de tarefas complexas, numa interação mais dinâmica com o usuário final. Isso foi possível porque as aplicações cliente/servidor são normalmente divididas em camadas, como interface com o usuário, camada de negócio e camada de persistência. Dessa forma, torna-se possível a execução na máquina do cliente apenas da interface, que se comunica pela rede com as demais camadas que podem estar em máquinas com mais recursos computacionais. O usuário (ou cliente), com uma aplicação "leve", pode comandar execuções que exigem muito processamento em máquinas remotas com maior capacidade. Além disso, trazendo parte da computação para próximo do usuário e provendo uma interface mais flexível e fácil de usar, sistemas cliente/servidor podem aumentar a produtividade, incrementar a satisfação do usuário e reduzir significativamente custos de treinamento [4].

Com o advento da Internet e a popularização da *World Wide Web* (WWW) — um caso especial e importante de sistema cliente/servidor — um número crescente de empresas, órgãos governamentais e até mesmo programadores têm disponibilizado na web as mais diversas aplicações. Um único tipo de aplicação, o navegador web, é suficiente para o acesso a bibliotecas e museus virtuais, comércio eletrônico (onde se pode comprar de livros a carros, entre diversos produtos), clientes de e-mail na WWW (*webmails*), jornais e revistas, serviços de pesquisa e localização em mapas, *sites* de relacionamento e milhares de outras opções que se encontram disponíveis na rede mundial. O crescimento da WWW levou à criação de um órgão normativo, o *World Wide Web Consortium* (W3C, http://www.w3.org/).

Inicialmente, a WWW era feita apenas de recursos estáticos, porém novas aplicações e serviços exigiram tecnologias para a construção dinâmica de páginas e documentos. Nesses sistemas, as páginas devem ser criadas a partir da interação com o usuário, ou seja, geradas a partir de parâmetros baseados nas ações do mesmo. Um exemplo simples disso é um sistema de buscas, onde a lista de resultados deve ser criada baseada nas palavras chave selecionadas pelo usuário — diferente dos antigos catalogadores de páginas, que eram serviços estáticos onde o usuário procurava um endereço baseado em categorias, onde as páginas não dependiam de entrada dos usuários.

Diversas tecnologias surgiram para permitir a criação dinâmica de páginas WWW na Internet, como CGI, PHP, JSP, ASP e outras. Cada uma tem suas qualidades e defeitos, mas todas se comunicam diretamente com o usuário através de uma linguagem (não padronizada) de descrição de interfaces web, o *Hypertext Markup Language* (HTML [1]). Apesar de o HTML ter sido um dos grandes responsáveis pela popularização da WWW e da Internet, a falta de padronização e a sua natureza voltada para a formatação da interface com o usuário torna muito difícil a interação entre aplicações.

Interoperabilidade entre aplicações é capacidade de dois programas de computador interagirem entre si, trocando informações e serviços de maneira a atingir os resultados esperados por ambos. A comunicação entre empresas e instituições de forma automatizada é muito importante no mundo globalizado e dinâmico de hoje. As empresas informatizadas necessitam que seus programas comuniquem-se diretamente com seus clientes, fornecedores e outras unidades, sem a intervenção humana. Por exemplo, compras e solicitações de produtos podem ser processadas pela aplicação de controle de estoques da empresa diretamente ao serviço de recebimento de pedidos do fornecedor, sem a necessidade da intervenção de um funcionário; um sistema de cartões de crédito pode consultar outros sistemas sobre o valor de moedas estrangeiras, evitando a necessidade de um usuário para a inserção dessa informação.

A interoperabilidade através da rede é um objetivo com uma longa história mas apenas sucessos parciais [5], como o *Distributed Component Object Model* (DCOM [2]), o *Common Object Request Broker Architecture* (CORBA [3]) e o *Java Remote Method Invocation* (RMI) sobre o *Internet Inter-ORB Protocol* (IIOP [4]). Essas soluções são normalmente específicas para uma linguagem ou tecnologia proprietária e trazem vários problemas para a comunicação cliente/servidor na Internet.

Para solucionar esses problemas o W3C, com o apoio de diversas empresas, criou um padrão de comunicação entre aplicações baseado em tecnologias conhecidas e pa-

[1] O W3C tenta padronizar o HTML com uma recomendação disponível em http://www.w3.org/TR/html4/, porém essa especificação nem sempre é respeitada pelos fabricantes de navegadores web. Para solucionar esse impasse, o W3C tenta tornar padrão uma reformulação do HTML baseada em XML, o XHTML (http://www.w3.org/TR/xhtml1/).

[2] Criado pela Microsoft: http://www.microsoft.com/com/default.mspx

[3] Definido pelo Object Management Group: http://www.omg.org/

[4] RMI-IIOP: http://java.sun.com/products/rmi-iiop/

dronizadas, os *Web Services*. Essa solução tenta tornar a comunicação entre aplicações tão fácil quanto a Web tornou a publicação de documentos [6]. Definindo um conjunto de formatos de dados estruturados e legíveis por humanos e fazendo-os abertos e independentes de qualquer linguagem ou plataforma particular, os projetistas de *Web Services* esperam criar um arcabouço para interoperabilidade programa-a-programa. Isso permite a criação de aplicações distribuídas com vários componentes espalhados pela Internet, trabalhando em cooperação para a realização de tarefas. Os cenários para a utilização dessas tecnologias são os mais variados: livrarias virtuais e outros serviços de comércio eletrônico podem receber pedidos de compras diretamente das aplicações de controle de estoque de seus clientes, consultando o valor do transporte na empresa responsável; sites podem implantar busca na web ou em dicionários, usando o processamento de serviços já conhecidos; etc. Tudo sem a necessidade de saber nenhum detalhe sobre como foram implementados esses serviços.

Outra característica visível de *Web Services* é a possibilidade de criar novas aplicações e serviços apenas compondo e combinando serviços já existentes. Vários são os serviços já disponíveis na Internet para realizar as mais diferentes tarefas, como consultar endereços, converter arquivos entre formatos diferentes, checar valor de câmbio e ações em bolsas de valores, consultar clima de determinadas regiões, entre outros [5]. Qualquer um pode tornar público um serviço para que outras aplicações o utilizem seguindo os padrões definidos. O objetivo de *web services* é que esses serviços possam ser facilmente compostos para gerar novas aplicações, sem que código precise ser desenvolvido para recompor essas operações.

A idéia de aplicativos que usam massivamente serviços independentes espalhados pela rede constituem o paradigma denominado Computação Orientada a Serviços (SOC, *Service Oriented Computing*). O modelo de SOC se baseia em arquiteturas de software orientadas a serviços (SOA, *Service Oriented Architecture*), que é uma maneira de reorganizar aplicações e infraestruturas de software em um conjunto de serviços interativos [7]. A idéia de SOA é relativamente nova, apesar de o conceito de serviço ser mais antigo. Esse estilo arquitetônico de fazer software promove o pouco acoplamento entre os componentes e a independência de tecnologias específicas para que eles sejam reutilizáveis. Algumas vantagens do uso de SOA são a adaptação das aplicações a tecnologias em constante evolução, a fácil integração de aplicações com outros sistemas, o reaproveitamento de investimentos em aplicações legadas e a criação prática e rápida de novos processos a partir de serviços existentes.

Como serviços provêem uma forma uniforme e largamente disponível (ubíqua) de distribuir informações para uma grande quantidade de dispositivos computacionais (como *notebooks*, PDAs, telefones celulares, entre outros) e plataformas de software

[5]Consulte, por exemplo, o *site* http://xmethods.com/ para uma lista de serviços públicos disponíveis.

(como UNIX ou Windows), eles constituem o próximo grande passo em computação distribuída [7].

O paradigma SOC/SOA pode ser aplicado para solucionar problemas nas mais variadas áreas de conhecimento [8, 9, 10, 11]. Um uso comum de SOA é na criação de aplicações com partes bem distintas, que interagem entre si e podem ser executadas em máquinas diferentes, como no caso da mineração de dados, que usaremos para ilustrar esse paradigma. Mineração de dados, como veremos em detalhes posteriormente, é o processamento de um conjunto de dados para a obtenção de conhecimento útil ao usuário. Segundo Raghu Ramakrishnan [12], o principal gargalo em mineração de dados é o elemento humano envolvido na análise dos dados, o que torna necessário o desenvolvimento de técnicas para gerenciar eficientemente esse processo, reduzindo o tempo para se chegar a descobertas úteis. Esse é o objetivo da criação da plataforma Tamanduá, apresentada neste documento. A separação do sistema em entidades identificadas no processo de mineração responsáveis pela gerência de bases de dados (originais e derivados), processamento de algoritmos e visualização de dados mostra-se adequada para a aplicação dos princípios de SOC/SOA. Uma melhor organização e distribuição dessas "tarefas" na forma de serviços oferecidos independentemente possibilita a configuração dinâmica e a flexibilização desse processo interativo.

1.2 Objetivos

Tendo em vista a discussão anterior, o objetivo deste trabalho é estudar arquiteturas orientadas a serviços (SOA) usando *Web Services* e as novas tecnologias que têm surgido para dar suporte a aplicações baseadas nesse paradigma. A criação sistemas desse tipo depende do estudo e avaliação de diversas plataformas, padrões e especificações existentes para que se possa escolher quais utilizar. É importante conhecer as várias opções e entender o seu funcionamento para proporcionar ao usuário a melhor experiência possível com o sistema.

Essa dissertação visa estudar a aplicabilidade e o desempenho de *Web Services* com um estudo de caso em uma arquitetura orientada a serviços real. A contribuição principal será a ilustração dos passos necessários para a concepção e o desenvolvimento de aplicações baseadas em serviços, apontando dificuldades e possíveis soluções para questões que fazem parte desse novo modelo de programação.

1.3 O estudo de caso

Aplicações de mineração de dados e descoberta de conhecimento vêm se tornando mais importantes a cada dia. Como veremos em posteriormente, a construção de um ambiente distribuído para a mineração de grandes bases de dados define diversos requisitos

interessantes, como localização e uso dinâmico de serviços remotos, que podem ser mapeados no modelo SOA.

Com base nessa observação, o caso de uso utilizado será o Tamanduá, serviço implementado com uma uma arquitetura baseada em *Web Services* que atende aos requisitos necessários para a mineração de dados de forma distribuída. Essa plataforma já se encontra em operação e a sua extensibilidade e escalabilidade serão medidas para demonstrar a viabilidade e as potencialidades desse tipo de sistema.

1.4 Síntese dos resultados

O Tamanduá é, então, um dos resultados mais visíveis desta dissertação, uma plataforma de mineração de dados baseada em *Web Services* sendo usada para minerar dados reais em projetos do governo e ligados à UFMG. A implantação do Tamanduá se mostrou complexa devido a requisitos específicos do problema, porém novas técnicas de desenvolvimento têm surgido para auxiliar a tarefa do programador. Este ainda não é um sistema completo de descoberta de conhecimento em bases de dados, porém demonstra o grande potencial do uso de SOA e *Web Services*. A estruturação do sistema permite a duplicação de servidores para melhorar o desempenho e a extensão da plataforma com a adição de novos serviços.

A plataforma Tamanduá está em operação e os experimentos apresentados na seção 5.3 mostram que ela possui bom desempenho e pode ser escalonada para atender ao aumento da demanda de usuários. A análise de desempenho realizada aqui serve também de apoio a novos trabalhos na área de *Web Services*.

1.5 Organização do texto

O restante desta dissertação está organizado da seguinte maneira: o capítulo 2 apresenta conceitos fundamentais para o completo entendimento da dissertação; o capítulo 3 fala sobre trabalhos relacionados aos assuntos aqui abordados; o capítulo 4 discute as opções existentes para a criação de arquiteturas orientadas a serviços e mostra como esse paradigma foi adotado na construção do Tamanduá; o capítulo 5 discute a adequação do uso de uma arquitetura de serviços ao problema de descoberta de conhecimento e os resultados dos experimentos realizados para a avaliação de desempenho e escalabilidade do sistema; finalmente, o capítulo 6 conclui o trabalho e indica algumas direções para trabalhos futuros.

Capítulo 2

Conceitos básicos

Neste capítulo são apresentados alguns conceitos importantes para o bom entendimento do texto. Para atingirmos nosso objetivo de discutir o uso de *Web Services* e SOA para a construção de aplicações distribuídas, falaremos de algumas das necessidades para a implementação de *Web Services* e das ferramentas utilizadas. Além disso, discutiremos outras tecnologias existentes, como grades computacionais, e como elas se inserem no conceito de arquiteturas orientadas a serviços. Mostraremos também uma visão geral sobre mineração de dados, que será utilizada como motivação para a construção da aplicação baseada em SOA, apresentada no capítulo 4.

Como estamos discutindo a criação de serviços na Internet, primeiro é necessário entender o que é um serviço. Serviço é um recurso abstrato que representa a capacidade de executar tarefas que possuam uma funcionalidade coerente na visão do provedor e do consumidor [13]. De forma geral, um serviço computacional (implementado por um programa de computador) pode ser oferecido através de uma interface com o usuário, como um aplicativo ou *site* Web, ou pode ser implementado e disponibilizado por um componente de software para ser usado por outros componentes, como um serviço de impressão utilizado por diversos aplicativos. Assim como a interface com o usuário deve ser de fácil entendimento, um serviço prestado a outro programa deve ter uma interface e um protocolo de comunicação bem definidos para que os clientes possam utilizá-lo.

Uma interface é uma especificação que existe entre componentes de software que define as formas de interação entre eles — muitas vezes chamada de contrato. Uma interface pode definir, por exemplo, métodos, constantes e tipos de dados. O protocolo de comunicação é uma especificação do conjunto de mensagens (quais as mensagens e qual o formato de cada uma) trocadas entre os aplicativos.

Erroneamente, pode-se considerar que serviço é um outro termo para denominar componentes. Muitas das boas práticas das metodologias de desenvolvimento e integração baseado em componentes são relevantes na definição de serviços, mas esses últimos são independentes de arcabouços técnicos específicos de linguagens ou pla-

taformas [14, 15]. Algumas definições sugerem que serviços é um outro termo para interfaces. Um serviço pode incluir uma interface, mas ele não assume que o consumidor pode utilizar mecanismos de implementação da maneira como interfaces vêm sendo tradicionalmente utilizadas.

Consideraremos então que serviços são funcionalidades expostas por componentes, através de um contrato (ou interface), para serem utilizados por outros componentes. Essa definição exclui serviços oferecidos diretamente ao usuário final, que serão chamados aqui de aplicativos ou programas.

2.1 *Web Services*

Para se aplicar os conceitos de *Web services* é preciso entender as idéias por trás da sua criação e conhecer as principais tecnologias e padrões envolvidas na sua criação.

2.1.1 Princípios

Como mencionado anteriormente, há uma grande demanda por soluções que viabilizem a integração entre programas de computador implementados por diferentes empresas e implantados em lugares distintos. Para que isso seja possível é necessário que aplicações diferentes, nas mais diversas plataformas, consigam solicitar serviços de outros programas mesmo em máquinas remotas, muitas vezes em outras organizações, através de trocas de mensagens que devem ser entendidas por todas as partes envolvidas na comunicação. Esse modelo de solicitação de serviços é bastante semelhante a uma chamada remota de procedimento (RPC, *Remote Procedure Call*), onde uma aplicação executa um procedimento localizado em outro programa através da rede.

Para a comunicação entre as empresas, é importante que os protocolos utilizados sejam amplamente difundidos para tornar possível e viável a integração entre os sistemas com a invocação de métodos remotos através da Internet, um ambiente extremamente heterogêneo. O uso da Internet para a comunicação nesse caso é uma boa opção, apesar da heterogeneidade, já que muitas delas já possuem conexão com a rede mundial. A WWW possibilitou a criação de diversas aplicações acessíveis através de um navegador, de tal forma que o usuário pode utilizar os mais diversos serviços somente com o cliente Web. Há servidores Web implementados para as mais diversas plataformas existentes. O protocolo de acesso a recursos remotos na WWW, o HTTP, é também uma boa via para a transmissão das mensagens entre as empresas, uma vez que sua implementação é bastante conhecida e padronizada.

A criação de serviços baseados na WWW para a comunicação entre componentes de software que sejam acessíveis através de protocolos bem definidos e amplamente acei-

tos na Internet é o que se entende normalmente pelo termo *Web Services* [1]. É difícil definir exatamente o que são *Web Services* devido à grande quantidade de tecnologias envolvidas e há sempre confusões. Segundo Almeida e Menascé [16], *Web Service* descreve funcionalidades de negócio específicas expostas por uma companhia, usualmente através de uma conexão Internet, com o propósito de prover uma maneira para que outras companhias ou programas usem o serviço. Chappell e Jewell [1] definem WS como uma peça da lógica de negócios, localizada em algum lugar na Internet, que é acessível através de protocolos da Internet baseados em padrões, como HTTP ou SMTP.

2.1.2 Arquitetura de serviços

Como já mencionado brevemente, uma Arquitetura Orientada a Serviços (SOA) é um conjunto de princípios para construção de software, no qual os mais importantes são a procura por uma modularidade apropriada e pouco acoplamento entre os componentes. Como veremos, há várias tecnologias para implementar serviços e qualquer uma delas pode ser usada para a montagem de aplicações baseadas em SOA. As principais características de SOA são:

- serviços são componentes de software que publicaram seus contratos (ou interfaces); esses contratos são independentes de plataforma, linguagem ou sistema operacional;
- consumidores podem encontrar serviços dinamicamente;
- serviços são interoperáveis.

Uma visão geral do paradigma SOA é mostrado na figura 2.1.

Uma aplicação que utiliza SOA normalmente é dividida em camadas, como na figura 2.2. A camada de serviços é responsável pela interface entre a camada de negócios (onde o processamento é realizado) e as aplicações externas. Pode-se por exemplo, criar uma camada abstrata baseada em *Web Services* para aplicações legadas e, posteriormente substituí-la, se for conveniente. Para acessar os serviços, pode ser utilizado uma interface com o usuário (GUI ou WWW), ou por outro componente responsável pelo gerenciamento do processo (a aplicação que está utilizando esse serviço).

O mais importante princípio de SOA é a interoperabilidade. É fundamental que todos os componentes se comuniquem independente da linguagem em que foram construídos, do sistema operacional em que estão sendo executados e da arquitetura de hardware. *Web Services* atendem bem a esse requisito já que, como visto anteriormente,

[1] Nessa dissertação seguiremos o uso que vem sendo comumente adotado em língua portuguesa, usando o termo *Web Service* (WS) para a designação do conjunto específico de tecnologias apresentadas nesta seção, enquanto o termo traduzido (serviço Web) será usado para a referência a serviços (ou aplicativos) sobre a WWW de forma mais genérica.

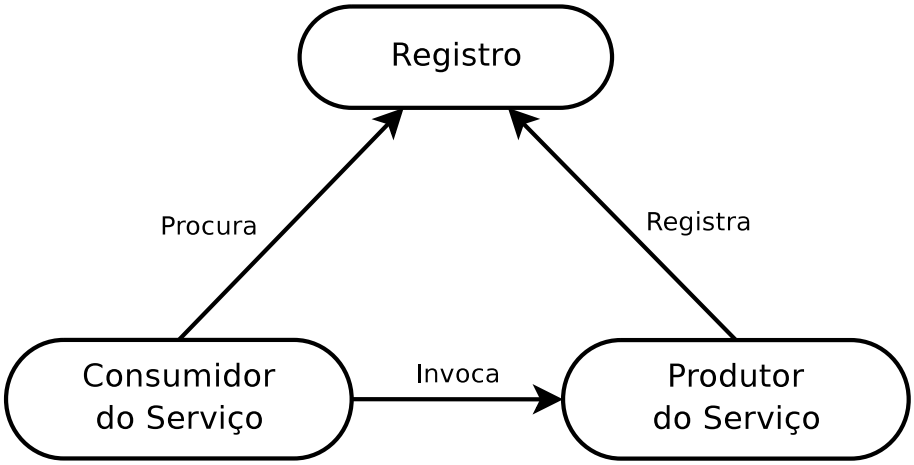

Figura 2.1: Paradigma básico para Arquiteturas Orientadas a Serviços (SOA)

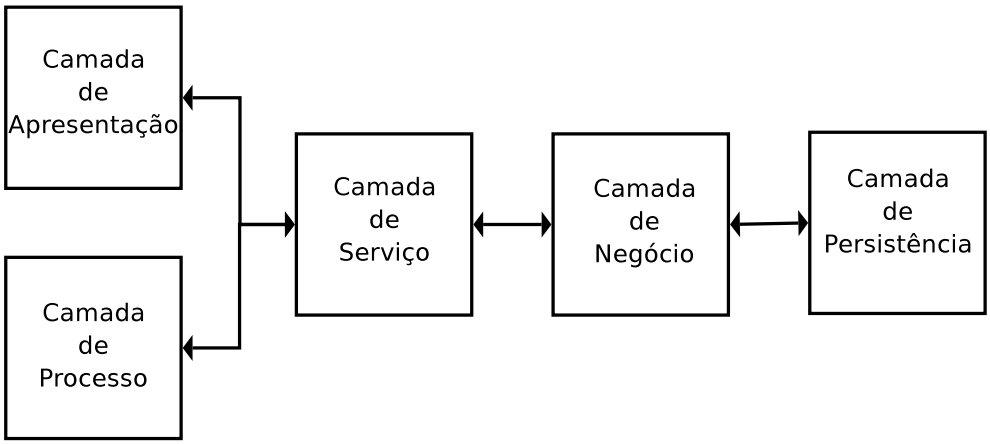

Figura 2.2: Diferentes camadas de uma aplicação orientada a serviços

utilizam protocolos e formatos de documentos padronizados e aceitos. Entretanto, apesar de a interoperabilidade ser o seu benefício chave, a diversidade de implementações de bibliotecas para suporte de *Web Services* e a flexibilidade permitida em alguns pontos da especificação torna difícil a concretização dessa característica. A Organização para Interoperabilidade de Web Services (WS-I, *Web Services Interoperability Organization* [2]) é uma organização da indústria para a promoção da interoperabilidade entre esses serviços baseada em definições e padrões XML relacionados aceitos por todos. O WS-I cria guias e ferramentas para ajudar desenvolvedores a criar e certificar *Web Services* compatíveis. O primeiro guia da organização foi o *WS-I Basic Profile 1.0*, que tenta aperfeiçoar a interoperabilidade pretendida na especificação.

[2]WS-I: http://www.ws-i.org/

2.1.3 Tecnologias componentes

Com base nos princípios de SOA, para se implementar serviços no modelo de *Web Services*, com ampla aceitação, fácil acesso, independência de arquitetura e fácil processamento por aplicações, um conjunto de elementos é necessário: um protocolo de transmissão de dados amplamente adotado, um formato de representação de dados independente de arquitetura, uma forma de se exprimir claramente a localização, funcionalidades e interface de acesso de cada serviço e um protocolo que expresse a semântica do processo de requisição de serviços e obtenção de respostas que se pretende implantar. Cada um desses elementos é discutido a seguir, com as opções normalmente adotadas para *Web Services*.

