Universidade Federal de Minas Gerias
Instituto de Ciências Exatas
Departamento de Ciências da Computação

Leandro de Sousa Nunes

Desenvolvimento de Sistemas Distribuídos Descentralizados Utilizando Distributed Hash Tables

Belo Horizonte, MG

17 de abril de 2012

Universidade Federal de Minas Gerias
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de Software

Desenvolvimento de Sistemas Distribuídos Descentralizados Utilizando Distributed Hash Tables

por

LEANDRO DE SOUSA NUNES

Monografia de Final de Curso

Prof. Marcos André Gonçalves
Orientador

Belo Horizonte, MG

17 de abril de 2012

Leandro de Sousa Nunes

Desenvolvimento de Sistemas Distribuídos Descentralizados Utilizando Distributed Hash Tables

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Especialista em Informática.

Área de concentração: Engenharia de Software

Orientador: Prof. Marcos André Gonçalves

Belo Horizonte, MG

17 de abril de 2012

Aos meus professores,
aos meus colegas de curso,
aos meus familiares e amigos
dedico este trabalho.

Agradecimentos

Aos meus pais que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

Ao professor Dr. Marcos André Gonçalves pela paciência na orientação e incentivo que tornaram possível a conclusão desta monografia.

A minha namorada Poliana, pelo carinho e companheirismo.

A MAV Tecnologia, em especial a equipe de desenvolvimento, Daniel, Marcos, Marco Túlio e Talles, pela amizade, pelo trabalho em equipe e pela inspiração deste trabalho.

Aos meus amigos e colegas, pelo incentivo e pelo apoio constantes.

“Procure ser um homem de valor,
em vez de ser um homem de sucesso.”
Albert Einstein

Resumo

Este trabalho baseia-se na crescente importância que a computação em nuvem vem adquirindo no cenário tecnológico mundial. Computação em nuvem consiste na utilização de servidores interligados e compartilhados através da internet para armazenamento e processamento de dados de forma a otimizar a utilização desses recursos, com garantia de disponibilidade, confiabilidade e a prova de erros. Uma das técnicas mais bem sucedidas de implementação de um sistema baseado em computação em nuvem é conhecido como *Distributed Hash Table* (DHT), que é similar à uma tabela *hash*: pares (chave, valor) são armazenados na nuvem, e qualquer nó participante pode recuperar o valor associado a uma dada chave. O foco deste trabalho é realizar um estudo sobre como desenvolver um sistema focado em computação na nuvem utilizando o DHT.

Palavras-chave: Tecnologia da Informação; Computação em nuvem; DHT; segurança; Internet; provedores; sistemas distribuídos; processamento paralelo.

Abstract

This dissertation is based on the increasing importance that cloud computing has acquired in the global technological landscape. Cloud Computing is the use of interconnected and shared servers through the Internet for storage and data processing to optimize the use of these resources, with guaranteed availability, reliability and error proof. One of the most successful implementation of a system based on cloud computing is known as Distributed Hash Table (DHT), which is similar to a hash table: pairs (key, value) are stored in the cloud, and any participating node can retrieve the value associated with a given key. The focus of this work is to conduct a study on how to develop a system focused on cloud computing using the DHT.

Keywords: *Information Technology; Cloud Computing; DHT; security; Internet; servers; distributed systems; parallel processing.*

Lista de Figuras

Lista de Siglas

API *Application Programming Interface*

CA *Certification Authority*

CRL *Certificate Revocation List*

DHT *Distributed Hash Table*

DNS *Domain Name System*

IP *Internet Protocol*

IaaS *Infrastructure as a Service*

P2P Ponto-a-ponto

PC *Personal Computer*

PKI *Private Key Infrastructure*

PaaS *Plataform as a Service*

QoS *Quality of Service*

SGBD Sistema de Gerenciamento de Banco de Dados

SHA-1 *Secure Hash Algorithm 1*

SLA *Services Level Agreement*

SaaS *Software as a Service*

TI Tecnologia da Informação

XOR ou-exclusivo

Lista de Tabelas

Sumário

1 Introdução

Confiabilidade em uma grande escala é um dos grandes desafios encontrados por serviços disponibilizados pela Internet à milhares de usuário, onde qualquer erro tem significantes consequências financeiras e impacto na confiança dos consumidores. Esses serviços são implementados em uma infraestrutura de dezenas de milhares de servidores e componentes de redes alocados em vários *data centers* pelo mundo. Neste escopo, componentes, pequenos ou grandes, falham constantemente e a preocupação em manter um estado persistente dos dados a partir dessas falhas leva a necessidade de sistemas de software confiáveis e escaláveis (DECANDIA et al., 2007).

Aplicações ponto-a-ponto (P2P) são sistemas distribuídos sem um controle centralizado ou organização hierárquica, onde o software executado em cada nó é equivalente em funcionalidades. As principais necessidades das recentes aplicações ponto-a-ponto constituem uma lista longa: armazenamento redundante, estabilidade, anonimato, busca, autenticação e etc. Apesar desta lista de necessidades, a principal operação em sistemas ponto-a-ponto é a localização eficiente de um dado item (STOICA et al., 2001).

Dados os problemas acima citados, é frequente a utilização de *Distributed Hash Tables* (DHT) para gerenciar os dados através da rede em um sistema ponto-a-ponto. As DHT armazenam dados no formato de (**chave**, **valor**) e a interface é constituída pelas operações ***get***, que retorna o valor associado à chave, ***put*** que armazena a informação dada, e ***rem***, que remove o par que contém uma determinada **chave**. Desta forma, aplicações P2P baseadas em DHT são amplamente usadas para compartilhamento de arquivos e dão suporte a uma funcionalidade de busca simplista norteada pela busca de um arquivo identificado por uma determinada **chave** (RIBAS et al., 2010).

O objetivo principal deste trabalho é apresentar um estudo sobre como utilizar o DHT para o desenvolvimento de sistemas distribuídos altamente confiáveis, escaláveis e a prova de erros. Para atingir este objetivo será necessário: realizar um estudo sobre a computação em nuvem e seus princípios; realizar um estudo sobre o DHT; planejar um sistema que utilize o DHT;

consultar trabalhos semelhantes realizados para embasamento; e descrever formas de aplicar tais princípios em projetos de software.

Para elaboração deste trabalho foi realizada uma pesquisa de materiais de auxílio na elaboração de um estudo bibliográfico sobre os assuntos relacionados: computação em nuvem, DHT, sistemas distribuídos. A pesquisa foi realizada em livros e na internet. Na pesquisa realizada na internet foram buscados artigos, monografias, dissertações, teses, normas e outros.

Após a reunião do material necessário foi feito um estudo sobre o mesmo para extrair informações relevantes para compor a fundamentação teórica do trabalho. Foram estudadas definições e conceitos intrínsecos sobre os assuntos descritos anteriormente.

Este trabalho está organizado da seguinte maneira. O capítulo 2 fala sobre a computação em nuvem, seus conceitos, características, modelos de serviços e de implantação e desafios. O capítulo 3 apresenta o DHT, com uma discussão sobre um modelo de sistema e as várias maneiras de se implementar as propriedades da computação em nuvem, um exemplo de implementação e seus resultados. O capítulo 4 fala sobre os trabalhos relacionados ao DHT. E finalmente, o capítulo 5 contém a conclusão obtida com este trabalho e sugestões para trabalhos futuros.

2 *Computação na Nuvem*

2.1 Introdução

O termo computação nas nuvens (do inglês *cloud computing*) está associado a um novo paradigma na área de computação. Basicamente, esse novo paradigma tende a deslocar a localização de toda a infraestrutura computacional para a rede. Com isso, os custos de software e principalmente de hardware podem ser consideravelmente reduzidos (VAQUERO et al., 2009).

Computação em nuvem é uma tendência recente de tecnologia que tem por objetivo proporcionar serviços de tecnologia da Informação através de um *cluster* de servidores descentralizados, geralmente conectados através da Internet. Todos já utilizam ou irão utilizar o poder da nuvem, desde o usuário final que hospeda seus documentos pessoais até empresas que terceirizarão toda a parte de TI para outras empresas. O potencial de um sistema que utiliza um sistema de computação em nuvem é gigantesco: não apenas recursos de computação e armazenamento são melhor utilizados, mas toda a pilha de computação pode ser aproveitada na nuvem, com mais segurança, disponibilidade e a prova de erros. Na Figura 2.1, podemos ter uma visão geral de uma nuvem computacional.

