Capítulo

3

Desafios em Cloud computing: Armazenamento, Banco de Dados e BIG Data

Rodrigo Elia Assad[1], Marco André Santos Machado[1,2], Paulo Fernando Almeida Soares[1,2], Anderson Fonseca e Silva[1] , Thiago Jamir e Silva[1,2], Vinicius Cardoso Garcia[2] Fernando Mota Trinta[3], Silvio Romeiro Lemos Meira[2]

[1]USTO.RE {assad@usto.re, marco@usto.re, paulo@usto.re, anderson@usto.re, tjamir@usto.re}
[2]Universidade Federal de Pernambuco { vcg@cin.ufpe.br, srlm@cin.ufpe.br }
[3]Universidade Federal do Ceará { fernando.trinta@lia.ufc.br }

Abstract

With increasing connectivity speed and Web systems evolution, emerges the Internet systems, which are more commonly called cloud computing. It´s designates a support platform that provides: management, on-demand use, fitness requirements, rational use of resources and automation of processes related to creation of infrastructure. On this context emerge systems of data storage in the cloud, database scalability and data search and retrieval now called as BIGDATA. This short course addresses these exemplifying how we implement and use such platforms.

Resumo

Com o aumento da velocidade de conectividade e evolução dos sistemas WEB, começa a surgir os sistemas de Internet, que mais comumente são chamados de Computação nas Nuvens. Este termo designa uma plataforma de suporte a sistemas de software que provê aos seus usuários: gerenciamento, uso sob demanda, adequação às necessidades, racionalização do uso dos recursos e automação dos processos relacionados a criação de infra-estruturas. Neste contexto surge os sistemas de armazenamento de dados em nuvem, escalabilidade de banco de dados e busca e recuperação que hoje chamamos de BIGDATA. Este minicurso aborda estes temos exemplificando como implementar e utilizar tais plataformas.

3.1 Introdução

Em 1990, Tim Berners-Lee deu início à implementação das linguagens, dos protocolos e dos artefatos de software necessários à construção da *World Wide Web*, ou simplesmente web, como conhecemos hoje [W3C, 2000]. Duas décadas depois, a web ganhou proporções mundiais, atingindo mais de um bilhão e meio de pessoas em todos os continentes[30]. Ao longo desta história, a mesma passou por constantes evoluções, permitindo uma variedade de aplicações não antes previstas por Tim Berners-Lee, as quais compõem o momento atual da web. A **Erro! Fonte de referência não encontrada.** apresenta a distribuição dos usuários de internet por continente em 2009.

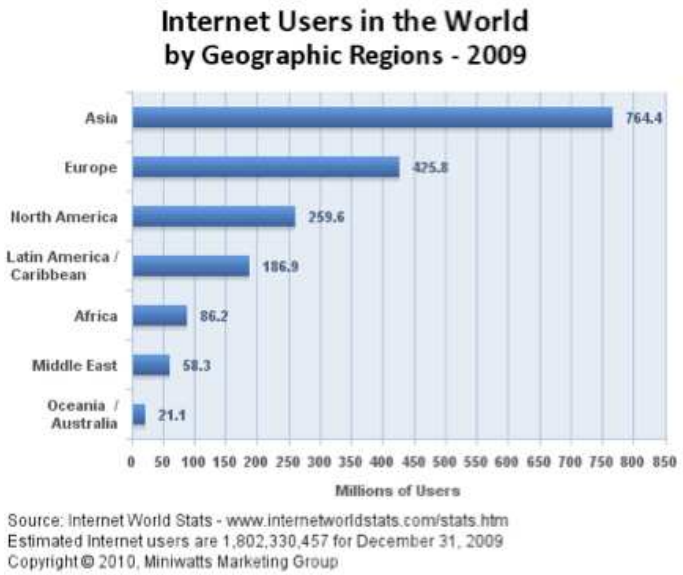

Figura 3.1 - Distribuição dos usuários de internet por continente em 2009

Adicionalmente, de acordo com um estudo do *The Economist*[31] no início de 2010 estimava-se que a quantidade de informação gerada no planeta, no ano de 2010, seria duas vezes maior do que o armazenamento disponível para guardá-la, caracterizando assim um "*dilúvio informacional*". Isto faz com que haja um grande aumento na demanda de volume de recursos para o apoio as tarefas de elicitação, compartilhamento, manipulação e exploração de grandes conjuntos de dados, bem como para o desenvolvimento e execução de serviços de apoio à Pesquisa.

A *McKinsey Quartely*[32] [Bughin et al., 2010], jornal de negócios da *McKinsey & Company*, divulgou artigo em que relaciona 10 tendências de tecnologia que devem estar no radar dos estrategistas das empresas para geração de novos modelos de negócios. O artigo faz um alerta aos altos executivos das empresas sugerindo que eles devem pensar estrategicamente sobre como preparar as organizações para o novo ambiente desafiador que se aproxima com o crescimento de tecnologias como computação em nuvem, internet das coisas, colaboração em escala entre outras.

[30] URL: http://bit.ly/MM7XnF, Acessado em 05/07/2012
[31] URL: http://econ.st/MM7YrC, Acessado em 05/07/2012
[32] URL: http://bit.ly/9stJjH, Acessado em 05/07/2012

Ainda segundo o artigo da *McKinsey*, a tecnologia continua a evoluir rapidamente. *Facebook*[33], em pouco mais de dois curtos anos, quintuplicou de tamanho a uma rede que atinge mais de 500 milhões de usuários. Mais de 4 bilhões de pessoas ao redor do mundo já utilizam telefones celulares, e para 450 milhões de pessoas a Web é uma experiência completamente móvel. A maneira como utilizamos tecnologia, como Computação em Nuvem (*Cloud Computing*) e virtualização, redistribuem os custos de tecnologia, criam novas formas para os indivíduos consumirem produtos e serviços e torna possível que empresários e empresas pensem em novos modelos de negócios que muitos consideram impossíveis como, por exemplo, marketing contextual.

A *McKinsey* também alerta que as empresas não devem se limitar a compreender as 10 tendências abaixo descritas. Elas precisam também pensar estrategicamente sobre como adaptar a gestão e estruturas organizacionais para atender as novas demandas proporcionadas por essas 10 tendências, pois muitas dessas tendências exigirão a necessidade de atravessar fronteiras organizacionais tradicionais fazendo com que líderes criem conexões entre as equipes em diferentes e espalhados cantos da organização, buscando maior interdisciplinaridade de forma que toda a empresa compreenda a necessidade de explorar plenamente essas tendências. Isso implica em transformar as organizações em pequenos laboratórios capazes de testar mais rapidamente e aprender mais rapidamente em pequena escala e depois expandir o sucesso rapidamente.

As 10 tendências citadas pela *McKinsey* [Bughin et al., 2010] são:

1. **Distribuição estimula a Co-Criação** (*Distributed cocreation moves into the mainstream*): Inspirados pelo pioneirismo da *Wikipedia* e uma comunidade de desenvolvedores de software de código aberto, a capacidade de organizar as comunidades da Web para desenvolver , comercializar produtos e serviços de apoio passou das margens da prática empresarial para o *mainstream*, onde nos dias de hoje diversas empresas tem pautado seu negócio (ou o relacionamento com seus clientes, no mínimo) que tem como objetivo redes abertas de compartilhamento de conhecimento;
2. **Transformar a organização em uma rede** (*Making the network the organization*): Muitas empresas estão indo além fronteiras estaduais, nacionais ou até continentais, construindo redes flexíveis que se estendem além das fronteiras internas e externas da própria organização. Barreiras culturais, burocráticas ou até processuais em torno de determinados silos organizacionais podem impedir que os gestores acessem os melhores talentos em toda a organização para resolver problemas críticos com soluções criativas. Utilizando tecnologias da Web, por exemplo, para expandir o acesso a especialistas de todo o mundo, as empresas podem montar comunidades de inovação entre as unidades de negócios em silos, acelerando a prestação de serviços e melhorando simultaneamente a qualidade destes serviços e do relacionamento com todos os seus *stakeholders*;
3. **Colaboração em escala** (*Collaboration at scale*): Com o advento da Internet e da WEB 2.0, o número de pessoas que realizam trabalho, muitas vezes colaborativamente, de conhecimento (voltado para inovação e criação de riqueza em cada setor da economia) tem crescido muito mais rapidamente do que os

[33] URL: http://www.facebook.com, Acessado em 05/07/2012

trabalhadores de produção de bens. Como resultado, é crescente o interesse em tecnologias de colaboração que prometem melhorar o rendimento e a eficácia desses trabalhadores;

4. **A crescente "Internet das Coisas"** (*The growing 'Internet of Things'*): Dispositivos, eletrodomésticos, embalagens, *containers*, entre outros objetos (ou genericamente "*coisas*"), embutidos de sensores, atuadores e capacidades de comunicações, em breve serão capazes de absorver e transmitir informações em grande escala e, em alguns casos, adaptar-se e reagir às mudanças no ambiente automaticamente. Estes ativos "inteligentes" podem tornar os processos mais eficientes, fornecendo novos recursos e novos modelos de negócio das empresas;
5. **Experimentação e Grandes Volumes de Dados** (*Experimentation and big data*): O volume de dados vem crescendo a taxas nunca antes vistas, e tecnologias para a captura e análise de informações estão amplamente disponíveis a preços cada vez mais baixos. Muitas empresas estão levando o uso desses dados para novos níveis, utilizando a Tecnologia da Informação e Comunicação (TIC) para suportar a experimentação constante de negócios que orientem as decisões e para testar novos produtos, modelos de negócios e inovações na experiência do cliente. Esta tendência tem o potencial de conduzir a uma transformação radical na pesquisa, inovação e na forma como as empresas conduzem seus negócios. Isso exigirá que TIC e Negócios não sejam mais compreendidos como áreas estanques e separadas, e sim como um único problema, ou seja, TIC não é mais parte do negócio, na verdade, é necessário um modelo de negócio que utilize a TIC como instrumento do negócio;
6. **Cabeamento para um mundo sustentável** (*Wiring for a sustainable world*): Mesmo com os marcos regulatórios continuando a evoluir, gestão ambiental e sustentabilidade são temas prioritários da agenda empresarial. Além do mais, a sustentabilidade está se tornando rapidamente uma importante métrica de desempenho corporativo, que os *stakeholders*, incluindo até mesmo os mercados financeiros começaram a dar mais atenção. TIC desempenha um duplo papel neste debate: ela é uma fonte significativa de emissões para o ambiente e um fator essencial de muitas estratégias para mitigar os danos ambientais. A pesquisa da *McKinsey* mostra, no entanto , que o uso da TIC em áreas como as redes de energia inteligentes, edifícios eficientes e melhor planejamento de logística poderia eliminar cinco vezes as emissões de carbono que a indústria produz;
7. **Imagine TUDO como Serviço** (*Imagining anything as a service*): Cada vez mais podemos afirmar que TIC é serviço e não produto. Consumidores desejam pagar apenas pelo uso e, dessa forma, evitar grandes gastos e futuras dificuldades de compra e manutenção de um produto. Na indústria de TIC, o crescimento de "*cloud computing*" [Armbrust et al., 2009; Galán & Sampaio, 2009; Leavitt, 2009; Smith et al., 2009; Sun, 2009] exemplifica esta mudança, além de software como serviço (SaaS) [Turner et al., 2003], que permite às organizações acesso a serviços como, por exemplo, a gestão de relacionamento com clientes. A aceitação pelos consumidores dos serviços de *cloud* baseada na Web para tudo, desde e-mail até armazenamento de vídeos e fotos, tende naturalmente a tornar-se universal, e as companhias estão seguindo essa tendência;

8. **A era do modelo de negócios multilaterais** (*The age of the multisided business model*): O maior exemplo desse modelo de negócios é o Google, que vende publicidade oferecendo serviços de TIC que permitem a busca e organizam a informação na Web. Outro exemplo, citado no artigo da *McKinsey*, é a empresa *Spiceworks*[34] oferece gerenciamento de aplicativos de TIC para 950 mil usuários, sem qualquer custo, enquanto vende espaço publicitário para empresas de B2B que querem acesso a profissionais de TIC. Quanto mais pessoas migrarem para atividades on-line, o valor dos modelos de negócio multilaterais são ampliados por estes efeitos proporcionados pela rede, sendo que quanto maior o número de usuários, utilizando os serviços gratuitamente, mais valioso o serviço torna-se para todos os clientes;
9. **Inovando na base da pirâmide** (*Innovating from the bottom of the pyramid*): A adoção de tecnologia é um fenômeno global, e a intensidade de seu uso é particularmente impressionante nos mercados emergentes. A pesquisa da *McKinsey* mostrou que os modelos de negócios disruptivos surgem quando a tecnologia se combina com condições extremas de mercado, tais como a demanda dos clientes por preços muito baixos, pouca infraestrutura, fornecedores de difícil acesso e curvas de baixo custo para o talento;
10. **Expandir serviços públicos** (*Producing public good on the grid*): É preciso entender que o custo de transação das empresas, como o trânsito caótico para movimentação das pessoas, cresce cada vez mais tornando-se inviável do ponto de vista macroeconômico e isso tende a se agravar, uma vez que cerca de metade da população mundial vive em áreas urbanas, e essa parte é projetada para atingir 70 por cento em 2050. Políticas públicas criativas, que incorporem as novas tecnologias, poderiam ajudar a aliviar as tensões econômicas e sociais da densidade populacional. Londres, Singapura e Estocolmo têm utilizado recursos inteligentes para gerenciar o congestionamento do tráfego em seus núcleos urbanos, e muitas cidades em todo o mundo estão implementando essas tecnologias para melhorar a confiabilidade e a previsibilidade dos sistemas de transporte de massa. Da mesma forma, redes de água inteligentes (*networked water smart grids)* serão fundamentais para responder à necessidade de água limpa [Baumgarten & Chui, 2009]. A plena exploração do potencial da tecnologia na esfera pública significa re-imaginar a forma como os serviços públicos são criados , entregues e gerenciados;

Torna-se imperativo que as instituições considerem adotar tais tendências de tecnologia para sua indústria de Tecnologia de Informação e Comunicação, tanto para atender de forma efetiva sua demanda interna, quanto para tornar-se uma potência exportadora de sistemas de software que, com as características citadas, teriam qualidade e rapidez superior aos seus concorrentes.

Como uma plataforma chave de fornecimento de serviço na área de computação de serviços [Zhang et al., 2007], Computação em Nuvem oferece ambientes que permitem o compartilhamento de recursos em termos de infraestruturas escaláveis, middleware e plataformas de desenvolvimento de aplicações, algumas com alto valor agregado ao negócio. Os modelos de operação podem incluir modelos de utilidade de

[34] URL: http://www.spiceworks.com/, acessado em 28/08/2010

pagamento pelo uso (*pay-as-go*), serviços de infraestrutura grátis com plataformas de serviço com valor agregado, serviços de infraestruturas baseados em taxas composto por serviços de aplicações com valor agregado ou de serviços gratuitos para os fornecedores, mas compartilhando as receitas geradas pelos consumidores.

Tipicamente, existem quarto tipos de recursos que podem ser provisionados e consumidos através da Internet [Zhang et al., 2008]. Estes recursos podem ser compartilhados entre os usuários por meio da economia de escala. O provisionamento é uma forma de compartilhar recursos com solicitantes através da rede. Um dos objetivos principais da computação em nuvem é aproveitar a Internet ou uma Intranet para provisão de recursos para os usuários.