O *HyperText Transfer Protocol* (HTTP [17]) é o protocolo utilizado na WWW para a transferência de recursos (como páginas, documentos e imagens). Como visto anteriormente, seu uso é bem padronizado e amplamente aceito, permitindo a flexibilidade necessária às aplicações Web. A sua adoção para a invocação de serviços remotos resolve um dos maiores problemas desse tipo de comunicação: a restrição imposta por administradores de rede, que limitam acesso externo a servidores, mantendo o acesso a portos TCP fechados a menos que sejam absolutamente necessários. O protocolo HTTP, por ser bastante utilizado, tem seu porto padrão (associado normalmente a servidores WWW) normalmente liberado na maioria das organizações, o que o torna um bom candidato como o canal de transferência de dados. Arquiteturas tradicionais de acesso remoto como DCOM, CORBA e RMI requerem múltiplos portos TCP abertos através de um *firewall*, o que muitas vezes impede conexões sofisticadas, inclusive entre organizações, para conexões entre os serviços. Por utilizarem HTTP, o uso de *Web Services* não requer portos adicionais ou permissões de acesso além daquelas de servidores WWW padrão.

Depois do protocolo de comunicação, um segundo problema a ser resolvido é o do formato de representação de dados a ser utilizado. Em um ambiente heterogêneo como a Internet é necessário um formato bem definido para utilização em protocolos e documentos, que possibilite a compreensão universal das mensagens trocadas entre os programas de computadores com as mais diversas plataformas. Nesse contexto é que surgiu o(a) XML (*Extensible Markup Language* [18]), um formato de texto baseado em marcações, aberto e flexível, padronizado pelo W3C [3]. O XML é hoje apoiado mundialmente pelas maiores empresas de tecnologia e usado para a troca e armazenamento de diversos tipos de dados [19].

Por ser baseado em marcações de texto (assim como o HTML) XML é, ao contrário de formatos binários, passível de entendimento por seres humanos, ao mesmo tempo em que pode ser processado por software ou hardware para a extração dos dados

[3]World Wide Web Consortium (W3C): http://www.w3c.org

auto-descritos. O sucesso de XML vem de sua sintaxe clara e simples e sua estrutura sem ambigüidades. O XML permite a verificação da qualidade de um documento, através de regras de sintaxe, verificação de vínculo interno, comparação com modelos de documento e tipos de dados [19].

A descrição, publicação e localização de modo uniforme é outro fator importante para a utilização de serviços distribuídos. O WSDL (*Web Services Description Language* [20]) é um formato de texto baseado em XML, criado para descrever os serviços disponibilizados a partir de um determinado endereço na Internet, incluindo seus parâmetros e protocolos sobre os quais o serviço pode ser solicitado. Essas descrições são documentos WSDL, utilizados pelos clientes para a criação das mensagens que serão utilizadas para solicitação dos serviços. Tais descrições podem ser publicadas em qualquer repositório, normalmente implementados utilizando o protocolo UDDI (*Universal Description, Discovery and Integration* [21]). Com o UDDI é possível descobrir dinamicamente, através de uma série de interfaces padronizadas, quais serviços são publicados por cada provedor de serviços e quais as interfaces oferecidas pelos mesmos [22].

Para o envio da requisição por serviços sobre HTTP com o controle da semântica de requisição e resposta específica desse tipo de comunicação pode ser utilizado o protocolo SOAP (*Simple Object Access Protocol* [23]), que é capaz de descrever completamente uma chamada remota de procedimento, com seus respectivos parâmetros, mesmo as mais complexas estruturas de dados. O SOAP é, em sua essência, um protocolo assíncrono baseado em XML para intercâmbio de informações em ambientes distribuídos [1] [4]. Fundamentalmente, há transmissões de via única do remetente para o receptor, e o modelo RPC é apenas um caso especializado que combina múltiplas mensagens assíncronas em um padrão de requisição e resposta. Existe também um outro modelo, menos utilizado, orientado a documentos, onde é enviado em uma mensagem SOAP um documento XML qualquer, que obedece um esquema determinado pelo dono do serviço (por exemplo, definido com *XML Schema*). Uma mensagem SOAP (também chamada envelope) pode incluir dados anexados (*SOAP with Attachments*, SwA), inclusive em formatos binários [24].

A conjunção de vários serviços para a realização de tarefas em comum torna necessário um componente que agrupe e comande esse conjunto. Esse processo é denominado orquestração de serviços e já existem algumas ferramentas e padrões propostos para a execução desses processos aglomerados. A OASIS (*Organization for the Advancement of Structured Information Standards*, http://www.oasis-open.org) tem um padrão chamado *Business Process Execution Language* (BPEL) que tem sido adotado por diversas empresas.

Como mencionado anteriormente, o *WS-I Basic Profile 1.0*, que tenta aperfeiçoar

[4] O SOAP pode, em princípio, ser utilizado sobre qualquer protocolo de transporte, como o SMTP; entretanto, o HTTP é o mais comum atualmente.

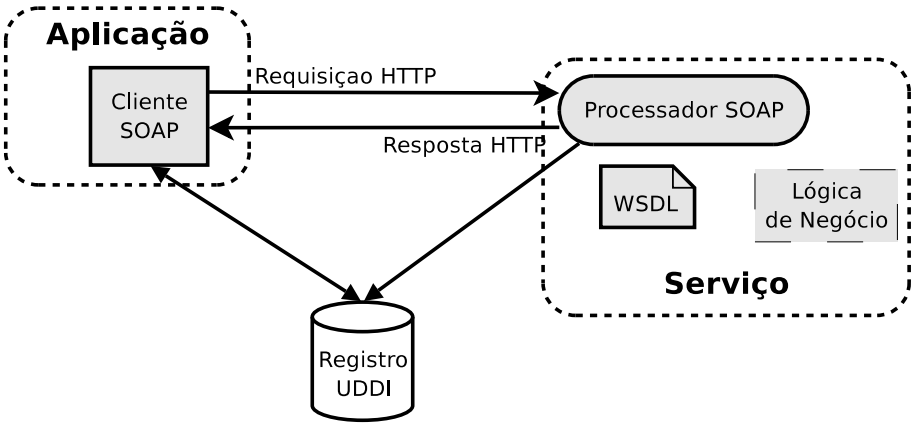

Figura 2.3: Interação simples de *Web Services* [1]

a interoperabilidade pretendida na especificação de *Web Services* define o uso de XML Schema 1.0, SOAP 1.1, WSDL 1.1 e UDDI 2.0. A figura 2.3 ilustra como se dá a interação entre esses componentes.

2.2 Tecnologias relacionadas

Como mencionado, apesar de *Web Services* (utilizando o conjunto HTTP, XML, SOAP, UDDI, etc.) ser o foco desta dissertação, essa não é a única forma de se implementar uma arquitetura orientada a serviços. A título de ilustração, esta seção discute outros mecanismos recentes que dão suporte à criação de serviços computacionais distribuídos, servindo de apoio para o desenvolvimento de aplicações baseadas em SOA.

2.2.1 *REpresentational State Transfer*

Roy Fielding, em sua tese de doutorado [25], propôs um outro estilo arquitetônico para a criação de serviços Web, diferente do modelo de chamada remota de procedimento adotado pelo SOAP, denominado REST (*REpresentational State Transfer*). REST é um conjunto coordenado de regras arquitetônicas que procura minimizar a latência e a comunicação em rede enquanto maximiza a independência e escalabilidade da implementações de componentes. Na visão do autor, o protocolo HTTP já é, em si, um protocolo do tipo requisição/resposta, por isso esse estilo descreve uma arquitetura que utiliza o que a Web tem de melhor, propondo o uso de um conjunto pequeno e abrangente de métodos remotos (mapeados nos métodos do protocolo HTTP: GET, POST, PUT, DELETE, etc.) aplicável a qualquer coisa — mais especificamente, qualquer recurso. Ou seja, segundo o autor, não são necessários outros métodos (por exemplo, os mapeados em objetos numa comunicação RPC) além daqueles já definidos pelo HTTP.

Isso porque, independente de qual seja o cliente e o servidor Web utilizado, sabe-se que com uma URI do tipo *http://www.exemplo.com/arquivo* pode-se retirar o documento associado usando o comando GET e substituí-lo por outro usando o comando PUT, sem nenhum conhecimento a mais além da especificação do protocolo HTTP.

Dessa forma, a comunicação é feita através da troca de recursos. Por exemplo, um documento pode ser acessado, alterado localmente e re-enviado ao servidor, alterando-se assim o seu estado. Em outro cenário, um sistema para empresas aéreas pode permitir que um documento descrevendo (com um esquema pré-conhecido) a origem, destino e hora do vôo seja postado no servidor, disparando o subsistema de emissão de passagem aérea, que gera um bilhete eletrônico a partir daqueles dados. Para consulta sobre a confirmação da passagem, o cliente acessa aquele documento passando parâmetros pela URL.

Esse estilo de criação de serviços também pode ser chamado de *Web Services*, pois permite a comunicação entre programas de computador através da troca de arquivos XML pela WWW. Tem ainda a vantagem de permitir ao usuário a realização de consultas aos recursos com o uso de um navegador WWW comum.

O REST vem sendo adotado por alguns provedores de serviços, normalmente em paralelo a uma versão usando SOAP. Esse é, por exemplo, o caso da Amazon.com, que utiliza *Web Services* com versões em SOAP e REST para acesso aos seus serviços (mais informações em http://www.amazon.com/webservices). Entretanto, não há uma padronização suficiente para a criação de um modelo único para que possa ser adotado largamente entre as empresas.

2.2.2 Grades computacionais (*Grids*)

O barateamento e a popularização dos computadores nos últimos anos vem tornando sub-utilizados seus recursos. Processadores, memória e até discos ficam ociosos a maior parte do tempo. Numa empresa, por exemplo, os computadores ficam desligados ou simplesmente parados fora do horário de expediente e durante os intervalos dos funcionários. O grande crescimento da banda disponível de rede interligando todos esses aparelhos pela Internet torna viável a utilização desses recursos por terceiros, remotamente. Por exemplo, um usuário que precisar de maior poder de processamento pode utilizar processadores que estão parados na rede para a execução de seus programas, tendo a sensação de dispor de um grande processador, e oferecer o seu para que outros o utilizem quando não estiver precisando. Esse é o princípio básico das Grades Computacionais (*Grid Computing*).

Grades vêm ganhando grande popularidade em sistemas distribuídos [5], oferecendo

[5]Há vários projetos, entre sistemas, *middlewares* e aplicações, para a implementação de grades [26]. Entre eles: Globus Toolkit (http://www-unix.globus.org/toolkit/), Legion (http://www.cs.virginia.edu/~legion/), o projeto brasileiro OurGrid (http://www.ourgrid.org/) e o Da-

tecnologias e infra-estruturas para suportar compartilhamento e uso coordenado de diversos recursos em organizações virtuais dinâmicas e distribuídas [27], proporcionando uma arquitetura para uso de recursos computacionais em escala global. Dentro do conceito de grades, é possível executar uma aplicação (ou várias, com um objetivo único) em diversos processadores espalhados pela Internet.

Ian Foster e outros [28] sugerem que para uma fácil virtualização — ou seja, o uso, através de uma mesma interface, de máquinas variadas, possivelmente de diferentes arquiteturas e operando sob diferentes sistemas operacionais —, todos os recursos em uma grade computacional podem ser vistos como serviços: recursos de processamento e de armazenamento, redes, programas, bancos de dados, etc. Para isso, os autores definem a idéia de Serviços em Grade (*Grid Services*): um *Web Service* que provê um conjunto de interfaces bem definidas e que seguem convenções específicas, com mecanismos para criação, nomeação e descoberta de instâncias de serviços na grade, transparência de localização e multiplicidade de protocolos e integração com plataformas já existentes [29].

Web Services e computação em grade estão atualmente bastante interligados. Ambos se propõem a criar ambientes de execução distribuídos, onde diversos componentes ou partes do sistema, que podem estar espalhados por diversas máquinas distintas, realizam uma fração do problema para alcançar o resultado esperado. Entretanto, essas tecnologias se diferenciam em alguns aspectos. Grades são planejadas para o compartilhamento de recursos, com manutenção de estado em cada componente, enquanto *Web Services* são orientados a serviços que podem ser totalmente independentes e descolados do restante da aplicação, onde podemos ter cada requisição totalmente independente da anterior.

Serviços em Grade podem ser implementados sobre *Web Services*, como acontece com a arquitetura de serviços do Globus, o *Open Grid Services Architecture* (OGSA). Baseada nos conceitos e tecnologias de Grid e *Web Services*, essa arquitetura define uma semântica unificada de serviços, mecanismos para criação, nomeação e descoberta de instâncias de serviços na grade, transparência de localização e multiplicidade de protocolos e integração com plataformas já existentes [29]. Entretanto, OGSA usa uma extensão de WSDL [30], não compatível com os padrões de *Web Services* atuais. Além do mais, ainda não há um padrão definitivo e aceito universalmente para a construção de grades computacionais e para o uso de serviços em grade, como há para *Web Services*. Novas tecnologias e especificações, ainda em estudo, vêm evoluindo e aperfeiçoando essas duas tecnologias, tornando-as mais próximas. Essas especificações fazem parte do *WS-Resource Framework* [30], ainda em desenvolvimento.

taGrid Project europeu (EU-DataGrid, http://eu-datagrid.web.cern.ch/eu-datagrid/).

2.2.3 Serviços *Peer-to-Peer*

Redes *peer-to-peer* (P2P) ganharam muita popularidade com os sistemas de trocas de arquivos — principalmente músicas e filmes — entre usuários da Internet. Desconsiderando as questões legais envolvidas, esses sistemas se mostraram eficientes e escaláveis, na medida em que formam redes virtuais entre diversas máquinas, replicando recursos e os disponibilizando para outros que estejam mais próximos. Dessa forma, novas rotas podem ser traçadas através da Internet, "pegando carona" em máquinas de outros usuários para evitar caminhos com muito tráfego.

Fundamentalmente, muitas das redes P2P modernas funcionam baseadas em um mecanismo de publicação dos recursos disponíveis em nós nomeados como repositórios (também conhecidos como super-nós), em uma busca pelo recursos desejados baseada em suas características (incluindo palavras chaves e tipo), e na recuperação desse recurso de várias entidades que o possuam. Esse modelo de acesso a elementos remotos pode também ser utilizado para a disponibilização de serviços, se eles forem tratados como mais um tipo de recurso.

Esse foi o modelo adotado pelo JXTA (http://www.jxta.org), um projeto livre, iniciado pela Sun Microsystems (http://www.sun.com) e apoiado por outras empresas e instituições. O JXTA enxerga um serviço como um recurso disponível numa rede virtual. O provedor do serviço pode publicar um anúncio na rede descrevendo suas funcionalidades e a sua interface. O cliente, ao localizar o serviço desejado, pode invocá-lo de um ou mais provedores através de uma conexão virtual (chamada *pipe*), que pode trafegar entre nós diferentes na tentativa de um caminho com menos tráfego, e até mesmo, ser deslocada para outro provedor do serviço, caso o anterior se torne indisponível, sem interrupção da comunicação.

O JXTA é boa solução para implementação de serviços distribuídos pois é independente de plataforma e linguagem, é bastante flexível e pode ser até implementado utilizando *Web Services* para a comunicação. Para transpor a barreira imposta por *firewalls*, o JXTA utiliza a tática de utilizar um ponto externo à rede como uma ponte, que é responsável por intermediar a comunicação entre os demais pontos e o ponto atrás da barreira. Assim como os serviços em grade, adiciona à simples publicação e utilização de serviços mecanismos mais elaborados, como a descoberta e localização ponto-a-ponto, a elaboração de rotas virtuais alternativas, etc. Entretanto, apesar de o JXTA ser um projeto aberto, ele ainda não é um padrão reconhecido para aplicações P2P, usado por todas as aplicações desse tipo. Há diversos outros produtos e protocolos sendo desenvolvidos com a mesma finalidade.

2.3 Mineração de dados

A disseminação e popularização do uso de sistemas computacionais em todas as áreas do conhecimento vêm gerando um enorme crescimento na quantidade de dados processados e armazenados em instituições e empresas de todos os tipos. Registros de compras de supermercados e de utilização de planos de saúde, informações sobre concessões de crédito, estoque, compras e vendas, dados coletados em pesquisas, entre outros, são exemplos de bases de dados que, sem nenhum tratamento especial, escondem por vezes conhecimentos relevantes.

2.3.1 O processo de descoberta de conhecimento

A grande quantidade de informação contida nesses arquivos ou banco de dados, ou mesmo outros tipos de armazenamentos (que chamaremos neste texto, genericamente, de bases de dados), possui pouco ou nenhum valor se não servir de fonte para a aquisição de novos conhecimentos. O processo de descoberta de conhecimento em bases de dados (KDD, *Knowledge Discovery in Databases*), definido inicialmente por Gregory Piatetsky-Shapiro [31], serve como uma base para guiar tarefas de exploração da informação e descoberta de conhecimento. KDD envolve diversas etapas [2], como ilustrado na figura 2.4: o pré-processamento (composto de limpeza e integração dos dados, seleção e transformação) para tornar os dados consistentes, completos e em conformidade com as necessidades do algoritmo; a mineração de dados, com a aplicação de um algoritmo concreto para encontrar estruturas e padrões; e a avaliação, apresentação e interpretação de resultados, com diferentes métricas, afim de filtrar padrões mais interessantes ao usuário e provê-lo com o conhecimento desejado. Desses, a mineração dos dados em busca de padrões interessantes é a mais importante, sendo muitas vezes confundida com o conceito geral de KDD.

O pré-processamento dos dados é necessário na maioria das vezes, para permitir que sejam utilizados corretamente na mineração. Para a operação correta do processo, é necessário a eliminação de erros nas bases, complementação de dados essenciais faltosos e a integração em um conjunto de dados comum. Alguns algoritmos exigem também que dados sejam transformados, por exemplo, discretizando valores contínuos (colocando-os em intervalos bem definidos, como 1-10, 10-20, etc.).

O passo de mineração de dados no processo de KDD envolve a aplicação de algoritmos, ferramentas e técnicas apropriadas a conjunto de dados pré-processados para facilitar a exploração de dados e extração de padrões úteis [32].

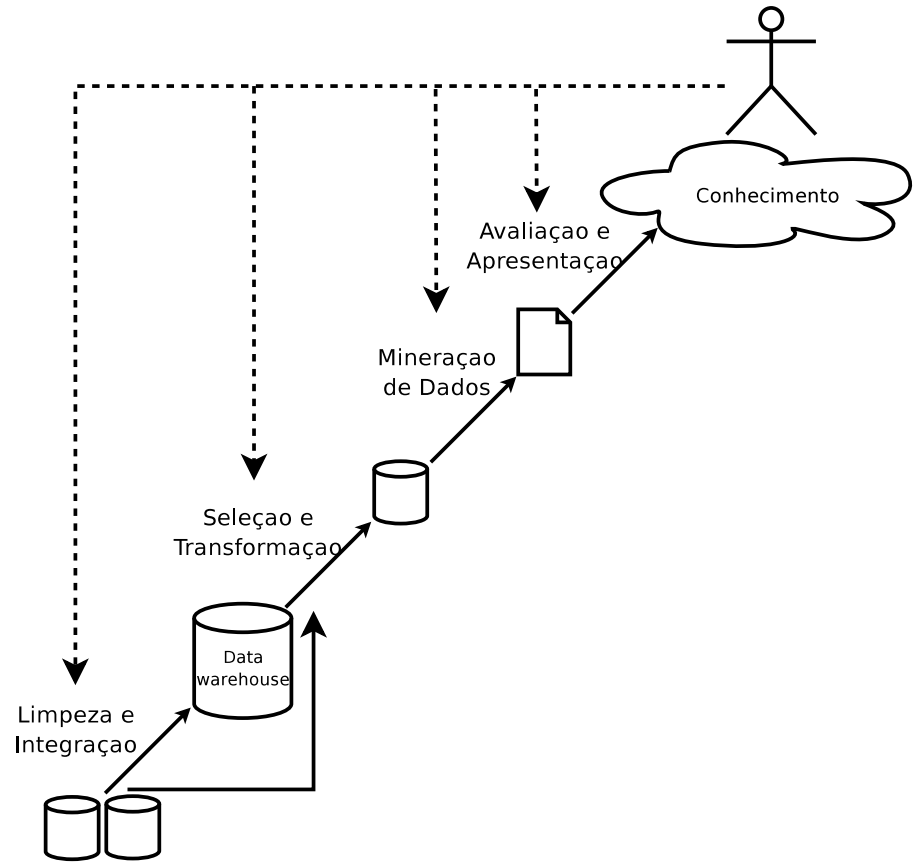

Figura 2.4: Processo de Descoberta de Conhecimento (KDD) [2]

2.3.2 Algoritmos de mineração

Há vários tipos de algoritmos de mineração de dados, como regras de associação (*association rules*), agrupamento (*clustering*) e árvore de decisão, entre outros.

Mineração de Regras de Associação (ARM, *Association Rule Mining*) é um dos algoritmos mais comuns e vem se tornando um dos principais em mineração de dados, segundo Agrawal e outros [33], referenciado por Zaki [34]. O uso de ARM permite extrair regras que relacionam itens que freqüentemente ocorrem juntos em uma transação, ou seja, definir como e quanto a presença de um subconjunto de itens influencia na presença de outro subconjunto. Com o uso de ARM podemos identificar regras como: 98% dos clientes que compram pneus e acessórios automotivos compram também o serviço completo de manutenção [35]; quem compra os produtos A e B, na maioria das vezes, compra também C; etc. O uso de ARM tem muito valor em diversas áreas, como por exemplo: em *marketing*, ajudando a identificar perfis de usuários ou escolher a melhor disponibilização de produtos nas lojas; em pesquisas médicas, identificando padrões em cadeias de DNA; na área policial, identificando características e padrões em comportamentos de criminosos; entre outras áreas.

Outra técnica importante é a de identificação de agrupamentos (*clustering*), quando grandes conjuntos de dados são divididos de forma a separar seus elementos em grupos distintos que apresentem características semelhantes entre si e diferentes daquelas de outros grupos. Com isso, padrões de separação dos dados podem ser identificados e os elementos podem ser analisados de forma simplificada em função dos grupos a que pertencem.

Processar todas essas informações tem um custo computacional muito alto, já que normalmente envolve enormes quantidades de dados. Mesmo os mais eficientes algoritmos seqüenciais podem se tornar ineficazes quando analisam bases muito grandes [36], gastando muito tempo no processamento e muitas vezes não conseguindo chegar ao resultado esperado. Assim, a implementação de algoritmos de mineração de dados em ambientes paralelos de alta performance é crucial para melhorar os tempos de resposta.

2.3.3 Visualização

A avaliação, apresentação e interpretação dos dados constituem um processo interativo, onde o usuário pode visualizar os resultados da mineração, adequando-os às suas necessidades. A ferramenta de visualização pode ser usada como uma sumarização para que o usuário final tenha uma visão geral de um conjunto de dados. Isso possibilita tomar decisões mais acertadas com relação à preparação de dados [32]. A visualização dos dados pode ser feita de diversas formas, de acordo com o tipo de informação disponível e a percepção do usuário [37, 38]. Resultados da busca por regras de associação devem ter uma metáfora visualmente diferente dos resultados de um agrupamento de dados (*clustering*). A criação de metáforas que correspondem ao desejo e entendimento do usuário ainda é um campo aberto e várias propostas podem ser encontradas na literatura [32]. Programas distintos podem representar de forma diferente um mesmo conjunto de dados, e seria interessante permitir que o usuário, com poucas informações, pudesse escolher a que melhor lhe convém.