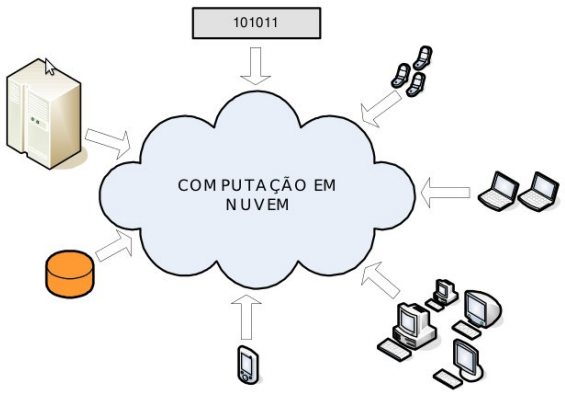

Figura 2.1: Visão geral de uma nuvem computacional (SOUSA; MOREIRA; MACHADO, 2009)

Com a computação em nuvem, os usuários estarão movendo seus dados e aplicações para a nuvem, podendo acessá-los de forma simples e de qualquer local. Isso é um caso de utilização de processamento centralizado.

Computação em nuvem é, portanto, uma maneira bastante eficiente de maximizar e flexibilizar os recursos computacionais. Além disso, uma nuvem computacional é um ambiente redundante e resiliente por natureza. Resiliente pode ser definido como a capacidade de um sistema de informação continuar a funcionar corretamente, apesar do mau funcionamento de um ou mais dos seus componentes (TAURION, 2009).

A computação em nuvem surge da necessidade de construir infraestruturas de TI complexas, onde os usuários não têm que realizar instalação, configuração e atualização de softwares. Além disso, recursos de computação e hardware são propensos a ficarem obsoletos rapidamente. Assim, a utilização de plataformas computacionais de terceiros é uma solução inteligente para os usuários lidarem com infraestrutura de TI.

2.2 Características Essenciais

As características essenciais são as vantagens obtidas quando é utilizada uma solução de computação nas nuvens. Elas são como os requisitos para um sistema baseado na computação em nuvem e o que o distingue de outros paradigmas. Abaixo veremos essas características.

2.2.1 *Self-service* sobre demanda

Um usuário pode adquirir unilateralmente recursos computacionais, como tempo de processamento ou armazenamento na rede na medida em que necessite e sem precisar de interação humana para de cada serviço. (MELL; GRANCE, 2011).

O *hardware* e o *software* dentro de uma nuvem podem ser automaticamente reconfigurados. Estas modificações são apresentadas de forma transparente para os usuários, que possuem perfis diferentes e assim podem personalizar os seus ambientes computacionais.

2.2.2 Amplo acesso à rede

Os recursos estão disponíveis através da rede e são acessados através de mecanismos genéricos que promovam o padrão utilizado por plataformas heterogêneas (como telefone celulares, *tablets*, *laptops* e PCs) (MELL; GRANCE, 2011).

A interface de acesso a nuvem não obriga os usuários a mudarem suas condições e ambientes de trabalho, como por exemplo, linguagens de programação e sistema operacional. Já os softwares clientes instalados localmente para o acesso à nuvem são leves, como um navegador de Internet.

2.2.3 *Pooling* de recursos

O recursos computacionais do provedor são agrupados para atender vários consumidores através de um modelo multi-locatário, com diferentes recursos físicos e virtuais atribuídos dinamicamente e novamente de acordo com a demanda do consumidor. Há um senso de independência local em que o cliente geralmente não tem nenhum controle ou conhecimento sobre a localização exata dos recursos disponibilizados, mas pode ser capaz de especificar o local em um nível maior de abstração (por exemplo país, estado ou *data center*). Exemplos de recursos incluem o armazenamento, processamento, memória, largura de banda de rede e máquinas virtuais (MELL; GRANCE, 2011).

2.2.4 Elasticidade Rápida

Os recursos podem ser adquiridos de forma rápida e elástica, em alguns casos automaticamente, caso haja a necessidade de escalar com o aumento da demanda, e liberados, na retração dessa demanda. Para os usuários, os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento (MELL; GRANCE, 2011).

2.2.5 Serviço Medido

Sistemas em nuvem automaticamente controlam e otimizam a utilização dos recursos, alavancando a capacidade de medição em algum nível de abstração adequado para o tipo de serviço (por exemplo armazenamento, processamento, largura de banda e contas de usuários ativos). O uso de recursos pode ser monitorado, controlado e relatado a existência de transparência para o fornecedor e o consumidor do serviço utilizado (MELL; GRANCE, 2011).

Podemos monitorar e controlar o uso de recursos, garantindo a transparência para o provedor e o usuário do serviço utilizado. Utiliza-se a abordagem baseada em nível de serviço *Services Level Agreement* (SLA) para garantir a qualidade de serviço (QoS). O SLA fornece informações sobre os níveis de disponibilidade, funcionalidade, desempenho ou outros atributos do serviço como o faturamento e até mesmo penalidades em caso de violação destes níveis.

2.3 Modelo de Serviços

A computação em nuvem é composta por três modelos de serviços. Estes modelos definem um padrão arquitetural para soluções que utilizam a nuvem. A Figura 2.2 exibe estes modelos.

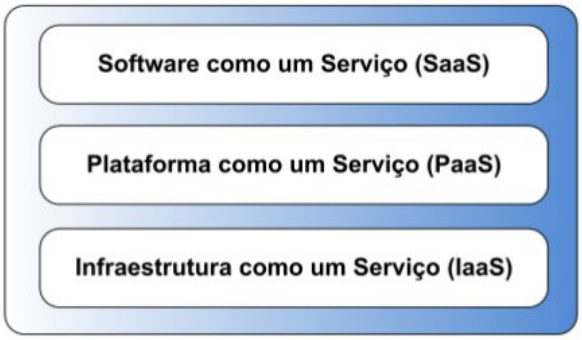

Figura 2.2: Modelos de Serviços (SOUSA; MOREIRA; MACHADO, 2009)

2.3.1 Software como um Serviço (SaaS)

A capacidade fornecida ao consumidor é de usar as aplicações do fornecedor que funcionam em uma infraestrutura da nuvem. As aplicações são acessíveis dos vários dispositivos do cliente através de uma relação do *thin client* tal como um navegador Web. No SaaS, o consumidor não administra ou controla a infraestrutura básica, incluindo nuvens de rede, servidores, sistemas operacionais, armazenamento, ou mesmo capacidades de aplicação individual, com a possível exceção de limitada aplicação específica e definições de configuração de utilizadores (MELL; GRANCE, 2011).

Podemos dizer que o SaaS, representa os serviços de mais alto nível disponibilizados em uma nuvem. Esses serviços representam as aplicações completas que são oferecidas aos usuários. Os prestadores de serviços disponibilizam o SaaS na camada de aplicação, o que leva a rodar inteiramente na nuvem e pode ser considerado uma alternativa a rodar um programa em uma máquina local, assim o SaaS traz a redução de custos, dispensando a aquisição de licença de softwares. Como o software está na Web, ele pode ser acessado pelos usuários de qualquer lugar e a qualquer momento, permitindo maior integração entre unidades de uma mesma empresa ou outros serviços de software. O *Google Docs* é um exemplo de Saas.

2.3.2 Plataforma como um Serviço (PaaS)

A capacidade fornecida ao consumidor é de desdobrar a infraestrutura da nuvem com aplicações adquiridas ou criadas pelo próprio consumidor usando linguagens de programação e

as ferramentas suportadas pelo fornecedor. O consumidor não administra ou controla a infraestrutura básica, incluindo nuvens de rede, servidores, sistemas operacionais, ou armazenamento, mas tem controle sobre os aplicativos utilizados e eventualmente hospedagem de aplicativos e configurações de ambiente (MELL; GRANCE, 2011).