O primeiro tipo de recursos são os recursos de infraestrutura, que incluem o poder de computação, armazenamento e fornecimento de máquinas. Por exemplo, o Amazon EC2 oferece interface de serviço web para facilmente solicitar e configurar capacidades online [Amazon EC2, 2012]. O serviço *Xdrive Box* oferece armazenamento online para os usuários [Xdrive, 2012]. O *Microsoft SkyDrive* oferece serviço de armazenamento gratuito, com um modelo integrado *offline* e *online* que mantém a privacidade relacionada com arquivos em discos rígidos, e permite às pessoas acessar esses arquivos remotamente [Microsoft, 2012]. Na área de compartilhamento de poder computacional, a computação em Grid tomou como seu foco principal a utilização de clusterização e tecnologias de computação paralela para compartilhar o poder computacional com os outros usuários (consumidores do serviço), com base no agendamento de tarefas quando os computadores estão ociosos.

O segundo tipo de recursos em computação em nuvem são os recursos de software, incluindo middlewares e recursos de desenvolvimento. O middleware consiste em sistemas operacionais centrados na nuvem, servidores de aplicação, banco de dados, entre outros. Os recursos de desenvolvimento compreendem plataformas de projeto, ferramentas de desenvolvimento, teste e implantação além de projetos de referência baseados em outros projetos de código aberto [Zhang and Zhou, 2009].

O terceiro tipo de recursos em computação em nuvem são os recursos de aplicação. As empresas de TIC estão migrando gradualmente aplicações e dados para Internet (nuvem). As aplicações de software são entregues através do modelo de Software como Serviço (*SaaS – Software as a Service*) ou *mashups*[35] de aplicações de valor agregado. Por exemplo, o Google tem utilizado computação em nuvem para oferecer aplicações Web para comunicação e colaboração [Google, 2012]. O pacote de escritório do Google, o Google Docs, moveu gradativamente para a Web as aplicações de produtividade anteriormente existentes apenas no modelo de licenciamento por máquina. Desta forma, fica claro que desenvolvendo uma arquitetura aberta de computação em nuvem reutilizável e customizável possibilitando um efetivo ambiente de desenvolvimento de aplicações como serviço é a chave para o compartilhamento (ou distribuição) de aplicações por meio da Internet.

O quarto tipo de recursos na computação em nuvem são os processos de negócio. Algumas aplicações podem ser compartilhadas (publicadas) como serviços públicos (*utilities*), ou seja, por meio de sub-processos com baixo acoplamento ou tarefas dentro de processos de negócio de clientes. O compartilhamento de processos de

[35] URL: http://pt.wikipedia.org/wiki/Mashup, Acessado em 09/07/2012

negócios é a terceirização de aplicativos orientados a negócios que dá suporte ao reuso, composição e provisionamento de aplicações.

Neste contexto, está claro o porque de grandes empresas estarem investindo recursos de esforço neste novo modelo computacional. Principalmente as fornecedoras de serviço como Google, Amazon e Microsoft que possuem hoje suas plataformas de serviços que permitem o aluguel de sua infraestrutura de máquinas para que terceiros possam hospedar aplicações, repassando questões como administração de máquinas, controle de escalabilidade, dentre outras, para as empresas contratadas.

Atualmente, boa parte da economia mundial já é baseada em sistemas de TIC, e a demanda por software vai aumentar ainda mais nas próximas décadas [Osterweil, 2007]. Estas mudanças se darão devido a fatores como o uso de software em larga escala por meio de infraestrutura de serviços, novos tipos de aplicações de integração de negócios e novas plataformas, além de combinação de serviços oferecidos via Web e dispositivos móveis.

3.1.1 Sistemas de Informação e os modelos de computação em nuvem

Sistemas de informação compreendem pessoas, componentes físicos (hardware) e lógicos (software) que, ligados por redes de comunicação, permitem a transformação de dados brutos em informação e conhecimento. A quantidade e diversidade dos dados requerem cuidados adicionais sobre seu armazenamento, como por exemplo, segurança, confiabilidade, entre outros, além de suporte a procedimentos comuns como: backups, salvamento de arquivos e sua recuperação.

Um sistema ao utilizar o novo paradigma de computação em nuvem, o faz por meio de três modelos básicos:

a) Software como Serviço (*SaaS* - do inglês, *Software as a Service*): Neste modelo, o provedor da infraestrutura de computação em nuvem cuida de todos os aspectos relacionados à disponibilidade, manutenção e adição de novas funcionalidades para possíveis clientes. Exemplos são *Google Docs*, *OfficeLive (+)* e o *SalesForce* que fornecem versões online de editores de texto, planilhas de cálculo e demais programas de escritório;
b) Plataforma como Serviço (*PaaS* - do inglês, *Platform as a Service*): Nesta outra abordagem, o provedor oferta uma plataforma que será utilizada pela aplicação do usuários final. Exemplos: *Amazon S3*, *AllMyData*, *Windows Azure* que fornecem sistemas de armazenamento;
c) Infraestrutura como Serviço (*IaaS*, do inglês, *Infrastructure as a Service*): O provedor oferece um ambiente virtualizado e gerenciável pelo usuário. Exemplos: *Amazon EC2*, *CloudBR* que fornecem sistemas virtualizados para processamento sobre demanda.

Em qualquer um dos modelos apresentados, um propósito básico é a racionalização da utilização dos recursos computacionais por uma organização ou um conjunto de organizações. A ideia é que clientes em um modelo de computação em nuvem possam melhor administrar o dimensionamento de seus recursos de infraestrutura, software, aplicação e/ou de negócio. Neste modelo, aplicações tornam-se escaláveis de forma automática, onde os encargos relacionados ao seu correto funcionamento são cobrados de acordo com a demanda das próprias aplicações.

No caso especifico de uma plataforma oferecida como serviço no paradigma de computação em nuvem, deve ser entendida com uma infraestrutura que permita a seus usuários a utilização de serviços por meio de uma interface pública, estabelecida através de uma interface de programação - *API* (do inglês, *Application Programming Interface*). Ao utilizar esta *API*, cada participante pode utilizar e oferecer recursos para os demais integrantes da nuvem, permitindo uma melhor utilização dos recursos compartilhados, otimização de processamento, armazenamento, transferência, dentre outros fatores.

Os dados de uma plataforma em modelo de computação em nuvem podem ser distribuídos entre os vários participantes da nuvem, aumentando a sua disponibilidade, assim como otimizando o uso de recursos subutilizados. Porém, há problemas a serem enfrentados, tais como escalabilidade horizontal segurança, disponibilidade.

Em geral, a infraestrutura de uma plataforma de computação em nuvem é construída sob uma tradicional forma da topologia cliente/servidor, onde o lado servidor da nuvem é composto por um alto número de máquinas que criam a abstração de um único provedor de serviços em cluster. Apesar de ser um modelo consolidado, existem problemas associados ao custo da manutenção desta infraestrutura, bem como do consumo de banda, energia, dentre outros fatores. Como alternativa para este modelo, ao longo dos anos tem se visto o crescimento de soluções baseados em topologia ponto a ponto - P2P (do inglês, *Peer to Peer*), onde não há a figura de um host ou conjunto de hosts que assume o papel de servidor ou coordenador. Os exemplos mais conhecidos de aplicações P2P são as redes de compartilhamento de arquivos, como *gnutella* ou *e-donkey*.

Aplicações P2P lidam melhor com questões de escalabilidade e tolerância a falhas do que aquelas no modelo cliente/servidor, que também contam com um custo adicional devido à complexidade necessária para o gerenciamento das buscas e roteamento dos dados entre seus participantes.

Unir os conceitos de redes P2P e computação em nuvem cria um cenário muito interessante de pesquisa para novas aplicações. Busca-se neste casamento trazer o melhor dos dois mundos. De computação em nuvem, temos **(i)** a alta disponibilidade de informações, **(ii)** a elasticidade das aplicações, ou seja, a alocação ou liberação de recursos de forma automática, **(iii)** a garantia de serviços e **(iv)** o acesso ubíquo através da rede. De redes P2P, tem-se a descentralização e a possibilidade de serviços mais tolerantes a falhas.

3.2 Fundamentação teórica

Este projeto tem por objetivo explorar os conceitos apresentados anteriormente, principalmente os ligados a abordagens P2P, para compartilhar recursos de armazenamento que estejam ociosos e disponíveis pelos nós (*peers*) que compõem uma rede para formar uma solução de backup, sincronização e compartilhamento de dados por meio de uma plataforma de computação em nuvem. Esta solução se baseia ainda no paradigma orientado a serviço e tem por princípio os principais benefícios e práticas de reutilização de software. Doravante chamaremos de USTO.RE a plataforma confiável em computação em nuvem para a realização de backups, sincronização e compartilhamento de dados que seja eficiente, segura e totalmente distribuída.

3.2.1 Reutilização e Arquitetura Orientada a Serviços

A ideia de reutilização de software não é nova. Em 1968, durante a conferência da NATO de engenharia de software, Mcllroy [McIlroy, 1968], em seu revolucionário artigo, intitulado: "*Mass Produced Software Components*", apresentou a tese que dizia: "*a indústria de software é fracamente fundada e um aspecto dessa fraqueza é a ausência de uma indústria de componentes de software*". O autor propunha investigar técnicas de produção em massa de software, conforme algumas ideias da construção industrial.

Mcllroy idealizava, com a indústria de componentes, encontrar catálogos de rotinas padronizadas, classificadas por precisão, robustez, performance; aplicar rotinas existentes no catálogo para uma variedade de máquinas; e, ter confidência na qualidade das rotinas.

Durante cerca de quatro décadas, diversas pesquisas têm sido apresentadas e discutidas em conferências, como: *International Conference on Software Engineering* (ICSE), *International Conference on Software Reuse* (ICSR) e, em periódicos, na área de engenharia de software, tentando seguir as direções apontadas por Mcllroy.

Dentre as principais razões para a distância entre as ideias de Mcllroy e o atual estado da arte, está o atraso da indústria de software, comparado a indústria de hardware em termos de princípios de manufatura e catálogo de padrões; a mudança cultural dos desenvolvedores, que, normalmente, utilizam o verbo desenvolver, ao contrário do verbo reutilizar; o fato dos atuais sistemas de repositórios raramente considerarem processos de Engenharia de Domínio ou Linha de Produtos para o desenvolvimento dos artefatos, além de processos de garantia de qualidade antes de publicar os artefatos para reutilização.

Há muito vem se formando uma consciência na comunidade de engenharia de software de que, a fim de se obter produtos com alta qualidade e que sejam economicamente viáveis, torna-se extremamente necessário um conjunto sistemático de processos, técnicas e ferramentas. Nesse conjunto, a reutilização está entre as técnicas mais relevantes. Ao longo dos últimos anos, muitas técnicas têm sido propostas para favorecer o reuso. Dentre estas técnicas, destacam-se: a engenharia de domínio, frameworks, padrões, o Desenvolvimento Baseado em Componentes (DBC) e, atualmente, Linhas de Produto de Software que tem atraído o maior foco de pesquisa e desenvolvimento [Clements & Northrop, 2001].

Segundo Papazoglou et al. [Papazoglou et al., 2007], o paradigma de desenvolvimento orientado a serviços utiliza serviços para o suportar o desenvolvimento efetivo, com baixo custo e alta interoperabilidade de aplicações distribuídas. Serviços são entidades autônomas e independentes de plataforma que podem ser descritas, desenvolvidas, publicadas e descobertas. Os serviços realizam funções que podem variar de simples requisições, como serviços meteorológicos, até executar processos de negócios sofisticados, que requerem relacionamentos ponto a ponto entre camadas de consumidores e serviços [Erl, 2008]. Quando tratamos de serviços, o processo natural de criação dos serviços passa por uma fase inicial conhecida como modelagem de serviços.

Segundo Bell [Bell, 2008], a disciplina de modelagem é um campo de conhecimento que oferece boas práticas, padrões e procedimentos para facilitar as

atividades de desenvolvimento orientado a serviços durante o ciclo de vida de um serviço. Em geral, as atividades de modelagem orientada a serviços são realizadas antes da construção e implantação de serviços. As disciplinas de modelagem orientada a serviços identificam os processos chave nos quais o negócio e as pessoas envolvidas com o desenvolvimento devem estar engajados para produzir os artefatos de projeto e implementação.

Anos atrás, com o objetivo de analisar o mercado de engenharia de software baseada em componentes, o *Software Engineering Institute* (SEI) analisou a indústria de software, com ênfase em componentes. O estudo [Bass et al., 2000], conduzido durante o período de setembro de 1999 até fevereiro de 2000, examinou componentes de software sob a perspectiva técnica e de negócios.

A partir dessa pesquisa, um distinto conjunto de inibidores para adoção de componentes de software foi identificado. Com base nestes dados e de entrevistas realizadas, o SEI identificou os seguintes inibidores para a adoção de componentes, apresentados aqui em ordem decrescente de importância:

1. Carência de componentes disponíveis;
2. Carência de padrões estáveis para a tecnologia de componentes;
3. Carência de componentes certificados; e
4. Carência de um método efetivo para construção de sistemas baseados em componentes.

Com o crescimento do paradigma de desenvolvimento de software orientados a serviços, podemos perfeitamente fazer uma associação destes inibidores para o reuso de componentes, com a reutilização de serviços, uma vez que os próprios serviços podem ser vistos como componentes de software (ativos) reutilizáveis.

Vale ressaltar também, que em uma recente pesquisa envolvendo o estado da arte e direções de pesquisa na área de desenvolvimento orientado a serviços, Michael Papazoglou, principal investigador da área, em conjunto com outros pesquisadores, constataram que um dos grandes desafios está na definição de processos voltados para o desenvolvimento de software orientado a serviço [Papazoglou et al., 2007]. Assim, analisando as visões das duas comunidades, tanto de reuso de software e linhas de produto, quanto de desenvolvimento orientado a serviços, pode-se identificar a relevância do tema e a oportunidade para o projeto.

Quando tratamos do desenvolvimento de software, principalmente em relação à reutilização (seja por meio do DBC ou de linha de produtos de software) um ponto chave é a arquitetura de software. No caso do paradigma orientado a serviços, novos desafios são identificados por meio das arquiteturas de software orientadas a serviços.

Uma arquitetura orientada a serviço estabelece um modelo arquitetural que visa aumentar a eficiência, agilidade e produtividade das empresas com a utilização de serviços representando os seus processos de negócio. Como uma tecnologia de arquitetura de software, uma implementação de uma arquitetura orientada a serviço pode consistir da combinação de tecnologias, produtos e *APIs* [Erl, 2008]. Web Services [Stal, 2002], uma opção tecnológica para se implementar SOA, tem sido utilizado na indústria nos últimos anos com o objetivo de construir aplicações que reutilizam serviços disponíveis na rede [Stal, 2002]. Esta tecnologia permite disponibilizar funcionalidades através de serviços, utilizando padrões conhecidos da Web, tais como, SOAP, XML e HTTP, provendo uma maior interoperabilidade entre as aplicações.

Desta maneira, o desenvolvimento de aplicações distribuídas através de serviços se torna mais rápido, aumentando a qualidade e diminuindo o custo e tempo de entrega do produto.

3.2.2 Arquiteturas Peer-to-Peer (P2P)

Os grandes responsáveis pelo impulso e popularização dos sistemas distribuídos P2P, foram o Napster [Napster, 2012] e Gnutella [Gnutella, 2012]. Além de protagonizar inúmeras questões judiciais quanto a direitos autorais e pirataria, foi através destas ferramentas que se evidenciou o potencial da tecnologia no que diz respeito a compartilhamento de recursos sem desprender maiores investimentos em hardware. Desde a época dos precursores a tecnologia sofreu diversas mudanças.

