Infaestrutura Autonômica Descentralizada como Suporte para Aplicações com Demanda Elástica

Marcia Pasin[1], Felipe Silvano Perini[1], Ana L. C. Bazzan[2]

[1]Centro de Tecnologia – Universidade Federal de Santa Maria (UFSM)
Av. Roraima 1.000 – 97.105-900 – Cidade Universitária – Santa Maria – RS – Brasil

[2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{marcia,fsperini}@inf.ufsm.br, bazzan@inf.ufrgs.br

Abstract. *Web applications are becoming extremely popular posing new challenges to distributed infrastructures. Thus, system management is turning more complex and costly. Services and data tend to be distributed, heterogeneous, and there are issues related to elastic demand, failures and load peaks. Moreover, because humans are usually in charge of system management, unavailability and low performance are frequent issues. Therefore, self-management – the management by the system itself - is claimed to be a solution. In this paper, we propose the use a group of independent self-managed services as support to Web applications running over distributed infrastructures. In order to solve conflicts generated by distributed actions, negotiating agents must agree before making decisions. In addition, group communication abstractions allow consistency and failure detection.*

Resumo. *É crescente a quantidade de aplicações disponíveis na Web gerando novos desafios para infraestruturas distribuídas. De fato, a gestão de infraestruturas está mais complexa e dispendiosa, devido à necessidade de tratar demandas elásticas, falhas e picos de carga. Além disso, porque humanos normalmente são encarregados da gestão, erros de configuração e baixa disponibilidade são frequentes. Neste contexto, auto-gerenciamento é uma solução promissora. Este trabalho apresenta um sistema implementado por uma coleção de serviços independentes e descentralizados para gerenciar automaticamente uma infraestrutura computacional que suporta aplicações Web com demandas elásticas. Com a finalidade de evitar ações conflitantes dos serviços independentes, é usado o suporte da teoria de agentes. Adicionalmente, comunicação de grupo facilita a execução de operações distribuídas e detecção de falhas.*

1. Introdução

É crescente a quantidade de aplicações disponíveis na *Web* colocando, então, novos desafios para infraestruturas distribuídas que as suportam. De fato, a gestão de infraestruturas de servidores está cada vez mais complexa e dispendiosa. Uma questão recorrente é que sistemas disponíveis na *Web* estão sujeitos a falhas e picos de carga, com milhares de usuários solicitando informações concorrentemente, enquanto que em certos períodos a demanda decresce drasticamente (feriados, madrugadas, por exemplo). Em outras palavras, a demanda que deve ser atendida por estes sistemas é elástica. Além

disso, atualmente, seres humanos são responsáveis pela maioria das tarefas de gestão de sistemas computacionais, o que pode gerar erros de configuração e ocasionar baixa disponibilidade de aplicações e de serviços.

Para tratar estas questões, uma abordagem promissora é a implementação de sistemas de suporte e gerenciamento como um serviço autonômico [Kephart e Chess 2003], através de um conjunto de propriedades de auto-gerenciamento. Basicamente, o sistema deve realizar adaptação, em face a um cenário, para atender às políticas de alto nível previamente definidas. Vantagens dessa abordagem incluem minimizar intervenção humana e adaptação automática apesar de falhas, picos de carga, ociosidade e demanda por escalabilidade.

Mudanças de demanda para servidores que suportam serviços para a *Web* podem ser evidenciadas, por exemplo, em aplicações do tipo ATIS (*Advanced Traveler Information System*). Essas aplicações oferecem informações a usuários de redes de transporte. Por exemplo, na saída de uma partida de futebol ou na saída de espetáculo, usuários tendem a fazer mais consultas sobrecarregando o sistema. Portanto, é interessante que a infraestrutura de suporte antecipe mudanças na demanda com a finalidade de prover serviço eficiente. Finalmente, outro desafio é tratar mudanças momentâneas como picos de carga abruptos e pouco duráveis, que talvez não justifiquem a inclusão de novo servidor na infraestrutura.

Este trabalho apresenta um sistema autonômico para gerenciar uma infraestrutura computacional que suporta aplicações *Web* com demandas elásticas. O sistema autonômico é implementado como uma coleção de serviços descentralizados. Baseado em informação coletada na infraestrutura, os serviços desencadeiam adaptação dinâmica quando necessário. Por exemplo, o serviço de reconfiguração adapta a quantidade de servidores disponíveis na infraestrutura com a finalidade de manter a qualidade do serviço (QoS).

De fato, o uso de sistemas autonômicos para suportar aplicações distribuídas não é um assunto novo [Roblitz *et al*. 2004, Bouchenak *et al*. 2006, Santana *et al*. 2010, Saleem *et al*. 2008]. Entretanto, como diferencial, o sistema autonômico aqui proposto é implementado por uma coleção de serviços independentes e replicados com gerenciamento totalmente descentralizado e, portanto, não apresenta os gargalos inerentes a implementações centralizadas nem representa um ponto único de falha (SPoF ou *Single Point of Failure*). Uma camada adicional, com um sistema de comunicação de grupo confiável permite tais características e assegura consistência entre réplicas.

Entretanto, dado que serviços são implementados e executados de forma independente, decisões tomadas por componentes individuais podem levar o sistema a um comportamento antagônico. Para resolver questões relacionadas com ações conflitantes, e garantir consistência, agentes negociadores são implementados como um serviço de meta-nível (sistema multiagente). Agentes devem concordar antes de tomar qualquer decisão relativa à infraestrutura gerenciada.

Mais especificamente, esta proposta destaca as seguintes características: (*i*) suporte ao tratamento de decisões conflitantes através um sistema multiagente (decisão não é somente tomada com base em métricas atuais, mas com base em dados históricos), (*ii*) economia de recursos: em períodos de ociosidade, servidores excedentes

são logicamente removidos da infraestrutura e podem ser alocados a outras aplicações ou podem operar em *standby*, e (*iii*) suporte estratégico para reconfiguração de servidores devido à demanda elástica e manutenção de configuração (número de servidores) em virtude de picos de carga momentâneos. O suporte à tomada de decisão é realizado através de um sistema multiagente. O suporte à reconfiguração é oferecido através de um sistema de comunicação de grupo confiável.

