Capítulo 4

Gerenciamento de Dados em Nuvem: Conceitos, Sistemas e Desafios [1] [2]

Flávio R. C. Sousa, Leonardo O. Moreira, José Antônio F. de Macêdo e Javam C. Machado

Universidade Federal do Ceará (UFC)

Abstract

Infrastructures, platforms and software are being offered as services, in cloud computing environments, with companies and users being able to rent computing power and storage in a transparent way and on-demand. These companies and users are moving their data and applications to the cloud so that they can access them anytime and anywhere. However, this new computing model requires significant changes in data management systems, since these systems demand scalability, availability, performance and cost. This minicourse will introduce the main concepts and technologies in cloud computing, focusing on data management and systems, and point research challenges and opportunities in this context.

Resumo

Infraestruturas, plataformas e software estão sendo disponibilizados como serviços, sendo estes fornecidos por ambientes de Computação em Nuvem, no qual empresas e usuários podem alugar capacidade de computação e armazenamento de forma transparente e sob demanda. Estas empresas e usuários estão movendo seus dados e aplicações para a nuvem de forma a acessá-los a qualquer momento e independente de localização. Entretanto, este novo modelo de computação requer grandes mudanças nos sistemas de gerenciamento de dados, pois estes sistemas necessitam de escalabilidade, disponibilidade, desempenho e custo. Este minicurso tem como objetivo apresentar os principais conceitos e tecnologias de computação em nuvem, destacando o gerenciamento de dados e sistemas, além de desafios e oportunidades de pesquisa neste contexto.

[1] Publicado no SWIB 2010.

[2] Versão revisada e ampliada em Dezembro de 2011.

4.1. Introdução

Com o avanço da sociedade humana moderna, serviços básicos e essenciais são quase todos entregues de uma forma completamente transparente. Serviços de utilidade pública como água, eletricidade, telefone e gás tornaram-se fundamentais para nossa vida diária e são explorados por meio do modelo de pagamento baseado no uso [Vecchiola et al. 2009]. As infraestruturas existentes permitem entregar tais serviços em qualquer lugar e a qualquer hora, de forma que possamos simplesmente acender a luz, abrir a torneira ou usar o fogão. O uso desses serviços é, então, cobrado de acordo com as diferentes políticas de tarifação para o usuário final. Recentemente, a mesma ideia de utilidade tem sido aplicada no contexto da informática e uma mudança consistente neste sentido tem sido feita com a disseminação de *Cloud Computing* ou Computação em Nuvem.

Computação em nuvem é uma tendência recente de tecnologia cujo objetivo é proporcionar serviços de Tecnologia da Informação (TI) sob demanda com pagamento baseado no uso. Tendências anteriores à computação em nuvem foram limitadas a uma determinada classe de usuários ou focadas em tornar disponível uma demanda específica de recursos de TI, principalmente de informática [Buyya et al. 2009]. Computação em nuvem pretende ser global e prover serviços para as massas que vão desde o usuário final que hospeda seus documentos pessoais na Internet até empresas que terceirizam toda infraestrutura de TI para outras empresas. Nunca uma abordagem para a utilização real foi tão global e completa: não apenas recursos de computação e armazenamento são entregues sob demanda, mas toda a pilha de computação pode ser aproveitada na nuvem.

A computação em nuvem surge da necessidade de construir infraestruturas de TI complexas, onde os usuários têm que realizar instalação, configuração e atualização de sistemas de software. Em geral, os recursos de computação e hardware são propensos a ficarem obsoletos rapidamente e a utilização de plataformas computacionais de terceiros é uma solução inteligente para os usuários lidarem com a infraestrutura de TI. Na computação em nuvem os recursos de TI são fornecidos como um serviço, permitindo que os usuários o acessem sem a necessidade de conhecimento sobre a tecnologia utilizada. Desse modo, os usuários e as empresas passaram a acessar os serviços sob demanda e independente de localização, o que aumentou a quantidade de serviços disponíveis.

Com isso, os usuários estão movendo seus dados e aplicações para a nuvem e assim podem acessá-los de forma simples e de qualquer local. Isso é novamente um caso de utilização de processamento centralizado. Cenário semelhante ocorreu há aproximadamente 50 anos: um servidor de tempo compartilhado acessado por vários usuários. Contudo, nas últimas décadas, quando os computadores pessoais surgiram, os dados e as aplicações começaram a ser utilizados localmente.