2.3.4 Organização de sistemas de mineração de dados

Sistemas de mineração de dados evoluíram a partir de sistemas *stand-alone* caracterizados por algoritmos simples com pouco suporte para o processo de descoberta de conhecimento, tornando-se sistemas integrados que incorporam diversos algoritmos de mineração, múltiplos usuários, vários formatos de dados e fontes de dados distribuídos [39]. A relação base de dados, algoritmo de mineração, visualização de resultados necessária para a transformação de dados puros em entendimento pode se dar através de diversas combinações entre essas entidades, num processo interativo. Resultados intermediários gerados por uma técnica de mineração de dados podem ser analisados por um analista humano, que irá, a seguir, decidir como proceder, talvez

usando uma segunda técnica [32]. Isso pode envolver re-seleção iterativa de conjuntos de dados e atributos, e qualquer passo de pré-processamento necessário.

Esses requisitos apontam para o uso de arquiteturas orientadas a serviço, onde cada etapa é executada em um ou mais componentes distribuídos que oferecem seus recursos (bases, algoritmos, implementação de visualização, etc.) aos demais. Como vimos, *Web Services* atendem aos requisitos para esse tipo de sistema, como localização dinâmica de serviços, padronização nos protocolos e na definição de como acessá-los.

2.4 Análise de desempenho

O desempenho apresentado por um serviço Web é fundamental para o seu sucesso. Um serviço com desempenho insatisfatório na visão do cliente tende a não ser mais utilizado. É necessário que o serviço esteja sempre disponível e atenda a requisitos claros de qualidade e desempenho.

A qualidade do serviço (QoS, *Quality of Service*) depende do que o cliente espera do sistema. Como o usuário do sistema não interage diretamente com os *Web Services*, sua percepção se refere à aplicação que faz uso deles. O usuário de um serviço distribuído deseja rapidez, tempo de resposta previsível e disponibilidade todo o tempo [40]. Portanto, para garantir a qualidade é preciso monitorar dados como o tempo de resposta, os erros que podem tornar o sistema indisponível e a utilização dos recursos da máquina.

O tempo percebido pelo usuário para obter uma resposta a uma requisição de um serviço WWW, o tempo de resposta fim-a-fim, é uma das principais métricas para avaliação do desempenho de sistemas Web e pode ser dividido em dois componentes principais [40]: tempo de rede e tempo de processamento no(s) servidor(es) responsável(is) pelo serviço. O tempo de rede inclui todo o tempo gasto na transmissão das mensagens entre o cliente o serviço; é composto de latência, que representa o tempo necessário para o envio da requisição e o retorno de um aviso de recebimento ou da resposta envolvidos na troca de mensagens — chamado *round trip time* (RTT) —, e o tempo de transmissão, gasto na transmissão dos dados, que depende da largura de banda no *link* mais lento no caminho entre cliente e servidor. O tempo no servidor pode ser dividido entre tempo de serviço — tempo em que a requisição está recebendo serviço de algum recurso como uma CPU, disco ou rede — e tempo de espera — a soma de todos os tempos de espera para acesso aos recursos, quando a requisição é enfileirada no sistema.

Outra métrica muito utilizada é o *throughput*, a quantidade de dados que podem ser processados e/ou enviados de um local a outro em um determinado espaço de tempo. Para serviços Web, podemos dizer que o *throughput* é a taxa na qual requisições HTTP são servidas (normalmente expressa em requisições por segundo).

No caso de uma plataforma baseada em serviços, o tempo de resposta percebido

pelo usuário é constituído do tempo de rede até a aplicação principal (se essa estiver localizada remotamente) e o seu processamento, mais o tempo para acesso aos serviços distribuídos. Para os serviços que agregam ou fazem uso de outros serviços, os acessos a esses serviços se somam ao do serviço principal. Portanto, quanto menor o tempo gasto com um serviço, melhor será o tempo de resposta total percebido pelo usuário. Vários fatores podem contribuir para degradar o desempenho de um *Web Service* [40], como: largura de banda insuficiente em períodos de pico, servidores sobrecarregados, cargas inconstantes no servidor, deficiência na comunicação com banco de dados, falhas de serviços de terceiros, etc.

Em um sistema que atende a vários usuários, como é o caso de sistemas Web, para evitar a perda de desempenho é necessário que ele seja escalável, ou seja, consiga se adaptar ao crescimento da demanda dos clientes. Em um sistema escalável é possível conseguir mais poder computacional adicionando-se a ele um maior número de processadores (ou máquinas), mais memória, mais largura de banda ou quantidade de armazenamento, outras réplicas de um serviço, etc.; ou seja, é feita uma mudança de escala. Por exemplo, um sistema distribuído escalável é aquele que pode começar com apenas algumas máquinas (ou uma) mas pode ser expandido para várias máquinas, de modo a atender mais usuários simultâneos. Escalabilidade é muito importante para um sistema porque significa que pode-se investir nele com poucos recursos e garantir que ele poderá crescer conforme a necessidade.

Capítulo 3

Trabalhos Correlatos

Nessa seção revisamos alguns dos trabalhos que vêm contribuindo para a evolução de arquiteturas orientadas a serviços (SOA) e do processo de descoberta de conhecimento distribuído, bem como para a análise de desempenho de *Web Services*. Os benefícios do uso de arquiteturas distribuídas e a idéia de aplicações baseadas em serviços que trabalham de forma cooperativa soam como um caminho evidente para aplicações que demandam grandes quantidades de dados e processamento, como é o caso da mineração de dados. Porém, a avaliação da escalabilidade e do desempenho de *Web Services* é fundamental para tornar o uso de SOA uma solução viável.

3.1 Computação Orientada a Serviços

Brereton e Budgen [41], segundo Turner *et alli* [42], verificaram que o desenvolvimento de novos estilos de arquiteturas de software baseados em formas estruturais como objetos e componentes não seriam suficientes para gerar avanços significativos para o desenvolvimento de software. Os autores previram um futuro em que desenvolvedores teriam uma visão radicalmente diferente de como o software entregaria a sua funcionalidade aos usuários. Bennett *et alli* [43] começaram um estudo sobre a criação de software baseado em serviços, para suportar as novas demandas pela a criação de aplicações que estão sempre em evolução. Os autores apontaram as necessidades e desafios dessa nova visão no desenvolvimento de software.

Chandrashekar *et alli* [44] criticam a arquitetura atual da Internet, mostrando que novos requisitos vêm se tornando essenciais para o funcionamento de aplicações emergentes. Esses requisitos incluem disponibilidade, confiabilidade, qualidade e segurança. Para tornar a Internet um arcabouço viável para a implantação de novos serviços, enfatizam uma Internet totalmente orientada a serviços (SOI, *Service Oriented Internet*), baseada em três principais abstrações: uma noção de nuvem de serviços, um novo esquema de endereçamento em dois níveis, independente de localização e uma nova camada abstrata de serviço, usada para encaminhar os pacotes para o serviço de destino

apropriado.

A necessidade de criação de protocolos de comunicação entre aplicações na Internet, com uso de XML, começou a ser discutida em 2000 pelos membros do W3C, e em 2002 Austin *et alli* [45] apresentaram o primeiro rascunho sobre o assunto, onde definem alguns requisitos básicos para a construção de serviços baseados nos protocolos mais utilizados da WWW, os *Web Services*.

Os desafios e as novas idéias trazidas por esse conjunto de tecnologias foram discutidas por Clay Shirky [6], que procurou enxergar a Internet como um grande sistema operacional (com diversos serviços disponíveis) onde não há separação entre o que é aplicação local e o que é global. Essa é uma visão bastante interessante, que torna mais acessível a percepção das possibilidades que estão sendo criadas com SOC. O autor antecipa também alguns requisitos que serão necessários nesse contexto, como confiança, semântica e coordenação de serviços. A importância de SOA e o uso de *Web Services* para implantação desse novo estilo arquitetônico, bem como os desafios a serem enfrentados, foi amplamente discutido por diversos autores [5, 42, 46, 7, 15], que serviram de guia para esse trabalho.

3.2 Serviços de Mineração de Dados

Um dos primeiros trabalhos sobre disponibilização de mineração de dados como serviços na Internet foi proposta por Sarawagi e Nagaralu [47]. Nele, os autores apresentam alguns desafios interessantes a serem analisados para sistemas desse tipo — que podem ser providos por qualquer um com dados e experiência adequados —, como padrões para descrição de modelos de mineração, segurança e confiabilidade, integração de modelos de origens distintas e personalização usando dados do usuário. O trabalho é focado em *Application Service Providers* (ASPs) — provedores de soluções corporativas em aplicações completas e independentes — para modelos de mineração de dados, apresentando alguns cenários de exemplo onde esses serviços poderiam ser utilizados, porém sem propor soluções.

Krishnaswamy *et alli* [39, 48] compararam diversos provedores de serviços (ASPs) de mineração de dados baseados na Internet, como digiMine [1] e Information Discovery [2], entre outros. Estes requerem que o usuário envie seus dados ao provedor de serviços (com quem normalmente a empresa possui um contrato) e acesse os resultados através de uma interface WWW. Após discutirem situações de uso desses serviços e questões relacionadas a cada um, propuseram um modelo de múltiplos provedores de serviço, definindo os protocolos de interação e um mecanismo de troca de informações

[1] http://www.digimine.com/

[2] O endereço indicado no artigo (http://datamining.aa.psiweb.com/) não estava disponível durante o nosso trabalho.

e descrição de requisição por tarefa baseado em XML. O modelo funciona na forma de uma "federação" de provedores de serviço, usando um componente de coordenação chamado *federation manager*, que gerencia as interações entre o cliente e os provedores, buscando o que melhor satisfaz as necessidades do usuário. O cruzamento das preferências do cliente com as capacidades do provedor de serviços é feito através de avaliações de expressões XPath [49], uma linguagem para endereçamento de partes de um documento XML que pode ser usada para casamento e teste de padrões.

Outro enfoque abordado por Krishnaswamy *et alli* [50], além dos anteriores, é o modelo para mineração de dados baseado em agentes móveis. A idéia é basicamente o agente ir até onde estão os dados e onde deve ser feita a mineração, comunicando-se entre si para a realização da várias tarefas do processo de mineração. Porém, essa abordagem não tem sido muito usada devido ao pequeno número de plataformas de agentes móveis disponíveis.

Os modelos de realização de mineração de dados por ASPs ou agentes são diferentes do adotado pelo Tamanduá. No primeiro caso, a mineração é feita inteiramente por um único provedor de serviço, que o usuário deve escolher conforme suas necessidades e se adaptar a ele. Isso implica, entre outras coisas, em enviar toda a base de dados para o *site* do executor da mineração. No segundo, a aplicação é que vai até os dados, exigindo um ambiente bem controlado e seguro para evitar acessos indevidos desses aplicativos. Já o Tamanduá, como veremos, utiliza diversos serviços distribuídos que trabalham em conjunto para a realização da mineração de dados.

O uso de grades computacionais (*Grid Computing*) também tem atraído a atenção dos trabalhos em mineração distribuída (veja seção 2.2). A primeira tentativa de mineração em grades parece ser de Mahinthakumar *et alli* [51], que implementaram um algoritmo de mineração de dados em cluster sobre o Globus para executá-lo em diversos computadores heterogêneos distribuídos geograficamente, usando uma versão do MPI para Globus (MPICH-G [3]). R. Moore [52] apresenta os conceitos de grades baseadas em conhecimentos (*knowledge-based Grids*).

A primeira definição de uma infraestrutura para mineração de dados baseada em *Grid Services* foi feita por Cannataro *et alli* [53, 54, 55]. O *Knowledge Grid* foi construído sobre um conjunto básico de serviços que dão suporte aos serviços de mineração. Esse conjunto básico é normalmente oferecido pelas plataformas de grid. O *Knowledge grid* possui uma interface gráfica que permite a visualização da execução das tarefas (VEGA, *Visual Environment for Grid Applications*), porém não foi apresentado nenhum uso em aplicações concretas.

O GridMiner, apresentado por Brezany *et alli* [56, 57, 58], é uma arquitetura mais recente para mineração de dados distribuída, baseado no Globus toolkit 3.0 e OGSA-

[3]Mais informações sobre MPICH-G2 em http://www3.niu.edu/mpi/ e http://www-unix.mcs.anl.gov/mpi/mpich

DAI. É um sistema completo e atende a todos os requisitos necessários para a mineração de dados, utilizando serviços distribuídos para a gerência de dados e execução de algoritmos. Porém em nenhum dos artigos encontrados sobre o assunto é apresentado estudo sobre a escalabilidade do sistema, o que é muito importante e será apresentado aqui para o Tamanduá.

A mineração de dados através de ASPs, agentes móveis ou grades não é o foco desse trabalho e representam outras formas de prover esse serviço, diferentes do usado no Tamanduá. Não foi encontrado nenhum trabalho que utilize as tecnologias de *Web Services* para a execução de um processo de KDD.

3.3 Análise de desempenho de *Web Services*

Menascé e Almeida [40, 4] discutem detalhadamente questões relacionadas à qualidade de serviço em todos os tipos de serviços WWW, usando uma abordagem quantitativa. Provêem um arcabouço para entender as complexas relações da WWW e como isso impacta no desempenho e disponibilidade dos serviços. O enfoque analítico adotado pelos autores, que usa cálculos matemáticos para a investigação profunda do desempenho desses serviços, não será adotado nesse trabalho, que possui um enfoque mais experimental.

O desempenho de protocolos para acesso a serviços remotos, no estilo RPC (*Remote Procedure Call*) foi avaliado por Govindaraju *et alli* [59, 60, 61, 62]. Foram levantados os requisitos para um protocolo usado na implementação de aplicações científicas — que contêm muitos números —, encontrando como denominador comum a confiabilidade, robustez, facilidade de leitura e uso, compatibilidade com códigos existentes e interoperabilidade.

O desempenho de SOAP, testado com a biblioteca Axis [63] e gSoap, foi avaliado pelos autores em comparação ao JavaRMI [64] e outras tecnologias proprietárias. Apesar de SOAP ter sido mais lento que os demais, mostraram que ele atende aos critérios estabelecidos. Posteriormente, examinaram os limites do desempenho de SOAP, destrinchando todos os passos necessários para a comunicação através desse protocolo — incluindo o processador XML e transmissão pela rede. Em cada passo, foram sugeridas soluções para melhorar o desempenho, gerando códigos específicos para a aplicação, o que permitiu comparar diversas opções e se mostrou bem mais eficiente que as demais bibliotecas testadas. Notaram que o desempenho baixo se deve, na maior parte, à conversão para texto de números de precisão dupla (*double*) em grandes arranjos (*arrays*), muito comum em aplicações desse tipo. Com isso, os autores mostram que o SOAP e as bibliotecas têm muito o que melhorar para garantir um bom desempenho, e sugerem algumas soluções.

Em um trabalho semelhante, Dan Davis e Manish Parashar [65] também procu-

raram identificar as fontes de ineficiência das implementações de SOAP. Para isso, mediram a latência dos serviços usando algumas bibliotecas — Apache SOAP e Axis Alpha (em Java), SoapRMI (em Java), SOAP::Lite (em Perl), Microsoft SOAP Toolkit (em Visual Basic) — em relação a JavaRMI e CORBA [66]. Testes foram feitos com cliente e servidor na mesma máquina e em máquinas separadas, para avaliar o impacto da transmissão pela rede. Os mesmos serviços foram implementados usando cada biblioteca em sua respectiva linguagem de origem, desprezando tempo para procura no registro ORB (do Corba) e criação de objetos. Foram medidos os tempos de três serviços sem parâmetros, porém um não retorna nada, outro retorna uma cadeia de caracteres (de tamanhos 200, 400, 800) e o último retorna um arranjo de inteiros (também de tamanhos 200, 400, 800). Perceberam que de 86% a 94% do tempo é gasto com o SOAP e processamento XML, e há grande impacto relacionado ao uso dos protocolos de transporte (TCP) e aplicação (HTTP) normalmente utilizados. Os autores propõem ainda a adoção do JavaRMI sobre SOCKS (para atravessar firewalls), até uma melhor maturação do SOAP.

Kohlhoff e Steele [67] analisaram o uso do SOAP para aplicações financeiras, procurando avaliar o seu desempenho em relação a outros padrões comerciais. FIX [4] e CDR [5], um formato binário usado para comparar o custo de usar um formato textual nesse processo. O estudo dos autores mostrou que o SOAP tem desempenho inferior aos outros dois protocolos, com tamanho maior de mensagens e menor *throughput*. Porém, sugerem que para aplicações não científicas, onde o uso de números é menor, a análise de Chiu *et alli* [60], citada anteriormente, de que a transformação de números de ponto flutuante para texto (codificação e decodificação) é o principal fator desse mal desempenho, não é válida, já que o FIX apresenta desempenho semelhante ao CDR. Propõem algumas práticas que podem ser adotadas pelos programadores para um melhor desempenho, como a compressão das mensagens SOAP (para economizar banda) e compactação das *tags* XML (a troca dos nomes das tags por cadeias de 2 a 4 caracteres reduziu o tamanho das mensagens SOAP em 25 a 35%).

Como vimos, muitos trabalhos já foram feitos comparando o desempenho de SOAP com outros protocolos de RPC, binários e textuais. Vários problemas já foram apontados e alterações foram propostas, e nesse trabalho aceitamos esses resultados e não repetiremos esses testes e experimentos. Entretanto, nenhum deles mostra a análise completa de desempenho de uma aplicação real baseada em SOA com *Web Services*, que será apresentada mais adiante.

[4] FIX (Financial Information eXchange) é um padrão textual de troca de mensagens desenvolvido especialmente para transações comerciais em tempo real. http://www.fixprotocol.org/

[5] O CDR (*Common Data Representation*) é um formato usado como base do CORBA [66]

Capítulo 4

Arquitetura Orientada a Serviços

Os métodos para a construção de aplicações orientadas a serviços diferem dos métodos tradicionais, em que desenvolvedores constroem softwares basicamente empregando alguma variação do ciclo editar-compilar-ligar [42]. Nesse novo modelo, o software a ser desenvolvido é descrito em função de um serviço oferecido e não mais das demandas de um usuário. Serviços são configurados para atender um conjunto específico de necessidades num determinado momento, executados e descartados — a visão de um serviço instantâneo [43]. Essa é uma mudança significativa de visão que pode trazer confusão ao desenvolvedor.

Este capítulo aborda o processo de criação de aplicações baseadas em arquiteturas orientadas a serviços (*Service Oriented Arquitectures, SOA*), mostrando como identificar os componentes que deverão prover serviços, quais as suas interfaces e quais as opções de protocolos e tecnologias podem ser empregadas. Por fim, apresentaremos o exemplo do Tamanduá, uma arquitetura baseada em serviços para a realização do processo de KDD, identificando as decisões tomadas em sua construção.

4.1 Organização de uma aplicação baseada em *Web Services*

A construção de uma arquitetura orientada a serviços começa com a definição dos objetivos da aplicação. Somente após os objetivos serem definidos é que podemos inferir quais os serviços deverão ser oferecidos e como esses serão agrupados em conjuntos de serviços relacionados, os quais serão aqui denominados servidores.

Cada servidor da arquitetura oferece serviços que deverão funcionar de forma coordenada, atendendo a uma parte bem definida do problema. Deve-se portanto identificar quais os elementos do problema podem ser organizados de forma a se tornar sub-problemas à parte, que poderiam ser solucionados através de um conjunto restrito de funcionalidades, disponibilizadas ao usuário como serviços. Os serviços de um ser-

vidor normalmente utilizam uma mesma base de dados e podem se comunicar através de mecanismos internos.

A arquitetura do sistema depende de como serão organizadas as interfaces, com que granularidade cada serviço será construído, como os dados serão transferidos entre servidores, como serão tratados os eventos assíncronos (aqueles que, por sua natureza devem ser processados sem interação direta com o cliente) e qual o ambiente será utilizado para a implementação do sistema.

As seções a seguir discutem as opções existentes nesse processo e os fatores a serem considerados nas decisões de projeto.

4.1.1 Concepção e distribuição de serviços

Para identificar quais serviços deverão ser disponibilizados pela arquitetura, é preciso pensar nas suas interfaces e nas funcionalidades que serão oferecidas. É importante buscar uma divisão de serviços que garanta o fraco acoplamento entre os diferentes servidores, pois quanto menos um serviço depender de outros, menor o impacto sobre ele quando alguma alteração tiver que ser realizada na arquitetura. Mesmo no caso de serviços que agregam outros serviços, deve-se tentar reduzir o número de serviços externos com os quais os serviços integrados trabalham. Além disso, em um ambiente SOA, onde os componente são distribuídos, é importante considerar o tráfego na rede e evitar transferências desnecessárias. Por outro lado, é preciso ponderar entre granularidade e desempenho, considerando a banda de passagem e a latência da rede.

Isolar a implementação de um serviço da forma como ele é exposto ao consumidor também permite que mudanças na implementação não interfiram no funcionamento do cliente, fator importante quando se tem muitos consumidores para um mesmo serviço, que podem frequentemente ser administrados por pessoas diferentes. Dessa forma, pode-se criar serviços genéricos o suficiente para atenderem a uma classe de problemas semelhantes que podem ser especificados conforme os parâmetros passados. Esse pode ser o caso, por exemplo, da execução de um algoritmo de mineração: um único método, cujo objetivo é executar uma tarefa de mineração, recebe como parâmetros o identificador do algoritmo que será executado e as opções a serem utilizadas, sendo capaz de acionar diversos algoritmos diferentes através de uma única interface. Uma outra forma de fazer isso seria a definição de um serviço para cada algoritmo. Na primeira opção, é necessário informar quais são os algoritmos disponíveis e quais os parâmetros para cada algoritmo. Já na segunda, como cada algoritmo é descrito por um serviço independente, a própria interface contém essas informações; entretanto, qualquer alteração na escolha e interação dos algoritmos e seus parâmetros de entrada do algoritmo torna incompatíveis os clientes já existentes.

É essencial também que a interface seja bem definida e atenda aos objetivos pro-

postos, retornando exatamente aquilo que o consumidor espera ou pode prever. Para isso, é importante ter bem documentada a responsabilidade de cada serviço, os pré-requisitos e parâmetros necessários para a sua invocação e qual o tipo de dado será retornado. Em caso de dados muito volumosos, deve-se também definir qual o protocolo será utilizado para a sua transferência.

Outro fator que deverá ser considerado é a granularidade do serviço [3]. Serviços de grão grosso oferecem uma frações das regras de negócio, ou seja, englobam várias funcionalidades para oferecer ao cliente uma maior abstração do problema. Eles isolam a lógica de negócio e acessos aos recursos no provedor do serviço. Serviços de grão fino são mais específicos e praticamente representam métodos de acesso a recursos, deixando a cargo do consumidor a implementação da lógica e o controle da interação entre eles. No primeiro caso é enviado um único pedido ao servidor com todas as informações necessárias para a realização total de uma tarefa, enquanto no segundo vários serviços são solicitados para se completar a tarefa. Na figura 4.1a vemos um serviço disponibilizado com granularidade grossa, que recebe vários dados de uma vez e é responsável pelo acesso aos recursos (como o banco de dados) para a execução completa da operação. Na figura 4.1b, temos a mesma tarefa dividida entre vários serviços, que recebem uma parte dos dados cada um, e a união de seus resultados representa a tarefa desejada.