O restante deste artigo está organizado como se segue. A seção 2 apresenta a aplicação-alvo usada como cenário (aplicação para transporte público). Trabalhos relacionados são discutidos na seção 3. A seção 4 apresenta a proposta de serviços autonômicos destacando a arquitetura do sistema, seus serviços independentes e resolução de conflitos. Questões referentes à implementação são relatadas na seção 5. Validação experimental é apresentada na seção 6, levando em conta a aplicação-alvo em um cenário livre de falhas, já que o suporte adequado para detecção de falhas é provido por uma camada desenvolvida por terceiros (sistema de comunicação de grupo). A seção 7 apresenta as conclusões e trabalhos futuros são descritos na seção 8.

2. Aplicação-alvo: ATIS para transporte público urbano

A aplicação-alvo deste trabalho é do tipo ATIS e processa informação sobre condições de tráfego para usuários de sistemas de transporte. Em geral um ATIS se refere a provimento de informações de transporte particular (automóveis, por exemplo). Entretanto, está se tornando comum o provimento de informações referentes a transporte público como ônibus e outros sistemas de trânsito rápido, incluindo informações sobre rotas e linhas, a partir de dados armazenados em um banco de dados. O banco de dados é alimentado por dados coletados por sensores. Esse tipo de sistema possui alta demanda pois atende grande número de usuários simultaneamente. Em geral, o provimento do serviço alterna períodos de sobrecarga (como saídas de jogos de futebol, final de tarde, etc.) com períodos de ociosidade (madrugadas, por exemplo). Basicamente, um ATIS coleta dados em tempo real sobre carros da linha e, com base nesses dados, calcula mediante solicitação do usuário informações de tempo de percurso.

ATIS podem ser aplicados como solução interessante e eficiente para facilitar acesso ao transporte público em cidades de todos os portes. Para fins de ilustração considere o seguinte cenário: um ônibus deixa a garagem e segue uma rota e um calendário previamente estabelecidos. A rota possui um conjunto de paradas de ônibus que deve ser servido. Paradas e ônibus são identificados unicamente com o uso de identificadores (IDs). Cada vez que um ônibus chega a uma parada, ele envia para o ATIS, seu ID, o ID de parada de ônibus, e o tempo em que este evento ocorreu.

Do ponto de vista do usuário do serviço de transporte (passageiro), o ATIS calcula o tempo que um ônibus chegará à próxima parada. Usuários podem acessar informações providas pelo ATIS utilizando um computador ou dispositivos portáteis, como telefones celulares, *smartphones* e PDAs. Apoiando o ATIS, há uma infraestrutura composta por *software* e *hardware*. A Figura 1 mostra o esquema simplificado desta infraestrutura.

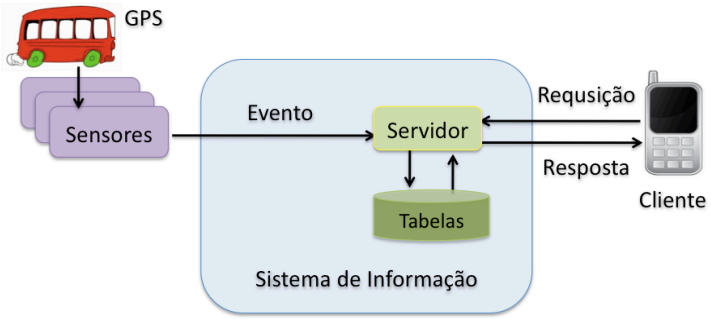


Figura 1 - Visão geral do esquema da infraestrutura que suporta o ATIS

Neste sistema, a infraestrutura de *hardware* é organizada da seguinte forma (arquitetura em três camadas): (*i*) um servidor para executar a aplicação com respectivo banco de dados associado (Tabelas), que é o principal componente do ATIS, (*ii*) telefones celulares, *smartphones*, e navegadores distribuídos geograficamente (cliente) e (*iii*) um conjunto de sensores para coletar informações sobre ônibus. Sensores podem ser de diferentes tipos, como sensores de rua usados para medir o número de veículos operacionais, e sensores associados a GPS (para saber quando um ônibus chega a uma determinada parada, a velocidade do ônibus, e a posição real do ônibus (latitude e longitude)). Um micro-controlador instalado em cada ônibus recolhe dados e os envia para o ATIS via tecnologia celular (GSM / GPRS) em tempo real. Quando a mensagem chega ao servidor, dados são usados para atualização do banco de dados com informação atual sobre o *status* da frota.

O servidor é responsável pelo processamento e utiliza como base informação sobre rotas previamente armazenadas nas tabelas do banco de dados e na informação recebida em tempo real via sensores. Mais precisamente, o servidor executa as seguintes atividades: (*i*) calcula o tempo aproximado que o próximo ônibus chegará em uma determinada parada de ônibus (em resposta de uma solicitação do cliente) e (*ii*) armazena valores captados pelos sensores instalados nos ônibus na base de dados.

O ATIS tem as seguintes características: (*i*) um servidor é sempre associado a um banco de dados que armazena uma réplica de uma tabela por linha de ônibus, e (*ii*) servidores são *multi-thread stateless* e processam dois tipos de requisição no banco de dados usando suporte transacional: somente leitura (*read only*) e somente gravação (escrita-cega). Transações do tipo *read only* não alteram o banco de dados e são geradas por usuários por meio de navegadores *Web* ou mensagens de texto de *smartphones*. Elas sempre têm uma mensagem de resposta associada. Transações de escrita-cega processam os dados gerados por sensores para atualizar o banco de dados sem qualquer operação de leitura prévia. Os servidores *stateless* facilitam a recuperação em caso de falhas pois não requerem a manutenção de estado conversacional.

A implementação da aplicação-alvo foi apresentada em [Bastos *et al*. 2010], e está sendo estendida para permitir todas as funcionalidades aqui descritas. Quando posto em operação, um cenário que provavelmente irá surgir é que tal sistema de informação estará sujeito à demanda elástica: picos de carga, com milhares de usuários solicitando informação ao mesmo tempo, contrastando com certos períodos onde a demanda decresce drasticamente. Provavelmente apenas um único servidor não esteja apto a oferecer serviço de forma apropriada, necessitando de replicação. O sistema, então, precisa contar com uma infraestrutura de suporte para sincronizar servidores (réplicas) e

se adaptar sob condições adversas visando economia de recursos e provisão de serviço eficiente quando ocorrer sobrecarga. É neste ponto que o sistema de auto-gerenciamento mostra-se importante.