Certamente o paradigma de computação em nuvem não é uma repetição da história. Há 50 anos os servidores de tempo compartilhado foram adaptados por questões de limitação de recursos. Já a computação em nuvem surge da necessidade de construir infraestruturas de TI complexas, onde os usuários têm que realizar instalação, configuração e atualização de sistemas de software. Além disso, recursos de computação e hardware são propensos a ficarem obsoletos rapidamente. Assim, a utilização de plataformas computacionais de terceiros é uma solução inteligente para os usuários lidarem com infraestrutura de TI. Na computação em nuvem os recursos de TI são fornecidos como um

serviço, permitindo que os usuários o acessem sem a necessidade de conhecimento sobre a tecnologia utilizada. Assim, os usuários e empresas passaram a acessa os serviços sob demanda e independente de localização, o que aumentou a quantidade de serviços disponíveis [Buyya et al. 2009].

Sistemas de gerenciamento de banco de dados (SGBDs) [3] são candidatos potenciais para a implantação em nuvem. Isso ocorre porque, em geral, as instalações destes sistemas são complexas e envolvem uma grande quantidade de dados, ocasionando um custo elevado, tanto em hardware quanto em software. Para muitas empresas, especialmente para *start-ups* e médias empresas, o pagamento baseado no uso do modelo de computação em nuvem, juntamente com o suporte para manutenção do hardware é muito atraente [Abadi 2009].

No final de 2010, a empresa Embarcadero realizou um *survey* [Embarcadero 2010] sobre tendência, principais iniciativas, ferramentas atuais e desafios de banco de dados realizado com profissionais de banco de dados de grandes organizações. Este *survey* destacou que as tecnologias de banco de dados em nuvem e virtualização terão o maior impacto na indústria de banco de dados, com 34% e 25% das respostas dos entrevistados, respectivamente. Isso indica uma previsão de adoção constante de tecnologias de banco de dados em nuvens.

Embora SGBDs em nuvem tenham reduzido o custo de armazenamento de dados e melhorado o acesso, existe uma enorme complexidade envolvida com os serviços de dados que possam escalar quando é necessário garantir operações consistentes e confiáveis considerando cargas de trabalho máximas. Além disso, o ambiente em nuvem tem requisitos técnicos para controlar centros de dados virtualizados, reduzindo os custos e aumentando a confiabilidade por meio da consolidação de sistemas em nuvem. Desafios tais como consistência e segurança dos dados são importantes para a computação em nuvem e acredita-se que futuras aplicações centradas em dados irão alavancar os serviços de dados em nuvem.

Esta seção apresenta as definições, as características essenciais da computação em nuvem e os modelos de serviço e implantação. A seção 4.2 caracteriza o gerenciamento de dados em nuvem e a seção 4.3 apresenta os principais sistemas de gerenciamento de dados em nuvem. A seção 4.4 discute alguns desafios do gerenciamento de dados em nuvem. Finalmente, a seção 4.5 apresenta as conclusões finais.

4.1.1. Computação em Nuvem

A computação em nuvem está se tornando uma das palavras chaves da indústria de TI. A nuvem é uma metáfora para a Internet ou infraestrutura de comunicação entre os componentes arquiteturais, baseada em uma abstração que oculta à complexidade da infraestrutura. Cada parte desta infraestrutura é provida como um serviço e, estes são normalmente alocados em centros de dados, utilizando hardware compartilhado para computação e armazenamento [Buyya et al. 2009].

A infraestrutura do ambiente de computação em nuvem normalmente é composta por um grande número, centenas ou milhares de máquinas físicas ou nós físicos de baixo

[3] SGBD refere-se um uma classe geral de armazenamento de dados, incluindo sistemas não-relacionais.

custo, conectadas por meio de uma rede como ilustra a Figura 4.1. Cada máquina física tem as mesmas configurações de software, mas pode ter variação na capacidade de hardware em termos de CPU, memória e armazenamento em disco [Soror et al. 2010]. Dentro de cada máquina física existe um número variável de máquinas virtuais (VM) ou nós virtuais em execução, de acordo com a capacidade do hardware disponível na máquina física. Os dados são persistidos, geralmente, em sistemas de armazenamento distribuídos.