Para exemplificar, imagine um cenário onde é necessário vários parâmetros para a execução de uma certa tarefa. Pode-se criar um serviço que espera todas as informações necessárias de uma só vez e executa a tarefa (granularidade grossa). Por outro lado, pode-se definir um conjunto de serviços com essa mesma finalidade (granularidade fina): um serviço para criar uma tarefa, outro para definir qual algoritmo será utilizado, um terceiro para receber um parâmetro (que pode ser chamado várias vezes) e um quarto para executar a tarefa. A opção pelo uso de serviços de grão grosso economiza tráfego na rede, pois há menos troca de mensagens entre cliente e provedor, porém não permite muita flexibilidade nem a fácil reutilização dos serviços, já que oferece um pacote fechado de funcionalidades ao usuário. Deve-se encontrar um equilíbrio para a granularidade dos serviços a fim de evitar o excesso de comunicação sem perder a característica de reuso.

Em SOA, os serviços são executados através da rede, seja uma LAN ou a Internet, e é necessário que não sejam demasiadamente demorados. Em um protocolo de invocação de métodos remotos, a chamada a um método deve retornar o mais rápido possível, para evitar a falsa impressão de que ele não conseguiu processar a requisição e está travado, causando inclusive o encerramento da conexão pelo protocolo de transporte. Caso seja necessário, a execução de um processamento demorado pode ser feita com o retorno de uma indicação de que a requisição foi recebida e será processada sem que a resposta seja enviada imediatamente, ou seja, uma execução assíncrona. O fim da

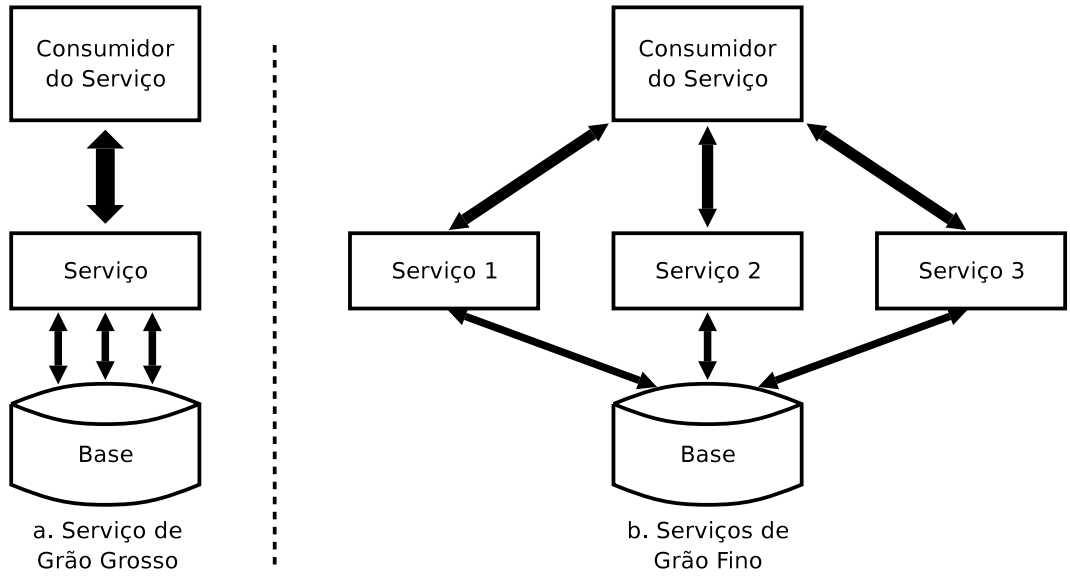

Figura 4.1: Granularidade de Serviços [3]

tarefa pode ser verificado com o envio de uma mensagem de término da execução por parte do provedor — o serviço é responsável por avisar que encerrou — ou através de consultas posteriores do cliente para verificar o andamento da execução da tarefa, ate seu término. Nesse caso, o servidor deve, ao aceitar a requisição, fornecer um identificador de tarefa que será utilizado pelo cliente para consultar o estado corrente da mesma.

No primeiro caso, o cliente não tem que se preocupar em consultar o estado periodicamente, o que gera menos tráfego, porém corre-se o risco de o cliente estar protegido por atrás de uma barreira — como um *firewall* ou mascaramento de endereço — e não receber a mensagem de aviso, ficando indefinidamente na espera do resultado. No segundo caso, o cliente consulta o provedor do serviço sobre o estado da tarefa em períodos pré-definidos e obtém o resultado após o seu término. Isso exige uma gerência de execuções e mais serviços, para a consulta e retirada do resultado, porém permite maior controle por parte do cliente.

4.1.2 Ambiente de Implementação

Como mencionando anteriormente, o paradigma de SOA prevê a independência de linguagem e plataforma. Com *Web Services* isso é feito utilizando protocolos baseados em XML, sem a dependência de uma tecnologia ou *framework* específicos de um fabricante. Dessa forma, é possível criar *Web Services* nas mais diversas plataformas, como J2EE, .Net, PHP e várias outras.

Cada uma dessas plataformas oferece aos desenvolvedores arcabouços e ferramen-

tas diferentes para a construção de *Web Services* e outras tecnologias para implementações de serviços (normalmente proprietárias). Na maioria dos casos, as interfaces dos serviços são mapeados para objetos construídos e diversas facilidades são oferecidas para que o desenvolvedor possa se preocupar apenas com a lógica do serviço.

Em Java/J2EE, a maneira mais simples de se criar *Web Services* é utilizar a biblioteca Axis [63] sobre um servidor de aplicação J2EE, como o Jakarta Tomcat [1].

O Axis é basicamente um *Servlet*, um módulo executado pelo servidor que responde às requisições por *Web Services*, processando as mensagens SOAP e repassando as chamadas às classes responsáveis pelos serviços (como especificado no documento WSDL). Algumas ferramentas acompanham o Axis para facilitar o trabalho do programador, como um gerador de descrições WSDL a partir de uma interface Java e vice-versa. Com essas ferramentas é possível criar classes clientes que funcionam como encapsulamento para o mecanismo de chamada remota (classe *stub*) e classes que auxiliam o servidor, mascarando a ligação entre a requisição e classe alvo (classes esqueleto). Há também um mecanismo para a geração de serviços diretamente a partir do código fonte de uma classe Java, em tempo de execução. Essa opção, entretanto, não é recomendada, pois a geração pode ser lenta e não produzir exatamente o resultado esperado. O ideal, nesse modelo, é a criação do documento WSDL e, a partir dele, a geração das classes cliente e servidor.

Para grandes aplicações corporativas, os melhores servidores J2EE oferecem ainda mais facilidades para a criação de *Web Services*. É possível criar componentes EJB (*Enterprise JavaBeans* [2]) que são acessados através de SOAP, apenas com algumas configurações adicionais ou anotações no código fonte, indicando quais métodos serão exportados.

4.1.3 Transferência de Dados

Muitas aplicações trabalham com grandes volumes de dados, como no caso da maior parte das tarefas de mineração. Em um sistema monolítico, todos os componentes podem fazer acesso a um mesmo arquivo que, por exemplo, pode ser gerado por uma tarefa e servir como entrada para outra, diferente. Por outro lado, em um ambiente SOA, composto de serviços distribuídos entre máquinas e redes diferentes, é improvável que todos eles tenham acesso um mesmo arquivo. Os dados, gerados ou armazenados em algum componente, podem ter que ser movidos para outra máquina para serem processados. É fundamental, portanto, encontrar meios eficientes para transferir esses dados. Há vários protocolos de transferência de arquivos em uso na Internet, porém, em um ambiente baseado em *Web Services*, é essencial que essa transferência seja

[1] Jakarta Tomcat é implementado em Java e atende às especificações para servidores Web em J2EE – incluindo *Servlets* e JSPs – e pode ser encontrado em http://jakarta.apache.org/tomcat.

[2] A especificação de EJB pode ser encontrada em http://jcp.org/en/jsr/detail?id=220.

feita através dos servidores WWW, seguindo padrões bem definidos, de forma segura e eficiente, a fim de que não se perca a característica fundamental de SOA: a possibilidade de uso entre organizações distintas — incluindo a garantia de passagem do tráfego através de *firewalls*.

As bases de dados para mineração, por exemplo, são conjuntos de dados separados em n registros, cada qual composto de m campos, representando uma tabela com $n * m$ valores, extremamente densa — possivelmente, todos os campos contêm dados. A transferência dessa base completa entre os componentes do sistema é necessária quando se quer minerar dados localizados remotamente. A mesma coisa acontece quando os resultados gerados a partir de uma mineração precisam ser armazenados remotamente. Nesses casos, questões como a forma de se incluir os dados no protocolo de comunicação, o formato de representação dos dados durante a transferência e as garantias de confiabilidade da comunicação precisam ser tratadas.

4.1.3.1 Opções para a transferência de arquivos

Como discutido anteriormente, a comunicação entre componentes em um ambiente SOA é usualmente realizada através do protocolo SOAP, baseado na linguagem XML, uma representação textual. Esse protocolo não foi desenvolvido originalmente para transferências maciças de dados. Torna-se então necessário definir uma forma de encapsular arquivos de dados no fluxo de comunicação entre os componentes.

Simplesmente inserir os dados no fluxo SOAP se torna um problema para garantir o processamento da comunicação. Se os dados forem codificados em XML, teremos um documento XML dentro de outro, com um esquema de codificação diferente, o que exige alterações no arquivo encapsulado e coloca uma demanda a mais sobre o processador XML usado pelo receptor, usualmente otimizados para campos de pequenas dimensões. Se os dados estiverem em binário é preciso recorrer a algum formato de codificação, pois a inserção de arquivos binários em um fluxo de caracteres também apresenta problemas de processamento para o receptor.

Se a inserção do arquivo diretamente no fluxo SOAP é um problema, uma opção, já que se trata de um ambiente baseado em *Web Services*, é através de um subconjunto do protocolo HTTP, já utilizado. Ao invés de codificar os dados de um arquivo no fluxo SOAP, nesse caso a comunicação é feita passando-se uma URL que identifica o arquivo com os dados que se deseja transferir. O componente que recebe a URL utiliza uma requisição HTTP tradicional para transferir o arquivo, usando os recursos usuais de HTTP para codificar os dados, se necessário.

Essa técnica possibilita que um arquivo muito grande seja retirado em partes, melhorando a velocidade de transmissão e possibilitando a recuperação de dados já transmitidos caso haja falhas na comunicação, além de permitir que o arquivo seja armazenado em *caches*, evitando retransmissões desnecessárias. Quando o endereço informado

aponta para o mesmo servidor — o que é o caso mais comum —, pode-se utilizar ainda a vantagem da reutilização da conexão HTTP aberta na requisição SOAP, melhorando o desempenho. Porém, é preciso garantir que o recurso estará realmente disponível no endereço indicado, pelo menos até um limite de tempo pré-definido. Isso pode tornar necessário técnicas complexas para a gerência desses arquivos.

Outra solução existente é a de inserção de anexos em partes bem distintas da requisição no fluxo SOAP. Essa solução, denominada *SOAP with Attachments* (SwA) apresenta-se como uma solução viável para o envio de arquivos através de *Web Services*. SwA é uma extensão do protocolo SOAP que utiliza o recurso de divisão de requisições em múltiplas partes, através de padrões como MIME (*Multipurpose Internet Message Encapsulation* [68]) ou DIME (*Direct Internet Message Encapsulation*, discutida brevemente no capítulo 2), em conjunto com um protocolo de transporte [24], como o HTTP. O envelope SOAP é gerado com o valor do dado anexado sendo representado como uma referência à parte MIME que contém o anexo, de forma similar aos arquivos anexos em e-mails. O arquivo com os dados a serem anexados é enviado, então, como uma outra parte MIME, no formato original ou codificado, permitindo que qualquer arquivo seja transmitido junto à mensagem sem problemas. Porém, não é simples guardar esse arquivo em *cache*, como no caso anterior, já que as respostas SOAP, que são geradas dinamicamente, também fazem parte da resposta HTTP, podendo ser diferentes em momentos diferentes. Quando as respostas são dinâmicas, não são armazenadas e reutilizadas pelo *proxy*.

4.1.3.2 Formato dos arquivos de dados

A forma de representação das bases e demais conjuntos de dados para a transferência entre os componentes do sistema influencia diretamente o tempo de transferência e a forma como os dados são interpretados entre transmissor e receptor. Formatos binários, apesar de serem frequentemente compactos, trazem problemas na sua inserção em fluxos SOAP e HTTP, como mencionado anteriormente.

Uma solução simples é a adotada em codificações com MIME sobre HTTP, por exemplo, que faz a conversão do arquivo enviado em um formato textual, como base64 [69]. A formato de codificação base64 converte um conjunto de bytes de qualquer tamanho em uma cadeia de caracteres. Basicamente, conjuntos de três octetos (24 bits) são transformados em quatro caracteres, cada um com apenas seis bits utilizados para representar a informação. Os dois bits restantes em cada caracter são normalmente zerados para garantir que os caracteres se refiram a letras, números e pontuação usuais. Com isso, dados codificados com base64 ocupam aproximadamente 33% mais espaço que os dados originais. Ao enviar um arquivo dessa forma, não só é necessário um acréscimo de tamanho, como também um maior processamento e consumo de memória.

O envio de dados binários apresenta outros problemas, mesmo com uma codificação

como base64. O principal deles é certamente o de interpretação dos dados quando as arquiteturas de origem e destino diferem em suas representações internas. Para resolver esse problema, considerando-se que HTTP e SOAP já são protocolos textuais, a solução é adotar formatos de codificação que transformam os dados em seqüências de caracteres.

Há dois formatos textuais de representação de dados bastante conhecidos, que foram avaliados para a transferência de dados entre os componentes do Tamanduá. O primeiro é o formato simples conhecido usualmente como CSV (*Comma Separated Values*), onde os registros são separados por caracteres delimitadores de linha e os campos de cada registro são separados por um caractere pré-selecionado — normalmente a vírgula. Esse formato é simples e permite a representação correta dos dados, desde que se garanta que o caractere pré-selecionado para separação de campos não exista na representação dos valores.

O segundo formato é baseado em XML, linguagem de formatação adotada largamente para troca de dados entre aplicações, inclusive nos padrões de *Web Services*. Como mencionado anteriormente, a linguagem XML é baseada em marcações de texto (ou *tags*), usadas para identificar e estruturar os dados (meta-informações sobre o conteúdo). Organizar os dados em uma estrutura bem definida pode ser vantajoso para garantir a correta utilização do arquivo, porém pode trazer grande acréscimo ao seu tamanho e torna necessário um mecanismo complexo de verificação dessa estrutura.

4.1.3.3 Transferência segura

Em um ambiente distribuído a segurança dos dados trafegados pela rede é um fator importante. No caso da mineração de dados, por exemplo, uma base pode conter informações importantes e sigilosas que não devam ser expostas a intrusos.

Para uma conexão segura, a criptografia dos dados trafegados é essencial. Isso pode ser feito para conexões TCP/IP utilizando-se os recursos de codificação inicialmente desenvolvidas pela Netscape, conhecidos como SSL (*Secure Socket Layer*). Os padrões atuais são denominados TLS [70, 71] (*Transport Layer Security*) e mantidos por organizações independentes na Internet, como o IETF (*The Internet Engineering Task Force* [3]).

Na WWW, inclusive com *Web Services*, é possível utilizar o protocolo HTTP sobre TLS [72], o HTTPS, que garante a confidencialidade dos dados trafegados na WWW através de criptografia. Porém, é necessário que se avalie o tempo devido ao processamento adicional para que não haja uma penalização excessiva sobre o desempenho da arquitetura.

Alguns trabalhos avaliaram o desempenho de HTTPS em relação ao HTTP. Schmitt *et alli* [73] perceberam que o HTTPS não perde muito em desempenho em relação

[3]IETF: http://www.ietf.org

ao HTTP. Apostolopoulos *et alli* [74] identificaram a sobrecarga adicionada por cada componente utilizando *benchmarks* conhecidos e propuseram duas técnicas para melhorar o desempenho do protocolo. Xubin He [75] mostrou que o HTTPS gasta muito mais recursos no cliente que no servidor. Isso faz com que seja necessário um número maior de clientes para sobrecarregar o servidor. Com isso, esses trabalhos mostram que o TLS adiciona uma carga ao protocolo HTTP, porém seu uso é viável desde que utilizado com moderação, ou seja, apenas onde realmente seja essencial. Khare e Lawrence [76] propuseram inclusive uma solução para intercalar requisições seguras e não seguras numa mesma conexão HTTP, evitando a abertura e fechamento de conexões TCP desnecessárias.

4.2 Plataforma Tamanduá

O processo de descoberta de conhecimento em bases de dados (KDD), como já visto, é um processo interativo e incremental, que pode exigir processamento pesado por longos períodos de tempo. Um sistema robusto para mineração de dados, capaz de atender à demanda de vários usuários simultaneamente, deve possibilitar a utilização de diversas bases de dados e diferentes algoritmos e apresentar os resultados obtidos de maneiras distintas, com o objetivo de proporcionar um bom entendimento pelo usuário. O sistema será ainda melhor se for capaz de localizar esses recursos num ambiente heterogêneo como a Internet, a fim de proporcionar ao usuário uma melhor experiência durante o processo.

O Tamanduá é uma plataforma interativa, multi-usuária, escalável e incremental para a mineração de dados. Sendo assim, ele permite a utilização simultânea por vários usuários, cada um tratando dos passos necessários para o processo incremental de descoberta de conhecimentos a partir de bases de dados. O acesso a bases, o processamento da mineração e a visualização dos dados são concebidos na forma de serviços disponibilizados aos clientes, que podem empregá-los dinamicamente no processo de KDD.

4.2.1 Arquitetura do sistema

A plataforma do Tamanduá foi projetada com componentes distribuídos que provêem serviços com interfaces bem definidas, podendo ser utilizados da forma que melhor atender às necessidades da aplicação. Para identificar quais seriam os serviços oferecidos e como eles seriam agrupados em servidores distintos, foram identificados três pontos importantes do problema de KDD que deveriam ser tratados pela arquitetura: a disponibilização das bases de dados, a execução da mineração propriamente dita e a visualização dos resultados. Cada um desses pontos constitui um sub-problema à

parte e envolve um conjunto de funcionalidades que devem ser expostas ao usuário como serviços.

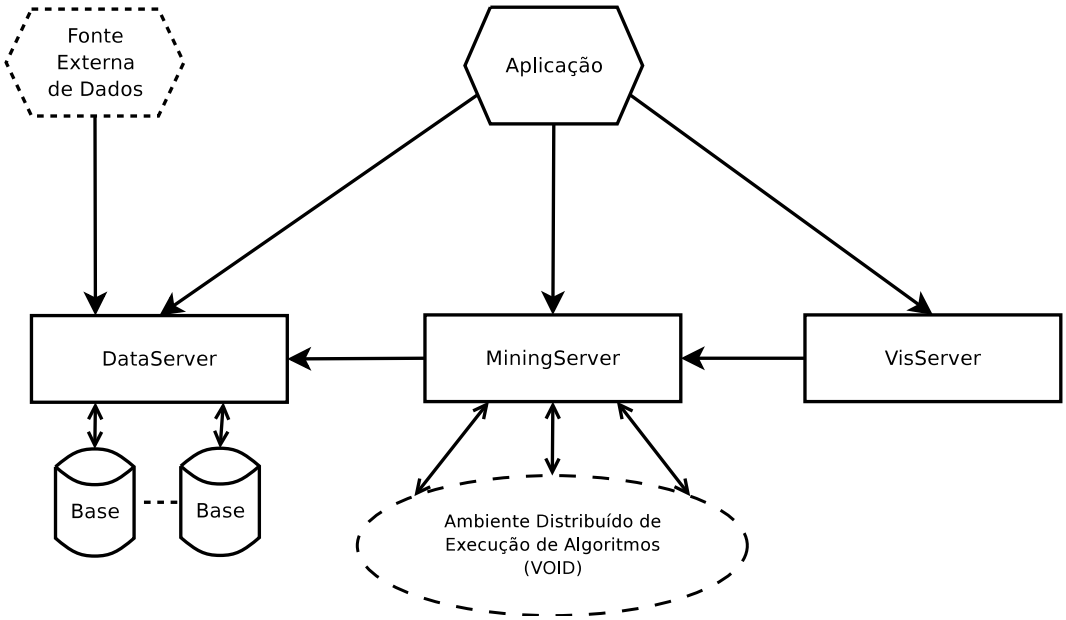

Figura 4.2: Arquitetura do Tamanduá

Dessa forma, o sistema foi construído com três componentes principais (figura 4.2): o servidor de dados, DataServer, responsável pela disponibilização das bases; o servidor de mineração, MiningServer, onde são processadas as tarefas de mineração e o servidor de visualização, VisServer, para a transformação de dados e modelos em metáforas visuais. Para completar a arquitetura, temos uma aplicação para controlar a interação com o usuário e a gerência do processo, que pode ser implementada em qualquer linguagem, desde que conheça as interfaces dos serviços e consiga se comunicar através de HTTP e SOAP. Os componentes são discutidos a seguir.

DataServer: responsável pela disponibilização das bases de dados e seus meta-dados. Bases são conjuntos de tabelas interrelacionadas, cada qual com seu conjunto de colunas. A base é disponibilizada no formato CSV, para economia de espaço (veja seção 4.1.3.2). São disponibilizados serviços para obtenção das bases existentes no servidor e seus meta-dados — como nome, tipo e descrição de cada coluna —, transferência de bases e consultas no estilo SQL.

MiningServer: componente para a mineração de dados propriamente dita, responsável pela execução das tarefas. Uma tarefa de mineração é composta das seguintes informações: identificação da base e colunas que interessam ao usuário, o algoritmo a ser utilizado na mineração e os parâmetros necessários ao algoritmo. As tarefas são executas de forma assíncrona, ou seja, o cliente não espera pelo processamento. Ao invés disso, recebe um identificador, com o qual pode consultar posteriormente o estado da tarefa. Como resultado de uma tarefa de mineração é gerado um modelo, também

disponibilizado em formato textual. O MiningServer foi projetado para servir apenas como uma camada de serviços acima da aplicação real de mineração de dados. Ou seja, sua principal função é a de prover acesso via *Web Services* a essa aplicação, que no caso do Tamanduá é implementada em um ambiente de processamento distribuído para garantir um bom desempenho originalmente sem uma interface SOAP.

VisServer: transforma dados em metáforas visuais apresentadas aos usuários. Tem como entrada modelos gerados numa tarefa de mineração e é responsável por gerar uma visualização, que pode assumir diversos formatos, específicos para cada tipo de algoritmo utilizado.