O PaaS tem por objetivo facilitar o desenvolvimento de aplicações destinadas aos usuários de uma nuvem, criando uma plataforma que agiliza esse processo. Ele oferece uma infraestrutura de alto nível de integração para implementar e testar aplicações na nuvem. Também fornece um sistema operacional, linguagens de programação e ambientes de desenvolvimento para as aplicações, auxiliando a implementação de softwares, já que contém ferramentas de desenvolvimento e colaboração entre desenvolvedores.

Em geral, os desenvolvedores dispõem de ambientes escaláveis, mas eles têm que aceitar algumas restrições sobre o tipo de software que se pode desenvolver, desde limitações que o ambiente impõe na concepção das aplicações até a utilização de sistemas de gerenciamento de banco de dados (SGBDs) do tipo chave-valor, ao invés de SGBDs relacionais. Do ponto de vista do negócio, a PaaS permitirá aos usuários utilizarem serviços de terceiros, aumentando o uso do modelo de suporte no qual os usuários se inscrevem para solicitações de serviços de TI ou para resoluções de problemas pela Web. Com isso, pode-se melhorar o gerenciamento do trabalho e as responsabilidades das equipes de TI das empresas. Podemos destacar o *Google App Engine* como um exemplo de PaaS

2.3.3 Infraestrutura como um Serviço (IaaS)

A capacidade fornecida ao consumidor é a prestação de processamento, armazenamento, redes e outros recursos computacionais fundamentais onde o consumidor seja capaz de implantar e executar programas arbitrários, que podem incluir sistemas operacionais e aplicativos. O consumidor não administra ou controla a infraestrutura de nuvem subjacente, mas tem controle sobre os sistemas operacionais, armazenamento, aplicativos implantados, e, eventualmente, o controle limitado de um grupo seleto de componentes de rede (por exemplo host de firewalls) (MELL; GRANCE, 2011).

O IaaS traz os serviços oferecidos na camada de infraestrutura, nestes serviços podemos incluir servidores, roteadores, sistemas de armazenamento e outros recursos de computação. Também é responsável por prover toda a infraestrutura necessária para a SaaS e o PaaS. O IaaS traz algumas características, como uma interface única para administração da infraestrutura, a aplicação API (Application Programming Interface) para interação com hosts, switches, roteadores e o suporte para a adicionar novos equipamentos de forma simples e transparente.

O termo IaaS se refere a uma infraestrutura computacional baseada em técnicas de virtualização de recursos de computação. Esta infraestrutura pode escalar dinamicamente, aumentando ou diminuindo os recursos de acordo com as necessidades das aplicações. Do ponto de vista de economia e aproveitamento do legado, ao invés de comprar novos servidores e equipamentos de rede para a ampliação de serviços, pode-se aproveitar os recursos disponíveis e adicionar novos servidores virtuais à infraestrutura existente de forma dinâmica. O *Amazon Elastic Cloud Computing* é um exemplo de IaaS.

2.3.4 Papéis na Computação em Nuvem

Os papéis servem para distinguir os diferentes usuários que estão envolvidos em um sistema de computação na nuvem quanto a responsabilidades, acesso e perfil. Na Figura 2.3, podemos entender melhor a computação em nuvem, classificando os atores dos modelos de acordo com os papéis desempenhados.

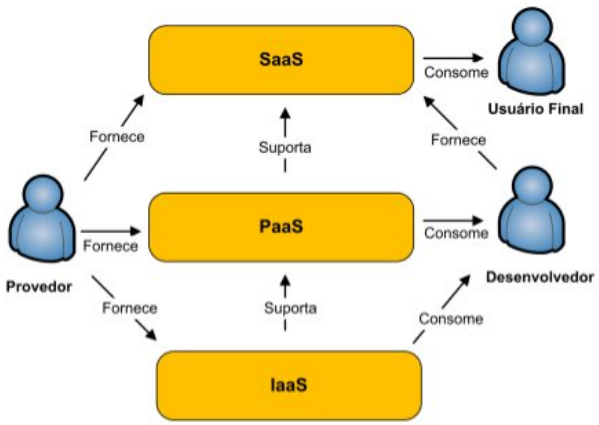

Figura 2.3: Papéis na Computação em Nuvem (SOUSA; MOREIRA; MACHADO, 2009)

- Provedor: responsável por disponibilizar, gerenciar e monitorar toda a estrutura para a solução de computação em nuvem;
- Desenvolvedor: utilizam os recursos fornecidos e disponibilizam serviços para os usuários finais;
- Usuário: utiliza os recursos fornecidos pela nuvem computacional.

A organização dos papéis ajuda a definir os atores e os seus diferentes interesses. Os atores podem assumir vários papéis ao mesmo tempo de acordo com os interesses, mas apenas o provedor fornece suporte a todos os modelos de serviços.

2.4 Modelo de Implantação

Quanto ao acesso e disponibilidade de ambientes de computação em nuvem, há diferentes tipos de modelos de implantação. A restrição de acesso depende do processo de negócio, do tipo de informação e do nível de visão. Percebe-se que certas empresas não desejam que todos os usuários possam utilizar determinados recursos no seu ambiente de computação em nuvem. Neste sentido, surge a necessidade de ambientes mais restritos, onde somente alguns usuários devidamente autorizados possam utilizar os serviços providos.

2.4.1 Nuvem Privada

A infraestrutura em nuvem é destinada a uso exclusivo de uma única organização compreendendo múltiplos consumidores. Ela pode pertencer, ser gerenciada e operada por esta organização, por terceiros, ou uma combinação dos dois, e isso pode acontecer com ou sem premissas (MELL; GRANCE, 2011).

2.4.2 Nuvem da Comunidade

A infraestrutura em nuvem é disponibilizada para uso exclusivo a uma comunidade específica de consumidores de organizações que compartilham interesses em comum. Ela pode pertencer, ser gerenciada e operada por uma ou mais organizações dentro da comunidade, por terceiros, ou uma combinação dos dois, e isso pode acontecer com ou sem premissas (MELL; GRANCE, 2011).

2.4.3 Nuvem Pública

A infraestrutura em nuvem é disponibilizada para uso aberto pelo público em geral. Ela pode pertencer, ser gerenciada e operada por uma empresa, instituição acadêmica, ou organização governamental, ou uma combinação delas. Ela existe nas premissas determinadas pelo provedor da nuvem (MELL; GRANCE, 2011).

2.4.4 Nuvem Híbrida

A infraestrutura em nuvem é uma combinação de duas ou mais infraestruturas em nuvem distintas (privada, comunidade, ou pública) que permanecem entidades únicas, mas estão unidas

pela tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicações (MELL; GRANCE, 2011).

2.5 Desafios

Computação em nuvem apresenta diversas vantagens, discutidas neste trabalho, mas também possui uma série de desafios a serem superados na utilização desse tipo de ambiente. A seguir destacamos alguns destes desafios.

2.5.1 Segurança dos Serviços de Dados

Como a computação em nuvem é um modelo que utiliza a Internet para disponibilizar seus serviços, torna-se mais complexo a utilização de recursos computacionais como diferentes domínios de redes, sistemas operacionais, software, criptografia, políticas de segurança, entre outros.

No que diz respeito à confiabilidade e responsabilidade, o provedor deve fornecer recursos confiáveis, especialmente se a computação a ser realizada é crítica e deve existir uma delimitação de responsabilidade entre o provedor e o usuário. Dessa forma, devem-se ter meios para impedir o acesso não autorizado a informações e que os dados sensíveis permaneçam privados, pois estes podem ser processados fora das empresas. Em geral, cada sistema tem seu próprio modelo de dados e política de privacidade destes dados (SOUSA; MOREIRA; MACHADO, 2009).

2.5.2 Gerenciamento de Dados

Um ponto crítico no contexto de computação em nuvem é o gerenciamento de dados. Os SGBDs relacionais não possuem escalabilidade quando milhares de sítios são considerados. Assim, aspectos de armazenamento de dados, processamento de consultas e controle transacional tem sido flexibilizados por algumas abordagens para garantir a escalabilidade, mas ainda não existem soluções que combinem estes aspectos de forma a melhorar o desempenho sem comprometer a consistência dos dados (SOUSA; MOREIRA; MACHADO, 2009).