3. Trabalhos relacionados

O uso de computação autonômica e adaptação dinâmica para gerenciar infraestruturas que suportam aplicações distribuídas elásticas não é novidade. Tipicamente, essas aplicações são estruturadas em camadas (cliente, servidor de aplicação, banco de dados), precisam suportar um grande número de usuários simultaneamente, e portanto, requerem replicação, o que torna o gerenciamento complexo. Sistemas que realizam adaptação de forma automática para arquiteturas em três camadas foram propostos por [Bouchenak *et al*. 2006, Bertini *et al*. 2007, Urgaonkar *et al*. 2008]. A ideia é que aplicativos legados possam ser distribuídos de forma autônoma, geridos e reconfigurados quando necessário. Esses sistemas diferem principalmente nas métricas usadas para a configuração de servidores. Em [Bouchenak *et al*. 2006] valores limiares de CPU são usados para o serviço autonômico iniciar ou remover um serviço (servidor de banco de dados ou servidor de aplicação ou servidor *Web*). Em [Urgaonkar *et al*. 2008], o mecanismo de previsão determina quantos recursos devem ser alocados para cada camada da arquitetura e outro mecanismo adapta o sistema em caso de falha e erros de previsão. O objetivo é reconfigurar o *cluster* para melhorar o tempo de resposta. Em contraste, neste trabalho a adaptação é feita totalmente em tempo de execução, sem utilizar uma pré-alocação de recursos. Em [Bertini *et al*. 2007], com base no tempo de resposta e no prazo de execução, a QoS é mantida acima de um patamar pré-estabelecido.

Um exemplo de sistema autonômico para aplicações genéricas é o DataGrid. O DataGrid [Roblitz *et al*. 2004] consiste em um conjunto de serviços para permitir auto-gestão de *clusters* e integração em um ambiente de *grid*. Auto-configuração é usada para adicionar automaticamente novos nós nos *clusters*, enquanto auto-cura garante integridade funcional do *software* e da infraestrutura de *hardware*.

Um serviço auto-configurável para atender especificamente um sistema com alta demanda de processamento foi proposto por [Saleem *et al*. 2008]. Um sistema de previsão de tempo é usado como aplicação-alvo. Baseado na prioridade das tarefas (a serem executadas e em execução), de acordo com a disponibilidade de recursos, o sistema aloca tarefas prioritárias a processadores e posterga ou interrompe tarefas com baixa prioridade. Em [Wang *et al*. 2008] é proposto um serviço de monitoramento em dois níveis: balanceamento de carga, e frequencia de utilização da CPU, para uso eficiente de recursos em um conjunto de servidores virtualizados.

Em [Santana *et al*. 2010] é apresentado um sistema para previsão de carga em um *cluster* visando economia de energia. Em contraste com este trabalho, a arquitetura aqui proposta é totalmente distribuída, com ausência de ponto centralizador para distribuição de carga e controle dos trabalhadores (*workers*).

Um trabalho que também defende a descentralização de serviços é o *Smart Grid* [Birman *et al*. 2011]. O *Smart Grid* é uma proposta descentralizada de um conjunto de serviços para gerenciar uma grade de computadores. Como diferencial ao *Smart Grid*, no trabalho aqui proposto, decisões de adaptabilidade são tomadas por uma camada de sistema multiagente, oferecendo suporte adequado à tomada de decisão.

Outra alternativa para suporte a aplicações com demanda de elasticidade é aproveitar serviços privados disponíveis comercialmente para *clouds* (Amazon, IBM, HP, entre outros). Uma dificuldade, no entanto, é escolher *a priori* um tipo de configuração de *cloud* adequado às necessidades da aplicação-alvo. Outro ponto contra o uso de serviços privados é que não se tem o controle da localização física dos dados ou processamento. As consultas de aplicação ATIS são relativamente ágeis e relativas a uma cidade específica, a um fuso específico e possuem algum tipo de restrição temporal. Por exemplo, é complicado saber em tempo hábil qual é o próximo ônibus que chegará em uma determinada parada da malha de transporte urbano de Porto Alegre a partir de um sistema executando na Ásia. A consulta teria mais sentido se fosse executada nos servidores de uma permissionária de ônibus que atende à parada em questão, ou seja, usando servidores locais na própria cidade de Porto Alegre, utilizando exclusivamente a base de dados in *situ* (sem qualquer tipo de consulta distribuída globalmente). Finalmente, outro ponto importante em se tratando de serviços para a população e provimento de mobilidade urbana, é a necessidade de gerar soluções de baixo custo com tecnologia aberta, que possam ser efetivamente usadas indiscriminadamente por prefeituras e permissionárias do serviço de transporte. Como diferencial aos trabalhos aqui relacionados, o sistema aqui proposto possui gerenciamento totalmente descentralizado e foi desenvolvido tendo em conta as particularidades de aplicações ATIS.

4. Arquitetura do sistema de auto-gerenciamento

O sistema de auto-gerenciamento (*self-managed system* ou SMS) proposto neste trabalho é formado por um conjunto de quatro camadas: (*i*) a infraestrutura de servidores gerenciáveis, que executa um sistema de informação (ATIS), (*ii*) suporte à comunicação confiável de grupo, (*iii*) serviços de auto-gerenciamento e (*iv*) agentes de *software*. A camada do ATIS foi anteriormente descrita (seção 2). A Figura 2 mostra um esboço desta arquitetura.

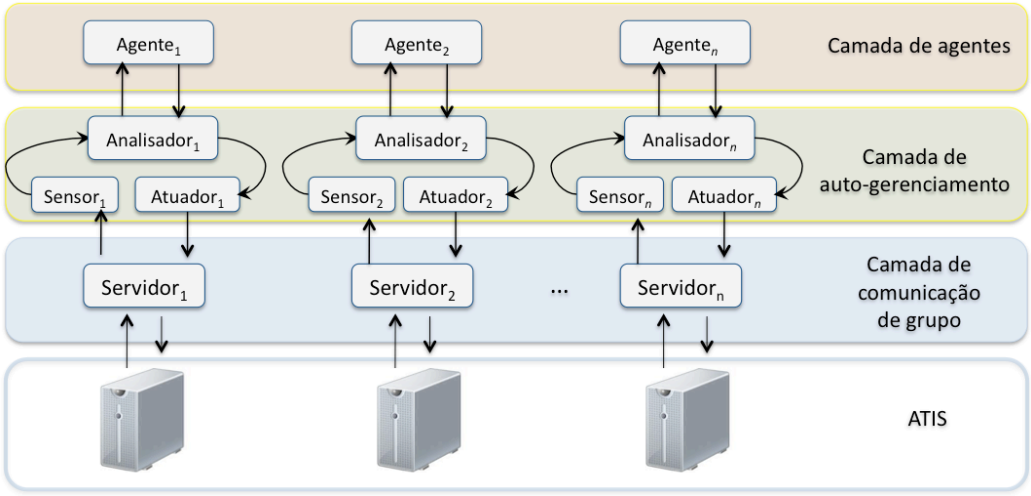


Figura 2 - A arquitetura do sistema

O serviço de auto-gerenciamento é uma camada presente em todos os servidores onde a aplicação-alvo executa. O conjunto desses servidores forma um grupo de servidores. Basicamente, na arquitetura proposta, a camada mais importante executa um sistema com serviços de auto-gerenciamento independentes que possuem sensores, analisadores