Pesquisas mostram que os sistemas tendem a sofrer um processo de descentralização contínuo, onde o primeiro grande avanço se deu com o modelo cliente-servidor. Este consiste em uma máquina central no qual disponibiliza algum serviço que é consumido por máquinas com um hardware com bem menos recursos. A segunda forma de descentralização mostra o surgimento das aplicações P2P, onde podem ser desenvolvidas sobre sua forma completamente descentralizada, denominada de pura, ou um modelo híbrido, onde se desenvolvem estruturas de controle centralizadas e a utilização de recursos descentralizados. Cada um dos modelos de descentralização possui suas vantagens e desvantagens conforme se pode acompanhar na sessão seguinte.

3.2.3 Arquiteturas Puras

As redes denominadas puras possuem como principal característica a não existência de nenhum tipo de controle central. Todo o funcionamento se dá com uso de um algoritmo descentralizado onde é possível localizar peers e/ou serviços [Schollmeier & Rudiger, 2002].

Esta localização é feita fazendo uso da técnica de enchente (flooding) [Schollmeier & Rudiger, 2002], onde a mensagem é enviada a todos os computadores diretamente ligados ao emissor e cada máquina que recebe a mensagem faz o mesmo. Para evitar que a rede entre em colapso (loop), um tempo de vida é atribuído a mensagem para que caso esta não chegue a seu destino seja descartada. Os pontos negativos aqui são:

- Algumas máquinas podem não receber a mensagem, negando assim um serviço que estaria disponível em tese;
- Há um grande volume de mensagens enviadas até que a mesma encontre a máquina de destino.

A técnica mais utilizada na implementação de arquiteturas puras que gerou um avanço significativo na área é o *Distributed Hash Tables* (DHT) [Balakrishnan et al., 2003]. Utilizada nas seguintes ferramentas: *pStore* [Oliveira, 2007], *Pastiche* [Mattos, 2005] e *Oceanstore* [Kubiatowicz et al., 2000], *PeerStore* [Landers et al., 2004] e *BitTorrent* [BitTorrent, 2012]. As DHTs pertencem à classe de sistemas distribuídos descentralizados e oferecem recursos de localização similar às *hash tables* (chave, valor). Um par de chave e valor é armazenado na DHT e qualquer participante da rede pode acessar o valor desejado apenas informando a chave associada. As primeiras

quatro especificações de *DHTs* (*Pastry* [Rowstron & Druschel, 2001], *Chord* [Stoica et al., 2001], *CAN* [Ratnasamy et al., 2001] e *Tapestry* [Zhao et al., 2004]) surgiram quase simultaneamente no ano 2001, depois disso, sua utilização se popularizou em aplicações destinadas ao compartilhamento de arquivos na internet. Um estudo mais aprofundado sobre as diferentes implementações de DHT pode ser encontrado em [Balakrishnan et al., 2003]. As *DHTs* possuem como principais características:

- Descentralização: os próprios nós criam e mantêm o sistema, sem a necessidade de um servidor;
- Escalabilidade: o sistema suporta a participação de um crescente número de nós simultaneamente;
- Tolerância a erros: o sistema deve ser confiável, mesmo com nós entrando e saindo continuamente da rede.

Para alcançar os objetivos supracitados, as redes *DHTs* utilizam a técnica de que um nó na rede deve estar em contato direto com apenas uma fração de todos os nós participantes. Isso reduz o custo de manutenção quando um nó entra ou sai do sistema.

Para armazenar um arquivo numa *DHT*, primeiro se calcula uma chave (geralmente o código *hash SHA-1* [FIPS, 1995] do seu nome ou do seu conteúdo), em seguida esse arquivo é enviado para a rede até ser encontrado o conjunto de nós responsáveis por seu armazenado. Para recuperá-lo, uma mensagem é enviada informando a chave do arquivo desejado, essa mensagem é encaminhada até um nó que possua o conteúdo desejado e então o mesmo é enviado como resposta. A seguir descreveremos uma das implementações de *DHT* mais utilizadas, o *Chord* [Stoica et al., 2001].

3.2.4 Arquitetura Chord

A implementação de *DHT* utilizando *Chord* se destaca pela sua simplicidade em oferecer uma única operação em que dada uma determinada chave, ela será mapeada para um nó na rede. Segundo Stoica [Stoica et al., 2001], sua arquitetura foi projetada para se adaptar facilmente a entrada e saída de novos *peers* na rede. Embora seja uma implementação que possui apenas uma tarefa (associar chaves a nós da rede), o *Chord* possibilita ao desenvolvimento de aplicações p2p uma série de benefícios:

- Balanceamento de carga, as chaves são distribuídas igualmente entre os nós da rede;
- Descentralização, nenhum nó é considerado mais importante que outro;
- Escalabilidade, o uso do *Chord* é viável mesmo com uma grande quantidade de nós;
- Disponibilidade, ajuste de sistema automático a entrada e saída de novos nós, fazendo que um nó sempre esteja visível na rede; e,
- Flexibilidade, não é necessário seguir nenhuma regra para o nome das chaves.

Para atribuir chaves aos nós, o *Chord* usa uma variação de *consistent hashing* [Karger et al., 1997] que cuida do balanceamento de carga, uma vez que cada nó tende a receber naturalmente um mesmo número de chaves. Em trabalhos anteriores ao *Chord* usando *consistent hashing* se assumia que cada nó possuísse conhecimento de todos os

outros, diferentemente, no *Chord* cada nó precisa ter conhecimento de apenas uma fração dos outros nós na rede.

Supondo uma rede de n nós, um *peer* mantém informações sobre *O (log n)* nós, e para encontrar um determinado nó na rede basta que ele possua apenas uma referência válida.

3.2.5 Arquiteturas Híbridas

Em alguns sistemas P2P é necessária a identificação dos peers conectados na rede. Para tal, sistemas como o *OurBackup* [Oliveira, 2007], que fazem uso de redes sociais para backup, é utilizado em sua arquitetura um servidor responsável pela autenticação dos usuários, manutenção da rede e dos *metadados* onde as cópias estão armazenadas. Pode-se ressaltar que a utilização de servidores não é obrigatória para a localização dos *peers* e dos *metadados*, podendo essa ser feita utilizando as DHT mencionadas anteriormente.

Nesses sistemas, o papel do servidor está em oferecer uma interface aos *peers* da rede com diversas operações tais como: autenticação do usuário; manipulação dos dados armazenados por outros *peers*, adicionar, remover, excluir, atualizar; manipulação dos usuários cadastrados no sistema; localização dos *peers* e relacionamento entre eles; dentre outras tarefas que venham a atender os requisitos do sistema. Em todos os casos geralmente é verificado se o cliente possui permissão para executar tais operações.

Essa centralização de informações pode trazer prejuízos de escalonamento no sentido de que sempre vai existir uma exigência maior do servidor à medida que se aumenta o número de requisições, usuários, *metadados* ou quaisquer outras informações delegadas ao serviço. Porém, sistemas como o *eDonkey* [Aidouni et al., 2009] se mostraram bastante eficientes quanto ao gerenciamento centralizado de informações dessa natureza. Se necessário esse escalonamento também pode ser resolvido com a utilização de clusters ou grids no lado do servidor, fazendo-se mais importante a disponibilização da interface de comunicação com a máquina cliente.

3.2.6 Detalhamento de ferramentas existentes

Em um estudo conduzido pela equipe deste projeto de pesquisa, foi constatado que existem diversas ferramentas implementadas e funcionais, ou seja, que fazem backup, sincronização, compartilhamento e recuperação em um ambiente P2P. Entretanto, foi constatado que existem apenas mecanismos que aumentam a disponibilidade. Em nenhuma das ferramentas investigadas foi identificado um o mecanismo que quantifique e garanta, em termos percentuais, a disponibilidade da recuperação. As alternativas que possuíam essa características só atendiam a esta necessidade por meio da adição de um componente extra (i.e. ferramentas, suítes ou ambientes).

Vignatti [Vignatti et al., 2009] faz um levantamento das soluções existentes e propõe um mecanismo de armazenamento digital a longo prazo baseado na escolha de repositórios confiáveis. Em essência a contribuição do trabalho é a preservação do dado, ou seja, que o mesmo não se corrompa durante sua transmissão, armazenamento, falha de hardware, entre outros. Segundo o autor muito pouca ênfase foi dada à recuperação dos itens.

O *BackupIT* [Loest et al., 2009] é uma ferramenta completa e funcional que faz uso de um mecanismo denominado de *Byzantine Quorum Systems* [Malkhi & Reiter, 1997] para controlar integridade e disponibilidade. No entanto, apenas há um aumento das chances de uma recuperação bem sucedida.

Colaço et al. [Colaço et al., 2008] especificou e desenvolveu um mecanismo de priorização dos arquivos mais usados por um usuário para diminuir o tempo que o sistema fica indisponível para o mesmo. A ideia central é que os arquivos mais importantes sempre estejam com uma disponibilidade maior que os demais.

O *pStore* [Batten et al., 2002] combina um algoritmo de roteamento chamado *Distributed Hash Table* (*DHT*) com quebra de arquivos em tamanhos fixos e uma técnica de controle de versão. A ideia é bastante completa, há suporte para encriptação de arquivos mantendo assim a privacidade nas máquinas clientes, bem como manter versões de arquivos já salvos, reaproveitando os pedaços de arquivos que se repetem em versões anteriores.

Samsara estimula a troca de recurso de maneira igualitária sem uso de uma terceira entidade no sistema para intermediar a comunicação [Cox & Noble, 2003]. O mecanismo utilizado para garantir a troca justa de recursos é um esquema punitivo para o *peer* da rede que por ventura venha a perder dados, atividade esta que é típica de usuários que querem utilizar e não disponibilizar recursos.

CleverSafe Dispersed Storange [Cleversafe, 2012] é uma aplicação *open source* desenvolvida para a plataforma *CentOS*. Faz uso do algoritmo de dispersão (*IDA*) e recuperação da informação injetada na rede. A ferramenta garante maior escalabilidade por não ter servidores centralizados e uma garantia de disponibilidade de acesso aos dados de 99,9999 (12 noves) por cento. Porém, a ferramenta faz uso de máquinas em ambientes controlados.

Dibs [Martinian, 2012] é uma ferramenta *open source* programada em *Python* que faz *backup* incremental, ou seja, de maneira inteligente o arquivo não é salvo novamente, apenas as modificações feitas no mesmo. Sua interação acontece mediante fechamento de contrato entre as partes, onde é comparado espaço mínimo disponível em disco.

Resilia [Meira, 2005] é um protótipo de sistema de backup que combina p2p com compartilhamento secreto e seguro de arquivos, faz uso de um mecanismo de replicação para aumentar a disponibilidade e permite a reconstrução de backups perdidos por falhas de nodos que compõe a rede.

Dropbox [Dropbox, 2012] é uma ferramenta de armazenamento de arquivos no qual faz uso da ideia de computação na nuvem, mantém um conjunto de servidores ligados em rede, com ambiente controlado. O salvamento dos arquivos dos usuários é feito por intermédio de um software instalado na máquina do cliente. Assim como o *CleverSafe,* o *Dropbox* também possui toda sua infraestrutura em ambiente controlado.

Disco virtual RNP [GTAR, 2012] é uma ferramenta Web que utiliza de servidores conectados montando um *Grid* de dados. O salvamento dos arquivos dos usuários é feito por intermédio de um software instalado na máquina do cliente ou Web.

Conforme pode ser constatado nesta sessão, diversos trabalhos vêm sendo desenvolvidos na área de salvamento e backup de dados distribuídos, A tabela abaixo

contextualiza de maneira mais detalhada os trabalhos levantados no estado da arte, onde foram tabuladas as principais características dos sistemas implementados para esse tipo de problema.

1. Garantia de disponibilidade: um mecanismo que quantifique e forneça estatisticamente que a restauração vai estar disponível em determinado horário.
2. Aumento de disponibilidade: mecanismo de redundância no qual aumente as chances de uma restauração bem sucedida.
3. Segurança: mecanismo de segurança dos dados salvos na rede de backup e compartilhamento.

   Inspeção dos dados: mecanismo de checagem que garante que os dados copiados em um *peer* não foram perdidos.
4. *Backup* Incremental: mecanismo no qual a similaridade entre diferentes versões é usada para salvar recursos de rede.
5. Ponto Central de controle: se o sistema teve sua arquitetura projetada para ser um sistema P2P híbrido.
6. Controle de espaço de armazenagem: Mecanismo no qual controla o espaço utilizado de *backup* e compartilhamento por usuário, evitando que o usuário utilize mais do que necessário e doe seus recursos.
7. Baseado em redes sociais: Método usado em busca de atingir maior índice de confiabilidade na rede. A proposta baseia-se em formar uma rede social de amigos (*friend-to-friend* [Li & Dabek, 2006]) e com isso evitar que seus arquivos sejam apagados.
8. Sistema de Armazenamento: Se o referido sistema tem em seu conceito ser utilizado como armazenamento ou backup.

3.2.7 Demais trabalhos relacionados

Além das ferramentas já apresentadas, várias são as soluções existentes para armazenamento de dados em nuvem no modelo de *DaaS*. Em geral, tais soluções são proprietárias e fechadas, o que dificulta uma análise técnica mais minuciosa sobre cada proposta. Analisando algumas das principais soluções existentes, como a *Amazon S3*, o *Megastore*, o *MSFSS* e o *Hadoop Distributed File System* (HDFS), nota-se que invariavelmente, todas utilizam uma estratégia comum baseada na replicação dos dados em diversos servidores, onde o número de servidores envolvidos na replicação de dados, varia de 3 a 7 vezes dependendo da solução. Esta replicação forçada é feita porque não há automação nos processos de replicação que possa garantir 99,9999999\% de disponibilidade [DeCandia et al., 2007].

O *Amazon S3* (*Simple Storage Service*) [Amazon, 2012] é o sistema de armazenamento por trás de muitos dos serviços da Amazon.com como o *Dynamo* [DeCandia et al., 2007], que recentemente teve seu banco de dados No-SQL lançado com o nome de *DynamoDB*[36]. O *DynamoDB* possui uma política de *backup* onde são feitas no mínimo três cópias do mesmo dado, onde duas são feitas na mesma zona e

[36] URL: http://bit.ly/xBc8mD, Acessado em 05/07/2012

uma terceira em uma zona externa, de modo a aumentar a disponibilidade da informação. Segundo dados de 2009, o sistema armazenava cerca de 40 bilhões de arquivos de 400.000 clientes. Seus desafios incluíam garantia de disponibilidade e o gerenciamento de falhas.

Outra plataforma, o *Megastore*, é um sistema desenvolvido com foco nos serviços *online* interativos [Baker et al., 2011]. Ele foi desenvolvido e é usado pela Google há muito tempo e manipula mais de 3 bilhões de escritas e 20 bilhões de transações de leitura diariamente, além de também armazenar um valor próximo a um *petabytes* de dados primários nos seus *datacenters* espalhados globalmente.