2.5.3 Autonomia

A computação em nuvem é um sistema autônomo gerenciado de forma transparente para os usuários. Hardware e software dentro de nuvens podem ser automaticamente reconfigurados, orquestrados e estas modificações são apresentadas ao usuário como uma imagem única. Essa autonomia é importante, pois reduz o custo de equipe de monitoramento do sistema tanto no âmbito centralizado quanto distribuído. Comparados com sistemas tradicionais, é possível identificar três fatores complexos: intervenção humana limitada, alta alternância na carga de processamento e uma variedade de infraestruturas compartilhadas (SOUSA; MOREIRA; MACHADO, 2009).

2.5.4 Disponibilidade de Serviços

A disponibilidade de serviços permite aos usuários acessar e utilizar a nuvem onde e quando desejarem. Os ambientes de computação em nuvem devem prover alta disponibilidade. Para tanto, esses podem utilizar técnicas de balanceamento de carga dinâmico e composição de nuvens de forma a atender as necessidades dos usuários. Por exemplo, podem-se construir aplicações altamente disponíveis com a implantação de duas ofertas de nuvem diferentes. Caso uma das nuvens falhe, a outra nuvem continua a apoiar a disponibilidade das aplicações (SOUSA; MOREIRA; MACHADO, 2009).

2.5.5 Escalabilidade e Desempenho

A escalabilidade foi uma das características fundamentais que conduziram ao surgimento da computação em nuvem. As nuvens de serviços e as plataformas oferecidas podem ser dimensionadas por vários fatores, tais como localizações geográficas, desempenho ou configurações. Apesar das limitações de rede e segurança, as soluções de computação em nuvem devem fornecer elevado desempenho, além de ser flexível para se adaptar diante de uma determinada quantidade de requisições. Como os ambientes de computação em nuvem possuem acesso público, é imprevisível e variá el a quantidade de requisições realizadas, tornando mais complexo fazer estimativas e garantias de QoS (SOUSA; MOREIRA; MACHADO, 2009).

3 *Distributed Hash Table*

3.1 Introdução

O desenvolvimento de uma nuvem computacional consiste no gerenciamento de várias máquina através de uma rede, a fim de disponibilizar todas as vantagens da computação em nuvem, vistas no capítulo anterior, de forma transparente ao usuário final. Um dos modelos de arquitetura de nuvem computacional mais bem aceitos e pesquisados, tanto em instituições acadêmicas como em empresas privadas, é o *distributed hash table*.

O *distributed hash table* (DHT) é um tipo de sistema distribuído e descentralizado que fornece um serviço semelhante a uma tabela *hash*. Tabelas *Hash* são estruturas de dados que usam uma função *hash* para mapear eficientemente identificadores conhecidos como chaves em valores associados (por exemplo, o nome de uma pessoa em seu número de telefone). Uma função *hash* permite realizar um cálculo trivial, que irá direcionar você para o valor desejado em vez da necessidade de procurar por todo o banco de dados.

No mundo da descentralização, DHTs recentemente tem obtido um efeito revolucionário. As caóticas topologias *ad hoc* da primeira geração de arquiteturas P2P foram substituídas por um conjunto de topologias emergentes, com propriedades demonstráveis e excelente performance. Conhecimento em algoritmos de DHT será um ponto chave no futuro do desenvolvimento de aplicações distribuídas (WILEY, 2003).

O DHT reside na camada de aplicação, embora ele não seja um aplicativo em si. Ele é usado em aplicações complexas como sistemas de arquivos distribuídos, provedores de e-mail, DNS (MOCKAPETRIS; DUNLAP, 1988), e mensageiros instantâneos. Um dos usos mais conhecidos para o DHT é o compartilhamento de arquivos por P2P *BitTorrent* (COHEN, 2001) e versões recentes do *Gnutella* (FRANKEL; PEPPER, 2000), por exemplo, usam DHT junto com outros meios tradicionais de distribuição de dados. Ele também é usado em sistemas de distribuição de conteúdo e Web Cache. Por exemplo, o *Coral Content Distribution Network* (FREEDMAN, 2004) mantém caches de web sites que possuem um grande tráfego. Existem

várias máquinas de busca experimentais como o *YaCy* (CHRISTEN, 2006), que usa o DHT para indexar conteúdo. Ele também é usado em redes anônimos como o Freenet (CLARKE, 2000), que pode ser usada para burlar a censura passando a informação diretamente através de vários nós.

O DHT fornece um jeito rápido e a prova de erros de armazenar e acessar grandes quantidades de dados reduzindo o custo à medida que a quantidade de dados aumenta.

Um grande número de DHTs foram desenvolvidos por universidades, escolhidos pela comunidade *Open Source* e implementados. Existem também algumas implementações proprietárias que são incorporadas em produtos comercialmente disponíveis. Pesquisas existentes sobre o assunto, como o *Chord* (STOICA et al., 2001), o *Kademlia* (MAYMOUNKOV; MAZIèRES,) e o *Dynamo* (DECANDIA et al., 2007), são o ponto de partida para o desenvolvimento de um sistema que queira utilizar o DHT.

3.2 Modelo do Sistema

Em um DHT, um nó é uma máquina na rede e qualquer nó participante pode recuperar rapidamente um valor baseado em uma chave. Em vez de um único servidor central, a responsabilidade por mapear é distribuída entre os nós, e é feito de tal forma que as mudanças nos nós conectados no sistema cause o mínimo de pertubação. Devido a esta estrutura, DHTs podem suportar grandes números de nós, enquanto gerenciam continuamente chegadas, partidas e falhas entre os nós. A seguir discutiremos como é a modelagem de um sistema DHT.

3.2.1 Distribuindo as Chaves

O DHT utiliza a técnica de *Consistent Hashing* (LEWIN, 1998). Essa técnica inclui o conjunto de todas as chaves possíveis, que são divididas entre os nós do sistema. Para mapear as chaves, precisa-se definir uma função *hash* que irá gerar um identificador de m-bits para cada nó e para cada chave, como por exemplo SHA-1 (FIPS, 1995). O identificador de um nó é obtido aplicando a função *hash* sobre qualquer valor que represente cada nó unicamente, como por exemplo o endereço IP do nó, enquanto o identificador de uma chave é obtido aplicando a função *hash* na própria chave. O tamanho do identificador deve ser grande o suficiente para se evitar que dois nós ou duas chaves não possuam o mesmo identificador.

Desse modo, associamos as chaves para os nós da seguinte maneira. O espaço dos identificadores é representado por um espaço circular ou um “anel”. A chave k é designada para o

primeiro nó cujo identificador é igual ou maior que (o identificador de) k no espaço dos identificadores. Este nó é chamado de o *nó sucessor* da chave k, denotado por *sucessor(k)*. Se os identificadores são um círculo de números de 0 até $2^m - 1$, então o *sucessor(k)* é o primeiro nó no sentido horário de k (STOICA et al., 2001). Por consequência o sucessor do maior identificador é o nó que possui o menor identificador entre os nós.

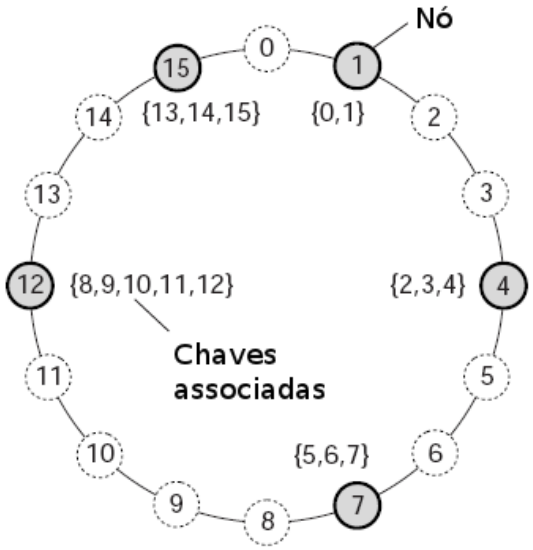

Figura 3.1: Um círculo de identificadores com 5 nós (YU, 2010).