e atuadores. Componentes são descentralizados para evitar gargalos e pontos únicos de falha. A camada de comunicação de grupo oferece suporte para a detecção de falhas. De fato, componentes podem estar replicados. As camadas que compõem a arquitetura serão descritas mais precisamente nos itens que se seguem.

4.1 Comunicação confiável de grupo

A abstração de grupos e o conceito de sincronia virtual (Birman 1993) oferecem uma camada-base que implementa comunicação confiável e serviço de detecção de falhas para os serviços distribuídos da camada de auto-gerenciamento. Na arquitetura proposta, o suporte a grupos é implementado pela biblioteca JGroups, (disponível em: http://www.jgroups.org/). O JGroups oferece suporte para a manutenção de uma lista de servidores não-falhos que corresponde ao grupo de servidores operacionais. Servidores em suspeita de falha são automaticamente excluídos do grupo de servidores pelo serviço de detecção de falhas provido pelo JGroups. Quando um novo servidor é inicializado, o JGroups automaticamente adiciona este servidor ao grupo de servidores.

4.2 Serviços de auto-gerenciamento

O núcleo do sistema de auto-gerenciamento consiste em um conjunto de serviços independentes que orquestra o grupo de servidores e respectivos bancos de dados associados usando informações históricas e atuais obtidas através de monitoramento da infraestrutura. O sistema de auto-gerenciamento é baseado nos conceitos de correlação de eventos, o que significa que uma ação leva à detecção de um padrão, ou seja, um conjunto de um ou mais eventos que, eventualmente, corresponde a uma regra ou a um conjunto de regras (estas regras foram esboçadas em trabalho anterior [Pasin *et al*. 2011]). Correlação de eventos é um mecanismo frequentemente usado para a tomada de decisão requerida pelos sistemas de auto-gerenciamento. Eventos são detectados por sensores localizados em cada nodo da infraestrutura e são analisados por um dos módulos analisadores usando um conjunto de regras pré-definidas. Eventos são informações extraídas de cada nodo como a taxa de ocupação da CPU, ocorrência de falha, pico de carga e ociosidade. Cada evento é coletado por um sensor diferente e é classificado em classes como por exemplo avisos ou alertas. Os alertas são manipulados pelos analisadores usando um sistema baseado em regras, e, possivelmente são combinados com outros eventos dentro de uma janela de tempo. Regras podem, então, acionar a execução de reação associada, que pode ser executada por atuadores (ou serviços) diferentes. Esse ciclo de execução, típico de sistemas autonômicos, é conhecido como MAPE [Kephart & Chess 2003]. Uma janela de tempo é implementada pela correlação de eventos atuais e históricos em um mecanismo de série temporal. Isso evita, em muitos casos, adaptação desnecessária devido a pico momentâneo de carga, por exemplo.

O sistema de auto-gerenciamento, presente em todos os servidores que compõem o grupo de servidores, dispõe de um conjunto de serviços independentes implementados por um conjunto de regras para garantir propriedades de auto-otimização e auto-cura. Auto-otimização significa adaptar o número atual de servidores ativos, em face à demanda elástica. Esta propriedade é implementada principalmente pelos serviços de reconfiguração e detecção de picos de carga e ociosidade. Auto-cura significa restaurar o serviço em presença de falhas e ataques. Esta propriedade é implementada principalmente pelos serviços de detecção de falhas e reconfiguração. Se

um servidor falha, ele não estará disponível para atender às solicitações dos usuários e será excluído logicamente da infraestrutura computacional.

Além dos serviços de reconfiguração, detecção de picos de carga e ociosidade e de detecção de falhas, outros serviços auxiliares são necessários, como balanceamento de carga, replicação e transferência de estado. Cada serviço é inicializado por um atuador diferente e implementado por um módulo diferente. A Tabela 1 sumariza os serviços. A descrição dos serviços segue no texto.

Serviço de balanceamento de carga. Este serviço distribui requisições de usuários entre os servidores operacionais, seguindo a política *round robin*. Servidores operacionais estão descritos em uma lista que é atualizada dinamicamente pela camada de comunicação de grupo (JGroups).

Tabela 1 - Serviços de auto-gerenciamento

Serviço	**Descrição do serviço**
Balanceamento de carga	Distribui a carga entre os servidores
Detecção de falhas	Detecta falhas de *crash* em nodos
Detecção de picos de carga e ociosidade	Detecção e antecipação de picos de carga e ociosidade
Reconfiguração de nodos	Remove ou adiciona um servidor de aplicação
Replicação de estado consciente	Garante consistência do banco de dados replicado
Transferência de estado	Atualiza uma base de dados defasada

Serviço de detecção de falhas. Este serviço executa a detecção de falhas de *crash* em nodos da infraestrutura. Cada servidor da infraestrutura de servidores continuamente anuncia para o serviço de detecção de falhas que ele está vivo, observando um período de tempo predefinido. Quando um *heartbeat* é perdido, o nodo é declarado como suspeito de falha pelo serviço de detecção de falhas. Por conseguinte, este servidor será marcado como danificado e será excluído da *lista de servidores operacionais* e não receberá mais qualquer solicitação de cliente até sua recuperação. Um sistema de comunicação de grupo serve como suporte para a implementação deste serviço.