Já o *MSFSS* é um sistema de arquivos distribuído de alta escalabilidade e flexível, desenvolvido para armazenar uma grande quantidade de pequenos arquivos [Yu et al., 2007]. A sua arquitetura é dividida em três componentes principais: *single master*, *storage nodes* e os *metadata servers*. Todo arquivo armazenado no sistema recebe um identificador de 128 bits (*FID - File ID*) gerado a partir da data de criação do arquivo. Esses arquivos são armazenados no sistema de arquivos local de cada *storage node*. Dentre seus requisitos, o *MSFSS* dá suporte à replicação de dados, assegura a consistência entre as réplicas e executa rápida sincronização para identificar réplicas obsoletas. O *FSI* (*File System Interface*), biblioteca que dá acesso ao sistema para os clientes externos, aloca uma grande quantidade de *FIDs* em processos *batch* e os deixa armazenados em memória local, para uso posterior [Yu et al., 2007]. Por padrão, o *MSFSS* armazena duas réplicas por arquivo, mas esse valor pode ser configurado para garantir maior disponibilidade e confiabilidade. Ele usa qualquer uma das réplicas no processo de leitura. Já no processo de escrita, todas as réplicas devem ser atualizadas automaticamente. Para minimizar a latência, os dados são armazenados próximo ao usuário e as réplicas próximas umas das outras. Ele separa os grupos por regiões e cria 3 ou 5 réplicas para os *datacenters*.

O *HDFS* é um sistema de arquivos utilizado pelo *Hadoop* [Shvachko et al., 2010] e seus projetos relacionados. *Hadoop* é um *framework* para análise e transformação de grande quantidade de dados usando *MapReduce* [Dean & Ghemawat, 2008]. Uma das principais características do *Hadoop* é o particionamento de dados e a computação dos mesmos utilizando milhares de *hosts*. O *cluster* do *Hadoop* no *Yahoo!*, por exemplo, alcançou 25.000 servidores (com *cluster* de até 3.500 servidores) e armazenavam 25 *Petabytes* de dados [Shvachko et al., 2010].

Os trabalhos aqui apresentados apresentam como característica comum a replicação excessiva de informação. Este fato reside na necessidade de se garantir uma alta disponibilidade dos dados, aumentar a confiabilidade e desempenho da recuperação de informações. Porém, o uso de réplicas não só traz benefícios. Elas criam um tráfego extra de dados na rede, tráfego este que pode ser tão excessivo ao ponto de se tornar o principal gargalo de uma aplicação, além de gerar um custo da banda tão alto que pode tornar a aplicação inviável [Yang et al., 2006].

Nestas soluções, infere-se que as mesmas exigem a aquisição de infraestrutura dedicada para prover o serviço e para garantir a réplica dos dados. De forma a melhorar este cenário, o objetivo do usto.re é permitir a criação de uma nuvem para armazenamento de dados utilizando tecnologia P2P que se baseia na disponibilidade de cada *peer* para, dinamicamente, criar federações e definir a quantidade de replicações de

cada *chunk*[37] de cada arquivo. Esta abordagem permite que em ambientes onde os *peers* tenham maior disponibilidade (taxa menor de falhas), se tenha uma replicação menor e, no caso de um ambiente mais dinâmico como a Intranet de uma empresa, se tenha uma replicação maior. A partir da próxima seção, este artigo detalha nossa proposta.

Tabela 3.1 - Tabela comparativa entre as soluções de backup/armazenamento p2p

	pStore	Pastiche	OurBackup	OceanStore	PAST	BackupIT	Samsara	Resilia	Clever Safe	Dibs	Drop box	Disco RNP
Garantia de disponibilidade												
Aumento de disponibilidade	x	x	x	x	x	x	x	x	x	x	x	x
Segurança	x	x	x	x	x	x	x	x	x	x	x	x
Inspeção dos dados				x		x	x	x	x			
Incremental	x	x	x				x	x		x	x	
Ponto Central de controle			x									x
Controle de espaço de armazenagem		x	x	x	x		x		x		x	x
Baseado em redes sociais			x			x						
Sistema de Armazenamento				x	x				x		x	x

3.3 USTO.RE: Um Sistema P2P confiável para Data Cloud

Como vemos, o principal desafio em se implementar um sistema que realize o armazenamento e backup remoto é garantir que o os dados estarão disponíveis, pois em todas as soluções analisadas existe a possibilidade de falhas acontecerem e os servidores estarem indisponíveis, tornando a replicação e a montagem de infraestruturas redundantes e consideravelmente custosas como a única solução. Neste caso, se

[37] *Chunk* é um fragmento de informação, ou seja, uma pequena "parte" de um arquivo armazenado. Arquivos são divididos em *chunks*, que por sua vez, são replicados em pontos de armazenamento na infraestrutura da nuvem.

utilizarmos os conceitos trazidos por sistemas P2P e computação em nuvem podemos implementar um sistema de backup menos custoso desde que consigamos resolver o problema dos dados estarem disponíveis quando os usuários solicitarem.

Sendo assim, uma abordagem que permite solucionar este problema é replicar os dados em outros *peers* de forma a utilizar seus recursos ociosos sem haver a necessidade de se adquirir novos recursos.

Consequentemente temos que definir a quantidade de *peers* em que devemos replicar os dados de forma que os mesmos estejam disponíveis quando for necessário realizar a sua recuperação. Se conseguirmos prever a probabilidade de falha/indisponibilidade de um *peer* na rede poderemos calcular a quantidade utilizada de *peers* para a replicação de dados. A formalização matemática desta solução está descrita abaixo:

$$f(t)=\lambda e^{-\lambda t}$$

Equação 1: Equação exponencial

$$\lambda = \frac{1}{MTTF}$$

Equação 2: Taxa de falhas

Onde λ é a taxa de falhas e MTTF é o tempo médio entre falhas. A probabilidade de falha para função exponencial no intervalo de 0 a t é dado pela equação F(t):

$$F(t) = \int_0^t \lambda e^{-\lambda t}\, dt = 1 - e^{-\lambda t}$$

Equação 3: Probabilidade de Falha

A **confiabilidade**, ou seja, probabilidade do sistema não falhar num instante de tempo t pode ser obtido pela função R(t), fazendo o complemento da probabilidade de falha:

$$R(t) = 1 - F(t)$$

Equação 4: Formulação geral da confiabilidade

Substituindo a equação 4 em 5, se tem:

$$R(t) = 1 - \left(1 - e^{-\lambda t}\right) = e^{-\lambda t} \quad (1.5)$$

Equação 5: Confiabilidade no modelo exponencial

Uma vez que se têm as taxas de falhas contabilizadas nos módulos, *Avaibility* e *SeftAvaibility* pode-se, a partir da equação 6, saber a probabilidade de um *Peer* não falhar em determinada hora. É sabido ainda que várias máquinas podem estar ligadas e que há um número mínimo de máquinas configurável no qual um pedaço tem que ser distribuído. Para se obter um resultado geral do algoritmo, é necessário fazer as somas das probabilidades. Meyer [Meyer, 1995] define a soma das probabilidades como sendo:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Equação 6: Soma das probabilidades

Por indução matemática em 7, se chega à equação geral de soma de probabilidade:

$$P(A_1 \cup A_2 \cup \ldots \cup A_K) = \sum_{i=0}^{K} P(Ai) - \sum_{i<j=2}^{k} P(Ai \cap Aj) + \sum_{i<j<r=3}^{k} P(Ai \cap Aj \cap Ar) + \cdots + (-1)^{k-1} P(A_1 \cap A_2 \cap \ldots \cap A_K)$$

Aplicando em conjunto as equações 6 e 7 o algoritmo consegue calcular a probabilidade de um conjunto de máquina não falhar em determinado horário e com base neste calculo definir com um percentual de confiança de que a mesma esteja ligada e funcionando.

Uma vez que já foram abordadas as entradas, saídas e a fundamentação matemática do algoritmo, na sessão seguinte será apresentado o processamento de toda a informação. Foi optado por deixar o código da forma como foi implementado por ter sido usado assim na prova de conceito.

3.3.1 Arquitetura do sistema

A arquitetura do usto.re foi especificada tendo como meta atingir um conjunto de atributos de qualidade peculiares aos sistemas de armazenamento distribuído e que atendesse aos principais benefícios oferecidos pelas aplicações P2P, a saber:

- **Escalabilidade**: atendendo a possibilidade de explorar recursos de hardware de um grande número de máquinas (*hosts*) conectados à rede, principalmente por meio do uso racional de recursos ociosos em grandes corporações.
- **Otimização** de iterações (troca de mensagens): a "distância" entre os *peers* que interagem no sistema tem impacto no desempenho geral na latência das interações individuais, desta forma a carga do tráfego da rede também sofre um impacto negativo por esta latência. Neste contexto, a escolha pela estratégia do uso de federações visa agrupar os *peers* relacionados e reduzir esta latência.
- **Disponibilidade**: sistemas P2P são baseados em computadores livres para se "unir" ao sistema a qualquer momento, bem como "sair" dele. Além disso, as conexões não são gerenciadas pelo sistema ou alguma autoridade que garanta a conectividade e a qualidade do serviço, neste contexto, uma estratégia de garantia de disponibilidade do serviço deve ser implementada considerando as características básicas de um sistema de armazenamento com alta disponibilidade, bem como as restrições que uma rede P2P impõe.
- **Segurança dos dados**: em se tratando de armazenamento de dados, políticas de segurança e proteção devem ser efetivamente adotadas de modo a garantir a privacidade e autenticidade dos usuários e dados do sistema.

Para detalhar o funcionamento do usto.re, esta seção apresenta uma visão geral da arquitetura do USTO.RE, seus os principais componentes, dependências e relacionamentos, conforme ilustrado na Figura 2. O USTO.RE é composto por um conjunto de cinco componentes, dispostos em uma arquitetura P2P híbrida estruturada e em três camadas. São eles: **(i)** *Super Peers Rendezou Relay* ou simplesmente *Super Peers*, **(ii)** Servidores, **(iii)** *Proxies*, **(iv)** Bancos de Dados Relacionais e Não-SQL e **(v)** *Simple Peers*. Estes componentes possuem funcionalidades distintas e interagem como um sistema distribuído descentralizado híbrido, de modo semelhante a uma rede P2P, onde cada nó realiza tanto funções de servidor quanto de cliente. Os componentes agrupam-se dinamicamente como federações de dados, onde os grupos são montados de modo a minimizar a troca de mensagens no sistema.

A Figura 3.2 apresenta o diagrama de sequencia do USTO.RE, onde vemos de maneira geral o processo de registro de *peer*, localização de serviços e como um *simple peer* obtém os dados.

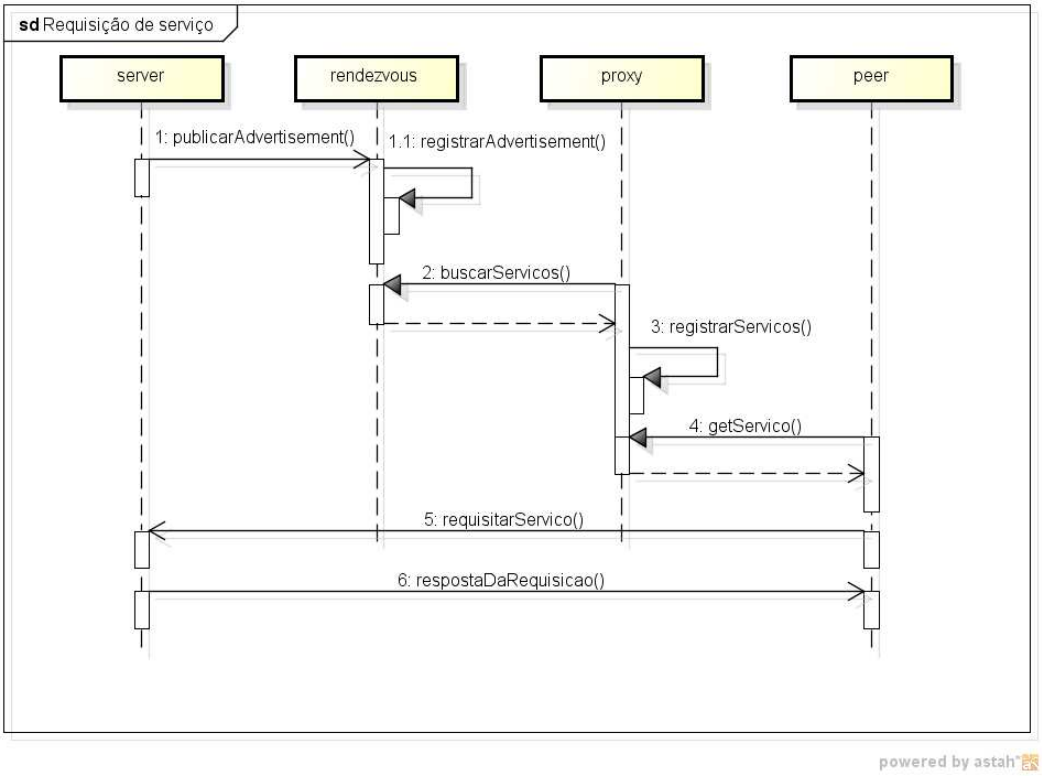

Figura 3.2 - Diagrama de sequencia USTO.RE

Na **Erro! Fonte de referência não encontrada.**.3 é apresentado este mesmo processo porém com a visão de processos isolados rodando em servidores com papeis específicos.

A organização do sistema nesta arquitetura multicamadas possibilita a distribuição do processamento, uma vez que os componentes estão fisicamente distribuídos. Entretanto, por se tratar de uma arquitetura híbrida P2P estruturada e multicamadas, o sistema possui uma distribuição dita horizontal. Nesta distribuição horizontal, em uma rede P2P, um cliente ou um servidor podem estar fisicamente divididos em partes logicamente equivalentes, onde cada um opera sobre a sua própria porção dos dados, o que propicia um balanceamento da carga.

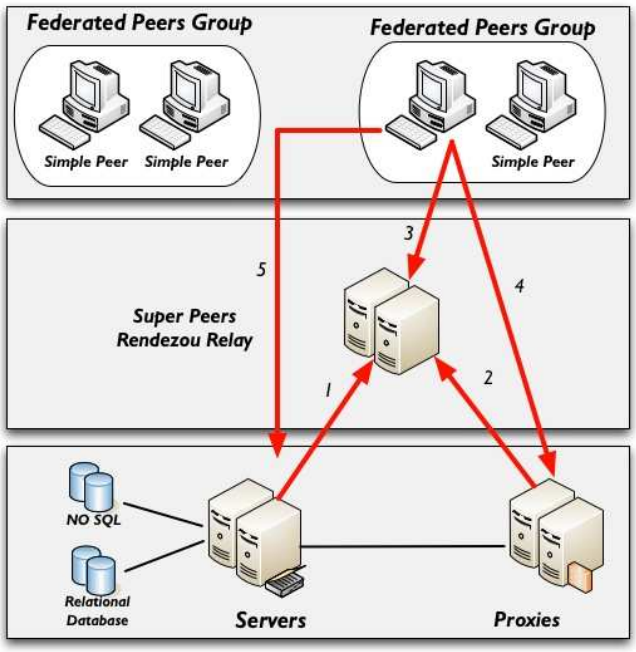

Figura 3.3 - Arquitetura do USTO.RE

Na Figura 3.4, é apresentado em detalhes cada um dos componentes de arquitetura do USTO.RE, separando cada um deles entre interfaces, bibliotecas compartilhadas, componentes da rede P2P e interfaces de consumo de serviços. Nas subseções a seguir, maiores detalhes sobre cada componente são fornecidos e para os componentes principais será acrescido o detalhamento da arquitetura.

3.3.2 Super peers

Os *super peers* funcionam como elementos de referência para os demais componentes da arquitetura, sendo a porta de entrada para a participação de servidores, *proxies* e *simples peers* no sistema. O papel do *super peer* é definir as federações de dados quando cada *peer* solicita conexão à rede. Para isso, os *super peers* devem ter sua localização previamente conhecida por todos os demais *peers* por meio de uma pré-configuração. Consequentemente, eles são os primeiros componentes a serem inicializados para o correto funcionamento do usto.re. Também como consequência, um *super peer* guarda informações sobre todos os servidores, *proxies* e *simple peers*, agrupando-os dinamicamente de acordo com o perfil de cada *peer*, a ser explicado adiante.