Aplicação: software capaz de comunicar e gerenciar os componentes anteriores, fazendo a interface entre o usuário final e os serviços. Atualmente, existe uma versão da interface com o usuário no Tamanduá, implementada para o acesso pela WWW. Qualquer aplicação pode ser integrada à plataforma e comandar o processo de KDD da forma mais conveniente à organização, seja pela WWW ou outro tipo de interface.

Na tabela 4.1 são mostrados os serviços implementados em cada componente. Esses componentes comunicam-se com a aplicação e entre si com uso de SOAP e do protocolo HTTP. Dessa forma, qualquer aplicação cliente pode ser utilizada sobre eles para gerenciar a interação do usuário com o processo de KDD, desde que conheça os padrões de *Web Services* adotados. Uma exceção é a adição de novas bases no servidor de dados, que é feita por uma fonte externa, ainda sem um mecanismo bem definido.

O uso de *Web Services* possibilita também que a implantação de um serviço não torne obrigatória a implantação dos demais. Cada um dos serviços pode ser disponibilizado à parte pelas organizações que possuírem recursos interessantes aos demais clientes e serviços da plataforma. Por exemplo, uma base pode ser adicionada ao ambiente pelo seu detentor apenas acrescentando um servidor que ofereça uma interface para ela. A partir de então, o usuário que desejar utilizar a base, solicita à aplicação que acesse aquele servidor para buscar informações e utiliza o servidor de mineração disponível para processá-la — como veremos, quando uma tarefa de mineração é criada, informa-se ao MiningServer a URL que será utilizada para acesso ao servidor de dados. O mesmo vale para algoritmos de mineração e sistemas de visualização.

4.2.2 Interação entre aplicação e serviços

Há três casos de uso do Tamanduá que mostram bem a interação entre os componentes da plataforma: prospecção de bases, criação e execução de tarefas e a visualização de resultados. Entender esses casos de uso, descritos a seguir, é importante para a compreensão da interação entre a aplicação e cada servidor da plataforma.

Prospecção de Bases: o usuário, ao entrar no sistema pela primeira vez, pode querer conhecer quais as bases estão disponíveis, de que tipo são e quais os atributos

Servidor	Serviço	Descrição
DataServer	`Database[] getDatabases()`	Busca as informações sobre as bases de dados disponíveis.
	`Metadata getMetadata(dbident)`	Busca meta-informações sobre uma base.
	`String loadData(dbident, tablename)`	Retorna a URL para acesso a uma base/tabela.
	`String[] getColumnValue(dbident, table, column)`	Retorna alguns valores de exemplo para uma coluna.
	`ResultSet queryData(dbident, table, offset, length, queryParameters[])`	Pesquisa valores em uma base, de acordo com parâmetros.
MiningServer	`Algorithm[] getSupportedAlgorithms()`	Busca meta-informações sobre os algoritmos disponíveis no servidor.
	`Parameter[] getParameters(algorithm)`	Busca meta-informações sobre os parâmetros para o algoritmo.
	`long executeTask(algorithm, arguments[], dataSource, columns[])`	Executa uma tarefa, indicando o algoritmo e seus argumentos, origem e formato dos dados.
	`int getStatus(taskId)`	Recupera o estado de uma tarefa.
	`String loadModel(taskId)`	Retorna uma URL para acesso a um modelo da tarefa.
	`void removeTask(taskId)`	Descarta uma tarefa, liberando recursos internos.
VisServer	`int setup(miningServer, taskId, taskType)`	Transfere os resultados de um servidor de mineração e armazena-os em um formato para consultas.
	`int getStatus(taskId)`	Retorna o estado do processo de configuração de uma tarefa.
	`Metadata getMetadata(taskId)`	Retorna a descrição dos dados presentes no resultado.
	`byte[] overview(taskId)`	Retorna uma imagem que representa o resultado da mineração.
	`byte[] filter(taskId, filterParams)`	Retorna uma imagem que restringe a informação segundo o filtro definido pelo usuário.
	`Details[] detail(point)`	Retorna as informações detalhadas a respeito de um elemento de visualização.
	`void removeTask(taskId)`	Descarta uma tarefa, liberando recursos internos.

Tabela 4.1: Serviços SOAP oferecidos na arquitetura Tamanduá

possui. Para isso, a Aplicação, sob ordem do usuário, solicita ao DataServer sua lista de bases, com nome e descrição. O usuário tem, então, a opção de solicitar o detalhamento de uma base, onde serão recuperadas as informações sobre as colunas, com descrição e tipo. O diagrama de seqüência para esse caso está na figura 4.3.

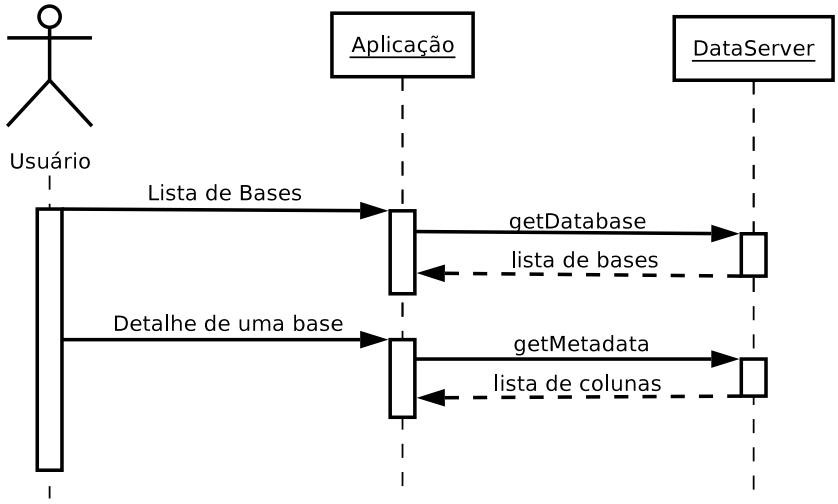

Figura 4.3: Diagrama de Seqüência para a prospecção de bases

Criação: Para criar uma tarefa de mineração de dados, o usuário precisa definir qual a base será processada e qual o algoritmo será utilizado para obter a informação que ele deseja. Como vimos, há vários tipos de algoritmos para mineração de dados e cada um trata os dados de forma diferente, de acordo com parâmetros que são informados. Além disso, nem todos os algoritmos são capazes de trabalhar com todos os tipos de valores, e o número de colunas investigadas, assim como os parâmetros informados, influenciam no tempo de processamento e no resultado encontrado. O usuário deve, então, escolher apenas as colunas que lhe interessam para serem analisadas e informar os parâmetros necessários de acordo com a sua intenção.

Quando o usuário manda executar a tarefa, esta é submetida ao MiningServer responsável pelo algoritmo escolhido com todas as informações definidas. O MS coloca a tarefa em uma fila de espera e retorna à Aplicação o seu identificador, usado para referências futuras. O diagrama de seqüência associado é apresentado na figura 4.4.

Execução: O processo de execução de tarefas na fila do MiningServer é assíncrono, ou seja, funciona sem a intervenção da Aplicação nem do usuário. A fila de tarefas é atendida pelo MS em ordem de chegada, e mais de uma tarefa pode ser executada paralelamente, caso haja mais de uma fila implantada. Caso ainda não tenha disponível localmente a base que será processada, o MS requisita ao DataServer responsável a URL onde esta pode ser obtida. A transferência para o repositório local é feita por uma requisição HTTP normal para recuperação de recursos. Com a base local, o algoritmo de mineração de dados é executada sobre esses dados segundo os parâmetros

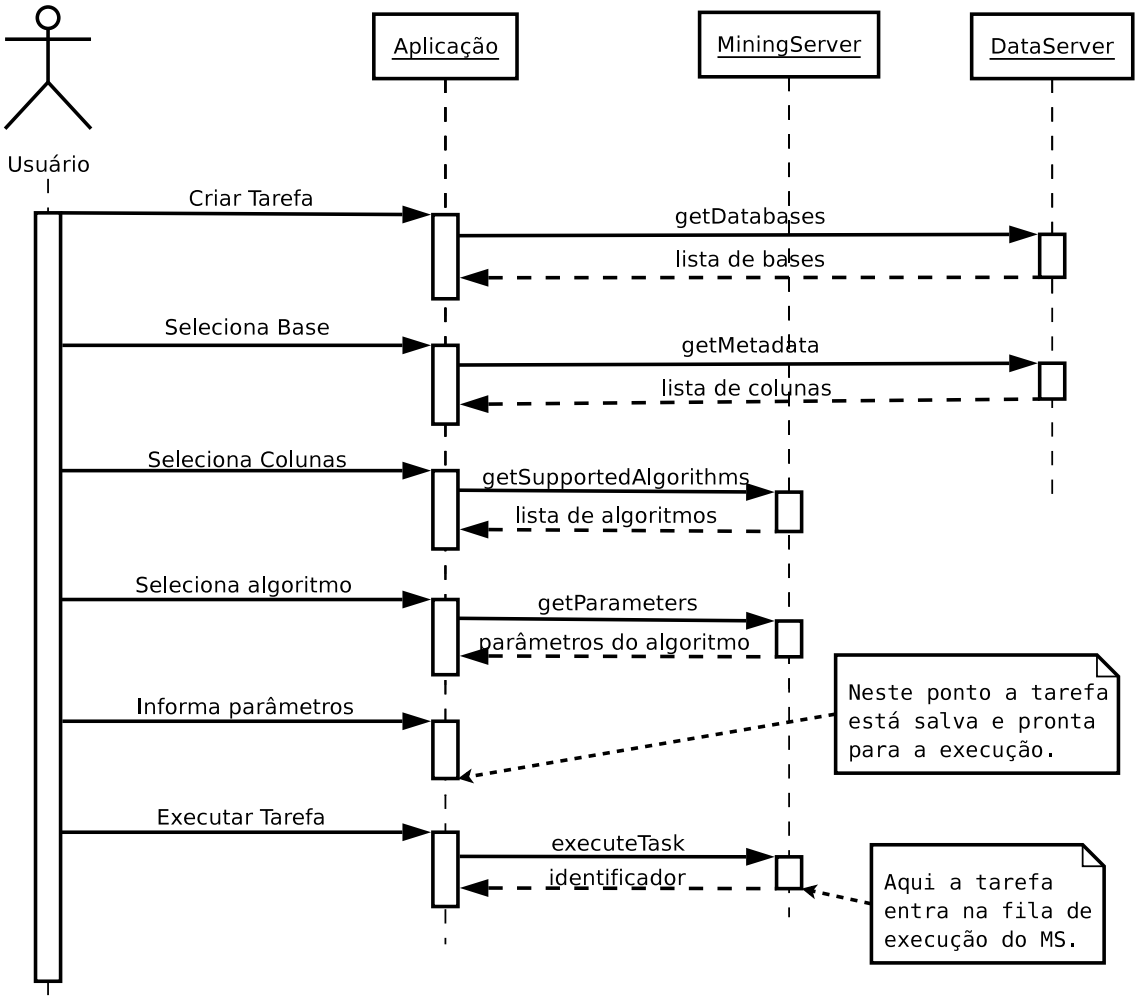

Figura 4.4: Diagrama de seqüência para a criação de tarefa

fornecidos na definição da tarefa.

Periodicamente, a Aplicação deve consultar o estado da tarefa com o identificador recebido na solicitação de execução; o servidor de mineração retorna informando se a tarefa está executando ou se ocorreu algum erro no processo. Quando a execução da tarefa finaliza, o MiningServer recupera do algoritmo o resultado e altera o estado da tarefa para "encerrada".

Ao perceber que a tarefa foi executada com sucesso, a Aplicação pode solicitar ao VisServer a preparação da visualização do resultado. Para isso, o VS requisita ao MS a transferência do arquivo com o resultado da tarefa, processa esses dados de forma a tornar a visualização mais eficiente e armazena em uma base local. A partir desse momento, o VS estará pronto para gerar a visualização para o usuário (figura 4.5).

Visualização dos Resultados: após a execução de uma tarefa, o usuário normalmente precisa entender os resultados encontrados. Ao ser informado que a tarefa está pronta para ser visualizada, o usuário pode pedir para ver o resultado, fazendo com

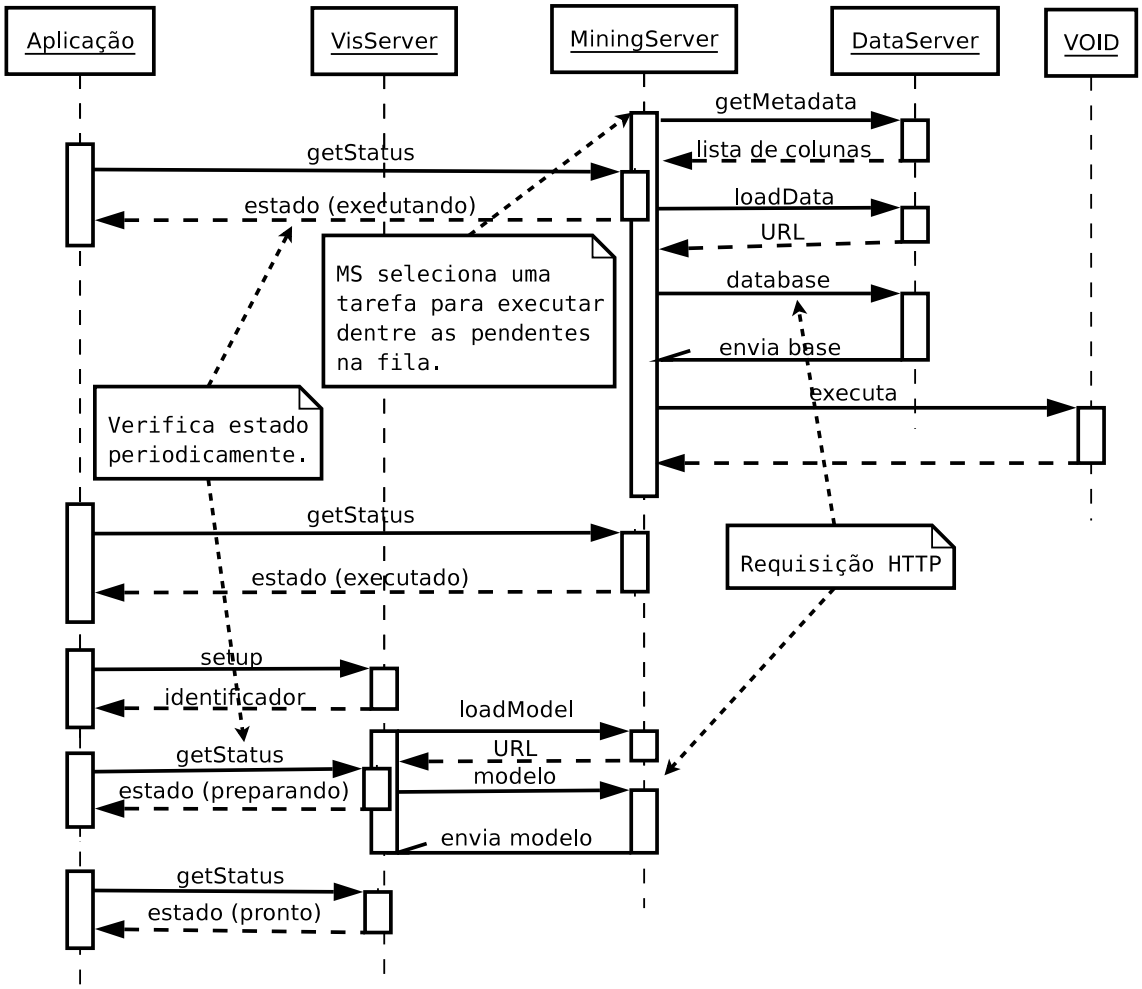

Figura 4.5: Diagrama de seqüência para a execução da tarefa

que a Aplicação recupere do VisServer os meta-dados referentes ao resultado. No caso de regras de associação, os meta-dados contém uma lista de atributos que são causa ou conseqüência de uma regra.

Para facilitar o entendimento do resultado, o servidor de visualização gera uma imagem contendo os pontos que representam as regras encontradas na mineração. Essa imagem, como discutido mais à frente, é transferida como resposta à requisição do serviço. Com isso, a Aplicação apresenta ao usuário uma tela onde ele pode selecionar um ponto que lhe pareça interessante e aplicar um filtro de regras e outras configurações [38]. Selecionado o ponto, a Aplicação exibe um resumo das regras naquela coordenada (normalmente confiança e suporte) e seus detalhes, obtidos do VS. Opcionalmente, o usuário pode também visualizar as tuplas que geraram uma determinada regra, a fim de entender melhor os resultados. O diagrama de seqüência é apresentado na figura 4.6.

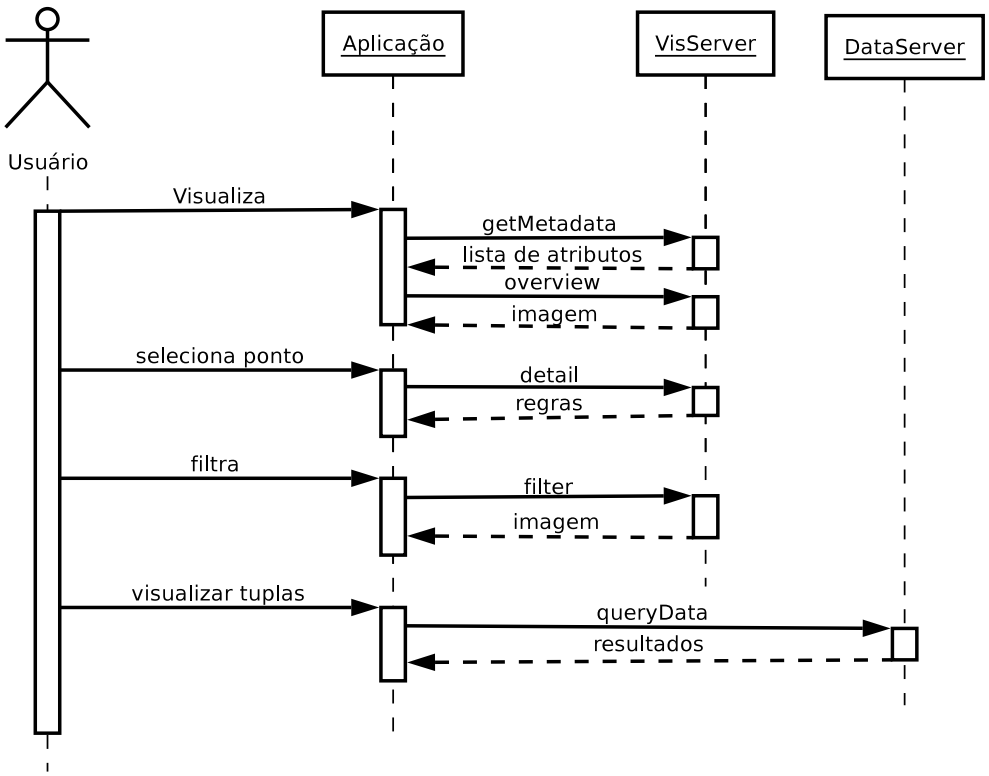

Figura 4.6: Diagrama de seqüência para a visualização de tarefa

4.2.3 Implementação

O Tamanduá foi implementado com um subconjunto da plataforma J2EE, utilizando o servidor de aplicação Jakarta Tomcat versão 5. A biblioteca Axis [63] 1.2 para Java foi utilizada como ferramenta para a implementação de *Web Services* no estilo RPC, para comunicação entre a aplicação e os componentes. Os serviços são mapeados em métodos de classes Java comuns, que chamaremos de classes alvo; as classes alvo são quem realmente implementa as operações dos serviços.

O Axis implementa um *Servlet* que é responsável pelo tratamento das requisições enviadas a uma URL (*Universal Resource Locators*) no servidor HTTP. Quando recebe um envelope SOAP, esse *Servlet* identifica, a partir de configurações locais, a classe que deve tratar aquela mensagem; essa classe auxiliar (comumente chamada de classe esqueleto) é responsável por extrair da mensagem as informações necessárias para a execução do método na classe alvo, acessá-la, e criar a mensagem de retorno à partir do resultado da execução.

Para acesso aos serviços remotos, o AxisClient (componente da biblioteca Axis) é utilizado por uma classe auxiliar (chamada de classe *stub*) para montar o envelope e enviá-lo ao destino, assim como processar a resposta e retornar o resultado. O uso da classe *stub*, que possui a mesma interface da classe alvo, possibilita que se acesse o serviço remoto de forma transparente, como se este fosse uma classe local.

O DataServer, MiningServer e VisServer são implementados como aplicações separadas — uma só acessa a outra através de *Web Services* — que não possuem interface direta com o usuário, apenas através de SOAP. A Aplicação do Tamanduá utiliza uma separação MVC (Modelo-Visão-Controle), com uma interface Web implementado com JSPs (*Java Server Pages*). Classes Java de modelo são responsáveis por acessar os serviços remotos e gerenciar os recursos disponíveis, fazendo a intermediação entre os usuários e os serviços.

Como já foi mencionado, o MiningServer é responsável apenas pelo gerenciamento da tarefa dentro do processo de KDD, servindo como uma camada de serviço para uma aplicação de mineração de dados. Portanto, o processamento dos algoritmos é realizado por um aplicativo à parte, que poderia ser qualquer uma já existente, normalmente utilizando processamento distribuído para melhorar o desempenho. Nesse caso, os algoritmos de mineração foram implementados na plataforma VOID [36, 77], um ambiente paralelo e distribuído de execução de aplicações em *cluster*.

A comunicação entre o MiningServer e o VOID é feito através da execução do programa na máquina local, com a passagem de parâmetros por linha de comando. Isso foi feito porque não há outros protocolos de comunicação implementados nos programas do VOID para esse fim, já que foge do escopo da aplicação: minerar, utilizando um *cluster* de máquinas, dados disponibilizados em um arquivo. Além disso, a implementação de *Web Services* diretamente nesse programa poderia impor uma perda de desempenho ao sistema. Portanto, o MiningServer gera, a partir dos dados recebidos do DataServer, um arquivo em um diretório temporário da máquina local. O caminho para esse arquivo é informado como parâmetro para o programa que dispara o processamento no VOID, juntamente com os demais parâmetros gerados a partir dos dados recebidos na criação da tarefa.

O método de execução de tarefas (`executeTask()`), portanto, é um método *stub* que faz a interface entre a plataforma baseada em *Web Services* com a execução por linha de comando do aplicativo. Para controlar a execução da aplicação é criada uma nova *thread*, que aguarda até que o programa seja encerrado e avisa ao sistema sobre o seu estado, indicando caso tenha ocorrido erros na execução. A ocorrência de erros é detectada através do código de retorno do programa, que retorna 0 (zero) em caso de sucesso ou um número diferente caso contrário. Se a execução ocorrer conforme o esperado, um novo arquivo é gerado pelo VOID na máquina local, como resultado do processamento. Esse arquivo é então disponibilizado pelo MiningServer ao VisServer para que seja gerada a visualização.

A transferência de arquivos de dados e modelos pelo ambiente foi implementada utilizando o método de envio de URL. Ou seja, é requisitado ao componente que retorne o endereço onde o recurso está disponível, e este é recuperado através de uma chamada HTTP comum. Isso porque o mesmo arquivo é utilizado várias vezes, e pode

ficar em *cache* por muito tempo, não sendo regerado a cada requisição. As imagens produzidas na visualização são transferidas como resposta ao chamado de método no SOAP, já que cada visualização retorna uma imagem diferente e possivelmente única, não sendo reaproveitada para outras requisições — isso tornaria complicada a gerência do armazenamento das imagens.