Serviço de detecção de picos de carga e ociosidade. Este serviço trata da detecção de picos de cargas momentâneos, em um cenário com grande quantidade de requisições de clientes em execução concorrentemente, e de eventos de ociosidade de serviço, onde servidores da infraestrutura podem ser realocados para outras atividades. Na implementação atual, este serviço possui dois módulos: um módulo local (presente em todos os servidores da infraestrutura) e um módulo distribuído globalmente. Localmente, cada servidor da infraestrutura é periodicamente monitorado (através de métricas, como taxa de utilização da CPU). Limites são previamente definidos para avaliar os valores máximos e mínimos para as métricas coletadas em nodos. Levando em conta essas métricas e limites, se o baixo limiar é alcançado, o nodo é marcado pelo módulo local como ocioso. Se o limiar elevado é alcançado, o nodo é marcado como sobrecarregado. O módulo global avalia valores coletados em todos os módulos locais e valores armazenados anteriormente para planejar ações futuras, ou seja, para decidir

sobre a adição ou remoção de um servidor na infraestrutura. Para evitar a adição ou remoção de um nodo por um pico momentâneo (ou a remoção de um nó, devido ao cálculo de um valor de carga momentânea baixo), a média dos últimos valores históricos obtidos nos servidores da infraestrutura é avaliada pelo módulo global em um mecanismo de séries temporais.

Serviço de reconfiguração. Este serviço inicia um novo servidor de aplicação em um nodo disponível na infraestrutura (se existe tal nodo) ou pára um servidor de aplicativos se o sistema estiver ocioso. Informações do estado atual do sistema e informação históricas armazenadas como uma série temporal são avaliadas para evitar reconfiguração devido um pico de carga momentânea ou ociosidade momentânea. O número de servidores ativos é sempre igual ou menor que o número de nós operacionais na infraestrutura. Se esse limite for atingido, o sistema começa a enfileirar pedidos dos clientes, para posterior execução. Finalmente, se a fila de espera está cheia, serviços são negados aos clientes. Nodos que não estão executando uma instância de um servidor de aplicação e banco de dados associado podem ser usados para outros fins.

De fato, a implementação atual dos serviços é baseada em séries temporais unidimensionais. Apenas um único parâmetro (taxa de uso de CPU) é levado em conta para avaliar quando um nodo está sobrecarregado ou ocioso. No entanto, esta implementação pode ser estendida para incluir um esquema de séries temporais multi-dimensionais [Xu *et al*. 2010], que comporta um conjunto de métricas tais como taxa de ocupação de memória, número de usuários por servidor, e largura de banda de rede. Séries temporais multi-dimensionais permitem maior precisão para capturar o comportamento do sistema dinâmico e podem ser usadas para prever o comportamento do sistema real e desencadeamento do sistema de auto-reconfiguração. No entanto, essa sobrecarga de atividades deve ser previamente avaliada.

Serviço de replicação de estado persistente. Levando em conta o uso de servidores *stateless*, e o uso de aplicações que não necessitam de provimento consistência tão severa (como os ATIS), a consistência do banco de dados pode ser feita através de um protocolo multiversão tradicional [Bernstein & Goodman 1981] executado individualmente em cada instância de banco de dados. Se uma operação de atualização é recebida pelo ATIS, ela é imediatamente enviada por *multicast* confiável via JGroups para todos os servidores operacionais. Então, ao receber a mensagem, os servidores geram um novo item de dado e depois substituem a versão antiga pela nova versão. Esta abordagem não exige protocolos de bloqueio e permite o acesso simultâneo da informação por diferentes clientes. Clientes possivelmente irão obter informações levemente defasadas, contudo isso não representa um entrave para muitas aplicações não-críticas (como ATIS), pois esta informação consiste em uma aproximação da situação real do sistema (e uma aproximação, neste caso, já é suficiente).

Serviço de transferência de estado. Um servidor de aplicativos que esteve fora de operação durante um período de tempo (por falha ou ociosidade) deve recuperar seu estado usando um procedimento chamado de transferência de estado. Durante este procedimento, o servidor que estava falho solicita a versão atual das tabelas armazenadas em algum servidor operacional. Este serviço deve ser implementado de forma gradativa, sem onerar demasiadamente o desempenho do sistema.

4.3 Resolução de conflitos (camada de sistemas multiagente)

Sobre a camada de serviços, agentes encapsulam analisadores e estabelecem comunicação com os demais agentes através de troca de mensagens. A troca de mensagens entre agentes/analisadores é implementada através de um sistema de comunicação de grupo. Conflitos podem ocorrer desde que serviços são implementados de forma independente e se um conjunto de eventos corresponde a um conjunto de duas ou mais regras. Se agentes detectam que o sistema poderá entrar em conflito, eles negociam para garantir consistência e que o sistema avança sempre para um estado seguro (isto é, o progresso deve ser garantido).

Desta forma, uma estratégia para detectar e resolver conflitos é necessária. Por exemplo, dado o seguinte cenário: se duas instâncias do *serviço de detecção de picos de carga e ociosidade* que está replicado devido à necessidade de tolerância a falhas detectam que a carga do sistema é baixa e solicitam a remoção de um servidor, então o sistema irá remover dois servidores quando a remoção de apenas um servidor seria suficiente. Como resultado, o número de servidores operacionais na infraestrutura reduz de forma excessiva, o que pode afetar negativamente o desempenho. É importante observar que a remoção do servidor é lógica e não física: a remoção de um servidor significa que ele não pertence mais aquele grupo de replicação e pode ser realocado para outro propósito.

Uma estratégia simples para resolução de conflitos é executar apenas uma ação e, em seguida, avaliar novamente a situação real do sistema antes de decidir sobre a execução de outra ação. Entretanto, a aplicação constante de ações seguidas de avaliações nem sempre garante progresso do sistema. De fato, o que se espera é que a execução em conjunto de uma ou mais ações garanta sempre o progresso do sistema. Essas ações seriam analisadas globalmente em um contexto específico e poderiam agregar informações sobre dados históricos. Neste sentido, o mecanismo de resolução de conflitos atualmente implementado é baseado em votação e agrega informação sobre a reputação dos agentes envolvidos na tomada de decisão.

Resolução de conflitos é um grande desafio para sistemas distribuídos. Muitas técnicas para detectar e resolver conflitos são descritas na literatura para sistemas centralizados (votação, FIFO, aleatória, prioridades, preferências previamente registradas, etc.) mas a aplicação destas técnicas em sistemas dinâmicos e/ou distribuídos não é tarefa trivial. Garantir QoS diante de tantas adversidades requer lidar com a dificuldade de manutenção de uma visão global do sistema. A estratégia inicial adotada neste trabalho é um mecanismo de votação distribuída. Se um analisador a_i detecta ação conflitante, ele desencadeia um processo de eleição em duas rodadas. Na primeira rodada, cada nodo não falho a_n recebe a mensagem do coordenador a_i solicitando um voto e envia seu voto para todos os nodos ativos a_n. Na segunda rodada, todos nodos ativos a_n recebem os votos de todos os demais nodos e decidem localmente que ação deve ser tomada com base nos votos recebidos.