Também é papel deste tipo de *peer*, escolher dinamicamente os *peers* e servidores das federações baseando-se em um algoritmo de proximidade [Duarte, 2010]. O agrupamento em federações permite o crescimento elástico e garante a escalabilidade do sistema, pois não existe um limite para a quantidade de federações que podem ser criadas. Crescimento elástico é a característica do sistema de crescer ou diminuir, em termos de capacidade e consumo de recursos, de maneira dinâmica e não intrusiva. Os *peers* se comunicam com a rede P2P através do protocolo *JXTA* [JXTA, 2012] e opcionalmente podem ofertar uma interface de serviço *REST* [Webber et al., 2010] para permitir a interoperabilidade com outras aplicações.

Figura 3.4 - Visão geral da arquitetura do sistema

3.3.3 Servidores

Os *peers* servidores são aqueles que oferecem um conjunto (ou subconjunto) de uma lista existente de serviços. Na ordem de configuração do usto.re, os servidores são os componentes que devem ser executados logo após a inicialização dos *super peers* (Passo 1 na Figura 3.2). Os *Super peers* estabelecem um esquema de sincronização fazendo com que a lista de servidores em cada um deles seja atualizada quando da entrada ou saída de um *peer* servidor.

A definição de *peers* com funcionalidades especificas na rede opõem-se à algumas propostas para sistemas P2P, onde cada *peer* deveria ser capaz de desempenhar todos os papéis no sistema, promovendo a ideia de uma *DHT* (*Distributed Hash Table*) completa [Oliveira, 2007; Loest et al., 2009]. No entanto, a implementação utilizando uma DHT em sua essência é bastante custosa e de difícil escalabilidade. Sendo assim, a proposta adotada no usto.re é a criação de níveis hierárquicos que implementam serviços bem definidos e que podem crescer horizontalmente. Dentre os serviços disponibilizados pelos servidores, pode-se citar:

- ***Autenticação***: usado para que cada *peer* se autentique;
- ***Disponibilidade***: permite checar a disponibilidade de cada *peer*;
- ***Chunk***: usado para o controle de *chunks*;
- ***Erro***: permite o controle de erros como, por exemplo, um serviço indisponível;
- ***Controle de Saída***: controla a saída voluntária de um *peer*, quando este desconecta-se voluntariamente da rede;
- ***Gerência de Diretórios***: utilizado para armazenamento e recuperação de diretórios inteiros;
- ***Gerencia de Arquivos***: utilizado para armazenamento e recuperação de arquivos;
- ***Busca***: procura por um conjunto de *peers* que obedeçam ao acordo de nível de serviço (do inglês, *Service-Level Agreement - SLA*), no caso do usto.re fortemente relacionado à disponibilidade do *peer*, para o salvamento de um arquivo;
- ***Árvore de Diretórios***: utilizado para visualização de diretórios inteiros;
- ***Segurança de Acesso***: controla a permissão de acesso aos *chunks*;
- ***Rastreabilidade***: mantém uma lista de usuários e arquivos a ser consultada quando a recuperação de um arquivo é solicitada.

Como se pode observar na Figura 2, os *peers* servidores utilizam dois tipos de banco de dados para manter a consistência do sistema. Um banco de dados tradicional relacional contém dados dos usuários do sistema, e um banco de dados No-SQL [Chang et al., 2006] que permite um crescimento horizontal e a recuperação mais rápida das informações relacionadas aos arquivos e *chunks* salvos. A escolha desta separação se dá pelas questões relacionadas ao desempenho do sistema, visto que, com o aumento do volume de arquivos e *chunks* salvos, o sistema de gerenciamento de banco de dados tende tornar-se um ponto de gargalo. A utilização de um sistema nativamente distribuído torna a solução viável e escalável [Chang et al., 2006].

Todas as informações referentes à autenticação e SLA dos *peers* são salvas em um banco de dados relacional devido à a garantia da integridade provida por este tipo de banco. Já o serviço do *File Tracker*, que permite a identificação de qual *peer* possui partes do arquivo a ser recuperado, é utilizado um banco de dados No-SQL, o que permite o seu crescimento horizontal. As instâncias dos bancos, quer seja este relacional ou Não-SQL, podem ser compartilhadas entre mais de um servidor.

Um *peer* servidor pode prover um ou mais serviços da rede, sendo assim, da mesma forma que na criação de federações de dados, pode-se iniciar *peers* servidores e *proxies* sobre demanda aumentando a escalabilidade e elasticidade do sistema. No atual estágio do projeto, este aumento deve ser feito manualmente por um humano, de acordo com o monitoramento do desempenho do sistema.

O relacionamento entre cada componente da solução descrita acima com suas interfaces e conectores está apresentada na Figura 3.5 abaixo. Nela se pode observar que um dos elementos centrais do sistema são as filas de envio e recepção de mensagens. No USTO.RE toda troca de mensagem passa por uma fila que é consumida sobre demanda e de acordo com os recursos disponíveis no sistema operacional. A quantidade de filas pode ser configurada, sendo assim, o USTO.RE pode ser utilizado em equipamentos com características diferentes, garantindo que as operações sempre ocorrerão. Quando se deseja mais performance aumenta-se a quantidade de filas desde que haja recursos disponíveis no sistema operacional.

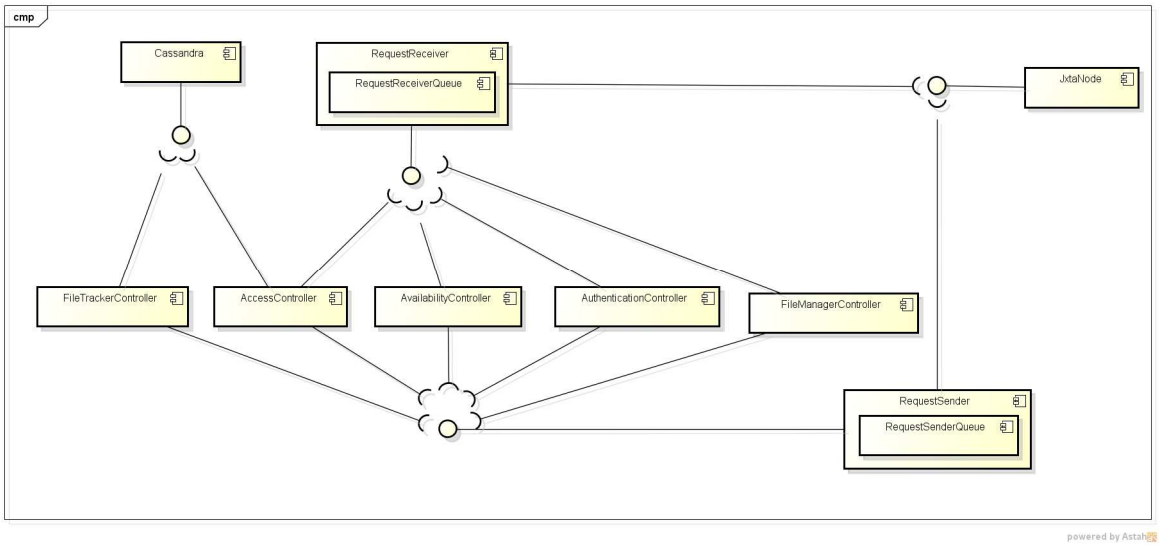

Figura 3.5 - Visão componente-conector do servidor

3.3.4 Proxies

Após a inicialização de *super peers* e servidores, o terceiro componente que precisa ser executado é o *proxy*. Cada *proxy* atua como um catálogo, um serviço de localização para serviços em execução nos diferentes servidores do usto.re. Um *proxy* ao se anunciar a um *super peer* (Passo 2 da Figura 3.2), recebe a lista de servidores registrados. Além disso, um proxy obtém a informação de quais serviços estão disponíveis em cada servidor. Desta forma, quando um *peer* deseja a informação sobre um determinado serviço, um *proxy* pode fornecer a referência para um servidor que atenda tal requisição. Desta forma, um *proxy* estabelece uma ponte de ligação entre

consumidores de serviços, tipicamente os *simple peers*, e o provedor de um serviço, no caso, os *peers* servidores.

3.3.5 Simple peers

Simple peers são aqueles responsáveis por armazenarem os *chunks* dos arquivos. Na visão alto nível da proposta, estes *peers* representam máquinas de usuários comuns que podem oferecer espaço de armazenamento para ser compartilhado entre vários usuários. Cada *simple peer* possui um perfil que define a sua disponibilidade e que lhe é atribuído quando de sua conexão com o sistema. Esta disponibilidade é relacionada ao período de tempo em que o *peer* esteja disponível para compartilhar dados. Como exemplo, um *peer* que esteja em uma *Intranet* de uma empresa pode ter em seu perfil, a disponibilidade atribuída como "***8:00 às 12:00 e 14:00 às 18:00***". Quando um *simple peer* se conecta ao sistema, ele recebe de um *super peer* a lista de *proxies* disponíveis (Passo 3 da Figura 3.2). Desta lista, o proxy busca por um serviço específico e obtém a referência de quais servidores ofertam determinado serviço (Passo 4 da Figura 3.2). Um servidor aleatório é escolhido, e o *simple peer* solicita o serviço desejado (Passo 5 da Figura 3.2). Caso um serviço não seja atendido por algum motivo, como um *timeout*, o *proxy* pode fornecer um novo servidor para o *simple peer*.

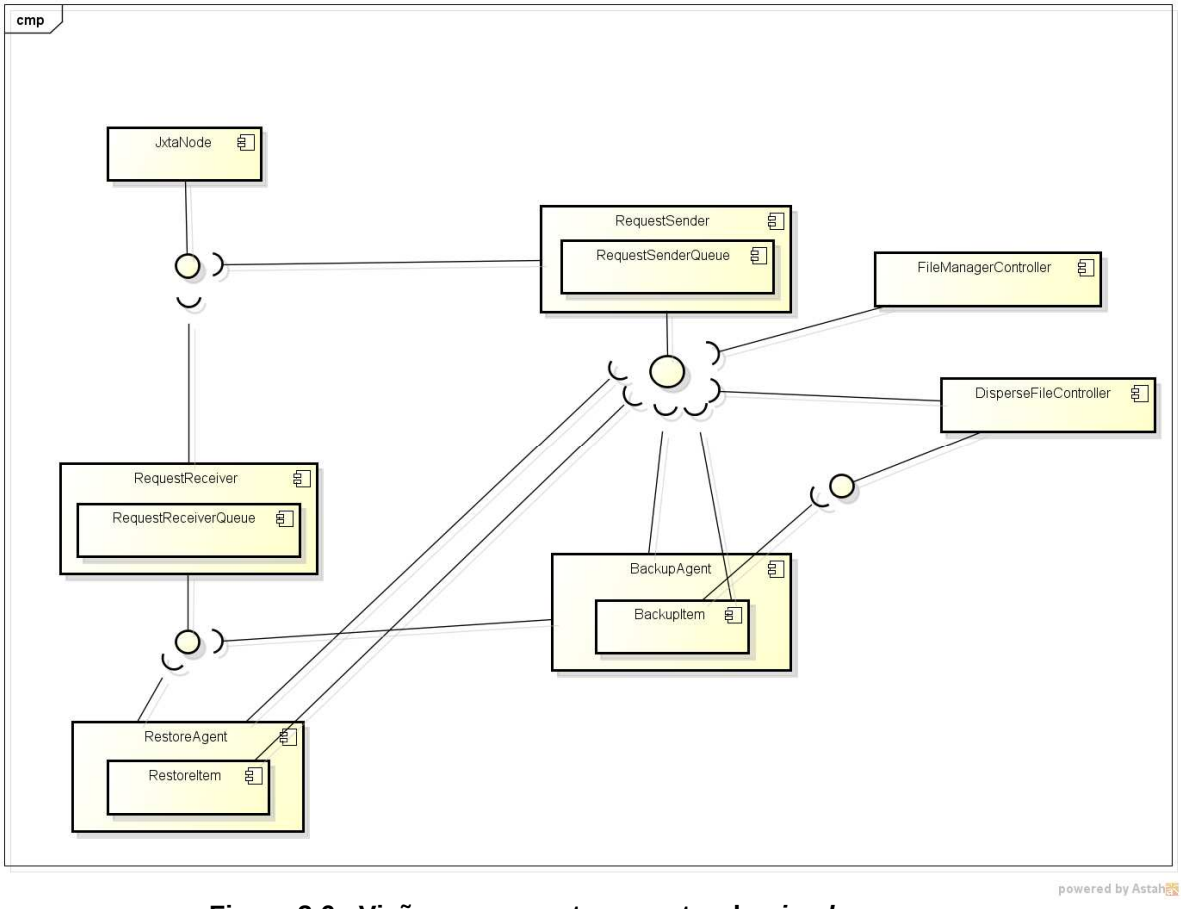

Figura 3.6 - Visão componente-conector do *simple peer*

Observando-se Figura 3.6, também se vê como elemento central do sistema o gerenciador de filas que neste caso também pode ser configurado trazendo os mesmos benefícios descritos para o servidor. Ainda como elementos centrais do sistema temos o

BackupAgent e o *RestoreAgent*. Cada componente deste representa uma ação recuperação (*restore*) ou *backup* (salvamento). Quando uma operação é solicitada uma instancia deste componente é criada e o componente é responsável por garantir que a operação solicitada seja executada. Para evitar que haja consumo elevado de memória ou recursos do sistema, apenas duas instancias de *BeckupAgent* e *RestoreAgent* pode ser executada por vez em um *simple peer.*

Como citado anteriormente, *simple peer* são agrupados em federações de dados pelos *super peers*. O objetivo de agrupá-los desta forma é minimizar a sobrecarga na rede e em cada *peer*, além de diminuir a quantidade de mensagens trocadas e permitir que uma federação desempenhe o papel de *backup* de outra federação. O agrupamento dos *peers* em federações obedece os seguintes critérios por ordem de relevância: proximidade; perfil de cada *simple peer*; latência de rede; latência da federação; georeferenciação; capacidade de cada *peer*; e, capacidade final da federação, que é definido pelos *super peers*.

Cada *peer* possui uma interface de serviço *REST* [Webber et al., 2010] que permite a autenticação do usuário, armazenamento, recuperação e remoção dos dados salvos. Tal característica apresenta como principal vantagem a possibilidade de compatibilizar o sistema com outras interfaces existentes, como *S3* da *Amazon*. Na atual arquitetura, o serviço de armazenamento dos dados pode ser modificado para outras alternativas (i.e. *Megastore*, *MSFSS* ou *Amazon S3*).

Para garantir o espalhamento de cada *chunk* de forma a garantir os níveis de serviço adequados, cada *peer* precisa periodicamente relatar seu estado atual com o objetivo de manter o SLA sempre atualizado. A **Erro! Fonte de referência não encontrada. (a)** apresenta o fluxo de funcionamento de um *peer* (*PL_1*), desde o momento em que ele anuncia o seu perfil à comunicação estabelecida com os demais pares por meio do servidor (*PS_1*). Periodicamente cada *peer* envia para um dos servidores uma mensagem de "*keep alive*" informando que está *online*. Desta forma, o servidor sabe se o *peer* está cumprindo com o perfil acordado (SLA) e o torna elegível para receber *chunks* no horário definido. Também periodicamente cada *peer* verifica com os demais *peers* existentes se os *chunks* que ele possui estão replicados na quantidade mínima de *peers* obedecendo o critério de disponibilidade exigida [Duarte, 2010]. Caso não esteja, ele replica o *chunk* em outro *peer*. Quando o *peer* que possui este *chunk* voltar a se conectar a rede, ele irá verificar que existe um excesso deste *chunk* e irá excluí-lo.

