UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE CIÊNCIAS EXATAS E DA TECNOLOGIA
CIÊNCIA DA COMPUTAÇÃO

BRUNO EMER

Implementação de alta disponibilidade em uma empresa prestadora de serviços para Internet

André Luis Martinotto
Orientador

Caxias do Sul
Abril de 2016

Implementação de alta disponibilidade em uma empresa prestadora de serviços para Internet

por

Bruno Emer

Projeto de Diplomação submetido ao curso de Bacharelado em Ciência da Computação do Centro de Ciências Exatas e da Tecnologia da Universidade de Caxias do Sul, como requisito obrigatório para graduação.

Projeto de Diplomação

Orientador: André Luis Martinotto
Banca examinadora:

Maria de Fatima Webber do Prado Lima
CCTI/UCS
Ricardo Vargas Dorneles
CCTI/UCS

Projeto de Diplomação apresentado em
x de x de 2016

Daniel Luís Notari
Coordenador

SUMÁRIO

LISTA DE SIGLAS

ECC *error correction code*
MTBF *mean time between failures*
MTTR *mean time to repair*
RAID *redundant array of independent disks*
SLA *service level agreement*
SPOF *single point of failure*
TI tecnologia da informação
VM *virtual machine*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Palavras-chave: .

1 INTRODUÇÃO

O crescente avanço tecnológico e o desenvolvimento da internet, provocou um aumento no número de aplicações ou serviços que dependem da infraestrutura de tecnologia da informação (TI). Além disso, percebe-se um aumento significativo no número de operações e negócios *on-line* que são realizados, tanto por organizações públicas ou privadas, quanto por grande parte da sociedade.

Desta forma, a sociedade está cada vez mais dependente da tecnologia, de computadores e de sistemas. De fato, pode-se observar sistemas computacionais desde em uma farmácia, até em uma grande indústria. Sendo assim, a estabilidade e disponibilidade destes sistemas tem uma grande importância em nosso dia-a-dia, pois um número significante de atividades cotidianas dependem deles.

Uma interrupção imprevista em um ambiente computacional poderá causar um prejuízo financeiro para a empresa que fornece o serviço, além de interferir na vida de pessoas que dependem de forma direta ou indireta deste serviço. Essa interrupção terá maior relevância para corporações cujo o serviço ou produto final é fornecido através da internet, como por exemplo, comércio eletrônico, *web sites*, sistemas corporativos, entre outros. Em um ambiente extremo, pode-se imaginar o caos e o possível risco de perda de vidas que ocorreria em caso de uma falha em um sistema de controle aéreo (COSTA, 2009).

Para essas empresas um plano de contingência é fundamental para garantir uma boa qualidade de serviço, além de otimizar o desempenho das atividades, e também para fazer uma prevenção de falhas e uma recuperação rápida caso essas ocorram (COSTA, 2009). De fato, hoje em dia a confiança em um serviço ou em um sistema é um grande diferencial para a empresa fornecedora deste serviço, sendo que a alta disponibilidade é fundamental para atingir esse objetivo.

A alta disponibilidade consiste em manter um sistema disponível por meio da tolerância a falhas, isto é, utilizando mecanismos que fazem a detecção, mascaramento e a recuperação de falhas, sendo que esses mecanismos podem ser implementados a nível de *software* ou de *hardware* (REIS, 2009). Para que um sistema seja altamente disponível ele deve ser tolerante a falhas, sendo que a tolerância a falhas é implementada frequentemente utilizando redundância. No caso de uma falha em um dos componentes evita-se a interrupção do sistema, uma vez que a redundância é feita através da replicação de componentes (BATISTA, 2007).

Neste trabalho será realizado um estudo sobre a implementação de um sistema de alta disponibilidade em uma empresa de hospedagens. Essa empresa oferece serviços pela internet, como por exemplo hospedagens de sites, *e-mail*, sistemas de gestão, *e-mail marketing*, entre outros. A empresa possui aproximadamente 55 servidores físicos e virtuais, e aproximadamente 9000 clientes, sendo que em períodos de pico

atende em torno de 1000 requisições por segundo.

Atualmente, a empresa possui redundância de conexões de acesso a internet, refrigeração e energia, com *nobreaks* e geradores. Porém, essa empresa não possui nenhuma redundância nos serviços que estão sendo executados nos servidores. Desta forma, caso ocorra uma falha de *software* ou *hardware*, os serviços ficarão indisponíveis. Neste trabalho será realizada uma análise dos serviços oferecidos pela empresa, sendo que mecanismos de alta disponibilidade serão desenvolvidos para os serviços mais críticos. Para a redução dos custos serão utilizadas ferramentas gratuitas e de código aberto.