A votação abriu a possibilidade para a inclusão de um mecanismo de reputação baseado em agentes que atribui um grau de reputação uns aos outros, afetando o protocolo de votação. Neste caso, um valor é associado a cada resultado de um processo de votação (para refletir o sucesso ou fracasso de uma escolha, ou seja, o nível de confiança de uma solução). Uma avaliação mais aprofundada destes valores pode ser usada para melhorar os resultados e decisões futuras.

Visando aprimorar o mecanismo de tomada de decisão, em [Pasin *et al*. 2014], avaliamos a decisão de adaptação da configuração, em relação ao número de servidores ativos, usando diferentes métricas (tempo de resposta, ocupação de CPU, ocupação de memória, número de *threads* ativas e número de *threads* na fila). As métricas são avaliadas com o uso de *Social Choice Functions (SCF)*. Basicamente, as preferências individuais são combinadas em uma única preferência global. Essa agregação pode ser realizada de forma centralizada ou descentralizada. Foi observado que a descentralização permite bons resultados sem a necessidade de manter o estado global.

5. Implementação

Para avaliar a arquitetura proposta, foi construído um protótipo em linguagem Java JDK 1.6. A camada de comunicação de grupo, com suporte para comunicação confiável e detecção de falhas, é provida pela biblioteca JGroups versão 2.12. A implementação dos serviços segue uma estratégia modular, onde cada componente pode ser facilmente substituído por outra versão, sem alterar a implementação dos demais serviços. Além do serviço de detecção de falhas, a implementação atual contempla os serviços de balanceamento de carga, detecção de picos de carga e ociosidade e reconfiguração, e uma primeira versão do serviço para a resolução de conflitos baseado em votação. Esta implementação usa suporte à comunicação de grupo para tolerar falhas de colapso durante a execução do algoritmo distribuído.

Para não sobrecarregar o desempenho do sistema de auto-gerenciamento, apenas a taxa local e a taxa global da ocupação da CPU são consideradas para o monitoramento de serviços na implementação atual. A grosso modo, se a carga global da CPU atinge determinado limite superior, um novo servidor é adicionado à infraestrutura. Se a carga global do sistema atinge determinado limite inferior, um dos servidores é escolhido ao acaso para ser desativado. Quando um servidor é avisado da sua desativação, ele finaliza o serviço em execução e abandona o grupo (isto é, não recebe mais requisições para execução de serviço). A taxa de ocupação local e global da CPU é computada através de séries temporais (uma sequencia de valores são medidos em momentos sucessivos e amostrados usando um intervalo de tempo uniforme). Isso significa que a carga da CPU atual individual de cada nodo é considerada para a obtenção de um valor de taxa de ocupação de CPU global, que agrega um conjunto de valores de carga de CPU previamente medidos. Com este mecanismo, uma decisão não é tomada apenas com informações atuais, mas com o apoio de estados anteriores.

6. Avaliação experimental

O sistema proposto foi avaliado através de experimentação. O primeiro experimento demonstra a reconfiguração e o segundo considera o serviço de detecção de picos de carga e ociosidade. Ambos experimentos foram realizados em cenário livre de falhas, dado que o serviço de transferência de estado, necessário para a recuperação pós-falha ainda não foi totalmente concluído. O tratamento de falhas, implementado com o auxílio do JGroups, não foi avaliado dado que o serviço de recuperação do servidor ainda não foi implementado.

Para conduzir os experimentos foram utilizados um conjunto de três computadores convencionais executando uma distribuição Ubuntu/Linux com processador 3.20 GHz *dual-core*, com 2G de memória principal, conectados em um

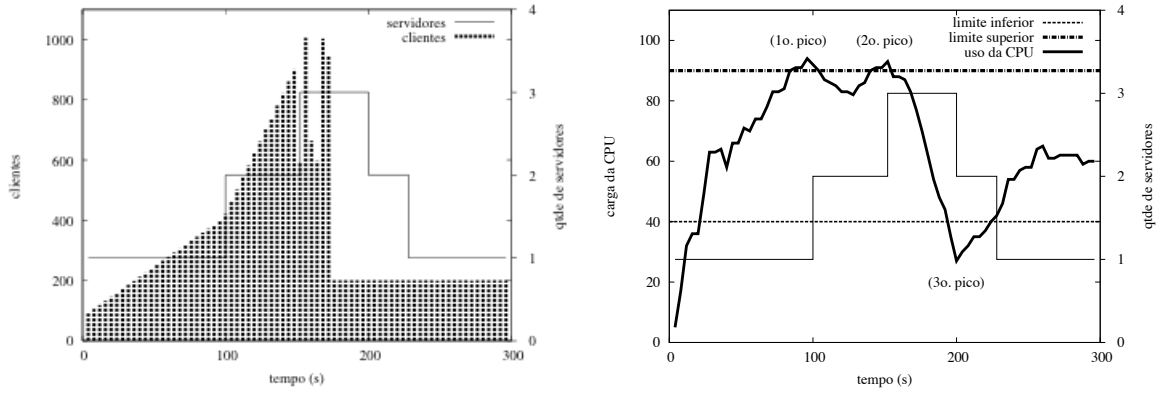
barramento 100 Mbps Fast Ethernet. Cada uma dessas máquinas pode executar no máximo uma instância de um servidor de aplicação e banco de dados associado.

Experimentos foram realizados em um ambiente controlado, usando o banco de dados PostGreSQL versão 8.3.4 para armazenamento das tabelas do sistema de informação. Para forçar sobrecarga na aplicação-alvo, servidores de banco de dados são bombardeados com dois tipos de requisições emitidas por clientes: escritas e leituras. As escritas simulam atualizações de dados provenientes dos sensores instalados nos ônibus, e as leituras simulam as consultas emitidas pelos clientes do sistema (usuários do sistema de transporte público). Inicialmente, a proporção entre escritas e leituras é de 50%. Para simular períodos de ociosidade, a quantidade de requisições clientes-servidores é reduzida.