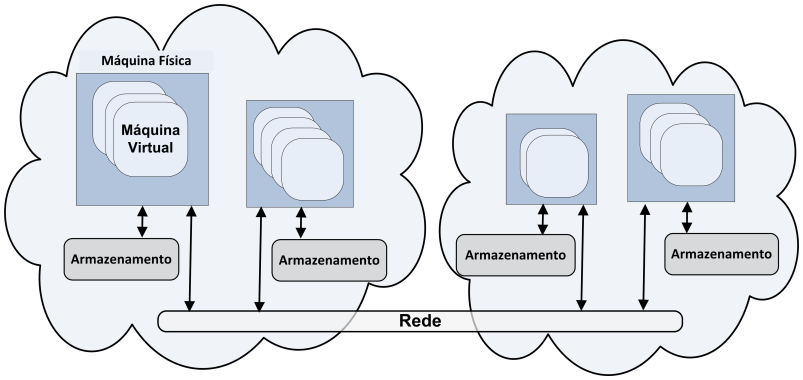

Figura 4.1. Ambiente de Computação em Nuvem

A computação em nuvem é uma evolução dos serviços e produtos de tecnologia da informação sob demanda, também chamada de *Utility Computing* [Brantner et al. 2008]. O objetivo da *Utility Computing* é fornecer componentes básicos como armazenamento, processamento e largura de banda de uma rede como uma "mercadoria" através de provedores especializados com um baixo custo por unidade utilizada. Usuários de serviços baseados em *Utility Computing* não precisam se preocupar com escalabilidade, pois a capacidade de armazenamento fornecida é praticamente infinita. A *Utility Computing* propõe fornecer disponibilidade total, isto é, os usuários podem ler e gravar dados a qualquer tempo, sem nunca serem bloqueados; os tempos de resposta são quase constantes e não dependem do número de usuários simultâneos, do tamanho do banco de dados ou de qualquer parâmetro do sistema. Os usuários não precisam se preocupar com *backups*, pois se os componentes falharem, o provedor é responsável por substituí-los e tornar os dados disponíveis em tempo hábil por meio de réplicas [Brantner et al. 2008].

Uma razão importante para a construção de novos serviços baseados em *Utility Computing* é que provedores de serviços que utilizam serviços de terceiros pagam apenas pelos recursos que recebem, ou seja, pagam pelo uso. Não são necessários investimentos iniciais em TI e o custo cresce de forma linear e previsível com o uso. Dependendo do modelo do negócio, é possível que o provedor de serviços repasse o custo de armazenagem, computação e de rede para os usuários finais, já que é realizada a contabilização do uso.

Existem diversas propostas para definir o paradigma da computação em nuvem [Vaquero et al. 2009]. O *National Institute of Standards and Technology* (NIST) argumenta que a computação em nuvem é um paradigma em evolução e apresenta a seguinte definição: *"Computação em nuvem é um modelo que possibilita acesso, de modo conveniente e sob demanda, a um conjunto de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapida-*

mente adquiridos e liberados com mínimo esforço gerencial ou interação com o provedor de serviços" [Mell and Grance 2009]. O NIST descreve que a computação em nuvem é composta por cinco características essenciais, três modelos de serviço e quatro modelos de implantação, detalhados a seguir.

4.1.1.1. Características Essenciais

As características essenciais são vantagens que as soluções de computação em nuvem oferecem. Algumas destas características, em conjunto, definem exclusivamente a computação em nuvem e fazem a distinção com outros paradigmas. Por exemplo, a elasticidade rápida de recursos, amplo acesso e a medição de serviço são características básicas para compor uma solução de computação em nuvem.

- *Self-service sob demanda*: O usuário pode adquirir unilateralmente recurso computacional, como tempo de processamento no servidor ou armazenamento na rede, na medida em que necessite e sem precisar de interação humana com os provedores de cada serviço.
- *Amplo acesso*: Recursos são disponibilizados por meio da rede e acessados através de mecanismos padronizados que possibilitam o uso por plataformas do tipo *thin*, tais como celulares, *laptops* e PDAs.
- *Pooling de recursos*: Os recursos computacionais do provedor são organizados em um *pool* para servir múltiplos usuários usando um modelo *multi-tenant* ou multi-inquilino, com diferentes recursos físicos e virtuais, dinamicamente atribuídos e ajustados de acordo com a demanda dos usuários. Estes usuários não precisam ter conhecimento da localização física dos recursos computacionais, podendo somente especificar a localização em um nível mais alto de abstração, tais como o país, estado ou centro de dados.
- *Elasticidade rápida*: Recursos podem ser adquiridos de forma rápida e elástica, em alguns casos automaticamente, caso haja a necessidade de escalar com o aumento da demanda, e liberados, na retração dessa demanda. Para os usuários, os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento.
- *Serviço medido*: Sistemas em nuvem automaticamente controlam e otimizam o uso de recursos por meio de uma capacidade de medição. A automação é realizada em algum nível de abstração apropriado para o tipo de serviço, tais como armazenamento, processamento, largura de banda e contas de usuário ativas. O uso de recursos pode ser monitorado e controlado, possibilitando transparência para o provedor e o usuário do serviço utilizado.