Os arquivos que representam as bases de dados são gerados no formato CSV, pelos motivos discutidos na seção 4.1.3.2 e que serão experimentados na seção 5.2.1. Os resultados da mineração, de forma semelhante, são transferidos para o serviço de visualização em formato textual.

Capítulo 5

Resultados Experimentais

Neste capitulo apresentamos alguns experimentos realizados com a plataforma Tamanduá. Primeiro temos uma avaliação qualitativa, cujo objetivo é avaliar a experiência obtida com o desenvolvimento de uma aplicação real baseada em SOA, mostrando uma avaliação pessoal com relação à metodologia utilizada nessa construção. Na segunda parte, será avaliada uma questão importante para sistemas desse tipo, porém não discutida em outros trabalhos, mostrando o impacto do formato utilizado no conjunto de dados que serão transferidos pela rede. Na última parte, será mostrado um teste de carga realizado no sistema para avaliar a sua escalabilidade e tentar quantificar o uso da plataforma.

5.1 Análise qualitativa

O uso de uma arquitetura orientada a serviços (SOA) se mostrou uma boa solução para a criação de uma plataforma de mineração de dados, o Tamanduá. Apesar dessa avaliação ser subjetiva, consideramos que esse modelo é melhor que o modelo tradicional, onde todos os componentes do sistemas estão embutidos em um único programa. O uso de computação orientada a serviços força o desenvolvedor a pensar de forma diferente, já que não há um lugar único para acesso às informações e aos dados do programa. Isso em um primeiro momento torna mais complexa a implementação, pois é importante a preocupação com a redução da transferência de dados entre os componentes para economizar tráfego na rede e com o controle de estado dos serviços e das versões dos dados. Essas decisões de projeto influenciam em muito a construção da arquitetura do sistema, que deve ser constituído de componentes independentes, na medida do possível, dos demais. Essa flexibilidade, entretanto, torna o processo de expansão do sistema mais simples posteriormente.

O uso de *Web Services* para a implantação de SOA proporciona flexibilidade para a distribuição dos serviços do sistema, tornando mais fácil a implantação deles em redes distintas, ao contrário de outras tecnologias semelhantes. As abstrações disponibiliza-

das pelo SOAP para a criação das interfaces de comunicação foram suficientes para a construção dos serviços, mesmo os mais complexos, atendendo aos requisitos impostos pelo problema.

A arquitetura do Tamanduá ainda não é a ideal, porém o problema proposto — descoberta de conhecimento em bases de dados — se adeqüa ao paradigma de serviços e o resultado esperado foi atingido: uma plataforma funcional de mineração de dados baseada em serviços. Com essa experiência, pudemos demonstrar o grande potencial do uso de SOA e *Web Services*. Por exemplo, a plataforma poderia ser expandida facilmente, com a definição de novos serviços, para atingir também a etapa de pré-processamento, anterior à disponibilização de dados, o que não foi feito ainda. O impacto para a disponibilização dessa nova funcionalidade na plataforma seria muito maior se não fosse utilizada SOA. Diversas melhorias têm sido propostas e serão implantadas futuramente.

A utilização do sistema em um ambiente real com seus serviços geograficamente distribuídos não pôde ser avaliada, devido à não implantação da plataforma nessa escala. Como o Tamanduá é um sistema novo e em implantação, todos os clientes da plataforma atualmente em operação utilizam versões locais, em uma máquina ou em um *cluster* de máquinas, de todos os serviços. Porém, percebemos que não há limitações conhecidas à sua implantação com os serviços distribuídos em diferentes redes pela Internet, desde que todas autorizem o acesso ao porto HTTP da máquina onde estão instalados.

5.2 Transferência de dados

Como vimos, a transferência de dados em um sistema distribuído é muito importante e tem grande impacto no funcionamento do sistema. Para trafegar dados pequenos, como parâmetros para um serviço, o uso do protocolo de comunicação usado na invocação dos serviços (como o SOAP) é suficiente, porém quando os dados são muito grandes nem sempre isso é verdade.

Num ambiente Web há alguns mecanismos para a troca de arquivos de dados, que vimos anteriormente, como o padrão *SOAP with Attachments* e a recuperação de um arquivo acessível por uma URL comum. Em termos de processamento, ambos são semelhantes, pois simplesmente enviam um fluxo de dados pela rede. Por exemplo, pode-se ler um arquivo e enviá-lo diretamente a quem estiver solicitando, sem nenhum outro processamento. O primeiro método adiciona um processamento à requisição pois acrescenta antes do arquivo a mensagem de resposta SOAP, não existente no segundo caso. Porém, este necessita também de uma requisição SOAP para identificar a URL onde poderá ser recuperado o arquivo. Nesse caso há a necessidade de duas requisições ao servidor para realizar a mesma transferência do caso anterior, porém, como mencionado, há a vantagem de se permitir que o arquivo seja armazenado em

mecanismos de *cache* existentes na rede. Isso pode ser útil nos casos em que o arquivo não muda constantemente.

Outra grande preocupação é com o formato usado para transferir os dados. A seção a seguir discute a influência que o formato usado no arquivo tem sobre o desempenho do sistema.

5.2.1 Formato dos arquivos de dados

Para verificar a questão dos formatos usados na transferência de dados, selecionamos duas bases reais volumosas, sendo a primeira formada por muitos registros com poucas colunas e a segunda com poucos registros, porém muitas colunas. Foram verificados inicialmente os tamanhos das bases no formato CSV, aparentemente menor. Posteriormente, foram construídos, a partir desses arquivos, 4 arquivos XML — nomeados XML1, XML2, XML3 e XML4 —, todos com elementos de identificação de registros com nome *reg* e identificação dos campos como abaixo:

XML1: Nomes comprimidos (1,2,3...), sem endentação.

XML2: Nomes comprimidos (1,2,3...), com endentação.

XML3: Nomes originais das colunas, sem endentação.

XML4: Nomes originais das colunas, com endentação.

Os arquivos sem endentação foram criados sem quebra de linha nem espaços entre *tags*. Os arquivos com endentação foram criados com quebra de linha após as *tags* (abertura e fechamento) de registro, um caractere de tabulação antes do elemento de campo e um fim de linha após, com o objetivo de torná-los mais legíveis. Ambos os formatos constituem arquivos XML válidos segundo os padrões recomendados [18].

As tabelas 5.1 e 5.2 mostram a comparação (tamanhos em *bytes*) do arquivo CSV com os arquivos XML gerados e o acréscimo percentual que esses últimos têm em relação ao primeiro. Na coluna "Compactado" está o tamanho dos arquivos compactados com o algoritmo *gzip* — utilizado opcionalmente pelo protocolo HTTP [17] para transferência de arquivos —, suas porcentagens em relação ao tamanho original e o acréscimo dos arquivos compactados em relação ao primeiro.

Tabela 5.1: Tamanho dos arquivos para a Base 1 (14506 registros com 16 colunas)

	Original	Acréscimo	Compactado		Acréscimo
CSV	2368351	-	240009	10,13%	-
XML1	4123577	74,11%	273699	6,64%	14,04%
XML2	4616781	94,94%	279657	6,06%	16,52%
XML3	6328489	167,21%	302674	4,78%	26,11%
XML4	6821693	188,04%	306886	4,50%	27,86%

Tabela 5.2: Tamanho dos arquivos para a Base 2 (3916 registros com 385 colunas)

	Original	Acréscimo	Compactado		Acréscimo
CSV	13861872	-	1032681	7,45%	-
XML1	28135692	102,97%	1854217	6,59%	79,55%
XML2	31158844	124,78%	1963007	6,30%	90,09%
XML3	46227612	233,49%	2873262	6,22%	178,23%
XML4	49250764	255,30%	2899760	5,89%	180,80%

Como podemos observar pelas tabelas, o número de colunas influencia muito no crescimento do arquivo XML, visto que cada coluna adiciona no mínimo duas vezes seu nome (*tag* de inicio e fim) mais cinco caracteres (delimitadores <, > e /). Para cada registro são adicionadas também as *tags* de inicio e fim <reg> e </reg> (11 caracteres). Portanto, o acréscimo no arquivo de dados necessário para a formatação XML, que tornaria o arquivo melhor estruturado, é considerável, mesmo compactado, principalmente quando se considera a transferência entre máquinas distantes na Internet.

Para decidir entre o uso ou não do XML nesses casos, deve-se levar em consideração a real necessidade da estruturação dos dados: o acesso ao serviço de agentes externos ao sistema exige uma boa estrutura dos dados para o entendimento do arquivo ou uma documentação bem elaborada do formato usado resolve o problema? Caso essa estruturação seja necessária, o uso de XML é indispensável na representação dos dados. Se apenas a documentação do sistema for suficiente para evitar erros no acesso aos dados — como um sistema que só será disponibilizado para uso restrito a um pequeno grupo de clientes —, um arquivo CSV pode ser utilizado na transferência sem maiores impactos.

5.3 Desempenho da Arquitetura

Criar serviços que atendem aos objetivos funcionais do sistema não é suficiente para o sucesso da arquitetura. É muito importante prover os serviços com a qualidade desejada pelo usuário, garantindo um tempo de resposta aceitável para o usuário e a escalabilidade do sistema. Como discutido anteriormente, o usuário espera que o sistema seja rápido e esteja disponível o tempo todo, sem erros.

Para verificar se é possível garantir isso aos clientes, precisamos medir qual a carga máxima que a arquitetura suporta em determinada montagem inicial. A carga de um sistema pode ser definida como o conjunto de todas as entradas recebidas do ambiente durante um período de tempo especificado [40]. No caso dos sistemas WWW, a carga é constituída pelas requisições enviadas ao serviço pelos clientes. Chamamos de monta-

gem do sistema a disposição dos serviços da arquitetura no ambiente distribuído (seja na Internet ou rede local), ou seja, onde estão localizados os serviços que serão utilizados. Como aplicações baseadas em SOA permitem a localização dinâmica de serviços, é necessário mostrar que opções alternativas para a montagem do sistema podem ser usadas para manter o tempo de resposta aceitável com o crescimento do número de sessões de usuários simultâneas. Isso demonstra que a arquitetura em questão é escalável.

O objetivo dessa seção é mostrar o estudo feito sobre o desempenho do Tamanduá, com o objetivo de provar que é um sistema escalável.

5.3.1 Metodologia

Para avaliar a escalabilidade do Tamanduá foram realizados vários experimentos com a geração de cargas crescentes no sistema. A carga foi adaptada a partir de dados coletados de sessões reais de usuários do sistema. Esses dados foram obtidos a partir de um trabalho de caracterização do Tamanduá que está sendo realizado no Departamento de Ciência da Computação da Universidade Federal de Minas Gerais [78]. Essa caracterização tem como objetivo, entre outros, a criação de grafos de modelos de comportamento dos usuários (CBMG, *Customer Behavior Model Graphs* [79, 40]) a partir de registros de ações armazenados para a auditoria do sistema.

Com essa técnica, conseguiu-se caracterizar três modelos de comportamentos de usuários do Tamanduá: o primeiro modelo, que representa 10% dos casos, de usuários que permanecem a maior parte do tempo criando e executando tarefas (figura 5.1); o segundo, 25% dos casos, de usuários que criam, executam e visualizam o resultado (figura 5.2); e o último, 65%, de usuários que visualizam detalhadamente os resultados das tarefas (figura 5.3).

O ambiente utilizado para a realização dos experimentos é um *cluster* formado de máquinas com processador Pentium 4 3.0 GHz, 1 GB de memória RAM, 1 disco rígido com capacidade de 120 GB, taxa de transferência de 1.5 Gb/s e tempo médio de busca de 8 ms. Todas utilizam o sistema operacional Linux, com *kernel* 2.6.7, com as mesmas configurações, e estão interligadas por um *switch* de 24 portas com capacidade de 100 Mb/s, formando uma rede isolada. A proposta dos testes é avaliar o desempenho e a escalabilidade da arquitetura orientada a serviços do Tamanduá sem considerar o tráfego da rede nem o processamento da mineração. Nesse ambiente, o algoritmo de mineração no VOID foi substituído por um programa simples que simula a execução, consumindo um tempo aleatório para completar a tarefa, porém sem impor maior uso da CPU e memória ao sistema, uma vez que no ambiente real esse processamento é realizado em um sistema independente.

A carga é gerada com o envio de várias requisições à aplicação WWW, como seria

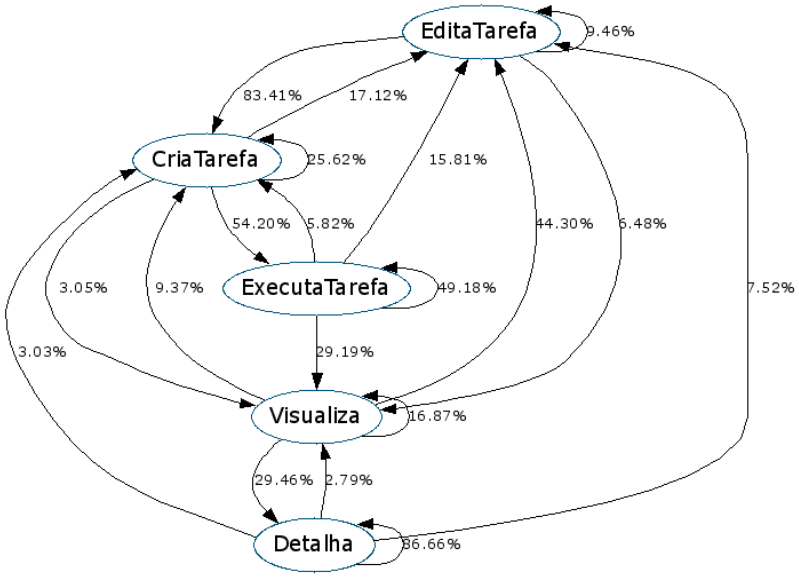

Figura 5.1: CBMG para usuários que, na maior parte do tempo, criam e editam tarefas.

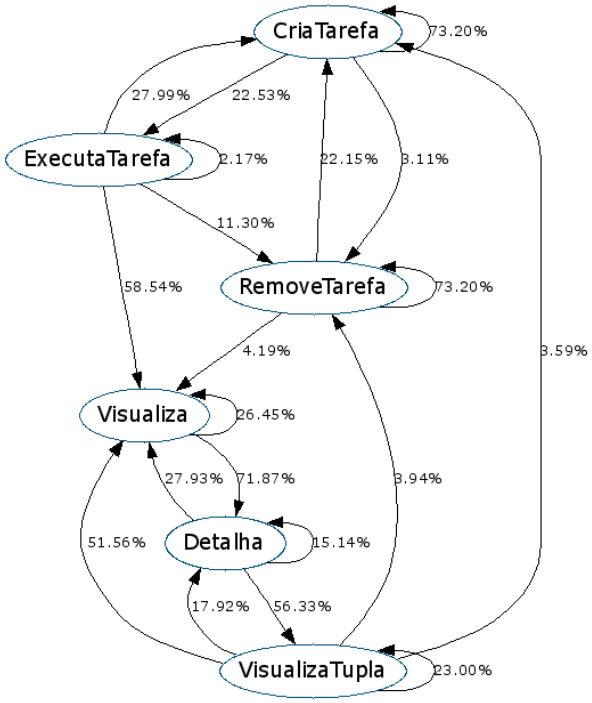

Figura 5.2: CBMG para usuários que criam e visualizam tarefas.

feito pelos usuários, utilizando o programa Jakarta JMeter [1]. Esse programa, desenvolvido em Java, permite a alocação de clientes em diversas máquinas, cada qual representando um determinado número de usuários simultâneos. Cada usuário segue um plano de testes pré-determinado, adaptado dos CBMGs apresentados, para tentar

[1] Jakarta JMeter: http://jakarta.apache.org/jmeter

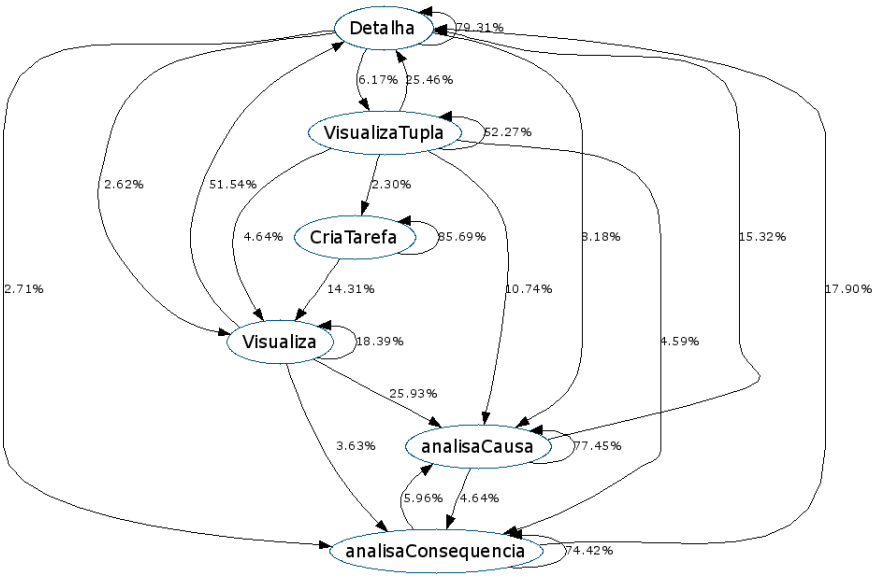

Figura 5.3: CBMG para usuários que só visualizam as tarefas.

simular o comportamento de clientes reais.

Os testes são executados por uma hora, com tolerância de 5 minutos para o início de todas as sessões de usuário. Para cada experimento foram medidos os seguintes dados: requisições feitas, com indicação de sucesso e tempo de resposta percebido pelo usuário (registrado no instante do recebimento da resposta completa), informado pelo JMeter; uso da CPU e memória das máquinas servidoras, coletado com o programa `vmstat` do Linux, somando os campos de tempo utilizado pelo usuário (us), tempo utilizado pelo sistema (si) e tempo esperando por E/S (wa); uso da CPU por processo de interesse (nesse caso, os servidores de aplicação e os de banco de dados), utilizando o comando `ps`, que faz a medida de uso da CPU dos processos dividindo o tempo em que ele está ativo pelo tempo em que teve a posse do processador — esse mecanismo não é muito preciso e pode apontar valores abaixo do real [2]; e tráfego da rede, utilizando o aplicativo `tcpdump`. Foram definidos como padrão de qualidade as métricas: tempo de resposta menor que 3 segundos e quantidade de erros percebidos até 10%.

O primeiro conjunto de testes foi feito com todos os serviços e a aplicação WWW em uma mesma máquina, como ilustrado na figura 5.4. Todos acessavam os respectivos dados em um único sistema gerenciador de banco de dados (SGBD). O objetivo foi iniciar a pesquisa com a montagem mais simples possível, para avaliar o limite de usuários simultâneos suportado pelo sistema. Em cada teste foram aplicadas ao sistema cargas de 10 a 1000 usuários simultâneos, buscando identificar o ponto de saturação do sistema. Um segundo conjunto de testes, mais direcionado, foi então realizado implantando cada serviço separado em uma máquina dedicada, utilizando um SGBD

[2]Mais informações sobre o `ps` em http://procs.sf.net.

local (veja figura 5.5). Com isso, conseguimos perceber uma melhora significativa no desempenho do sistema, demonstrando que a arquitetura é escalável. Apresentaremos a seguir os resultados mais interessantes encontrados.

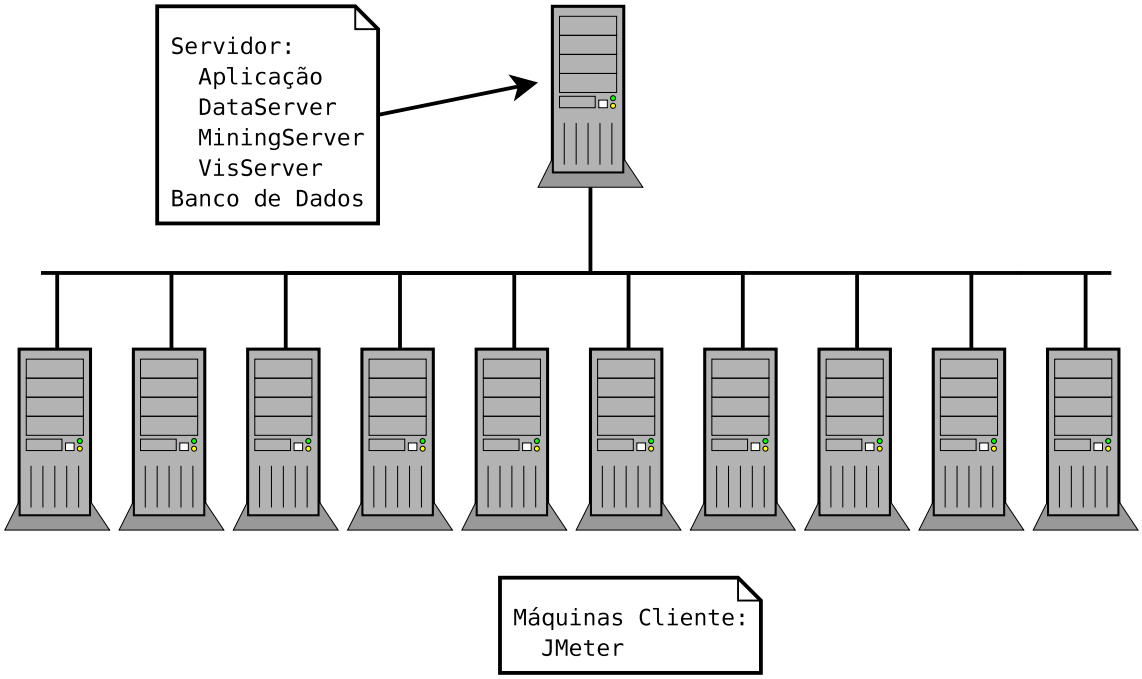

Figura 5.4: Montagem 1 do Tamanduá: todos os serviços numa mesma máquina.

5.3.2 Resultados

A primeira bateria de testes, realizada com todos os serviços em uma mesma máquina, foi feita com testes de 100 a 1000 usuários simultâneos. O número de requisições totais e o número de requisições com erros verificados nos experimentos mais significativos estão na tabela 5.3. São consideradas requisições com erros aqueles que não foram respondidas pelo servidor ou cuja resposta não corresponde ao esperado.

Podemos verificar que a partir de 400 usuários simultâneos começamos a ter muitos erros (acima de 10%), com 500 menos de 70% das requisições têm tempo de resposta menor que 3 segundos e a partir de 600 usuários temos uma queda no número total de requisições, que vinha crescendo constantemente. O aumento dos erros e do tempo de resposta são um demonstrativo de que o sistema já não está conseguindo processar corretamente as requisições[3].