1.1 Objetivos

Atualmente a empresa não possui nenhuma solução de alta disponibilidade para seus serviços críticos. Desta forma, neste trabalho será desenvolvida uma solução de alta disponibilidade para estes serviços, sendo que essa solução será baseada no uso de ferramentas de código aberto e de baixo custo.

Para que o objetivo geral seja atendido os seguintes objetivos específicos deverão ser realizados:

- Identificar os serviços críticos a serem integrados ao ambiente de alta disponibilidade;
- Definir as ferramentas a serem utilizadas para implementar tolerância a falhas;
- Realizar testes para a validação do sistema de alta disponibilidade que foi desenvolvido.

2 ALTA DISPONIBILIDADE

Alta disponibilidade é muito conhecida, sendo cada vez mais empregada nos ambientes computacionais. O objetivo de promover alta disponibilidade resume-se em garantir que um serviço esteja sempre a disposição quando o cliente solicitar ou acessar (COSTA, 2009). A alta disponibilidade é definida como a redundância de *hardware* ou *software* para que o serviço fique mais tempo disponível. Quanto maior for a disponibilidade desejada maior deverá ser a redundância no ambiente, assim reduzindo os pontos únicos de falha, que em inglês são chamados de *single point of failure* (SPOF).

A alta disponibilidade está diretamente relacionada a dependabilidade, confiabilidade, disponibilidade e tolerância a falhas.

- Dependabilidade indica a qualidade do serviço fornecido e a confiança depositada nele. A dependabilidade envolve vários atributos como segunrança de funcionamento, segurança, manutenabilidade, testabilidade e comprometimento do desempenho (WEBER, 2002);
- Confiabilidade é o mais importante atributo, pois transmite a ideia de continuidade de serviço (PANKAJ, 1994). A confiabilidade refere-se a probabilidade de um serviço funcionar corretamente durante um dado intervalo de tempo;
- Disponibilidade é a probabilidade de um serviço estar operacional no instante em que for solicitado (COSTA, 2009);
- Tolerância a falhas tenta garantir a disponibilidade de um serviço utilizando mecanismos capazes de detectar, mascarar e recuperar falhas, e seu objetivo é alcançar a dependabilidade, assim indicando uma boa qualidade de serviço (COSTA, 2009).

Uma das principais palavras-chave da alta disponibilidade é a tolerância a falhas, que será melhor discutida na seção 2.1.

2.1 Tolerância a falhas

Sabe-se que o *hardware* tende a falhar devido a fatores físicos, por isso utiliza-se métodos como prevenção de falhas e tolerância a falhas. A abordagem prevenção de falhas melhora a disponibilidade e a confiabilidade de um serviço, isto é, tem como objetivo prever e eliminar o maior número de falhas possíveis antes de colocar o sistema em uso. Mas essa prevenção não resolverá todas as possíveis falhas. Sendo assim, a tolerância a falhas fornece disponibilidade de um serviço mesmo com presença de falhas. Enquanto a prevenção de falhas tem foco em projeto, teste e

validação, a tolerância a falhas tem como foco usar componentes para mascarar as falhas (PANKAJ, 1994).

O objetivo da tolerância a falhas é aumentar a disponibilidade de um sistema, isto é, aumentar o tempo que os serviços fornecidos aos clientes ou usuários ficam disponíveis. Um sistema é dito tolerante a falhas se ele pode mascarar a presença de falhas ou recuperar-se de uma falha sendo que frenquentemente a tolerância a falhas é implementada utilizando redundância que será detalhada na seção 2.2. Um exemplo bastante utilizando atualmente é tolerância a falhas em servidores de virtualização, onde normalmente existem dois servidores com os seus dados replicados. Caso um dos servidores falhe por algum motivo um *software* de monitoramento fará a transferência das maquinas virtuais para o outro servidor evitando assim a indisponibilidade do serviço. Este tema, virtualização, será detalhado no capítulo 3.

A tolerância a falhas pode ser dividida em duas classes que são o mascaramento e a detecção, localização e reconfiguração. Na primeira classe o mascaramento trata as falhas na origem e manifesta-se na forma de erro. Um exemplo são os códigos de correção de erros, em inglês *error correction code* (ECC) que são utilizados em memórias para detecção e correção de erros. Na segunda, geralmente necessita de menor redundância, e consiste em detectar, localizar e reconfigurar o *software* ou *hardware* e por fim resolver a falha (WEBER, 2002).