A Figura 3.1 mostra um círculo de identificadores com $m = 4$. O círculo tem 5 nós: 1, 4, 7, 12 e 15. O sucessor do identificador 1 é o nó 1, então a chave 1 estaria localizada no nó 1. De modo similar, a chave 2 estaria localizada no nó 4, e a chave 9 no nó 12.

Cada nó se torna responsável pela região no anel entre ele mesmo e seu nó predecessor. A maior vantagem desse modelo é que a chegada e partida de um nó somente afeta os seus vizinhos enquanto os outros nós permanecem inalterados. Para manter a consistência, quando um nó n entra na rede, algumas chaves com identificadores que pertencem ao sucessor de n passam a pertencer a n. Quando o nó n deixa a rede, todas as suas chaves passam a pertencer ao seu sucessor. Nenhuma outra mudança entre os nós é necessária.

3.2.2 Procurando uma Chave

Todo nó na rede de um DHT deve ter a capacidade, de quando requerido, retornar o valor de uma determinada chave, mesmo que a chave não lhe pertença. Para isso o DHT age como uma rede virtual sobreposta que reside no topo da rede física. Na rede sobreposta, os nós são conectados de forma esparsa e um identificador de pesquisa de uma chave é passada de nó em nó até ela atingir um nó que é responsável por aquele identificador. Para qualquer chave, ou um

nó é responsável por seu identificador ou o nó possuí um link para outro nó que é responsável pelo identificador. A seguir, discutiremos os vários modelos de como os nós serão conectados entre si para realizar a pesquisa.

O método mais simples é pesquisar os nós sequencialmente (ou linearmente), desse modo cada nó precisa apenas saber da existência de seu sucessor no círculo. As pesquisa por um identificador são passadas sequencialmente para o nó sucessor até se encontrar o nó responsável pelo identificador pesquisado. Logo, cada nó precisa manter uma referência para o nó que o sucede para garantir que a pesquisa retorne corretamente. Porém, esse método é ineficiente; pode ser necessário percorrer todos os nós da rede para encontrar o nó responsável pelo identificador pesquisado, uma ordem de complexidade $O(N)$ (com N sendo o número total de nós na rede). Geralmente ele é utilizado em conjunto com outros métodos mais eficientes como complemento.

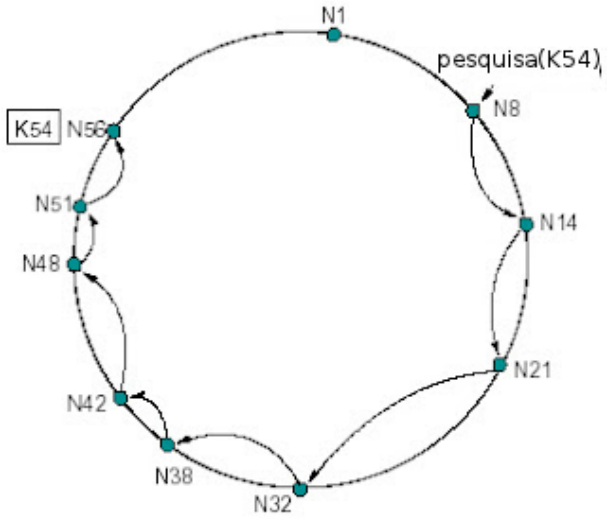

Figura 3.2: Pesquisa sequencial entre os nós (WERAPUN, 2008).

A Figura 3.2 mostra como é a pesquisa pelo identificador $K54$ a partir do nó $N8$. A pesquisa é repassada sequencialmente entre de nós sucessores até encontrar o nó responsável identificador $K54$, no caso é o nó $N56$.

Para melhorar a busca pelo um nó responsável de um dado identificador o Chord usa um método escalável que utiliza uma tabela de roteamento (*finger table*). Sendo m o número de bits utilizado para gerar os identificadores, cada nó possui uma tabela com no máximo m entradas. A i-ésima entrada da tabela de um dado nó n contém o identificador do primeiro nó, s, que sucede o identificador de n por pelo menos 2^{i-1} no anel de identificadores, ou seja, $s = sucessor(n + 2^{i-1})$, onde $1 <= i <= m$. Toda entrada nessa tabela contém o identificador do nó que ela representa quanto o seu endereço IP. Com isso cada nó possui conhecimento de um pequeno

números de nós na rede, e a ordem complexidade para se achar o nó responsável de um dado identificador cai para $O(log(N))$ (STOICA et al., 2001).

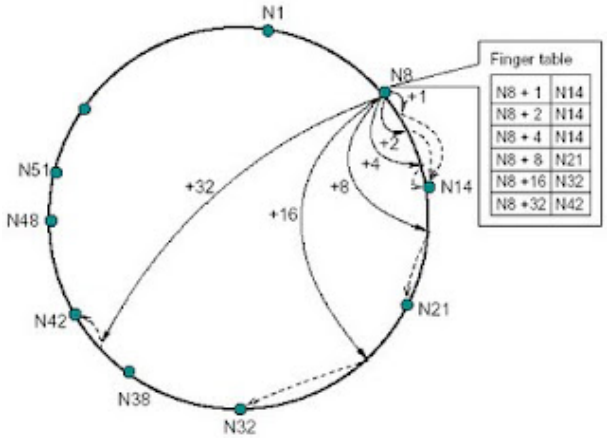

Figura 3.3: Montagem da tabela de roteamento (*finger table*) (WERAPUN, 2008).

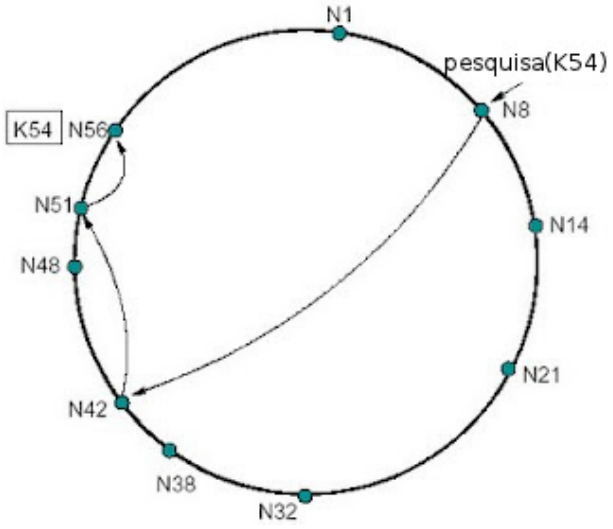

Figura 3.4: Pesquisa escalável entre os nós (WERAPUN, 2008).

A Figura 3.3 mostra como é a construção das 6 primeiras entradas da *finger table* do nó $N8$. E a Figura 3.4 monstra como é a pesquisa escalável pelo identificador $K54$ a partir do nó $N8$.

3.2.3 Adicionando e Removendo Nós da Rede

Para que a busca sequencial e a busca escalável funcionem, as referências para os nós sucessores e as tabelas de roteamento, respectivamente, de todos os nós precisam estar sempre atualizadas. No caso da busca escalável, uma tabela desatualizada não faz com que uma pesquisa de identificador retorne um resultado errado, desde que cada nó mantenha a referência para seu sucessor, mas a busca só é logarítmica se a tabela de roteamento estiver correta. Para

simplificar a chegada e partida dos nós, cada nó também deve manter uma referência atualizada para o seu predecessor no círculo.

Adicionando um Nó

Desse modo, o nó n precisa contactar um nó n' existente na rede. Então o nó n usa o n' para inicializar a sua inserção. Deve-se realizar três etapas quando um nó n é adicionado à rede (STOICA et al., 2001):

1. **Inicialização do novo nó:** O nó n faz uma pesquisa linear por seus predecessor e sucessor e uma pesquisa escalável por cada entrada de sua tabela de roteamento através do nó n', para suas respectivas inicializações. O que resulta em uma ordem de complexidade $O(mlog(N))$.

2. **Atualização dos nós existentes:** São feitas a atualização das referências do predecessor e sucessor dos nós vizinhos de n. A seguir o nó n precisa ser inserido na tabela de roteamento dos outros nós da rede. Interpretando a tabela de roteamento de um nó arbitrário p, a i-ésima entrada da tabela é o primeiro nó que sucede p a uma distância de pelo menos 2^{i-1} no sentido horário do nó p, então basta o nó n procurar, de 1 até m, pelos primeiros nós que o precedem pela mesma distância mínima porém em sentido contrário. Esses nós terão a i-ésima entrada de suas tabelas de roteamento atualizadas para o nó n. O que leva a uma ordem de complexidade $O(log^2(N))$.