Como a implementação atual da aplicação-alvo (ATIS) usa apenas um único servidor, e a infraestrutura autonômica atua sobre diferentes servidores (replicados), o ATIS foi simulado para operar sobre a infraestrutura distribuída. Transações do tipo *read only* que não alteram o banco de dados são resolvidas em um único nodo e transações de escrita-cega são processadas em todos os nodos da infraestrutura através do suporte à comunicação de grupo. Nenhum protocolo de consistência, a nível de aplicação, é usado para processar escritas nos servidores replicados

Experimento 1: reconfiguração devido à mudança de carga no sistema. O cenário inicial para este experimento possui um único servidor executando em uma máquina, recebendo requisições de clientes. A carga de requisições é aumentada para sobrecarregar a configuração inicial.

Quando a ocupação da CPU nos nodos operacionais atinge o limite máximo de 90% de ocupação (com 400 clientes), ocorre a inicialização automática de um servidor adicional. A tarefa de inicialização de servidor se repete (em 900 clientes) até que o número máximo de nodos da infraestrutura é atingido. Analogamente, se a carga de requisições diminuir (no experimento foi reduzida e mantida a 200 clientes ativos), e a ocupação da CPU reduz e quando atinge 40%, servidores são desalocados deste sistema. A Figura 3 ilustra a execução deste experimento através de gráficos mostrando requisições de clientes (a) e taxa de ocupação da CPU em servidores (b), respectivamente.



(a) Clientes ativos

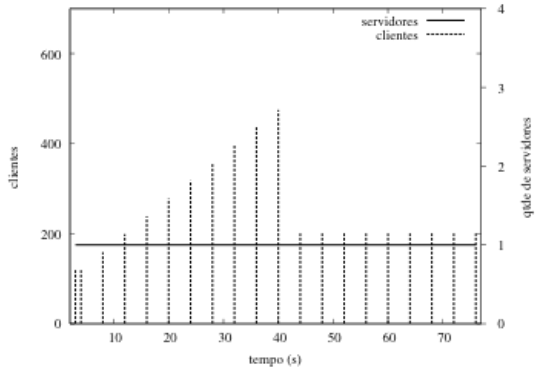
(b) Taxa de ocupação da CPU em servidores

Figura 3 - Reconfiguração de servidores devido à mudança de carga

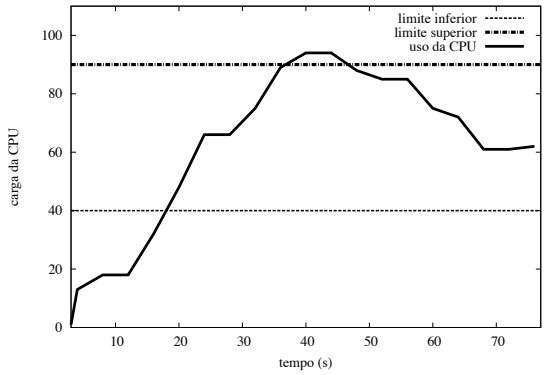
Experimento 2: tolerância em função de picos de carga e ociosidade. O cenário inicial apresenta um servidor alocado em uma máquina executando requisições emitidas por clientes. A carga de requisições é aumentada para sobrecarregar o único servidor ativo e depois é restabelecida abruptamente, sem ocorrer inicialização automática de novo servidor. A Figura 4 ilustra este cenário em gráficos mostrando requisições de clientes (a) e taxa de ocupação da CPU em servidores (b), respectivamente.

Como a oscilação de carga foi abrupta (observe que os valores do eixo tempo (s) é mais compacto na Figura 4 do que na Figura 3), ao contrário do cenário anterior (onde ocorreu ativação e desativação de servidor), no cenário da Figura 4, o número de servidores permaneceu estável. Apesar da quantidade crescente de requisições neste cenário, devido à redução abrupta de demanda, apenas uma unidade foi necessária para a manutenção do serviço.

Desta forma, os experimentos mostraram que mudanças de pico abruptas são bem toleradas pela manutenção de quantidade de servidores operacionais. Entretanto, mudanças mais significativas necessitam da remoção ou da adição de novos servidores para a manutenção de QoS, comprovando as hipóteses de trabalho.



(a) Quantidade de clientes ativos



(b) Taxa de ocupação da CPU para um servidor

Figura 4 - Tolerância em função de picos de carga e ociosidade

7. Conclusões

Aplicações executadas via *Web* estão se tornando cada vez mais populares e acessíveis. Este tipo de aplicação normalmente atua sobre uma infraestrutura distribuída ou replicada em *clusters* para permitir a disponibilidade e escalabilidade. Gestão de tais infraestruturas é tarefa complexa e onerosa porque serviços estão sujeitos à demanda elástica, falhas e picos de carga. Acredita-se que computação autonômica possa ser usada para aliviar a complexidade da gestão de sistemas computacionais, adaptando o serviço de acordo com a demanda.

Apesar da crescente atenção que sistemas de gerenciamento autonômico têm recebido nos últimos anos, sua aplicação prática continua sendo um grande desafio. Configuração, instalação e manutenção manual são tarefas recorrentes. Além disso, muitas implementações atuais de infraestruturas distribuídas ainda não conseguem manter disponibilidade *contínua* nem oferecem serviço eficiente em presença de

demanda elástica. Automação, seja total ou pelo menos parcial, ainda é um objetivo a ser alcançado.

De fato, a construção de sistemas de auto-gerenciamento não é trivial. Sistemas atuais são caracterizados por (*i*) demandas escalares, (*ii*) dinamismo, (*iii*) infraestrutura heterogênea e (*iv*) distribuição, dificultando a tarefa de gerenciamento automático. Há a necessidade de intensificar o desenvolvimento de técnicas de análise de eventos e ferramentas que possibilitem tratar de forma mais eficaz sistemas distribuídos com demandas de grandes quantidades de dados e eventos.

Neste trabalho, foi proposto um sistema de auto-gerenciamento focado nestas questões. Suas principais características são (*i*) a prestação de serviços relacionados com aplicações sujeitas a demandas elásticas e mudanças abruptas de demanda, (*ii*) a implementação de serviços através do uso de algoritmos leves visando não sobrecarregar o sistema, (*iii*) independência de falha e (*iv*) ausência de gargalos centralizados no sistema de auto-gestão. Em adição, resolução de conflitos é necessária dado que serviços são independentes e podem ser replicados. O sistema de auto-gerenciamento proposto considera um mecanismo de séries temporais para aplicar a correlação de eventos e evitar a ativação ou desativação desnecessária dos servidores da infraestrutura.