4.1.1.2. Modelos de Serviços

O ambiente de computação em nuvem é composto de três modelos de serviços. Estes modelos são importantes, pois eles definem um padrão arquitetural para soluções de com-

putação em nuvem.

- *Software como um Serviço (SaaS)*: O modelo de SaaS proporciona sistemas de software com propósitos específicos que são disponíveis para os usuários por meio da Internet e acessíveis a partir de vários dispositivos do usuário por meio de uma *interface thin client* como um navegador Web. No SaaS, o usuário não administra ou controla a infraestrutura subjacente, incluindo rede, servidores, sistema operacional, armazenamento ou mesmo as características individuais da aplicação, exceto configurações específicas. Como exemplos de SaaS podemos destacar os serviços de *Customer Relationship Management* (CRM) da Salesforce e o Google Docs.

- *Plataforma como um Serviço (PaaS)*: O modelo de PaaS fornece sistema operacional, linguagens de programação e ambientes de desenvolvimento para as aplicações, auxiliando a implementação de sistemas de software. Assim como no SaaS, o usuário não administra ou controla a infraestrutura subjacente, mas tem controle sobre as aplicações implantadas e, possivelmente, as configurações de aplicações hospedadas nesta infraestrutura. *Google App Engine* [Ciurana 2009] e *Microsoft Azure* [Azure 2011] são exemplos de PaaS.

- *Infraestrutura como um Serviço (IaaS)*: A IaaS torna mais fácil e acessível o fornecimento de recursos, tais como servidores, rede, armazenamento e outros recursos de computação fundamentais para construir um ambiente de aplicação sob demanda, que podem incluir sistemas operacionais e aplicativos. Em geral, o usuário não administra ou controla a infraestrutura da nuvem, mas tem controle sobre os sistemas operacionais, armazenamento, aplicativos implantados e, eventualmente, seleciona componentes de rede, tais como *firewalls*. O Amazon *Elastic Cloud Computing* (EC2) [Robinson 2008] e o Eucalyptus [Liu et al. 2007] são exemplos de IaaS.

4.1.1.3. Modelos de Implantação

Quanto ao acesso e à disponibilidade, há diferentes tipos de modelos de implantação para os ambientes de computação em nuvem. A restrição ou abertura de acesso depende do processo de negócios, do tipo de informação e do nível de visão desejado.

- *Nuvem privada*: a infraestrutura de nuvem é utilizada exclusivamente por uma organização, sendo esta nuvem local ou remota e administrada pela própria empresa ou por terceiros.

- *Nuvem pública*: a infraestrutura de nuvem é disponibilizada para o público em geral, sendo acessado por qualquer usuário que conheça a localização do serviço.

- *Nuvem comunidade*: fornece uma infraestrutura compartilhada por uma comunidade de organizações com interesses em comum.

- *Nuvem híbrida*: a infraestrutura é uma composição de duas ou mais nuvens, que podem ser do tipo privada, pública ou comunidade e que continuam a ser entidades

únicas, mas conectadas por meio de tecnologia proprietária ou padronizada que permite a portabilidade de dados e aplicações.

4.1.2. Multi-Inquilino

O termo multi-inquilino é um dos principais conceitos relacionados a SaaS. Um inquilino em um cenário de SaaS é um usuário que utiliza SaaS. Ele refere-se ao uso do mesmo software e instâncias por vários usuários e empresas de forma simultânea. O objetivo dessa abordagem é disponibilizar os mesmos recursos de software para um número maior de usuários.

Esta abordagem é baseada no conceito da "Cauda Longa" [Microsoft 2010], utilizada por diversas empresas de internet que conseguem obter renda tanto com produtos de nicho como produtos tradicionais. Com o seu artigo "The Long Tail", o escritor Chris Anderson popularizou a idéia da "longa cauda" ao explicar por que os varejistas on-line como Amazon.com estão posicionados de uma forma singular para atender uma imensa demanda que os varejistas tradicionais não podem atender de uma maneira econômica. Os fornecedores de soluções complexas de software de linha de negócios se defrontam com uma curva de mercado semelhante. A Figura 4.2 mostra este cenário.