3. **Transferência de chaves:** Transferir a responsabilidade de todas as chaves e valores que agora o nó n é o sucessor, que previamente eram responsabilidade do sucessor de n.

Removendo um Nó

Quando um nó n é removido da rede intencionalmente, é executado um processo parecido como quando se adiciona um nó:

1. **Atualização dos nós existentes:** Os nós predecessor e sucessor do nó n tem as referências de sucessor e predecessor, respectivamente, atualizadas para referenciarem um ao outro. A seguir, são atualizadas todas as tabelas de roteamento que referenciam o nó n para apontarem para o sucessor de n. Utiliza-se o mesmo método usado na adição de um nó para encontrar os nós que tem n em sua tabela de roteamento.

2. **Transferência de chaves:** Transferir a responsabilidade de todas as chaves e valores do nó *n* para o seu sucessor.

3.2.4 Resolvendo Falhas dos Nós

Durante o uso de DHT podem ocorrer falhas nas comunicações entres os nós, como por exemplo a máquina física que representa o nó ficar sem energia, ou ter uma falha de hardware ou perda do acesso à rede, Internet ou rede local. Geralmente estas falhas são temporárias e o nó falho deverá voltar a funcionar normalmente depois de um período. Caso haja uma falha em um nó específico *n* do DHT, todas as pesquisas feitas por identificadores que *n* é responsável e pesquisas que são roteadas através *n* irão falhar.

Um solução para esse problema é replicar os dados de um nó em múltiplos nós. O nó *n* deve ser responsável por replicar os dados das suas chaves para outros *R* nós na rede. Ao salvar o valor de uma chave que lhe é responsável, o nó *n* replica esses valores para os seus primeiros *R* sucessores. Desse modo, cada nó na rede se torna responsável pela região do anel entre ele mesmo e seu *R*-ésimo predecessor (DECANDIA et al., 2007).

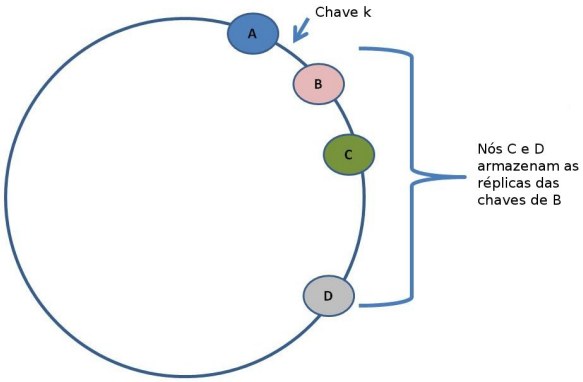

Figura 3.5: Replicação das chaves em mais de um nó (GANESH, 2012).

A Figura 3.5, mostra que a chave *k*, pertencente ao nó *B*, será replicada para os nós *C* e *D*.

Desse modo, quando um nó *n* falha, os nós que mantém *n* em suas tabelas de roteamento devem encontrar o sucessor de *n*. Para isso, cada nó deverá manter uma lista de sucessores contendo os seus *R* primeiros sucessores no anel (STOICA et al., 2001).

O valor exato de réplicas a se utilizar pode variar de acordo como for a necessidade da prevenção de falhas e para algum uso mais simples do DHT talvez não seja necessária o uso desta técnica. Caso as réplicas sejam utilizadas, as ações de adição e remoção de um nó da rede

devem ser atualizadas para incluir também a transferência das réplicas.

3.2.5 Segurança entre os Nós

Dependendo o uso de um sistema DHT, a entrada e saída de nós da rede, assim como os outras formas de comunicação entre nós, não pode ser permitida para qualquer computador. É necessário que somente computadores confiáveis acesse os serviços do DHT. Quando um nó n iniciar alguma comunicação com outro nó n', n' só irá dar continuação na comunicação caso reconheça n como confiável.

Um modelo simples para autenticar um nó como confiável é a utilização de criptografia RSA (JONSSON; KALISKI, 2003). Para isso, cada nó na rede deve possuir um par de chaves RSA, uma pública e outra privada, assim como as chaves públicas dos outros nós na rede. Com isso, a autenticação do nó n na comunicação com n' se dará com os seguintes passo:

1. n' envia um texto aleatório para n;
2. n assina o texto utilizando a sua chave RSA privada e envia a assinatura para n';
3. n' verifica a assinatura com a chave RSA pública de n;
4. Caso a assinatura seja comprovada a comunicação segue normalmente, senão, n' encerra a tentativa de comunicação.

Não é necessário realizar a autenticação entre nós para todo a comunicação entre eles. Basta que seja realizada no primeiro contato entre eles, e que depois seja expirada depois de um período de tempo.

Embora simples, esse modelo utilizando assinaturas RSA é muito trabalhoso quando se tem um número grande de nós no DHT, já que a entrada e saída de um nó resultaria em todos os nós atualizarem suas listas de chaves públicas. Pelo mesmo motivo também seria trabalhoso quando a chave privada de algum nó fosse comprometida e deixa-se de ser confiável. Um modelo mais completo que resolve essas falhas é o *Private Key Infrastructure* (PKI) (COOPER et al., 2008).

O PKI é um modelo utilizado em vários sistemas para verificar autenticação como em Smart Cards, protocolos de Internet (HTTPS, IPSEC), etc. A utilização do modelo de PKI consiste em terceirizar a responsabilidade da autenticação. Cada nó na rede, em vez de possuir apenas um par de chaves RSA, deve possuir um certificado digital. Para validação desse certificado, ele precisa ser assinado por uma entidade que será considerada confiável por todos os nós da rede,

chamada de Autoridade Certificadora (CA, do termo em inglês). Desse modo, os nós precisam guardar apenas os certificados das CAs confiáveis do sistema em vez das chaves RSA públicas dos outro nós, que será um número bem menor que o número de nós na rede. Com o certificado da CA, um nó tem condições de determinar se um certificado digital qualquer foi assinado pela CA ou não. Logo, a autenticação do nó n na comunicação com o n' passa a adotar os seguintes passos:

1. n envia o seu certificado digital para n';
2. n' verifica se o certificado de n foi assinado por alguma CA em que confie;
3. Caso confirmado a assinatura a comunicação segue normalmente, senão é encerrada a comunicação.

Com o uso de um PKI os nós precisam apenas armazenar alguns certificados de CAs, além dessa estrutura dar suporte para a negação de certificados digitais comprometidos através de *Certificate Revocation Lists* (CRLs). A Figura 3.6 resume o funcionamento da autenticação entre os nós utilizando um PKI.

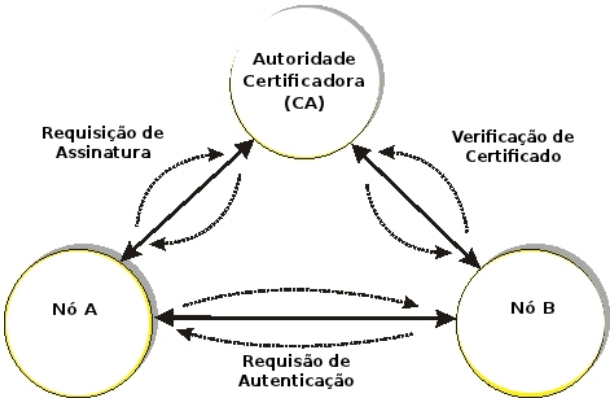

Figura 3.6: Funcionamento de um PKI (Próprio Autor).

3.3 Implementação

Nesta seção falaremos sobre a implementação de um modelo simplificado do DHT, a qual fiz parte do planejamento e desenvolvimento feita pela empresa MAV Tecnologia, e mostrarei os resultados obtidos.