8. Trabalhos futuros

Trabalhos futuros incluem finalizar serviços ainda não implementados, aprimorar serviços já implementados e realizar novos testes usando outros cenários. Por exemplo, o serviço de transferência de estado ainda precisa ser implementado, bem como um mecanismo que decida quando este serviço deve ser executado. Transferência de estado pode envolver cópia parcial ou cópia completa de uma base de dados. No esquema de cópia parcial apenas a informação desatualizada é transferida entre servidores. Um mecanismo para agilizar o processo de transferência de estado é realizar *backup* a frio, para assegurar que a versão dos servidores desativados não seja excessivamente defasada em relação aos servidores ativos. Outro ponto importante é a definição de uma janela de tempo para a realização deste serviço no intervalo adequado. A definição desta janela de tempo não é tarefa trivial pois depende de estimativas como, por exemplo, quantidade de dados a serem transferidos e taxa de transferência de dados.

Uma implementação inicial do serviço de resolução de conflitos foi concluída, mas será aprimorada em versão futura. Foi implementado um protocolo de eleição distribuída que considera informações de reputação com suporte à comunicação de grupo. O objetivo neste trabalho, quanto à solução para o problema de conflitos, não é apenas o uso de técnicas tradicionais já propostas, mas também explorar técnicas mais sofisticadas baseadas em agentes (ver [Pasin *et al*. 2014]). Negociação entre agentes pode ser implementada em uma variedade de formas [Capra *et al*. 2002, Fabregues *et al*. 2010], mas sua aplicação em sistemas distribuídos e dinâmicos ainda não é muito explorada. Procedimentos mais sofisticados também poderão ser implementados depois de avaliar o impacto na comunicação.

Outra solução mais sofisticada, considerando melhorias para o esquema de séries temporais, é avaliar o comportamento do sistema durante um período mais longo de tempo, analisando, assim, uma quantidade maior de dados históricos. Com esta informação, a adição de um novo servidor no sistema pode ser antecipada devido à

sobrecarga abrupta (como o fim de uma partida de futebol ou a hora do *rush*), por exemplo, quando usuários potencialmente podem sobrecarregar o sistema com inúmeros pedidos. O serviço de auto-gerenciamento será capaz de prever estes cenários para garantir que o ATIS continue operando de forma eficiente.

Apontamentos

Este texto é uma versão estendida do trabalho apresentado no VIII Simpósio Brasileiro de Sistemas de Informação (SBSI 2012), São Paulo – SP, intitulado "Self-management as support to an advanced traveler information system". Esta pesquisa foi apoiada pelos projetos PRONEX FAPERGS/CNPq RS-SOC - Rede Estadual de Simulação Social número 10/0049-7 e SIMTUR RNP CTIC - Cidades Inteligentes 2012.

Bibliografia

Bernstein, P. A. e Goodman, N. (1981). Concurrency control in distributed database systems. ACM Computing Surveys. Vol. 13, No. 2, pp. 185-221.

Bastos, R. e Jaques, P. (2010). ANTARES: Um sistema *Web* de consulta de rotas de ônibus como serviço público. Revista Brasileira de Computação Aplicada, Vol. 2, pp. 41-56.

Bertini, L., Leite, J. C. B. e Mosse, D. (2007). Statistical QoS guarantee and energy-efficiency in Web server clusters. In: Real-time Systems. ECRTS '07. 19th Euromicro Conference on, pp. 83-92.

Birman, K. P. (1993). The process group approach to reliable distributed computing. In: Communications of the ACM (CACM) 16:12. Dezembro.

Birman, K. P., Ganesh, L. e van Renesse, R. (2011). Running smart grid control software on cloud computing architectures. In: Workshop on Computational Needs for the Next Generation Electric Grid, Cornell University, Abril 19-20. Ithaca, NY.

Bouchenak, S., De Palma, N., Hagimont, D. e Taton, C. (2006). Autonomic management of clustered applications. In: Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2006), Barcelona, Espanha.

Capra, L., Emmerich, W. e Mascolo, C. (2002). A Micro-economic Approach to conflict resolution in mobile computing. In: Proceedings of the 10th International Symposium on the Foundations of Software Engineering (FSE-10), pp. 31-40, ACM Press.

Fabregues, A. e Sierra, C. (2010). An agent architecture for simultaneous bilateral negotiations. In: Proceedings of the 8th European Workshop on Multi-agent Systems (EUMAS 2010), Paris, França.

Kephart, J. O. e Chess, D. M. (2003). The vision of autonomic computing, In: IEEE Computer, Vol. 36. pp. 41-50.

Pasin, M., Perini, F. S. e Bazzan, A. L. C. (2011). Towards a self-managed distributed infrastructure to an advanced traveler information system. In: Anais do Autosoft - II Congresso Brasileiro de Software: Teoria e Prática (CBSoft 2011). São Paulo, Brasil. pp. 33-39.

Pasin, M., Bazzan, A. L. C. e Mendoza, M. R. (2014). Supporting self-managed infrastructures: a classification-based approach using social choice functions. In:

Proceedings of the 10th International Workshop on Agents and Data Mining Interaction (ADMI-14), Paris, França.

Roblitz, T. *et al*. (2004). Autonomic management of large clusters and their integration into the grid. In: Journal of Grid Computing - GRID, Vol. 2, No. 3, pp. 247-260.

Sallem, S. K. e Chen, S.-C. (2008). Towards a self-configurable weather research and forecasting system. In: Proceedings of the 5th IEEE International Conference on Autonomic Computing (ICAC 2008), pp. 195-196.

Sant'Ana, C. H., Leite, J. C. B. e Mossé, D. (2010). Previsão de carga para economia de energia em aglomerados de servidores *Web*. In: Anais do XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2010), Gramado - RS, Brasil.

Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P. e Wood, T. (2008). Agile dynamic provisioning of multi-tier Internet applications. In: ACM Transactions on Autonomous and Adaptive Systems, Vol. 3, No. 1, Article 1 (Março 2008), 39 p.

Wang, Y., Wang, X., Chen, M. e Zhu, X. (2008). Power-efficient response time guarantees for virtualized enterprise servers. In: IEEE Real-Time Systems Symposium, pp. 303-312, Barcelona, Espanha.

Xu, W., Qi, Y. e Hou, D. (2010). Multi-dimensional time series-based application server aging model. In: Proceedings of 2010 International Conference on Wavelet Analysis and Pattern Recognition (ICWAPR), pp. 238-243.