2.1.1 Fases da tolerância a falhas

A classificação das fases de tolerância a falhas mais comuns são detecção, confinamento, recuperação e tratamento. Essas fases excluem o mascaramento de falhas (WEBER, 2002).

- Detecção: a detecção de erro faz o monitoramento e aguarda uma falha se manifestar na forma de erro, para então passar para a próxima fase. Um exemplo de detecção de erro é o cão de guarda (*watchdog timer*), ele recebe um sinal do programa ou serviço monitorado e caso este sinal não seja recebido, devido alguma falha, o *watchdog* irá se manifestar na forma de erro. Outro exemplo é o esquema de duplicação e comparação, onde é feita replicação do componente que realiza as operações sobre os mesmos dados e compara os resultados de saída. Caso ocorra alguma diferença nos resultados um erro é gerado.
- Confinamento: esta fase é responsável pela restrição de um erro para que esse não se propague para todo o sistema, pois entre a falha e a detecção do erro há um intervalo de tempo. Neste intervalo pode ocorrer a propagação do erro para outros componentes do sistema, por isso antes de executar medidas corretivas é necessário definir os limites da propagação.
- Recuperação: após a detecção de um erro ocorre a recuperação, onde o estado de erro é alterado para estado livre de erros. A recuperação pode ser feita de duas formas:

  - *forward error recovery* ou recuperação por avanço: ocorre uma condução para um novo estado não ocorrido anteriormente. É o mais eficiente, porém complexo de ser implementado pois a ação deve ser precisa.
  - *backward error recovery* ou recuperação por retorno: ocorre um retorno para um estado anterior que deve estar livre de erros. Para retornar ao

estado anterior pode ser utilizado pontos de recuperação (*checkpoints*), e quando ocorrer um erro, um *rollback* é executado, assim retornará a um *checkpoint* anterior.

- Tratamento: a última fase serve para prevenir que futuros erros aconteçam. Nesta fase ocorre a localização da falha para descobrir o componente que originou a falha e após é feito um diagnóstico sendo que a remoção do componente danificado pode ser feita de duas formas, manual ou automática. O reparo manual é feito por um operador, e o automático quando existe um componente em espera para substituição. Exemplo de um reparo manual é um operador fazer a troca de um disco de um servidor. E um exemplo de reparo automático é um disco configurado como *hot spare*, que de acordo com (ROUSE, 2013) *hot spare* é definido como um componete de *backup* que assumirá o lugar de outro imediatamente após o componente principal falhar. Em *storages* ou servidores a *hot spare* pode ser configurada em uma *redundant array of independent disks* (RAID) juntamente com o *array* de discos.

2.2 Redundância

Redundância pode ser feita através da replicação de componentes, para reduzir o número de SPOF e garantir o mascaramento de falhas. Na prática, se um componente falhar ele deve ser reparado ou substituído por um novo, sem que haja uma interrupção no serviço. A redundância também pode ser através do envio de sinais ou *bits* de controle junto aos dados, servindo assim para detecção de erros e até para correção (WEBER, 2002).

Segundo (NøRVåG, 2000) existem quatro tipos diferentes de redundância que são:

- *Hardware*: utiliza-se a replicação de componentes, sendo que caso um falhe outro possa assumir seu lugar. Para fazer a detecção de erros a saída de cada componente é constantemente monitorada e comparada à saída de outros componentes. Um exemplo prático de redundância de *hardware* é servidores com fontes redundantes, normalmente são duas fontes ligadas em paralelo, caso uma falhe a outra suprirá a necessidade de todo o servidor;
- Informação: ocorre quando uma informação extra é enviada ou armazenada para possibilitar a detecção e correção de erros. Um exemplo clássico são os *checksums* (soma de verificação). Esses são calculados antes da transmissão ou armazenamento e recalculado ao recebê-los ou recuperá-los, assim sendo possível verificar a integridade dos dados. Outro exemplo bastante comum são os *bits* de paridade que são utilizandos para detectar falhas simples, que afetam apenas um *bit* (WEBER, 2002);
- *Software*: redundância de *software* não é muito útil pois replicando um *software* as falhas ou *bugs* estarão presentes em todas replicas. Existem algumas técnicas que podem ajudar com esse problema. A programação de n-versões é uma delas, esta técnica consiste em criar n versões para um mesmo *software* assim possibilitando o aumento da disponibilidade pois elas provavelmente não apresentarão os mesmos erros. Por outro lado a programação de n-versões possui um custo muito elevado devido a complexidade da sua manutenção.