A MAV Tecnologia é uma empresa especializada em segurança digital com sede em Belo Horizonte. Faço parte da equipe de desenvolvimento da MAV Tecnologia desde abril de 2008 e

participei do planejamento e desenvolvimento do seu principal produto atualmente, o MAV 5. O MAV 5 é um programa com o intuito de controlar e manter segura a rede interna do cliente, um *gateway* que é barreira entre a rede pública dos clientes da MAV e a Internet.

Um dos subprodutos do MAV 5 é o Mail Security, especializado no controle e segurança do tráfego de e-mails. Uma das funcionalidades do Mail Security é o sistema de anti-spam que consiste em uma análise se um e-mail é um e-mail comercial enviado em massa enviado sem a solicitação do destinatário. A análise do anti-spam no Mail Security é feita em duas etapas: primeiro o remetente passa pela aprovação do Sistema de Reputação; e em segundo é feito a análise textual do e-mail, assim como é feito pela maioria dos ”softwares” de anti-spam no mercado.

O Sistema de Reputação foi feito para definir uma reputação dos remetentes de e-mail através da Internet. Logicamente, um remetente conhecido por enviar muitos spams tem uma baixa reputação. Com isso o Mail Security pode recusar a conexão de um remetente com a reputação baixa antes mesmo de se começar o envio do e-mail, economizando a banda de nossos clientes.

Por trás do Sistema de Reputação está um método estatístico que necessita de vários dados do remetente ao longo do tempo para determinar a sua reputação, como por exemplo a probabilidade do anti-spam de texto, a média de destinatários inválidos nos e-mails desse remetente, quantas vezes ele foi considerado spam, etc. Outra necessidade do Sistema de Reputação é que ele não pode ficar indisponível, pois acarretaria em uma grande queda na qualidade do anti-spam do Mail Security além do aumento do tráfego nos clientes. Por isso o Sistema de Reputação foi baseado em um DHT.

Essa primeira implementação do DHT recebeu o nome interno de *Simple DHT*, pois não conta com a maioria das funcionalidades discutidas nesse capítulo. Ele foi desenvolvido para solucionar os requisitos do Sistema de Reputação e também para servir para base de estudos e como um protótipo para projetos futuras da MAV Tecnologia que pretendem usar todas as funcionalidades de um DHT.

O *Simple DHT* do Sistema de Reputação utiliza uma função *hash* que gera identificadores de 12 bits. Ele conta com 6 nós e não foi planejada nenhuma adição de novos nós, portanto não foi implementada a funcionalidade de adição de nós na rede. Por isso, e pelo pequeno número de nós também não foi necessário implementar a tabela de roteamento, cada nó tem o conhecimento de todos os outros e envia as requisições diretamente. Contra falhas temporárias dos nós, cada dado salvo no *Simple DHT* é replicado para 2 nós na rede, porém um nó não atualiza os seus identificadores quando ele volta de uma falha. Cada nó possui um par de chaves

RSA e conhece a chave RSA pública dos outros nós e a autenticação das requisições entre eles utiliza o primeiro método discutido no item 3.2.5 deste trabalho. Para armazenar os dados dos remetentes necessários para o método estatístico de reputação e para o processamento de tais dados, a chave para o armazenamento é o endereço IP do remetente.

Desse modo, para cada e-mail que chega para o servidor do cliente, o Mail Security faz uma requisição para qualquer nó do Sistema de Reputação usando como chave de pesquisa o endereço IP do remetente. A requisição é repassada para um dos dois nós responsáveis pelo identificador da chave passada que recupera os dados necessários e faz o cálculo da reputação do remetente. O valor retornado para o Mail Security é a probabilidade de o remetente ser um enviador de spam ou não. Então, de acordo com o nível de corte configurado pelo cliente, o Mail Security decide se aceita ou não a conexão.

A alimentação dos dados do Sistema de Reputação se deu inicialmente por uma base de e-mails internos da MAV. Para manter o Sistema atualizado, além de retornar uma probabilidade para cada requisição, também é retornado um valor booleano, aleatório, que informa ao Mail Security se os dados daquele remetente devem ser reportados. Caso esse valor retorne verdadeiro, o Mail Security envia os dados do remetente que interessam o Sistema de Reputação utilizando novamente o endereço IP do remetente como chave. Essa requisição então é repassada para o nó responsável pelo identificador da chave que deve replicar os dados para outro nó.

3.4 Resultados

Como explicado anteriormente, o Sistema de Reputação foi planejado e desenvolvido sobre uma implementação simplista de um DHT. As 6 máquinas que constituem o DHT estão localizadas em um *Data Center* em Miami, Flórida, e foram nomeados com a numeração de 1 a 6 respectivamente, mas sem relação com a posição lógica de cada nó no anel de identificadores.

Atualmente os nós 2 e 6 apresentaram falhas de *hardware* e não é possível obter acesso remoto a elas enquanto não houver a oportunidade de realizar a manutenção *in loco*. Embora 2 máquinas estejam fora do sistema o Sistema de Reputação continuou disponível e funcional, o que era um dos objetivos no início do planejamento.

A Tabela 3.1 descreve o desempenho dos nós restantes do Sistema de Reputação com relação à quantidade de IPs que cada nó é responsável, a quantidade de requisições que cada nó recebe dos clientes e a quantidade de requisições que cada nó recebe de outros nós da rede.

Nó	IPs	Requisições de Clientes	Requisições entre nós
1	806.890	160,06/min	318,89/min
3	849.138	241,04/min	561,46/min
4	881.882	69,67/min	360,06/min
5	910.720	78,60/min	572,12/min
Média	**862.157,50**	**137,34/min**	**453,13/min**

Tabela 3.1: Resultados Individuais dos Nós no Sistema de Reputação usando o *Simple DHT* (Próprio Autor).

Pode-se perceber pela Tabela 3.1 que os IPs estão bem distribuídos através dos nós do *Simple DHT*, atingindo o balanceamento de armazenamento de dados que era um requisito do Sistema de Reputação.

O número de requisições vindas de clientes não foi muito homogênea entre os nós, porém obteve uma boa média de 137,34 requisições por minuto. Um bom resultado que mostra o que o Sistema de Reputação possa talvez aguentar um ataque de negação de serviço.

Houve maior discrepância nos resultados do número de requisições entre os nós, mas isso é justificável pois os nós 3 e 5 são responsáveis por armazenar outros dados necessários para o método estatístico de reputação, como o cálculo de constantes. Os outros nós lhes consultam diretamente por essas informações.

4 *Trabalhos Relacionados*

Neste capítulo são mostrados alguns trabalhos relacionados. Existem atualmente vários sistemas e arquiteturas que dão suporte ao armazenamento e distribuição de um grande volume de dados, tais como os banco de dados em *cluster*, os sistemas *noSQL* e algumas implementações utilizando uma DHT como o Chord, o Kademlia e o Dynamo.

A primeira geração de sistemas P2P, como o Freenet (CLARKE, 2000), eram usados predominantemente como compartilhadores de arquivos. Eles eram exemplos de redes P2P não estruturadas onde os links entre os nós eram estabelecidos arbitrariamente. Nessas redes, uma busca por uma chave era normalmente espalhada por toda a rede para achar o maior número possível de nós que compartilhavam o dado. Sistemas P2P evoluíram para uma nova geração que é conhecida como redes P2P estruturadas. Essas redes usam um protocolo consistente para garantir que qualquer nó pode traçar uma rota de busca eficiente para os nós que possuam o dado desejado.

O Chord (STOICA et al., 2001) mapeia as chaves em nós, outros serviços tradicionais mapeiam chaves em valores. Um valor pode ser um endereço, um documento, ou um dado arbitrário. O Chord implementa facilmente essa funcionalidade armazenando cada chave/valor par em um nó onde a chave é mapeada, como por exemplo um sistema de DNS (MOCKAPETRIS; DUNLAP, 1988). O DNS provê um mapeamento entre nome do host e um endereço IP. O Chord disponibilizaria o mesmo serviço com o nome representando uma chave e o endereço IP associado representando um valor. Nesse caso é que o Chord não depende de nenhum servidor especial, enquanto um serviço de DNS depende de um conjunto de servidores especiais. Sistemas como o Chord usam mecanismos de roteamento para garantir que as consultas podem ser respondidas dentro de um número limitado de nós a serem buscados.