A **Erro! Fonte de referência não encontrada. (b)** apresenta o fluxo de funcionamento entre *peers* e servidores desde a conexão do *peer* local à rede, ao salvamento dos arquivos solicitados. Após o *peer* (*PL_1*) ao se conectar à rede *P2P_1*, o *super peer* indica um *peer servidor* para ele se autenticar. Com a autenticação bem sucedida, um processo de identificação dos pares para formar as federações é executado. Com a federação (agrupamento de *peer*) estabelecida, o sistema está apto a receber arquivos. Ao receber um arquivo ("*arq1.zip*"), o *peer PL_1* informa a necessidade de armazená-lo no sistema, para isto é feita uma segmentação do arquivo (em *chunks*) e estes segmentos são enviados para serem salvos na rede *P2P_1*. Em seguida, para efetuar o salvamento, uma rotina para medir a confiabilidade analisa o estado dos *peer* e, consequentemente, da disponibilidade dos segmentos do arquivo na rede e caso exista uma combinação de *peer* que atenda ao SLA para o armazenamento, os segmentos do

arquivo são enviados para estes *peer*. Se não houverem mais segmentos de arquivos a serem salvos, o *peer servidor PS_1* comunica ao *peer PL_1*, que solicitou o salvamento do arquivo, que o mesmo foi salvo com sucesso.

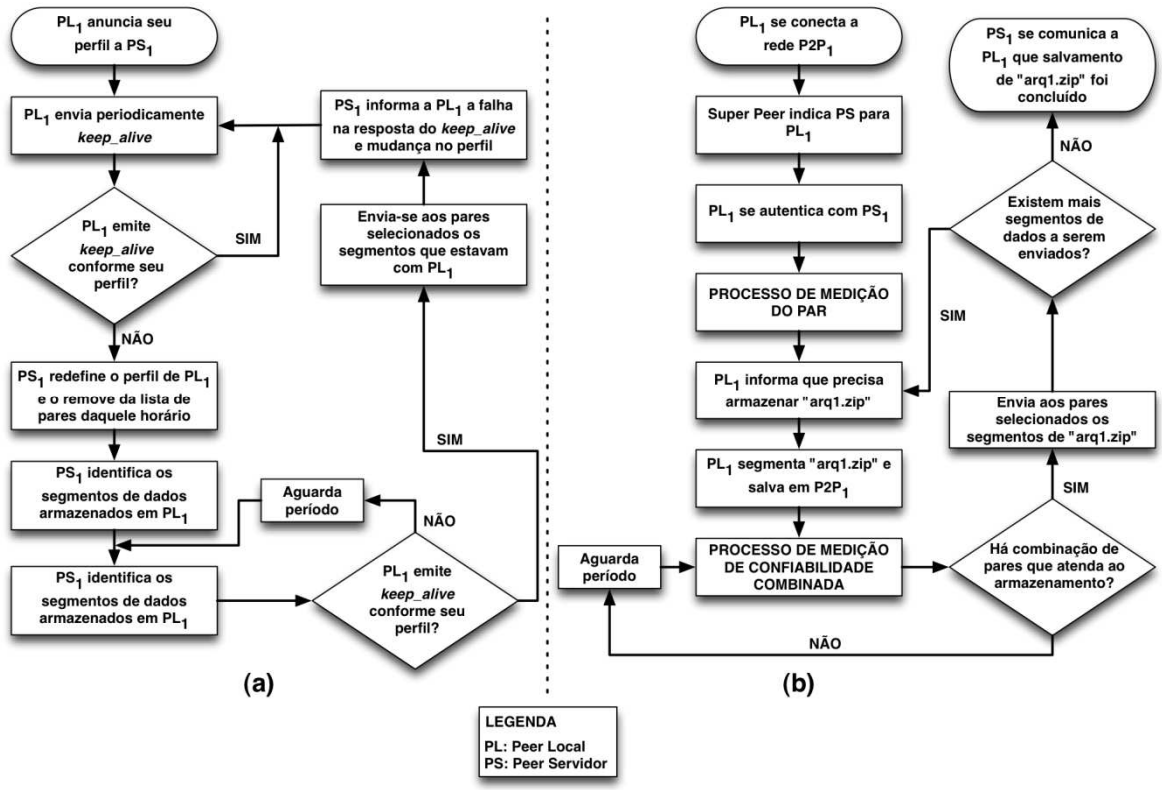

Figura 3.7 - a) Funcionamento de um *peer* b) Funcionamento *peer Server proxy*

3.3.6 Plataforma USTO.RE S3

Ainda como resultado deste projeto, utilizando os conceitos trazidos pelo reuso de software e conforme foi apresentado, pode-se enxergar a plataforma USTO.RE como sendo um sistema que permite a utilização dos recursos ociosos nos discos nos computadores que possuem o software instalado. Este sistema também pode ser visto como uma plataforma de armazenamento em nuvem. Sendo assim, se pode ofertar o uso desta plataforma como sendo um serviço (*PaaS*) na nuvem para pesquisadores que necessitam de muitos *Gigabaytes* para armazenar os dados ou backup dos seus projetos .

Neste caso será provido aos pesquisadores uma interface *SOAP/REST* compatível com as interfaces providas por outros serviços como *Allmydata*, *Amazon S3*, entre outros, de forma que seja possível utilizar os *peers* que compõe a rede como um sistema para armazenamento de dados como *PaaS*. Isto se torna possível porque a plataforma USTO.RE provê a garantia da recuperação dos dados. Neste sentido, será investigado a possibilidade de permitir aos usuários que utilizam plataforma de armazenamento a um custo mais elevado, utilizar o sistema USTO.RE S3 através de um custo mais acessível.

Pode-se afirmar que existe a diminuição de custo porque ao analisar as soluções descritas anteriormente, em todas elas existe a necessidade de se prover infraestrutura dedicada, mesmo que a baixo custo, para suportar o sistema. Neste caso, o custo diminui bastante visto que será utilizado parte do espaço disponível no disco dos usuários.

3.4 Banco de dados

Seguindo este mesmo principio a escalabilidade de banco de dados pode ser dada através da alocação de múltiplos sistemas de banco de dados em paralelo e s associando aos conceitos de sistemas multi-tenant.

Neste contexto a proposta da criação de sistemas multi-database, como adotado facebook se mostra extremamente viável.

Os objetivos deste modelo de funcionamento são:

Redistribuir e crescer sistemas de banco de dados dinamicamente sem a necessidade de alteração no código da aplicação.

a) Diminuição da carga no SGBD's
b) Crescimento elástico
c) Sistema de provisionamento baseado nos princípios de cloud computing

Como funciona:

I. A aplicação ao iniciar consulta a lista de banco de dados existentes
   a. Esta lista contém a URL de conexão do BD e qual informação ele possui
II. O pool de conexão conecta nestes bancos
III. Baseado no identificador de segmentação o sistema ao realizar o getConnection para o pool de conexão informa o identificador e baseado nesta informação o pool seleciona o DB
IV. Quando administrador do SGBD encontrar a necessidade de dividir o banco ele envia o comando através da interface de gerenciamento dos banco de dados provida pela aplicação
V. O banco de dados e colocado em read-only mode e um snapshot e feito. As consultas continuam porém novas transações serão rejeitadas
   a. Um mecanismo de tratamento de exceção precisa ser implementado/alterado para suportar o erro gerado
VI. Um novo servidor de SGBD é solicitado para o gerenciador de cloud
VII. O snapshot do BD em read-only é copiado para este novo banco
VIII. Queries de limpeza do Banco rodam para limpar os dois bancos novos.
IX. O sistema de particionamento atualiza o banco de dados com os hashes
X. O sistema de particionamento notifica a aplicação para atualizar seus hasehs.
XI. O pool de conexão referente ao servidor que foi particionado é ressetado

a. Assim as conexões existentes na aplicação gerarão um erro forçando outro getConnection a acontecer

A representação deste fluxo esta descrito na Figura 3.8

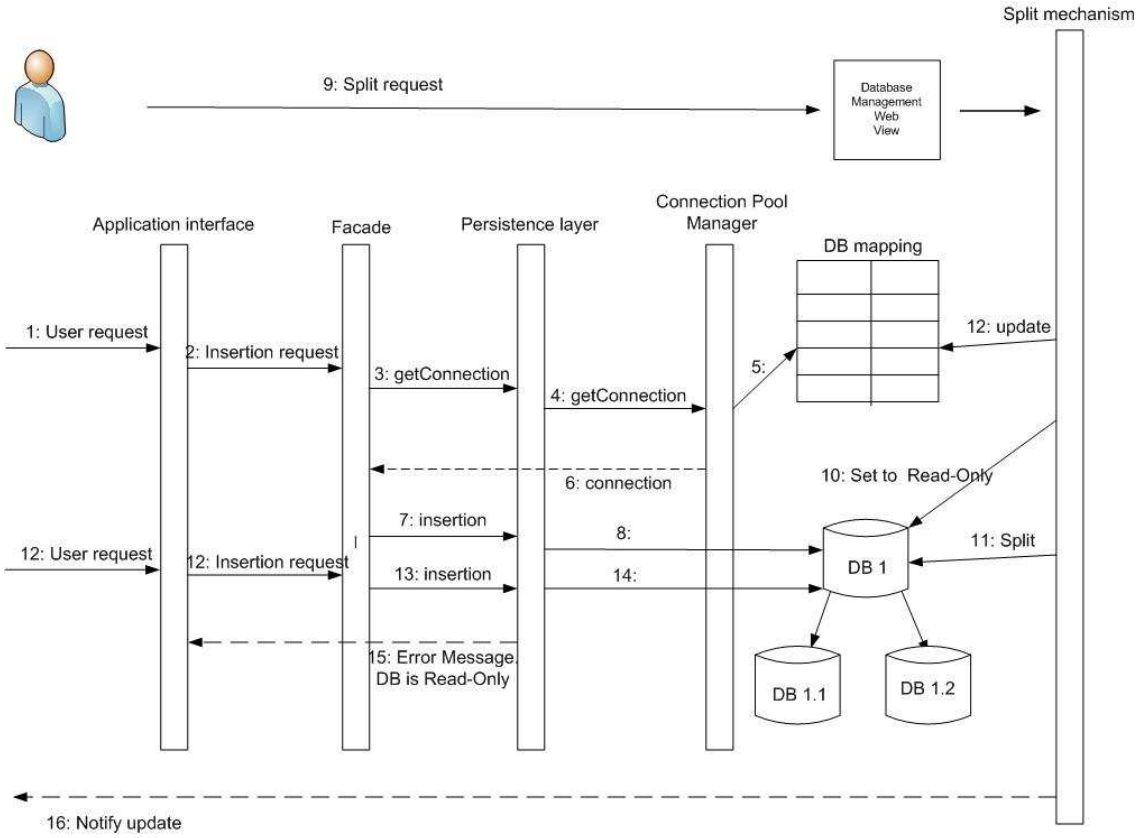

Figura 3.8 - Sistemas Multi-Database

3.5 Bigdata

Os conceitos relacionados a BIGDATA estão intrinsicamente ligados as necessidades de busca, recuperação e recomendação em uma base de dados desestruturados. Este tipo de dado, normalmente visto como um arquivo, agora esta salvo em sistemas de cloud storages como descrito nas sessões anteriormente.

Sendo assim é necessário que os mesmos conceitos ofertados por sistemas de arquivos tradicionais, como EXT4 do Linux e NTFS do Windows sejam portados para os sistemas de nuvem e a eles acrescidos informações extras que permitam a realização das operações desejadas nas plataformas de BIGDATA.

O começo da solução deste problema, esta na definição de metadados que contenham as informações mínimas de sistemas de arquivo, acrescidas de informações para a busca e recuperação, como índices e palavras chave.

3.5.1 Trabalhos Relacionados

Tanenbaum (2007) lista alguns atributos que podem compor os metadados de um sistema de arquivos para ajudar a superar os desafios existentes. Segundo o autor

supracitado, nenhum sistema existente dispõe de todos os atributos, porém cada um deles está presente em algum sistema. Na lista aparecem atributos ligados à proteção do arquivo e indicam quem poderá acessá-lo, bem como a senha necessária para isso, caso seja necessário. Há ainda flags para indicar se o arquivo é somente para leitura, oculto, se já foi feita cópia de segurança, além disso há campos para indicar datas de criação, último acesso e tamanho do arquivo.

O Microsoft Windows 8 utiliza como sistema de arquivos o ReFS[38](Resilient File System) que foi desenvolvido utilizando como base o NTFS. O ReFS utiliza árvores B+ para armazenar todos os dados e metadados do mesmo. A reutilização de código foi amplamente utilizada para manter uma alta compatibilidade com um subconjunto de funcionalidades do NTFS, como alguns atributos dos metadados (padrão de informações, nome do arquivo, valores de algumas flags, etc), interface do sistema de arquivos (leitura, escrita, abrir, fechar, modificar, notificar, etc), manter arquivos na memória, imposições de segurança e memória cache. Muitos dos atributos dos metadados utilizados no ReFS vem do NTFS, como os atributos para nome, dados, lista de controle de acesso, informações padrões (datas de criação, modificações e último acesso). Mas há atributos exclusivos do novo sistema de arquivos como o INTEGRITY_STREAM e o NO_SCRUB_DATA. O atributo INTEGRITY_STREAM é criado unicamente para gerenciar a integridade de diretórios e arquivos pertencentes a aplicativos que preferem gerenciar o armazenamento de arquivos cuidadosamente e, para isso, dependem de um determinado layout de arquivo no disco. Como os fluxos de dados realocam blocos sempre que o conteúdo de um arquivo é alterado, o layout do arquivo fica muito imprevisível para esses aplicativos. Para combater os casos de corrupções de disco e falhas no armazenamento, o ReFS periodicamente depura todos os metadados e dados em um volume do ReFS. O atributo de arquivo NO_SCRUB_DATA é utilizado para indicar que o depurador deve ignorar o arquivo. Esse atributo é útil para os aplicativos que mantém suas próprias informações de integridade[39].

O EXT4 é um sistema de arquivos de journaling para Linux desenvolvido como uma extensão para o EXT3 com o objetivo de revolver problemas de escalabilidade, desempenho e confiabilidade. O EXT3 possui usa 32 bits para representar blocos de números e tem por padrão um bloco de 4K, isso faz com que o sistema de arquivos fique limitado a 16 TB, o que hoje, devido à enorme massa de dados crescente existente, é uma capacidade de armazenamento pequena. O EXT4 ampliou o bloco de número para 48 bits, o que, em teoria, permite que o sistema de arquivos suporte até 1 Exabytes (1 milhão de TB) (Mathura, 2007). Além disso, o mesmo possui uma alta compatibilidade com versões futuras e antigas do sistema de arquivos, como o EXT3, e aprimoramentos na resolução e no intervalo do registro de data e hora[40]. Os metadados do EXT4 são compostos pelo número do inodes que o diretório aponta, tamanho do diretório, nome, tipo de dado armazenado no inode (arquivo, diretório, *socket*, *link* simbólico), além da lista de controle de acesso (que é guardada separadamente, em um *inode* especial) e atributos para a configuração do sistemas de arquivos, como o

[38]http://bit.ly/KzTYdZ
[39]http://bit.ly/KMEEQV
[40]http://ibm.co/KQQBB3

EXT4_INDEX_FL que indica se os diretórios serão armazenados em árvores B ou em um *array* linear[41].