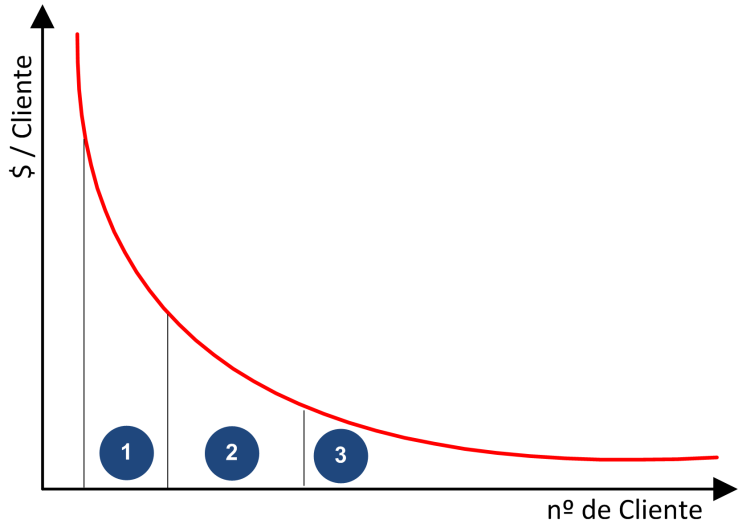

Figura 4.2. Gráfico da Cauda Longa

A primeira parte do gráfico refere-se a grandes clientes e a segunda parte a clientes típicos. A terceira parte trata-se de uma grande quantidade de pequenos clientes que devido à diminuição dos custos podem utilizar software como um serviço. O gráfico demonstra que, conforme diminui-se o custo de adoção, um número maior de clientes pode adotar uma solução, sendo que este número tende ao infinito, uma vez que a curva não toca o eixo "x". Assim, no modelo SaaS de fornecimento de software, precisa-se desenvolver soluções e infraestruturas de baixo custo, com alto aproveitamento de recursos por um número muito grande de usuários, para assim atingir um público não suportado hoje em dia, devido os custos proibitivos de entrada.

Outro conceito importante do modelo é o "micro-pagamento". Na cauda longa, um número muito grande de usuários poderá adotar nossa solução pagando pelo uso, por

demanda, o que deve gerar um valor muito baixo de pagamento. Neste contexto, procura-se o chamado “milhões de mercados de poucos” ao invés dos atuais “poucos mercados de milhões”.

Um ponto importante no modelo de SaaS é o apoio para vários inquilinos. A fim de explorar economias de escala, ou seja, permitir que um provedor ou desenvolvedor de SaaS ofereça a mesma instância de um aplicativo SaaS para vários inquilinos, a aplicação deve ser multi-inquilino consistente. Multi-inquilino consciente significa que cada inquilino pode interagir com o aplicativo como se ele fosse o único usuário da aplicação. Em especial, um inquilino não pode acessar ou visualizar os dados de um outro inquilino.

A forma de organização dos inquilinos e das aplicações ajuda a determinar a qualidade de uma solução multi-inquilino. De acordo com o nível de implementação do conceito de multi-inquilino, pode-se definir a maturidade do SaaS. A Figura 4.3 mostra o modelo de maturidade SaaS, sobre a evolução do suporte multi-inquilino [Microsoft 2010]. Em cada quadrante é possível observar as diferenças em relação ao suporte multi-inquilino: ad-hoc/personalizado configurável, configurável e eficiente para vários inquilinos e escalonável, configurável e eficiente para vários inquilinos, respectivamente.