A queda no número de requisições indica que o sistema está saturado, provavelmente devido ao tempo de resposta muito alto. Para verificar isso, o gráfico de requisições enviadas à aplicação (figura 5.6) foi comparado ao gráfico de variação do tempo de

[3]Posteriormente verificou-se que o servidor Jakarta Tomcat gerava um certo número de páginas de erro quando sua carga ultrapassava um certo valor, o que não pode ser isolado nos dados coletados

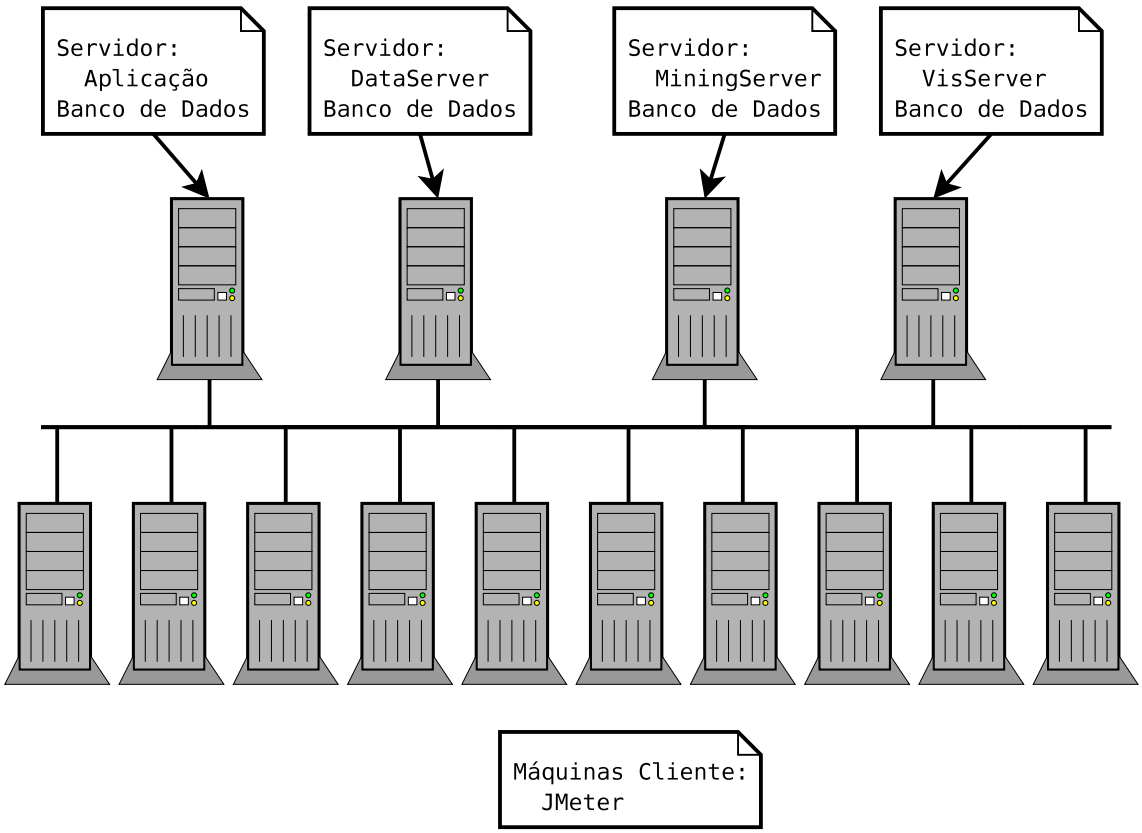

Figura 5.5: Montagem 2 do Tamanduá: cada serviço numa máquina dedicada.

Tabela 5.3: Comparativo de requisições feitas ao sistema

Núm. Usuários	Total	Erros (%)	Requisições com tempo de resposta até 3s (%)
100	34041	9,5	84,5
150	51239	8,2	94,5
200	62113	10,4	89,6
400	136271	24,6	75,2
500	149630	26,6	64,6
600	133048	23,9	52,8
700	94907	24,0	49,0
800	109812	23,7	48,6
900	60246	33,9	43,5

resposta (figura 5.7) no experimento de 600 usuários simultâneos. O primeiro mostra a média de requisições que receberam um resposta completa (ou tiveram erros) em determinado momento; o segundo apresenta o tempo de resposta médio percebido pelo usuário, desde o momento de envio da requisição até o recebimento completo da resposta. Há uma clara relação entre esses gráficos, com picos de tempo de resposta elevado no mesmo instante em que há quedas bruscas no número de requisições. Isso

se repete nos demais experimentos, demonstrando que realmente a queda no número de requisições se deve ao longo tempo que os clientes precisam esperar para fazer uma nova requisição. Observe, por exemplo, que quando os tempos de resposta ultrapassam 60 segundos, não é registrado nenhuma requisição com sucesso na média do instante seguinte.

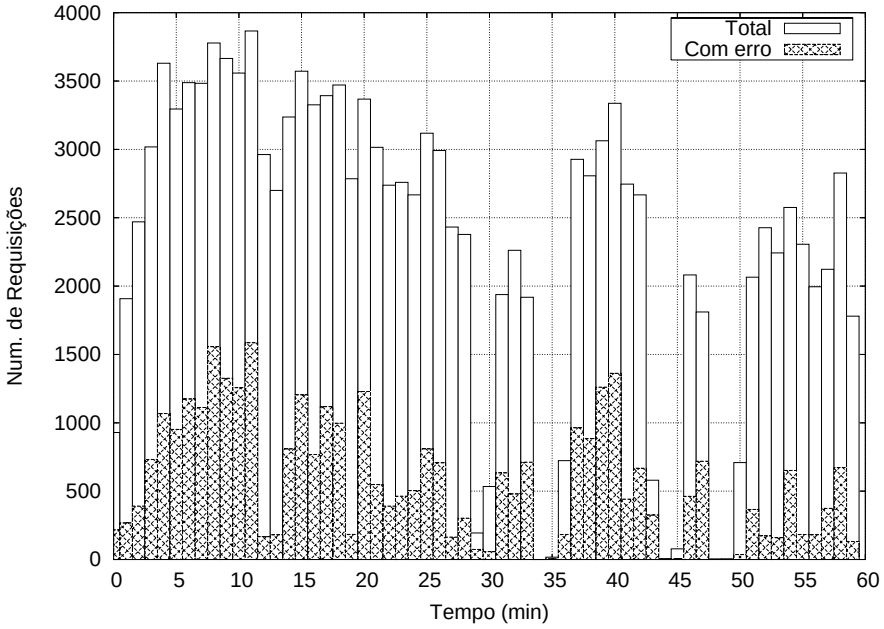

Figura 5.6: Gráfico de requisições médias por minuto para 600 usuários.

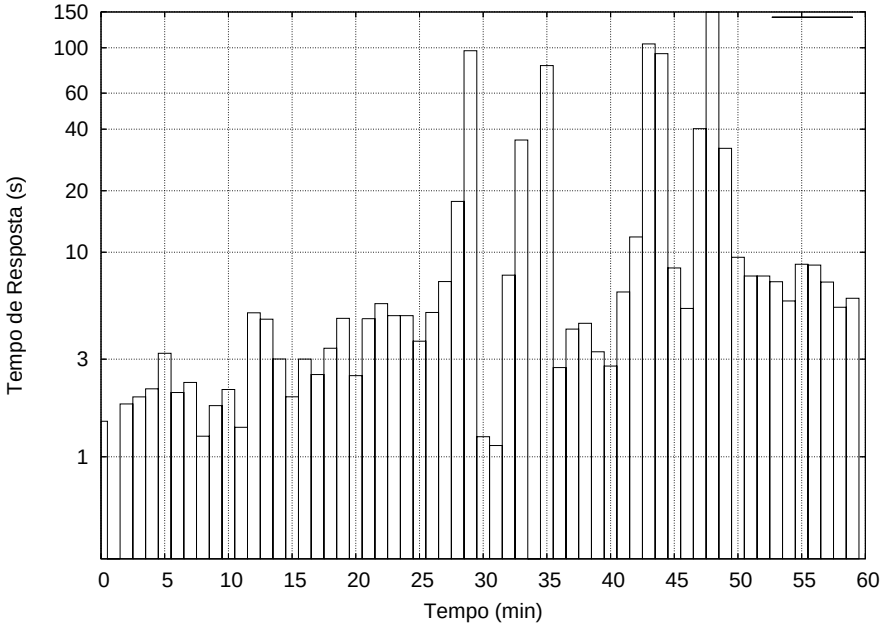

Figura 5.7: Variação do tempo de resposta médio para 600 usuários.

É necessário entender agora os fatores que tornam o tempo de resposta tão elevado.

Comparando os gráficos das figuras 5.8 5.9 que mostram o uso da CPU medido pelo `vmstat` no experimento com 400 (antes da saturação) e com 600 usuários (após a saturação), percebemos que há momentos em que não há processamento. Provavelmente isso é devido à grande fila de espera de requisições de entrada e saída na máquina durante os períodos em que o tempo de resposta estava saturado, porém não foram coletados dados que nos permitam afirmar isso com segurança.

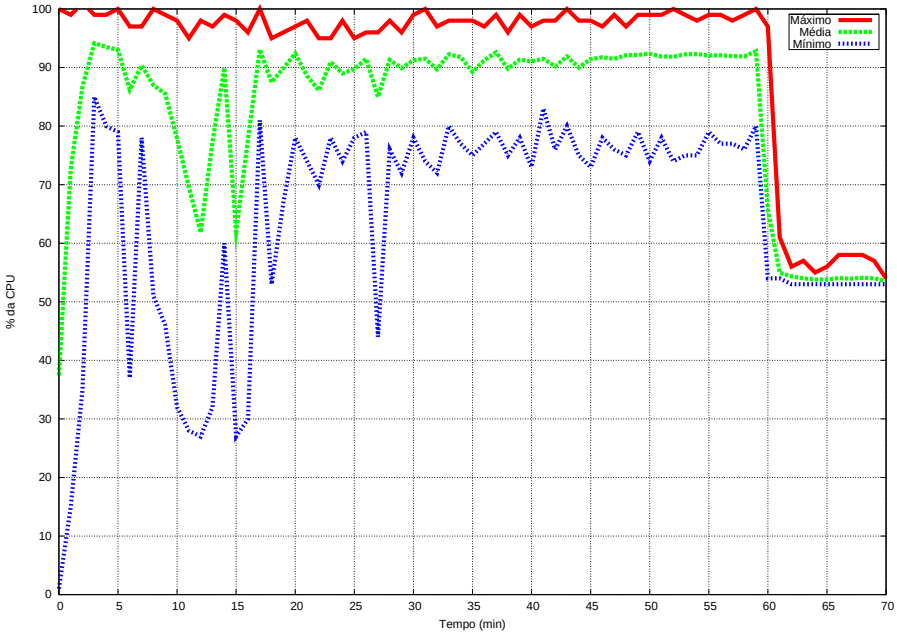

Figura 5.8: Gráfico da média de uso da CPU por minuto para 400 usuários.

Para identificar qual o programa que responsável pela maior parte da carga, foram coletadas informações de uso da CPU pelos processos do servidor de aplicações e do SGBD. Nas figuras 5.10 e 5.11 temos dois exemplos dos dados coletados com o programa `ps` durante os testes. Apesar de os dados informados por esse programa reportarem valores mais baixos que o real, o efeito é o mesmo para todos os programas. Com base nessa análise, podemos identificar que o servidor de aplicações é o responsável pelo uso excessivo da CPU com o crescimento do número de usuários.

A grande utilização da CPU pelo servidor de aplicações, que nessa bateria de testes é responsável pela aplicação Web e os três servidores (DataServer, MiningServer e VisServer), torna natural a escolha pela próxima configuração do sistema, onde cada conjunto de serviços é colocado em uma máquina separada. Nessa nova configuração, temos agora cada servidor com seu próprio SBGD.

Podemos notar pela tabela 5.4 que o número de requisições agora cresce até 1000 usuários e se estabiliza nesse patamar, pelo menos até 1500 usuários. O número de erros, em comparação ao caso anterior, diminuiu, porém sem grandes melhorias[4]. Já o

[4]Novamente, há uma contribuição da sobrecarga do Tomcat, que não pôde ser isolada devido à

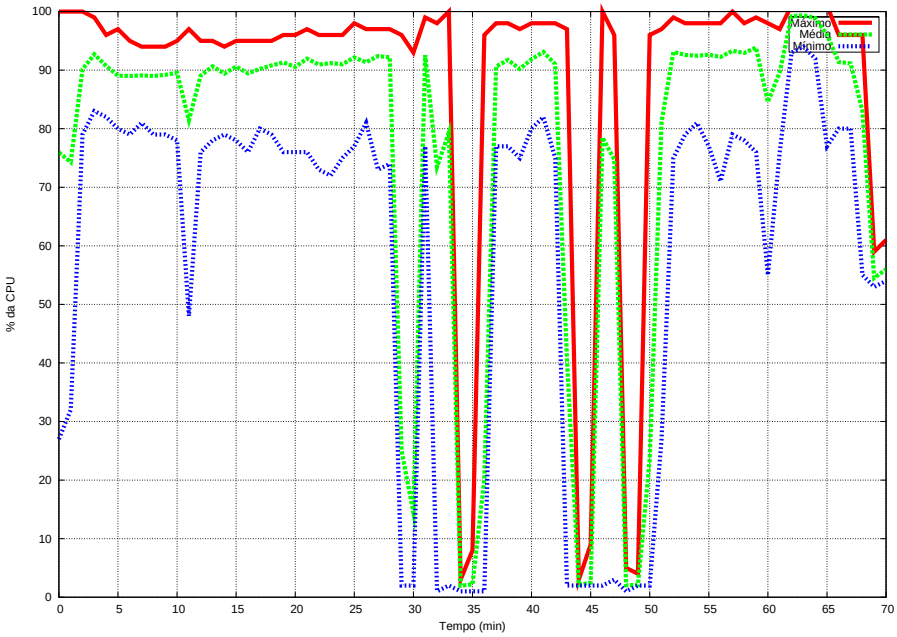

Figura 5.9: Gráfico da média de uso da CPU por minuto para 600 usuários.

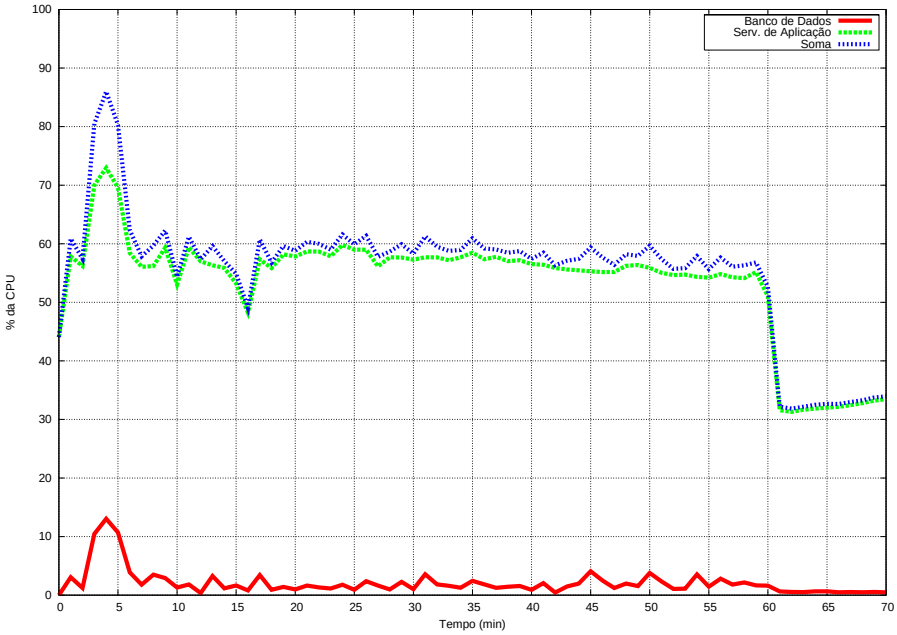

Figura 5.10: Gráfico da média de uso da CPU medida pelo `ps` para 400 usuários.

tempo de resposta melhorou bastante e até 900 usuários, cerca de 90% das requisições tiveram a resposta em até 3 segundos. Vemos nas figuras 5.12 e 5.13 os gráficos de requisições por minuto e tempo de resposta para o experimento de 1000 usuários. Pode-se perceber que nesse experimento, mesmo com tantos usuários, não há instantes sem requisições como no experimento de 600 usuários mostrado anteriormente (figura 5.6)

forma como os dados foram coletados

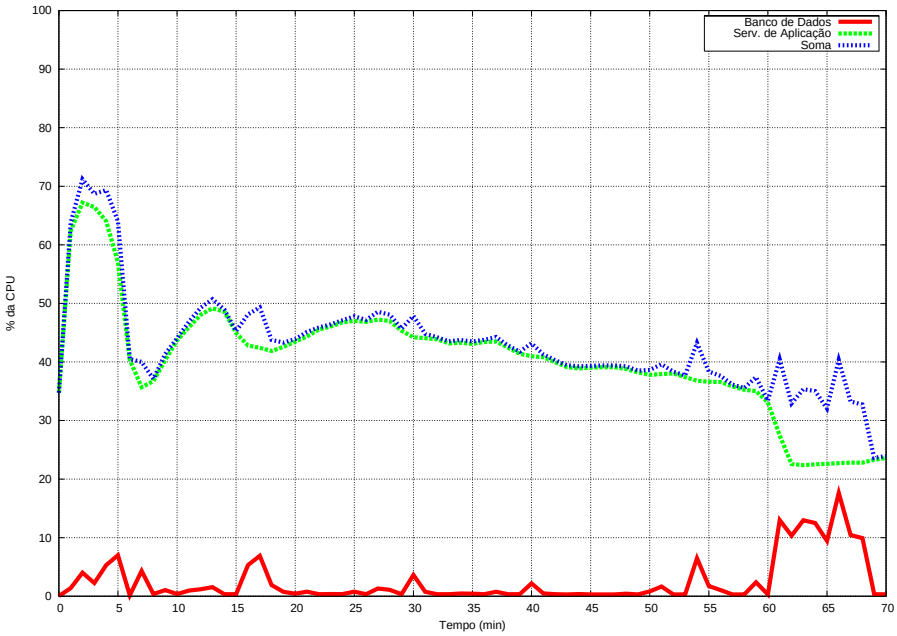

Figura 5.11: Gráfico da média de uso da CPU medida pelo `ps` para 800 usuários.

e não há picos de tempos de resposta muito elevados.

Tabela 5.4: Comparativo de requisições feitas ao sistema com servidores separados

Núm. Usuários	Total	Erros (%)	Requisições com tempo de resposta até 3s (%)
500	185263	12.1	95.7
600	226262	18.6	94.3
700	266047	23.4	93.2
800	285289	25.4	88.2
900	277272	21.8	94.9
1000	302690	25.5	68.7
1100	314487	26.0	58.5
1200	307941	25.2	50.2
1300	314781	25.9	48.2
1400	299849	24.8	42.3
1500	305189	25.2	39.8

O uso médio da CPU (medido pelo `vmstat`) nos testes de 900, 1000 e 1500 usuários estão nos gráficos das figuras 5.14, 5.15 e 5.16. Estão representadas cada máquina responsável pelos serviços, o que nos ajuda a identificar o uso que é feito por cada componente do sistema. Podemos notar que o VisServer é responsável por grande parte do processamento total do sistema — o que era esperado, já que a maioria dos usuários visualizam os resultados — e o DataServer não consome quase nada. Percebe-se também que o uso da CPU na máquina da aplicação se estabiliza a partir

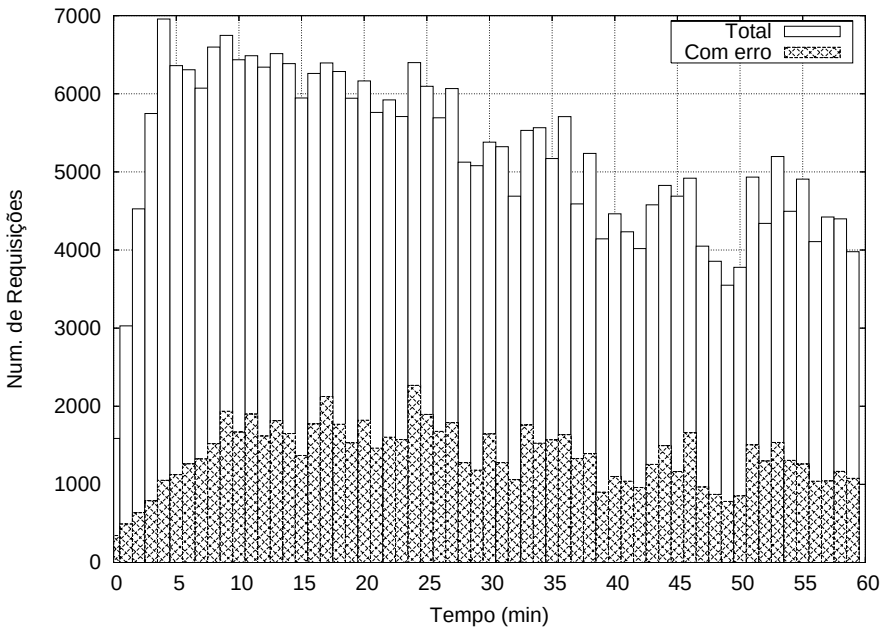

Figura 5.12: Gráfico de requisições médias por minuto para 1000 usuários.

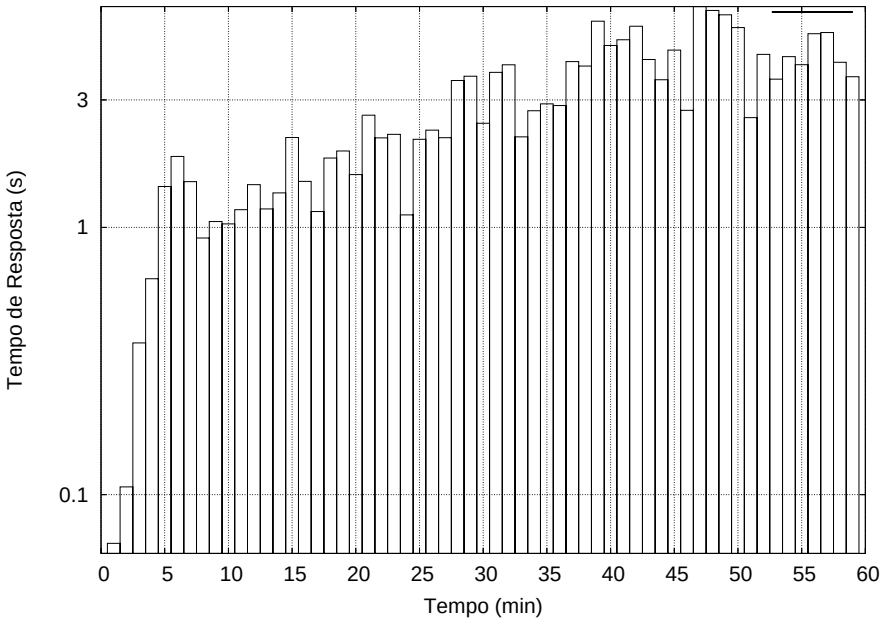

Figura 5.13: Variação do tempo de resposta médio para 1000 usuários.

de 1000 usuários, devido ao limite atingido pela aplicação, que não consegue servir mais requisições dos clientes, forçando-os a esperar mais tempo para fazerem novas requisições.