Tabela 2.1: Níveis de alta disponibilidade e exemplos de sistemas

Nível	Uptime	Downtime por ano	Exemplos
1	90%	36.5 dias	computadores pessoais
2	98%	7.3 dias	
3	99%	3.65 dias	sistemas de acesso
4	99.8%	17 horas e 30 minutos	
5	99.9%	8 horas e 45 minutos	provedores de acesso
6	99.99%	52.5 minutos	CPD, sistemas de negócios
7	99.999%	5.25 minutos	sistemas de telefonia ou bancários
8	99.9999%	31.5 minutos	sistemas de defesa militar

- Tempo: este é feito através da execução de instruções várias vezes no mesmo componente, assim detectando falha caso ocorra. Essa técnica necessita tempo adicional, e é utilizado onde o tempo não é crítico. Por exemplo um *software* de monitoramento de serviços e servidores, ele faz o teste de cada serviço e caso ocorra uma falha o *software* poderá executar uma ação corretiva para reestabelecer o serviço. Diferentemente de redundância de *hardware* e informação ela não requer um *hardware* extra para sua implementação (COSTA, 2009).

2.3 Cálculo da alta disponibilidade

Um ponto importante sobre alta diponibilidade é como medi-la. Para isso são utilizados os valores de *uptime* e *downtime*, que são respectivamente o tempo que os serviços estão funcionando normalmente e o tempo que não estão funcionando. A alta disponibilidade pode ser expressa pela quantidade de "noves", isto é, se um serviço possui quatro noves de disponibilidade este possui uma disponibilidade de 99,99% (PEREIRA FILHO, 2004).

A Tabela 2.1 possui alguns níveis de disponibilidade enumerados, seguido da porcentagem do *Uptime*, o *Downtime* por ano representado na medida de tempo e na última coluna possui alguns exemplos de serviços relacionados ao nível de disponibilidade. Pode-se observar que para alguns serviços como sistemas bancários ou sistemas militares é necessário um alto nível de disponibilidade.

A alta diponibilidade pode ser calculada através da equação

$$d = \frac{MTBF}{(MTBF + MTTR)} \tag{2.1}$$

onde d é a porcentagem de disponibilidade. O *mean time between failures* (MTBF) é o tempo médio entre falhas, correspondendo ao tempo médio entre as paradas de um determinado serviço. Já o *mean time to repair* (MTTR) é o tempo médio de recuperação, isto é, o tempo entre a queda e a recuperação de um serviço (GONçALVES, 2009).

A alta disponibilidade é um dos principais fatores que fornece dependabilidade dos clientes ou usuários, sendo extremante importante em empresas que fornecem serviços *on-line*. Por isso existe o *service level agreement* (SLA), que é um acordo de nível de serviço, que garante que o serviço fornecido atenda as expectativas dos

clientes. Um SLA é um documento contendo uma descrição e uma definição das características mais importantes do serviço que será fornecido. Esse acordo também deve conter o nível de serviço exigido pelo negócio, sendo que este deve ser minuciosamente definido. Por exemplo, um SLA pode conter descrição do serviço, requerimentos, horário de funcionamento, disponibilidade esperada, entre outros (SMITH, 2010).

3 VIRTUALIZAÇÃO

O conceito virtualização surgiu na década de 60, sendo que um dos principais motivos foi a necessidade de um grande servidor, conhecido como *mainframe*, executar uma variedade de *softwares*. Isso ocorreu pois cada *mainframe* necessitava do próprio sistema operacional, pois cada *software* possuia além da aplicação todo o ambiente operacional no qual executava. Assim sendo necessário a criação de máquinas virtuais, mais conhecida como *virtual machine* (VM) (CARISSIMI, 2008).

A virtualização é definida como uma camada entre o *hardware* e o sistema operacional que possibilita a divisão e proteção dos recursos físicos. Virtualização utiliza abstração em sua arquitetura, por exemplo, ela transforma um disco físico em dois discos virtuais menores, sendo que esses discos virtuais são arquivos armazenados no disco físico. Sabendo que arquivos são uma abstração em um disco físico, pode-se dizer que virtualização não é apenas uma camada de abstração do *hardware*, ela faz a emulação do *hardware* (SMITH; NAIR, 2005).