O Hadoop Distributed File System (HDFS) é um sistema de arquivos distribuídos projetado para rodar sob hardwares comuns. Diferentes de alguns outros sistemas de arquivos, o HDFS é altamente tolerante a falhas e foi projetado para ser executado em hardwares de baixo custo. Todos os servidores são totalmente conectados e a comunicação acontece através de protocolos baseados em TCP.

Diferentemente de outras abordagens de sistemas de arquivos distribuídos, o armazenamento e processamento do HDFS é feito em cada nó do sistema. Assim como outros sistemas de arquivos, o HDFS guarda os metadados separados dos arquivos da aplicação. O metadados são armazenados em um servidor dedicado chamado NameNode e os dados da aplicação são armazenados em outros servidores, chamados DataNodes. O NameNode possui inodes, que representam os arquivos e diretórios. Além disso, os inodes guardam atributos como permissões, data de acesso e modificações, namespace onde os arquivos estão e espaço disponível em cada disco (Shvachko, 2011).

O XtreemFS[42] é um sistema de arquivos globalmente distribuído e replicado. Ele é desenvolvido como parte do projeto XtreemOS EU, o qual tem o objetivo de criar um sistema operacional open source para ser utilizado em ambientes de grid. XtreemFS é baseado em objetos, com metadados e dados armazenados em diferentes tipos de nós. Além disso, ele é compatível com o padrão POSIX, e possui sistema de tratamento de falhas. Ele replica os objetos para tolerância a falhas e faz cache dos dados e metadados para melhorar o desempenho no caso de alta latência (Hupfeld, 2007 e 2008). O XtreemFS faz a replicação a partir do arquivo completo, ao invés de utilizar as partes já divididas do arquivo. Isso faz com que haja um aumento considerável na comunicação interna para manter as réplicas sincronizadas. O XtreemFS utiliza o BabuDB como banco de dados para armazenar os metadados do sistema de arquivos. Os atributos utilizados nos metadados incluem uma identificação única para arquivos e diretórios, nome, tipo, além de atributos relacionados a data de criação e modificação, tamanho do arquivo, localização do conteúdos do arquivos, localização das réplicas[43], dono, controle de acesso e atributos estendidos.

O Google File System (GFS) é um sistema de arquivos distribuídos escalável para grandes aplicações distribuídas com uma grande quantidade de dados. Ele provê tolerância a falha e um acesso confiável, eficiente ao dados utilizando um conjunto de hardwares comuns. Este sistema de arquivos é otimizado para trabalhar com grandes arquivos, normalmente 100MB ou mais, sendo muito utilizado para arquivo com tamanho superior a 1GB. Sua arquitetura consiste de múltiplos nós, sendo eles de dois tipos: um nó Master e um grande ChunkServer. Cada arquivo armazenado é divido em chunks de tamanho fixo, os quais são armazenados no ChunkServer. Cada chunk criado é replicado para outros servidores do mesmo tipo, no mínimo, três vezes. O nó Master armazena todos os metadados associado com os chunks, como uma tabela onde há o mapeamento da localização dos chunks com os arquivos, a localização das cópias dos chunks, quais processos estão acessando os chunks (Ghemawat, 2003).

[41] http://bit.ly/LrsF6p

[42] http://www.xtreemfs.org

[43]http://www.xtreemfs.org/arch.php

O Amazon S3[44](*Simple Storage Systems*) é um de armazenamento oferecido pela Amazon Web Service que funciona através de interfaces *web services* baseadas em *REST*, *SOAP* e *BitTorrent*.

O Amazon S3 é projetado para fornecer escalabilidade, alta disponibilidade e baixa latência. Ele é formado por *buckets*, que é similar a um diretório or um *container*. Cada objeto armazenado é composto pelo nome, um *blob* com o conteúdo (até 5 Gb) e os metadados do mesmo, composto por atributos que representam o tipo de dado armazenado, associação com usuários e permissões de acesso, datas de criação, acesso e modificação, localização de objetos relacionados, classificação e comentários feitos pelo usuário, rótulos de área, assunto e geolocalização[45].

O Amazon S3 só permite buscas limitadas a consultas simples baseadas no nome dos objetos e dentro de um único *bucket*, não permite buscas baseadas em metadados e nem no conteúdo dos objetos (Palankar, 2008).

Existem outros sistemas focados em backup nas nuvens como Dropbox, SugarSync, Google Drive, mas os mesmos são soluções proprietárias e não possuem informações relacionadas com metadados disponíveis, o que dificulta uma análise técnica aprofundada sobre os mesmos, mas é possível dizer que os mesmos não possuem uma indexação que permita que consultas sejam realizadas baseadas no conteúdo de cada arquivo armazenado.

Os diversos sistemas mostrados acima são utilizados para diferentes fins: alguns são utilizados em ambientes tradicionais (disco local), outros são focados em ambientes de rede ou distribuídos e outros são direcionados para serem utilizados em ambientes de computação nas nuvens, por isso, eles possuem uma estrutura de metadados que varia de acordo com a finalidade desejada. Os metadados descritos possuem atributos utilizados para diversos fins, como: identificar os arquivos e diretórios, controlar o acesso aos mesmos, indicar a forma de armazenar os dados, localizar os *chunks* e réplicas, armazenar as datas de criação, acesso e modificação, controlar a integridade. Como pode ser observado, não há uma estrutura definida para metadados voltada para sistemas de *cloud storage* que atenda aos requisitos apresentados anteriomente na seção 2 (motivacao), principalmente, no que se refere a indexação do conteúdo armazenado.

3.5.2 Solução Proposta

Um sistema de arquivos é um conjunto de estruturas lógicas e de rotinas, que permitem ao sistema operacional controlar o acesso ao disco rígido[46] e, para isso, Tanembaum (2007) define que é necessário que estas característas estejam presentes:

- Segurança ou permissões
- Listas de controle de acesso (ACLs, em inglês, *Access Control Lists*)
- Mecanismo para evitar a fragmentação
- Capacidade de enlaces simbólicos (*symbolic links*) ou duros (*hard links*)

[44]http://aws.amazon.com/s3/
[45]http://bit.ly/JeaAZg
[46]http://bit.ly/JebDZe

- Integridade do sistema de arquivos (*Journaling*)
- Suporte para arquivos dispersos
- Suporte para quotas de discos
- Suporte de crescimento do sistema de arquivos nativo

Já para o desenvolvimento de um sistema de data cloud, como o usto.re, outras característas diferentes das mencionadas acima foram identificadas. Um sistema de *cloud storage* deve ser preocupar em como armazenar e recuperar os dados, além de identificar qual a máquina utilizada, indexação do conteúdo, como provisionamento sob demanda (habilidade de alocar espaço de acordo com a necessidade), snapshots (criar uma imagem instantânea do disco sem criar uma cópia completa), replicação e clonagem[47].

Portanto, o usto.re, deve ser visto como um sistema de *cloud storage* que implementa metadados como forma de facilitar o gerenciamento e acesso aos arquivos armazenados.

3.5.3 METASTORE

Assim como no modelo tradicional, onde um disco local possui o sistema de arquivo, é necessário que seja definido um conjunto de metadados associados ao arquivo que contém as informações sintáticas.

No usto.re os metadados foram descritos utilizando JSON (*JavaScript Object Notation*) devido a facilidade de representação de objetos e ao fato de consumirem menos recursos para serem processados, favorecendo assim a troca de dados, quanto comparado com o XML. Os atributos dos metadados foram definidos tendo como base os requisitos definidos anteriormente na seção motivacao, os atributos levantados na seção de trabalhos relacionados, além de atributos ligados às necessidades da computação nas nuvens para identificação e recuperação de informações.

A partir da análise dos sistemas de arquivos na seção de trabalhos relacionados observa-se que para o usto.re atender aos requisitos funcionais especificados são necessários atributos para identificação do dono do arquivo, caminho relativo do arquivo para o diretório monitorado, código hash para identificar se modificações foram feitas, a data e hora da última modificação. Além disso, também foi preciso criar um perfil para identificar a máquina a partir da qual o usuário acessa o sistema no momento do backup, para que o sistema pudesse identificar se o usuário está autenticado a partir de uma máquina diferente. Isso é necessário devido ao requisito de monitoramento de diretórios, já que uma máquina diferente, possivelmente, não possui a mesma organização hierárquica de arquivos e diretórios, podendo, assim, fazer com que o sistema tentasse monitorar um diretório que não existe. O perfil da máquina é composto pelo nome da máquina, sistema operacional e arquitetura do mesmo.

Também é necessário incluir atributos para identificar quais as opções de segurança e compartilhamento relacionadas ao arquivo. Estas opções de segurança

[47]http://bit.ly/KVuGsU

dizem respeito à forma como os outros usuários verão este arquivo (público ou privado) e como interagirão com ele (somente leitura, escrita, controle total). Através desse atributo é possível também determinar que o arquivo fique acessível apenas para o grupo ao qual o usuário pertença.

Além das informações relacionadas ao arquivo e sobre a maquina que o usuário está autenticado, outra característica identificada como necessária é a indexação do arquivo de forma a facilitar a localização de arquivos. Esta funcionalidade de indexação de dados em sistemas de data cloud é uma das funcionalidaes mais desejadas (Buyya, 2009), (Leung, 2009) e (Grossman,2008), porém de difícil implementação, visto que os arquivos não são salvos integralmente, mas sim quebrados em pedaços menores (*chunks*) e cada pedaço salvo em um servidor da nuvem de dados diferente.

Na implementação do usto.re, os *peers* que recebem os arquivos e os salvam na *data cloud*, são responsáveis também por indexá-los, antes que os mesmos sejam quebrados em partes menores para serem armazenados, com o objetivo de extrair o conteúdo dos mesmos e salvá-lo em um arquivo separado, formando, assim, um índice onde as buscas serão realizadas.

Após a geração dos arquivos necessários para armazenar o índice, um serviço provido pelo sistema é responsável por copiar esta informação para o servidor, onde esses dados ficarão disponíveis para a realização de buscas.

Para a indexação de arquivos foi utilizado o Apache Lucene[48], por ser um motor de buscas de alto desempenho escrito em Java. O Lucene contém apenas o núcleo do "motor" de busca. Por isso, ele não inclui um *Web crawler* ou um *parser* para diferentes formatos de documentos. Para resolver o problema do \textit{parser} foi utilizado o Apache Tika[49], que é um detector e extrator de conteúdo de metadados e texto estruturado, podendo ser utilizado com arquivos de diversos formatos como: HTML, XML, OLE2 e OOXML do Microsoft Office, OpenDocument Format, PDF, ePub, RTF, arquivos compactados e empacotados.

Sendo assim, quando um usuário realizar uma busca pelo conteúdo de um arquivo no usto.re ele poderá consultar todos os arquivos do sistema, porém:

- O usuário somente poderá acessar os arquivos pertencentes ou compartilhados com o mesmo;
- Somente poderá visualizar que um ou mais arquivos tem a informação procurada, desde que no metadado do mesmo seja informado que o arquivo poderá ser indexado;
- Caso um usuário deseja acessar um arquivo que não lhe pertence deverá solicitar ao dono que o compartilhe;
- Poderá existir um número N de arquivos que o usuário não tem acesso que contém esta informação;

[48]http://lucene.apache.org/
[49]http://tika.apache.org/

Assim, a estrutura dos metadados para o usto.re é composta pelos seguintes atributos:

- Nome do arquivo;
- Código *Hash*;
- Data da última modificação do arquivo;
- *Flag* para identificar se o arquivo deve ser indexado ou não;
- Atributos de segurança e compartilhamento;
- Dono do arquivo;
- Grupo ao qual o dono pertence;

Detalhes da Implementação do METASTORE

A Figura 3.3 representa o processo de backup, geração de metadados e indexação. No usto.re quando a confirmação de um backup chega ao *appClient*, é executado um método para gerar os metadados do arquivo salvo. Com a posse desses dados, o *appClient* os adiciona ao arquivo que representa o mapeamento do diretório do qual faz parte e onde se encontram todos os metadados dos arquivos cujo *backup* foi concluído com êxito. Após isso, o *appClient* envia esse mapeamento para que seja salvo no banco de dados juntamente com as informações de qual é o caminho completo para o diretório monitorado, caminho relativo e código *hash* do mapeamento.

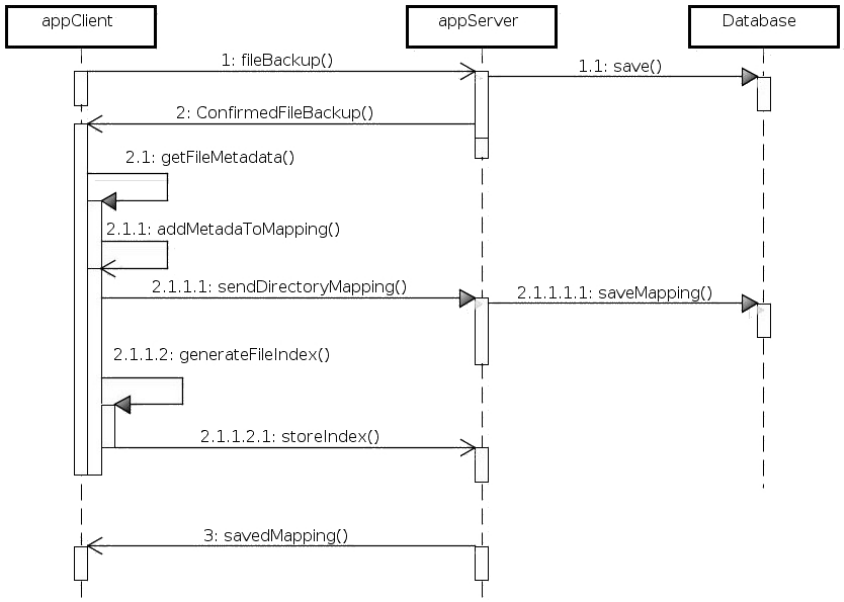

Figura 3.9 - Diagrama de sequencia metastore

Paralelamente à atividade descrita anteriormente é feita a indexação dos dados do arquivo e os dados gerados são enviados para o diretório *index* no servidor. Esta indexação extrai o conteúdo dos arquivos utilizando o Apache Tika e junta isso com a informação, repassada pelo appClient, de qual o nome do usuário que está fazendo backups no momento, o qual representa o dono dos arquivos e o nome junto com o caminho relativo para cada arquivo.

Na Figura 3.9 é apresentada a sequência executada quando é necessário efetuar uma busca por um arquivo no usto.re. A requisição feita ao *appClient* é repassada para o *appServer*. Nele o método *searcher* executa a busca utilizando a indexação gerada pelo Apache Lucene para isso. A busca pode ser realizada através do nome do arquivo, parte do conteúdo ou nome do dono. Se a busca for efetuada utilizando o nome do arquivo ou parte do conteúdo como parâmetro, o método retornará uma lista de arquivos contendo o nome dos mesmo e a quem pertence cada um deles. A depender das permissões de acesso do usuário, alguns arquivos podem deixar de ser exibidos na listagem ou podem aparecer arquivos aos quais o usuário não poderá acessar sem a permissão do dono.