Como demonstrado nos gráficos, o sistema satura (e as requisições feitas ao sistema se estabilizam) por que o servidor de visualização está saturado e depois outros servidores também alcançam o ponto de saturação. O uso da arquitetura orientada a serviços

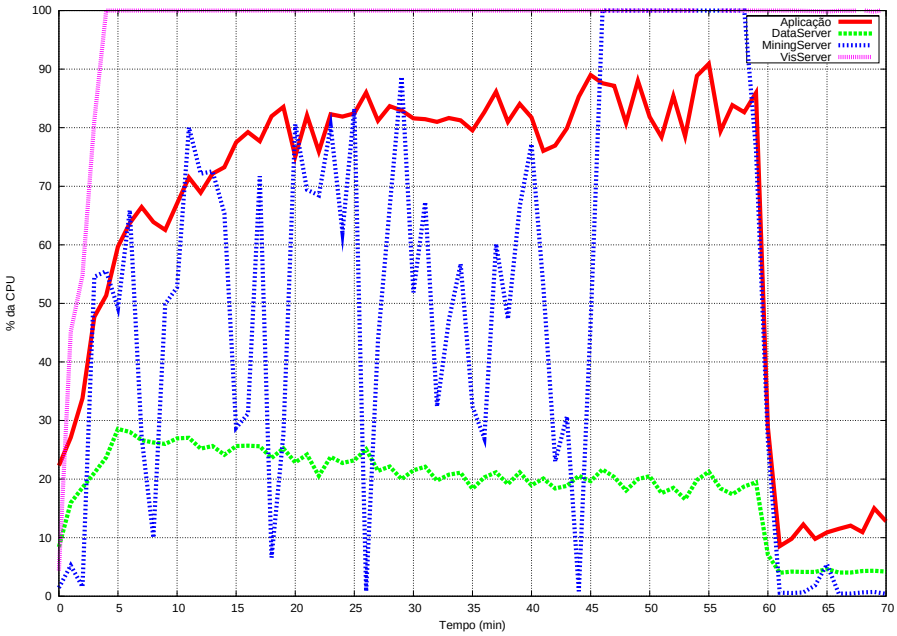

Figura 5.14: Gráfico da média de uso da CPU por minuto para 900 usuários.

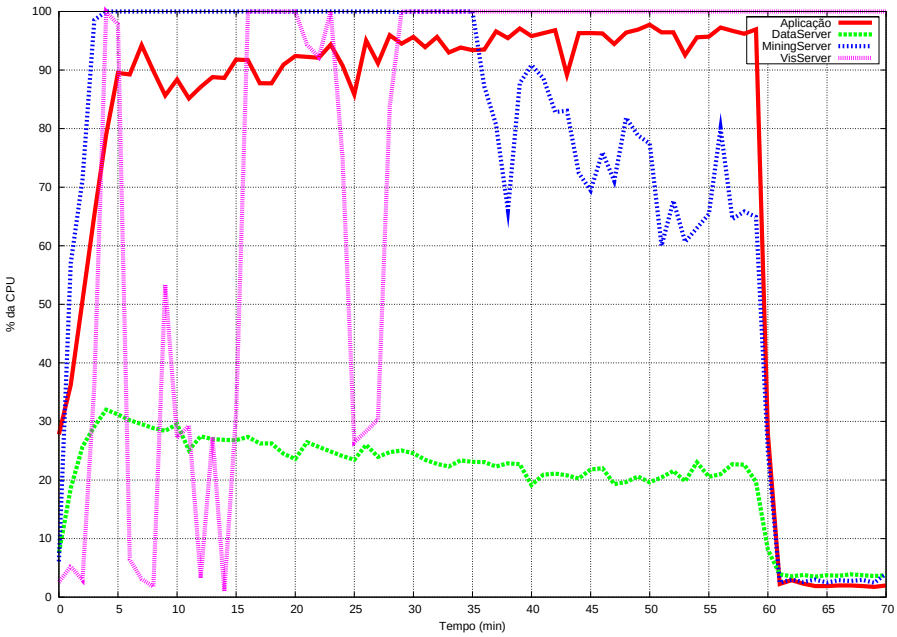

Figura 5.15: Gráfico da média de uso da CPU por minuto para 1000 usuários.

e a forma como o sistema foi implementado nos permitiria resolver esse problema simplesmente acrescentando um novo VisServer (e depois outros servidores), para que eles dividissem a carga, atendendo assim a mais clientes.

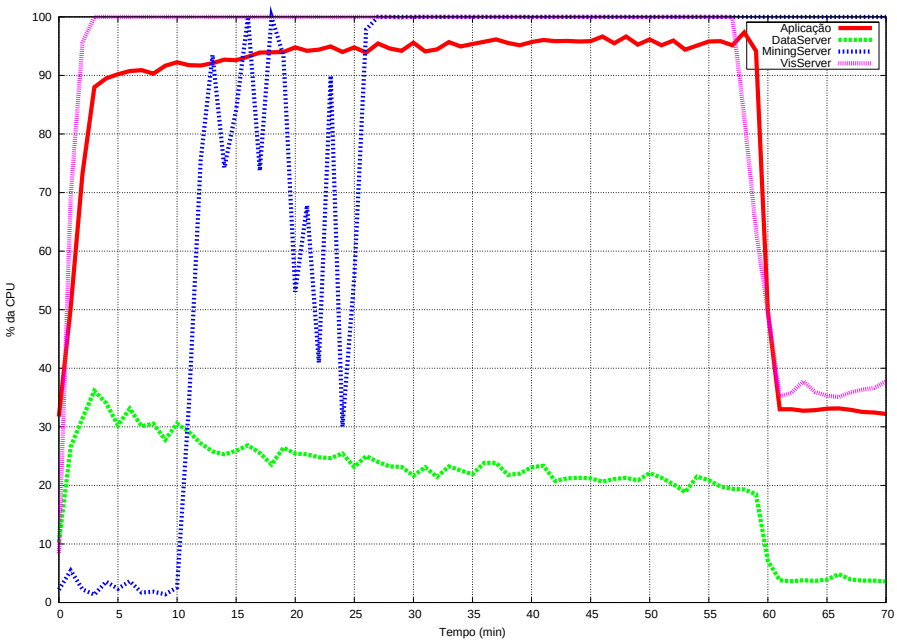

Figura 5.16: Gráfico da média de uso da CPU por minuto para 1500 usuários.

Capítulo 6

Conclusões e Trabalhos Futuros

Esse trabalho procurou analisar a viabilidade de sistemas baseados no paradigma de Arquitetura Orientada a Objetos (SOA). A motivação principal foi o crescente interesse de empresas e instituições na interação em tempo real entre seus diversos aplicativos — e com os de outras empresas —, muitas vezes localizados em espaços físicos distantes, porém interligados pela grande rede mundial, a Internet.

A grande quantidade de opções existentes para a implantação de SOA nos levou a identificar entre elas a que melhor atende às necessidades apresentadas, de forma simples e objetiva. Por isso, essa dissertação teve seu foco em um conjunto de tecnologias que são conhecidas como *Web Services* e utilizadas para a criação de serviços que podem ser acessados através da WWW. Foi demonstrado que *Web Services* oferecem mecanismos eficientes para a comunicação entre processos através da Internet, utilizando protocolos conhecidos, como HTTP, para ultrapassar barreiras comumente impostas pelos administradores de rede. Além disso, a sua padronização permite a independência de linguagem de programação ou plataforma de implantação, permitindo maior flexibilidade para a interligação de diversos sistemas heterogêneos.

Outro objetivo era o de apresentar a criação de uma arquitetura baseada em *Web Services*. Para isso, foi apresentado o Tamanduá, uma plataforma interativa e multi-usuária de mineração de dados, que permitiu a demonstração de como é criado e projetado um sistema desse tipo. Essa plataforma foi testada em um ambiente controlado, com a geração de carga baseada em comportamentos reais de usuários do sistema. Pelos resultados apresentados na seção 5.3 pode-se verificar a escalabilidade da arquitetura, podendo sofrer alterações na disposição dos serviços para suportar uma demanda maior de usuários.

Acredito que a criação dessa arquitetura e os experimentos realizados demonstram o grande potencial do uso de orientação a serviços para a implantação de arquiteturas distribuídas e de *Web Services* como tecnologia para implantá-las.

Há vários trabalhos que podem ser realizados a partir desta dissertação. Um dos principais seria a avaliação e implantação de mecanismos de autenticação e segurança

para acesso aos serviços. Como visto, esses são requisitos fundamentais em sistemas distribuídos, principalmente os que executam sobre a Internet. Um sistema distribuído de mineração de dados não pode trafegar dados importantes dos clientes sobre a Internet sem proteção a devida proteção. A avaliação do arcabouço conhecido como *WS-Resources* para a substituição de *Web Services* também deverá ser considerada em trabalhos futuros porque é uma tecnologia que promete trazer várias vantagens disponíveis no paradigma de computação em grade para a construção de serviços Web.

A plataforma Tamanduá, ainda recente, tem muito o que avançar em direção ao processo completo de descoberta de conhecimento em bases de dados. Alguns desses avanços são a re-seleção iterativa de conjuntos de dados e atributos em qualquer passo de pré-processamento necessário e definição de prioridades para a fila de execução no MiningServer, tentando prever quais as tarefas serão mais rápidas. Além disso, poderia ser adicionado aos serviços a possibilidade de envio apenas de partes alteradas e adicionadas dos arquivos de dados transferidos, com o objetivo de melhorar o tempo de transmissão de bases e resultados.

Na área de análise de desempenho pode-se avaliar o impacto de uso de um sistema de *cache* (ou *proxy*) na transferência de arquivos de dados. Isso se torna interessante considerando que as decisões tomadas na implementação do Tamanduá, de transferir bases de dados e resultados por uma requisição HTTP padrão, possibilitam a aplicação de tais técnicas uma vez que o recurso disponibilizado através de uma dessas URLs dificilmente é alterado.

Referências Bibliográficas

[1] David A. Chappell and Tyler Jewell. *Java Web services*. 2002. Using Java in service-oriented architectures.

[2] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 2001.

[3] Marco Righetti. Arquitetura orientada a serviços. *MundoJava*, (06):57–62, 2004.

[4] Daniel A. Menascé and Virgílio A. F. Almeida. *Capacity Planning for Web Performance: metrics, models and methods*. Prentice Hall PTR, 1998.

[5] Jen-Yao Chung, Kwei-Jay Lin, and Richard G. Mathieu. Guest editors' introduction: Web services computing - advancing software interoperability. *IEEE Computer*, 36(10):35–37, 2003.

[6] Clay Shirky. Web services and context horizons. *IEEE Computer*, 35(9):98–100, 2002.

[7] Mike P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *WISE*, pages 3–12. IEEE Computer Society, 2003.

[8] A service oriented architecture for a health research data network. In *SSDBM '04: Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, page 443, Washington, DC, USA, 2004. IEEE Computer Society.

[9] S. R. Amendolia, F. Estrella, R. McClatchey, D. Rogulin, and T. Solomonides. Managing pan-european mammography images and data using a service oriented architecture. In *IDEAS-DH '04: Proceedings of the IDEAS Workshop on Medical Information Systems: The Digital Hospital (IDEAS-DH'04)*, pages 99–108, Washington, DC, USA, 2004. IEEE Computer Society.

[10] P. Baglietto, M. Maresca, A. Parodi, and N. Zingirian. Deployment of service oriented architecture for a business community. In *EDOC '02: Proceedings of the Sixth International ENTERPRISE DISTRIBUTED OBJECT COMPUTING Conference (EDOC'02)*, page 293, Washington, DC, USA, 2002. IEEE Computer Society.

[11] Gang Dang, Zhi-Quan Cheng, Shi-Yao Jin, Tao Yang, and Tong Wu. A service-oriented architecture for tele-immersion. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 646–649, Washington, DC, USA, 2005. IEEE Computer Society.

[12] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Ramasamy Uthurusamy. Summary from the KDD-03 panel: data mining: the next 10 years. *SIGKDD Explorations*, 5(2):191–196, 2003.

[13] Web services architecture. Technical report, World Wide Web Consortium (W3C), 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211, visitado em 06/2004.

[14] David Sprott. Service oriented process matters, 2002. em CBDi Forum Newsletter, referenciado por Perrey *et al.* [15].

[15] Randall Perrey and Mark Lycett. Service-oriented architecture. In *SAINT-W '03: Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops)*, page 116, Washington, DC, USA, 2003. IEEE Computer Society.

[16] Virgílio A. F. Almeida and Daniel A. Menascé. Capacity planning: An essential tool for managing web services. *IT Professional, IEEE*, 4(4):33–38, 2002.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1. Technical report, Network Working Group, 06 1999.

[18] Extensible Markup Language (XML) 1.0. Technical report, World Wide Web Consortium (W3C), 02 2004. http://www.w3.org/TR/2004/REC-xml-20040204/, visitado em 01/2005.

[19] Erik T. Ray. *Aprendendo XML.* Campus, 2001.

[20] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1. Technical report, World Wide Web Consortium (W3C), 03 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315, visitado em 01/2005.

[21] UDDI executive overview: Enabling service-oriented architecture, 10 2004. http://uddi.org/pubs/uddi-exec-wp.pdf.

[22] JianJun Yu and Gang Zhou. Dynamic web service invocation based on uddi. In *CEC-EAST '04: Proceedings of the E-Commerce Technology for Dynamic E-Business, IEEE International Conference on (CEC-East'04)*, pages 154–157, Washington, DC, USA, 2004. IEEE Computer Society.

[23] SOAP version 1.2 part 0: Primer. Technical report, World Wide Web Consortium (W3C), 06 2003. http://www.w3.org/TR/2003/REC-soap12-part0-20030624, visitado em 01/2005.

[24] John J. Barton, Satish Thatte, and Henrik Frystyk Nielsen. SOAP messages with attachments. Technical report, World Wide Web Consortium (W3C), 12 2000. http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211, visitado em 02/2005.

[25] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

[26] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *Softw. Pract. Exper.*, 32(15):1437–1466, 2002.

[27] Ian T. Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. *IEEE Computer*, 35(6):37–46, 2002.

[28] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Publicado online em http://www.globus.org/research/papers/ogsa.pdf, 01 2002.

[29] Ian T. Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. Technical report, 2002.

[30] Karl Czajkowski, Don Ferguson, Ian Foster, Jeff Frey, Steve Graham, Tom Maguire, David Snelling, and Steve Tuecke. From open grid services infrastructure to ws-resource framework: Refactoring and evolution. 2004. http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf.

[31] Gregory Piatetsky-Shapiro. Knowledge discovery in real databases: A report on the IJCAI-89 workshop. *AI Magazine*, 11(5):68–70, 1991.

[32] Usama Fayyad, Georges G. Grinstein, and Andreas Wierse. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers Inc., 2001.

[33] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. *Advances in Knowledge Discovery and Data Mining*, 1995.

[34] Mohammed J. Zaki. Parallel and distributed association mining: A survey. *IEEE Concurrency*, 7(4):14–25, 1999.

[35] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Intl. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[36] Adriano Veloso, Wagner Meira Jr., Renato Ferreira, Dorgival Guedes Neto, and Srinivasan Parthasarathy. Asynchronous and anticipatory filter-stream based parallel algorithm for frequent itemset mining. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *PKDD*, volume 3202 of *Lecture Notes in Computer Science*, pages 422–433. Springer, 2004.

[37] Tom Soukup and Ian Davidson. *Visual Data Mining: Techniques and Tools for Data Visualization and Mining*. John Wiley & Sons, Inc., 2002.

[38] Fernando Almir Nascimento Júnior. Visualização de regras de associação. Master's thesis, Universidade Federal de Minas Gerais, 2005.

[39] Shonali Krishnaswamy, Arkady B. Zaslavsky, and Seng Wai Loke. Towards data mining services on the internet with a multiple service provider model: An xml based approach. *J. Electron. Commerce Res.*, 2(3):103–130, 2001.

[40] Daniel A. Menascé and Virgílio A. F. Almeida. *Capacity Planning for Web Services: metrics, models and methods*. Prentice Hall PTR, 2002.

[41] Pearl Brereton and David Budgen. Component-based systems: A classification of issues. *Computer*, 33(11):54–62, 2000. referenciado em [42].

[42] Mark Turner, David Budgen, and Pearl Brereton. Turning software into a service. *IEEE Computer*, 36(10):38–44, 2003.

[43] Keith H. Bennett, Paul J. Layzell, David Budgen, Pearl Brereton, Linda A. Macaulay, and Malcolm Munro. Service-based software: the future for flexible software. In *APSEC*, pages 214–221. IEEE Computer Society, 2000.

[44] Jaideep Chandrashekar, Zhi-Li Zhang, Zhenhai Duan, and Yiwei Thomas Hou. Service oriented internet. In Maria E. Orlowska, Sanjiva Weerawarana, Mike P. Papazoglou, and Jian Yang, editors, *ICSOC*, volume 2910 of *Lecture Notes in Computer Science*, pages 543–558. Springer, 2003.

[45] Web services architecture requirements. Technical report, World Wide Web Consortium (W3C), 2002. http://www.w3.org/TR/2002/WD-wsa-reqs-20020429, visitado em 02/2005.

[46] M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing: Introduction. *Commun. ACM*, 46(10):24–28, 2003.

[47] Sunita Sarawagi and Sree Hari Nagaralu. Data mining models as services on the internet. *SIGKDD Explor. Newsl.*, 2(1):24–28, 2000.

[48] Shonali Krishnaswamy, Arkady B. Zaslavsky, and Seng Wai Loke. Federated data mining services and a supporting xml-based language. In *HICSS*, 2001.

[49] XML Path Language (xpath) 1.0. Technical report, World Wide Web Consortium (W3C), 11 1999. http://www.w3.org/TR/1999/REC-xpath-19991116, visitado em 01/2005.

[50] Shonali Krishnaswamy, Arkady B. Zaslavsky, and Seng Wai Loke. *Architectural Issues of Web-enabled Electronic Business*, chapter Chapter 7: Internet Delivery of Distributed Data Mining Services: Architectures, Issues and Prospects, pages 114–128. Idea Group Publishing, 2003.

[51] G. Mahinthakumar, Forrest M. Hoffman, William W. Hargrove, and Nicholas T. Karonis. Multivariate geographic clustering in a metacomputing environment using globus. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, page 5. ACM Press, 1999.

[52] Reagan W. Moore. Knowledge-based grids. In *MSS '01: Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, page 29. IEEE Computer Society, 2001. referenciado por Brezany *et al.* [57].

[53] Mario Cannataro, Domenico Talia, and Paolo Trunfio. Knowledge grid: High performance knowledge discovery on the grid. In Craig A. Lee, editor, *GRID*, volume 2242 of *Lecture Notes in Computer Science*, pages 38–50. Springer, 2001.

[54] Mario Cannataro, Domenico Talia, and Paolo Trunfio. Design of distributed data mining applications on the knowledge grid. In *National Science Foundation Workshop on Next Generation Data Mining*, 2002.

[55] Mario Cannataro and Domenico Talia. The knowledge grid. *Communications of the ACM*, 46(1):89–93, 2003.

[56] Peter Brezany, Juergen Hofer, A Min Tjoa, and Alexander Woehrer. Towards an open service architecture for data mining on the grid. In *Conference on Database and Expert Systems Applications*. IEEE Computer Science Press, 2003.

[57] Peter Brezany, Juergen Hofer, A Min Tjoa, and Alexander Woehrer. Gridminer: An infrastructure for data mining on computational grids. In *Conference and Exhibition on Advanced Computing, Grid Applications and eResearch*. APAC, 2003.

[58] Guenter Kickinger, Peter Brezany, A. Min Tjoa, and Juergen Hofer. Grid knowledge discovery processes and an architecture for their composition. In *IASTED Conference 2004*, fevereiro 2004.

[59] Madhusudhan Govindaraju, Aleksander Slominski, Venkatesh Choppella, Randall Bramley, and Dennis Gannon. Requirements for and evaluation of rmi protocols for scientific computing. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 61. IEEE Computer Society, 2000.

[60] Kenneth Chiu, Madhusudhan Govindaraju, and Randall Bramley. Investigating the limits of soap performance for scientific computing. In *HPDC '02: Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02)*.

[61] Madhusudhan Govindaraju, Aleksander Slominski, Kenneth Chiu, Pu Liu, Robert van Engelen, and Michael J. Lewis. Toward characterizing the performance of soap toolkits. In *GRID*, pages 365–372. IEEE Computer Society, 2004.

[62] Nayef Abu-Ghazaleh, Michael J. Lewis, and Madhusudhan Govindaraju. Differential serialization for optimized soap performance. In *HPDC*, pages 55–64. IEEE Computer Society, 2004.

[63] The Axis Development Team. Apache axis 1.2. http://ws.apache.org/axis/, visitado em 06/2004.

[64] Sun Microsystems. Java remote method invocation. http://java.sun.com/products/jdk/rmi/index.jsp, visitado em 01/2005.

[65] Dan Davis and Manish Parashar. Latency performance of soap implementations. In *CCGRID*, pages 407–412. IEEE Computer Society, 2002.

[66] Object Management Group. The common object request broker architecture: Core specification, version 3.0, 2002. referenciado em [67].

[67] Christopher Kohlhoff and Robert Steele. Evaluating soap for high performance business applications: Real-time trading systems. In *WWW*, 2003.

[68] N. Freed and N. Borenstein. RFC 2045 - Multipurpose Internet Mail Extensions (MIME) part one: Format of internet message bodies. Technical report, Network Working Group, 11 1996.

[69] RFC 3548 - the base16, base32, and base64 data encodings. Technical report, Network Working Group, 07 2003.

[70] T. Dierks and C. Allen. RFC 2246 - The TLS Protocol version 1.0. Technical report, Network Working Group, 01 1999.

[71] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. RFC 3546 - Transport Layer Security (TLS) Extensions. Technical report, Network Working Group, 06 2003.

[72] E. Rescorla. RFC 2818 - HTTP over TLS. Technical report, Network Working Group, 05 2000.

[73] Andrew Schmitt, Arthur P. Goldberg, and Robert Buff. Comparison of http and https sever performance. In *Int. CMG Conference*, pages 226–231. Computer Measurement Group, 1998.

[74] George Apostolopoulos, Vinod G. J. Peris, and Debanjan Saha. Transport layer security: How much does it really cost? In *INFOCOM*, pages 717–725, 1999.

[75] Xubin He. A Performance Analysis of Secure HTTP protocol. Technical report, STAR Lab, Department of Electrical and Computer Engineering, Tennessee Tech University, 03 2003.

[76] R. Khare and S. Lawrence. RFC 2817 - Upgrading to TLS within HTTP/1.1. Technical report, Network Working Group, 05 2000.

[77] Luiz Thomaz do Nascimento, Renato A. Ferreira, Wagner Meira Jr., and Dorgival Guedes. Scheduling data flow applications using linear programming. In *International Conference on Parallel Processing*, 2005.

[78] Leonardo Chaves Dutra da Rocha. Uma metodologia de caracterização de serviços de mineracao de dados. Trabalho em Andamento.

[79] Daniel A. Menasce and A. F. Almeida Virgilio. *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.