Em muitos casos empresas utilizam serviços distribuídos entre servidores físicos, como por exemplo servidores de e-mail, hospedagens e banco de dados. Com isso existe uma ociosidade grande de recursos. Uma das grandes vantagens da virtualização é o melhor aproveitamento destes recursos, alocando vários serviços em uma única máquina gera um melhor aproveitamento do *hardware* (MOREIRA, 2006). Além disso pode-se ter uma redução de custos com administração e manutenção dos servidores. Em um ambiente heterogênio pode-se também utilizar virtualização, pois ela permite a instalação de diversos sistemas operacionais em um único servidor.

Atualmente o elevado custo da energia elétrica realça a necessidade de redução do consumo, que pode ser feito através da implantação de servidores mais robustos para substituir dezenas de servidores comuns, assim reduzindo o consumo de energia elétrica. Outros fatores como refrigeração do ambiente e espaço físico utilizado também podem ser reduzidos com a implantação de virtualização de servidores, e consequentemente reduzem os custos de energia também.

A virtualização favorece a implementação do conceito um servidor por serviço, que consiste em ter um servidor para cada serviço. Mas porque não colocar todos serviços em um único servidor? Muitas vezes com uma variedade de serviços é necessário diferentes sistemas operacionais, ou os serviços necessitam rodar nas mesmas portas, portanto isto se torna inviável. Outro fator relevante que também favorece a implementação de um servidor por serviço é, caso exista uma falha de segurança em apenas um serviço, essa vulnerabilidade poderá comprometer todos os outros serviços (CARISSIMI, 2008).

3.1 Funcionamento

3.2 Tipos de virtualização

REFERÊNCIAS

BATISTA, A. C. **Estudo teórico sobre cluster linux**. 2007. Pós-Graduação(Administração em Redes Linux) — Universidade Federal de Lavras, Minas Gerais.

CARISSIMI, A. Virtualização: da teoria a soluções. In: **Minicursos do Simpósio Brasileiro de Redes de Computadores**. Porto Alegre: XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2008.

COSTA, H. L. A. **Alta disponibilidade e balanceamento de carga para melhoria de sistemas computacionais críticos usando software livre**: um estudo de caso. 2009. Pós-Graduação em Ciência da Computação — Universidade Federal de Viçosa, Minas Gerais.

GONçALVES, E. M. **Implementação de Alta disponibilidade em máquinas virtuais utilizando Software Livre**. 2009. Trabalho de Conclusão (Curso de Engenharia da Computação) — Faculdade de Tecnologia e Ciências Sociais Aplicadas, Brasilia.

MOREIRA, D. **Virtualização**: rode vários sistemas operacionais na mesma máquina. <Disponível em: `http://idgnow.com.br/ti-corporativa/2006/08/01/idgnoticia.2006-07-31.7918579158/#&panel1-3`>. Acesso em 5 de abril de 2016.

NøRVåG, K. **An Introduction to Fault-Tolerant Systems**. 2000. IDI Technical Report 6/99 — Norwegian University of Science and Technology, Trondheim, Noruega.

PANKAJ, J. **Fault tolerance in distributed system**. Nova Jérsei, Estados Unidos: P T R Prentice Hall, 1994.

PEREIRA FILHO, N. A. **Serviço de pertinência para clusters de alta disponibilidade**. 2004. Dissertação para Mestrado em Ciência da Computação — Universidade de São Paulo, São Paulo.

REIS, W. S. dos. **Virtualização de serviços baseado em contêineres**: uma proposta para alta disponibilidade de serviços em redes linux de pequeno porte. 2009. Monografia Pós-Graduação(Administração em Redes Linux) — Apresentada ao Departamento de Ciência da Computação, Minas Gerais.

ROUSE, M. **Hot spare**. <Disponível em: http://searchstorage.techtarget.com/definition/hot-spare>. Acesso em 12 de abril de 2016.

SMITH, J. E.; NAIR, R. The architecture of virtual machines. **IEEE Computer**, [S.l.], v.38, p.32–38, 2005.

SMITH, R. **Gerenciamento de Nível de Serviço**. <Disponível em: http://blogs.technet.com/b/ronaldosjr/archive/2010/05/25/gerenciamento-de-n-237-vel-de-servi-231-o.aspx/>. Acesso em 25 de março de 2016.

WEBER, T. S. **Um roteiro para exploração dos conceitos básicos de tolerância a falhas**. 2002. Curso de Especialização em Redes e Sistemas Distribuídos — UFRGS, Rio Grande do Sul.