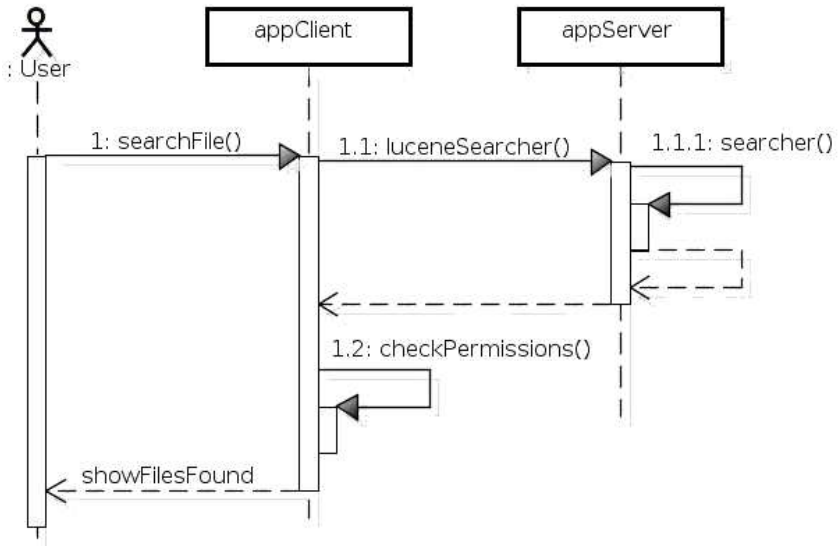

Figura 3.10 - Sincronização de diretorio

A sincronização de diretório só é possível porque o banco de dados armazena o caminho completo para os diretórios monitorados, isso permite chegar até os diretórios e, a partir disso, passar a monitorá-los. Quando um usuário loga, o usto.re faz uma consulta ao banco de dados pelos arquivos que o mesmo possui e quais são os diretórios monitorados. O usto.re traz o mapeamento armazenado no banco de dados e compara com os arquivos existentes nos diretórios monitorado, caso haja alguma diferença entre eles, será feita uma sincronização de arquivos, o que pode incluir adicionar e, até mesmo, remover arquivos.

Caso o usuário esteja em um cliente que não é sua máquina de trabalho (ou seja, que não possui seus arquivos) o usto.re criará uma pasta chamada "usto.re" dentro da pasta do usuário logado e realizará a recuperação de todos os arquivos do usuário. Isso permite que o mesmo tenha acesso a todos seus arquivos e diretórios independente do cliente utilizado.

3.6 Conclusão

Computação nas nuvens (Cloud Computing), consiste de um conjunto de tecnologias que agrupadas trazem vantagens significativas para a gerenciamento dos ambientes de Tecnologia da Informação, consequentemente a diminuição dos custos.

No entanto, a idéia de que qualquer aplicação pode ser migrada para a nuvem sem a necessidade de alterações, consiste de uma visão simplista do problema. Esta afirmativa, é verdadeira desde que os gestores de tecnologia abdiquem das questões relacionadas a performance (no caso de computação nas nuvens o crescimento elástico) e segurança, caso estejam em execução em ambientes públicos.

Este capítulo de livro apresentou como se define e implementa sistemas de armazenamento de massa em nuvens, cloud storages, como se pode definir e implementar sistemas de banco de dados escaláveis e uma abordagem para a incorporação dos conceitos de BIGDATA em sistemas de armazenamento de dados em nuvens.

3.7 Referências

[Aidouni et al., 2009] F. Aidouni, M. Latapy, and C. Magnien, "Ten weeks in the life of an eDonkey server," Proceedings of HotP2P'09, 2009, pp. 1-5.

[Armbrust et al., 2009] M. Armbrust et al. Above the Clouds: A Berkeley View of Cloud Computing. EECS Department, University of California, Berkeley. Technical Report No. UCB/EECS-2009-28. February 10, 2009.

[Amazon EC2, 2012] Amazon Elastic Compute Cloud (Amazon EC2), URL: http://aws.amazon.com/ec2/, Acessado em 05/07/2012.

[Amazon, 2012] Amazon. Amazon Simple Storage Service (Amazon S3), URL: http://aws.amazon.com/pt/s3/, last access 05/07/2012.

[Baker et al., 2011] J. Baker, C. Bond, J. Corbett, J. J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In CIDR'11, pages 223–234, 2011.

[Balakrishnan et al., 2003] Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R., and Stoica, I. Looking up data in P2P systems. Communications of the ACM 46, 2 (2003), 43.

[Bass et al., 2000] L. Bass, C. Buhman, S. Dorda, F. Long, J. Robert, R. Seacord, K. Wallnau, Market Assessment of Component-Based Software Engineering, Software Engineering Institute (SEI), Technical Report, Vol. 01, May, 2000, pp. 41.

[Batten et al., 2002] Christopher Batten, Kenneth Barr, Arvind Saraf, Stanley Trepetin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCSTM-632, Massachusetts Institute of Technology Laboratory for Computer Science, October 2002.

[Baumgarten & Chuí, 2009] Jason Baumgarten and Michael Chui, E-government 2.0, mckinseyquarterly.com, July 2009.

[Bell, 2008] M. Bell, Service-Oriented Modeling, 2008, pp. 366.

[BitTorrent, 2012] BitTorrent. Site Ofical. http://www.bittorrent.com/. Acessado em 09/07/2012.

[Bughin et al., 2010] Jacques Bughin, Michael Chui, and James Manyika, Clouds, big data, and smart assets: Ten tech-enabled business trends to watch, August 2010.

[Chang et al., 2006] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7, pages 15–15. USENIX Association, 2006.

[Clements & Northrop, 2001] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2001, pp. 608.

[Cleversafe, 2012] CLEVERSAFE. Site Oficial. Cleversafe dispersed storage project. http://www.cleversafe.org/dispersed-storage. Acessado em 09/07/2012.

[Colaço et al., 2008] COLAÇO, Eduardo José Moreira ; OLIVEIRA, Marcelo Iury ; SOARES, A.; BRASILEIRO, F ; GUERRERO, D. . Using a file working set model to speed up the recovery of Peer-to-Peer backup systems. ACM SIGOPS Operating Systems Review, v. 42, p. 64-70, 2008.

[Cox & Noble, 2003] Cox, L. P. and Noble, B. D. 2003. Samsara: honor among thieves in peer-to-peer storage. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (Bolton Landing, NY, USA, October 19 - 22, 2003). SOSP '03. ACM, New York, NY, 120-132.

[Dean & Ghemawat, 2008] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51:107–113, Jan. 2008.

[DeCandia et al., 2007] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon’s highly available key-value store. SIGOPS Oper. Syst. Rev., 41:205–220, Oct. 2007.

[Dropbox, 2012] Dropbox, URL: https://www.dropbox.com/. Acessado em 09/07/2012.

[Duarte, 2010]M. DUARTE. Um algoritmo de disponibilidade em sistemas de backup distribuído seguro usando a plataforma peer-to-peer. Dissertação de mestrado, Centro de Informática, Universidade Federal de Pernambuco, Recife-PE, Brazil, 2010.

[Erl, 2008] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, 2008, pp. 760.

[FIPS, 1995] FIPS 180-1. Secure Hash Standard. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, Apr. 1995.

[Galán & Sampaio, 2009] F. Galán, A. Sampaio et al. Service Specification in Cloud Environments Based on Extensions to Open Standards. In 4th International Conference on Communication System Software and Middleware (COMSWARE 2009), 2009.

[Grobauer et al., 2011] B. Grobauer, T. Walloschek, and E. Stocker. Understanding cloud computing vulnerabilities. IEEE Security and Privacy, 9:50–57, 2011.

[Gnutella, 2012] Gnutella, Site Oficial, http://rfc-gnutella.sourceforge.net/, Acessado em 09/07/2012.

[Google, 2012] Google Web Applications for Communication and Collaborations. http://www.google.com/apps. Acessado em 09/07/2012.

[GTAR, 2010]GTAR, DVN: Disco Virtual Nacional. URL: http://bit.ly/PCP3kj, Acessado em 09/07/2012.

[JXTA, 2012] JXTA. Jxta protocol, 2012. URL: http://java.net/projects/jxta/, Acessado em 05/07/2012.

[Karger et al., 1997] Karger, D., Lehman, E., Leighton, F., Levine, M., Lewin, D., and Panigrahy, R. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. 29th Annual ACM Symposium on Theory of Computing, (1997), 654-663.

[Kubiatowicz et al., 2000] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. 2000. OceanStore: an architecture for global-scale persistent storage. SIGOPS Oper. Syst. Rev. 34, 5 (November 2000), 190-201.

[Landers et al., 2004] Landers, M., Zhang, H., and Tan, K. PeerStore: better performance by relaxing in peer-to-peer backup. Proceedings of the Fourth International Conference on Peer-to-Peer Computing, (2004), 72-79.

[Leavitt, 2009] N. Leavitt, Is Cloud Computing Really Ready for Prime Time? IEEE Computer, 2009. 42(1): p. 15-20.

[Li & Dabek, 2006] J. Li and F. Dabek. F2F: Reliable storage in open networks.In Intl Workshop on Peer-to-Peer Systems, Santa BarbaraCA, Feb. 2006.

[Loest et al., 2009] S. R. Loest, M. C. Madruga, C. A. Maziero, and L. C. Lung. Backupit: An intrusion-tolerant cooperative backup system. In Proceedings of the 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science, pages 724–729. IEEE Computer Society, 2009.

[Malkhi & Reiter, 1997] D. Malkhi and M. Reiter. Byzantine quorum systems. In ACM Symposium on Theory of Computing, 1997.

[Martinian, 2012] Emin Martinian. Site Oficial. Distributed internet backup system (dibs). http://www.mit.edu/~emin/source_code/dibs/index.html. Acessado em 09/07/2012.

[Mattos, 2005] Mattos, Antonio Carlos M. Sistemas de Informação: uma visão executiva. São Paulo. Saraiva. 2005.

[McIlroy, 1969] M. D. McIlroy, "Mass Produced Software Components," in Software Engineering: Report on a conference sponsored by the NATO Science Committee, 1969, pp. 138--155.

[Meira, 2005] Meira, Fernando. Resilia: A safe & secure backup-system. Final year project, Engineerng Faculty of the University of Porto, May 2005

[Meyer, 1995] Meyer, Poul L. (1995) “ PROBABILIDADE Aplicações à Estatística”. Livros Técnicos e Científicos Editora. 2 Edição. Rio de Janeiro.

[Microsoft, 2012] Microsoft Skydrive service, http://skydrive.live.com/, 2012.

[Napster, 2012] Napster, Site Oficial, http://www.napster.com, Acessado em 09/07/2012.

[Oliveira, 2007] M. Oliveira. OurBackup: A P2P backup solution based on social networks, MSc Thesis, Universidade Federal de Campina Grande, Brazil, 2007

[Osterweil, 2007] Osterweil, L. J. 2007. A Future for Software Engineering?. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 1-11

[Papazoglou et al., 2007] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. 2007. Service-Oriented Computing: State of the Art and Research Challenges. Computer 40, 11 (November 2007), 38-45.

[Ratnasamy et al., 2001] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. 2001. A scalable content-addressable network. In Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications (San Diego, California, United States). SIGCOMM '01. ACM, New York, NY, 161-172.

[Rowstron & Druschel, 2001] Rowstron, A. and Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science, November 2001 (2001), 329-350.

[Schollmeier & Rudiger, 2002] Schollmeier, Rudiger. “A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications”. In: IEEE Internet Computing, 2002.

[Shvachko et al., 2010] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pages 1–10. IEEE Computer Society, 2010.

[Smith et al., 2009] D. M. Smith et al. Gartner Report ID Number G00163522, Predicts 2009: Cloud Computing Beckons, 2009.

[Stal, 2002] M. Stal, Web Services: Beyond Component-Based Computing, Communications of the ACM, Vol. 45, No. 10, October, 2002, pp. 71-76.

[Stoica et al., 2001] Stoica, I., Morris, R., Karger, D., Kaashoek, M., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, (2001), 160.

[Sun, 2009] Sun Microsystems, Introdution to cloud computing architecture whitepaper, June 2009.

[Turner et al., 2003] M. Turner, P. Brereton, and D. Budgen, "Turning Software into a Service," Computer, vol. 36, 2003, pp. 38-44.

[Vignatti et al., 2009] VIGNATTI, T. ; BONA, L. C. E. ; VIGNATTI, A. ; SUNYE, M. . Long-term Digital Archiving Based on Selection of Repositories Over P2P Networks. In: Ninth International Conference on Peer-to-Peer Computing (P2P'09), 2009, Seattle. Proceedings of Eight IEEE International Conference on Peer-to-Peer Computing, 2009.

[XDriver, 2012] XDriver Box service, http://www.box.net/xdrive, Acessado em 05/07/2012

[W3C, 2000] W3C, A Little History of the World Wide Web, 2000. URL: http://www.w3.org/History.html, Acessado em 05/07/2012.

[Webber et al., 2010] J. Webber, S. Parastatidis, and I. Robinson. REST in Practice: Hypermedia and Systems Architecture. O'Reilly Media, 2010.

[Yang et al., 2006] Q. Yang, W. Xiao, and J. Ren. Prins: Optimizing performance of reliable internet storages. In Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, pages 32–. IEEE Computer Society, 2006.

[Yu et al., 2007] L. Yu, G. Chen, W. Wang, and J. Dong. Msfss: A storage system for mass small files. In W. Shen, Y. Yang, J. Yong, I. Hawryszkiewycz, Z. Lin, J.-P. A. Barthes, M. L. Maher, Q. Hao, and M. H. Tran, editors, 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pages 1087–1092, Los Alamitos, CA, USA, April 2007. IEEE Computer Society Press.

[Zhang et al., 2007] Liang-Jie Zhang, Jia Zhang, Hong Cai, Services Computing, Springer and Tsinghua University Press, 2007, ISBN: 978-3-540-38281-2, July 2007

[Zhang et al., 2008] Liang-Jie Zhang, Carl K Chang, Ephraim Feig, Robert Grossman,Keynote Panel, Business Cloud: Bringing The Power of SOA and Cloud Computing, pp.xix, 2008 IEEE International Conference on Services Computing (SCC 2008), July 2008.

[Zhang and Zhou, 2009] Liang-Jie Zhang and Qun Zhou. 2009. CCOA: Cloud Computing Open Architecture. In Proceedings of the 2009 IEEE International Conference on Web Services (ICWS '09). IEEE Computer Society, Washington, DC, USA, 607-616.

[Zao et al., 2004] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 22, NO. 1, JANUARY 2004 41

[Grossman 2008] R. Grossman and Y. Gu. Data mining using high performance data clouds: experimental studies using sector and sphere. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '08, pages 920–927, New York, NY, USA, 2008. ACM.

[Ghemawat 2003] ,S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In ACM SIGOPS Operating Systems Review, volume 37, pages 29–43. ACM, 2003.

[Buyya 2009] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25(6):599 – 616, 2009.

[Palankar 2008] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon s3 for science grids: a viable solution? In Proceedings of the 2008 international workshop on Data-aware distributed computing, DADC '08, pages 55–64, New York, NY, USA, 2008. ACM.

[Ghemawat 2003] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In ACM SIGOPS Operating Systems Review, volume 37, pages 29–43. ACM, 2003.

[Mathur 2007] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier. The new ext4 filesystem current status and future plans. 2007.

[Steinacker 2001] A. Steinacker, A. Ghavam, and R. Steinmetz. Metadata standards for web-based resources. IEEE Multimedia, 8:70–76, 2001.

[Shvachko 2011] K. V. Shvachko. Apache Hadoop: The Scalability Update. Pages 7–13, 2011.

[XtreemFS 2008]F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario. The XtreemFS architecture: a case for object-based file systems. Concurrency and computation: Practice and experience, 20(17):2049–2060, 2008.