O Kademlia (MAYMOUNKOV; MAZIèRES,) se assemelha bastante ao Chord, a diferença entre os dois está no mecanismo de roteamento mais eficiente. Enquanto o Chord utiliza uma métrica modular, o Kademlia utiliza uma métrica baseada na operação *ou-exclusivo* (XOR). A vantagem da métrica do Kademlia é que ela é simétrica, ou seja, a distância entre a e b é a

mesma que a distância de b e a, o que não é verdade para a métrica adotada pelo Chord. Com isso o Kademlia economiza na construção da tabela de roteamento e na comunicação entre os nós pois um nó a que tenha b em sua tabela de roteamento implicará que a esteja na tabela de roteamento de b, o que é consequência da simétrica da métrica XOR.

O Dynamo (DECANDIA et al., 2007) difere dos sistemas descentralizados de armazenamento citados acima. Seu foco é para aplicações que precisam de uma alta disponibilidade para escrita, onde nenhuma atualização de dados são perdidas por causa de erros ou escritas concorrentes. Para isso foi necessário evitar requisições de roteamento entre múltiplos nós, que é a causa pelo aumento na variabilidade nos tempos de respostas das consultas, portanto aumentando a latência. O Dynamo pode ser caracterizado como um DHT sem roteamento entre nós, onde cada nó mantém informação suficiente para fazer a consulta para o nó apropriado diretamente.

5 *Conclusão e Trabalhos Futuros*

Recentemente tem crescido a importância que o conceito de computação em nuvem vem adquirindo no cenário tecnológico mundial. Diversas empresas apresentaram suas iniciativas na promoção da computação em nuvem. A comunidade científica também tem apresentado algumas soluções, principalmente com foco em suas necessidades.

Este trabalho apresentou os principais aspectos de computação em nuvem e alguns conceitos e aplicações relacionadas com estes ambientes. Os benefícios obtidos com esta tecnologia tem sido expressivos, tanto para grandes corporações, com um grande apelo econômico além da flexibilidade e dinamicidade proporcionada, quanto para o usuário comum, que usufrui principalmente da mobilidade, integração e inteligência das aplicações.

Foi discutido também um dos modelos mais adotados e aceitos para o desenvolvimento de uma nuvem computacional, o *Distributed Hash Table*. O DHT é similar à uma tabela *hash*; pares (chave, valor) são armazenados na nuvem, e qualquer nó participante pode recuperar o valor associado a uma dada chave. Foram discutidos vários quesitos no planejamento de um DHT para garantir as propriedades que se esperam na computação em nuvem como a distribuição descentralizada e homogênea do dados e do processamento entre os nós, a escalabilidade do sistema, como otimizar a comunicação dentro da rede, prevenção a falhas e a segurança do serviço prestado.

Em seguida foi apresentado os detalhes de como foi o planejamento, implementação e resultados de uma aplicação que fez uso dos recursos de um DHT. Essa implementação do DHT teve como objetivo fixar os conceitos mas não incluía todos os itens discutidos ao longo desse trabalho. Porém ela foi suficiente para comprovar as vantagens em se utilizar um DHT.

Com este estudo ficou claro os benefícios obtidos ao utilizar um DHT para uma aplicação que engloba os conceitos da nuvem computacional. Pode-se obter grandes resultados mesmo com uma implementação simplificada do DHT e quem for utilizá-lo pode moldar as suas funcionalidades de acordo com os requisitos que necessita.

Por esse grande poder de customização e o grande potencial computacional que o DHT

proporciona, ele se tornou referência no desenvolvimento da computação em nuvem. Ele já vem sendo utilizado por várias grandes empresas e seus clientes e ainda é um constante tema para pesquisa no meio acadêmico.

5.1 Trabalhos Futuros

Fica para um trabalho futuro a implementação de um DHT completo que abrange todos os tópicos discutidos, talvez com algumas mudanças de modelos baseadas em pesquisas acadêmicas semelhantes. Para obtenção dos resultados, também fica como futuro trabalho a utilização do DHT em outros tipos de aplicações que tenham requisitos funcionais e não-funcionais que possam ser solucionados pelo DHT e que possam aproveitar da maioria suas vantagens, como por exemplo um sistema de backups em nuvem, ou uma hospedagem de e-mails.

Outra linha de pesquisa sobre o assunto seria fazer um estudo de caso sobre os tipos de aplicações utilizam outras soluções para os problemas abordados por um DHT. Para isso seria necessário um estudo se é possível implementar o DHT para essas aplicações e discutir as vantagens e desvantagens nesses casos.

Pode-se também pesquisar sobre métodos para se construir casos de teste para o DHT a fim de determinar metas de qualidade para o sistema. Fazer otimizações a fim de conseguir a melhor distribuição de chaves entre os nós ou reduzir o número de comunicações entre eles para manter o sistema estável.

Referências

CHRISTEN, M. *YaCy, decentralized web search*. 2006. http://yacy.net/en/.

CLARKE, I. *Freenet, the free network*. 2000. http://freenetproject.org/.

COHEN, B. *BitTorrent*. 2001. http://www.bittorrent.org/.

COOPER, D. et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. [S.l.], Maio 2008. Disponível em http://tools.ietf.org/html/rfc5280.

DECANDIA, G. et al. *Dynamo: Amazon's Highly Available Key-value Store*. Outubro 2007. Disponível em: www.allthingsdistributed.com/2007/10/amazons_dynamo.html. Acessado em 23 de maio de 2011.

FIPS. *Secure Hash Standard*. Springfield, Abril 1995. Disponível em http://www.itl.nist.gov/fipspubs/fip180-1.htm.

FRANKEL, J.; PEPPER, T. *Gnutella, file sharing and distribution network*. 2000. http://rfc-gnutella.sourceforge.net/index.html.

FREEDMAN, M. *The Coral Content Distribution Network*. 2004. http://www.coralcdn.org/.

GANESH, T. V. Design principles of scalable, distributed system. *IBM Developer Works*, Março 2012. Disponível em https://www.ibm.com/developerworks/mydeveloperworks/blogs/theTechTrek/entry/design_principles_of_scalable_distributed_system5?lang=en.

JONSSON, J.; KALISKI, B. *Public-Key Cryptography Standards (PKCS) 1: RSA Cryptography Specifications Version 2.1*. [S.l.], Fevereiro 2003. Disponível em http://tools.ietf.org/html/rfc3447.

LEWIN, D. M. *Consistent Hashing and Random Trees: Algorithms for Caching in Distributed Networks*. Dissertação (Mestrado) — Massachusetts Institute of Technology, Março 1998.

MAYMOUNKOV, P.; MAZIèRES, D. *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*. Disponível em www.cs.rice.edu/Conferences/IPTPS02/109.pdf.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. Gaithersburg, Setembro 2011.

MOCKAPETRIS, P.; DUNLAP, K. J. Development of the domain name system. *SIGCOMM '88 Symposium proceedings on Communications architectures and protocols*, p. 123–133, 1988.

RIBAS, E. A. et al. Um sgbd com armazenamento distribuído de dados baseado em dht. *Simpósio Brasileiro de Banco de Dados*, 2010.

SOUSA, F. R. C.; MOREIRA, L. O.; MACHADO, J. C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. *ERCEMAPI*, 2009.

STOICA, I. et al. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*. Novembro 2001. Disponível em: `http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf`. Acessado em 23 de maio de 2011.

TAURION, C. *Cloud Computing - Computação em Nuvem: Transformando o mundo da Tecnologia da Informação*. 1ª edição. ed. Rio de Janeiro: Brasport, 2009.

VAQUERO, L. M. et al. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, v. 39, p. 50–55, Janeiro 2009.

WERAPUN, W. Dht: Chord routing. *My Research in Toulouse*, Setembro 2008. Disponível em `http://wwarodomfr.blogspot.com.br/2008/09/chord-routing.html`.

WILEY, B. Distributed hash tables. *Linux Journal*, Outubro 2003. Disponível em `http://www.linuxjournal.com/article/6797`.

YU, T. L. Distributed systems architecture. *CS 660 OS Theory Concepts*, Março 2010. Disponível em `http://www.csci.csusb.edu/tongyu/courses/cs660/notes/distarch.php`.