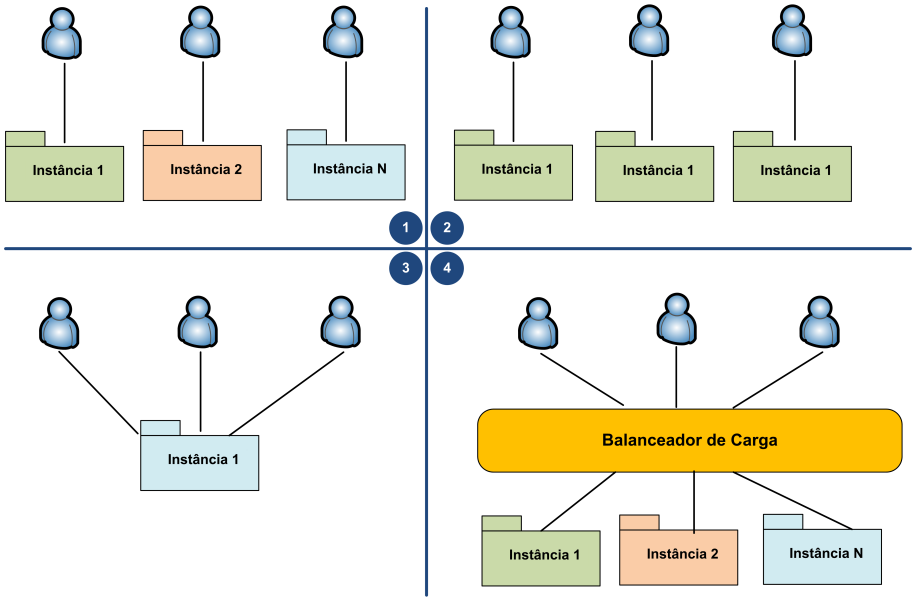

Figura 4.3. Modelo de maturidade SaaS, sobre a evolução do suporte multi-inquilino [Microsoft 2010]

No primeiro quadrante, existe um software destinado para cada inquilino. Isso garante o atendimento das demandas do usuário, mas custo elevado devido a grande customização e disponibilidade de recursos individuais. O segundo quadrante apresenta uma única solução destinada para todos os inquilinos separadamente. Neste caso, não há nenhuma customização presente, garantindo menor custo de manutenção.

No terceiro quadrante, existe uma única solução para todos os inquilinos e, diferente do quadrante dois, esse solução é única e disponivel para todos. Neste caso, uma solução multi-inquilino, onde ocorre total compartilhamento de recursos.No quarto quadrante, o atendimento é diferenciado para inquilinos que exigem elevada demanda de

recursos, havendo uma carga balanceada na infra-estrutura do provedor da solução SaaS.

A fim de permitir aplicações multi-inquilinos, a aplicação deve ser dividida em duas partes diferentes. A primeira parte da aplicação descreve os artefatos que são fixos e podem ser usados por todos os inquilinos. A segunda parte descreve a parte flexível, correspondente às configurações e denominada "metadados". Em relação aos dados, estes podem ser gerenciados de diversas formas, discutidas ao final deste texto.

4.2. Gerenciamento de Dados em Nuvem

SGBDs em nuvem estão começando a ser utilizados e têm o potencial de atrair clientes de diversos setores do mercado, desde pequenas empresas com o objetivo de reduzir o custo total, por meio da utilização de infraestrutura e sistemas de terceiros, até grandes empresas que buscam soluções que para gerenciar milhares de máquinas e permitir o atendimento de um aumento inesperado de tráfego [Abadi 2009].

A infraestrutura de SGBDs em nuvem possui várias vantagens para os usuários: (i) previsibilidade e custos mais baixos, proporcional à qualidade do serviço (QoS) e cargas de trabalho reais, (ii) complexidade técnica reduzida, graças a *interfaces* de acesso unificado e a delegação de *tuning* e administração de SGBDs e (iii) a elasticidade e escalabilidade, proporcionando a percepção de recursos quase infinitos. Por outro lado, o provedor tem que garantir (i) a ilusão de recursos infinitos, sob cargas de trabalho dinâmicas e (ii) minimizar os custos operacionais associados a cada usuário [Curino et al. 2010].

Diversos sistemas e arquiteturas estão sendo desenvolvidos para suprir as novas demandas de aplicações com diferentes requisitos de processamento e armazenamento [Abouzeid et al. 2009]. Estes novos sistemas tentam fornecer uma visão de armazenamento e escalabilidade infinitos, mas tem que tratar o problema de provisionar recursos. Este problema, que em SGBDs tradicionais consiste em determinar quais recursos são alocados para um único banco de dados, no ambiente em nuvem torna-se um problema de otimização, onde se tem uma grande quantidade de usuários, múltiplos SGBDs em nuvem e grandes centros de dados. Isso fornece uma oportunidade sem precedentes para explorar a economia em escala, balanceamento dinâmico de carga e gerenciamento de energia.

Esse aumento no número de abordagens disponíveis de SGBDs em nuvem agrava o problema da escolha, implantação e soluções de administração para a gestão de dados. Com isso, os SGBDs em nuvem estão sendo disponibilizados como serviços, que encapsulam a complexidade do gerenciamento por meio de formas de acesso simples e garantias de acordos de nível de serviço (SLAs).

4.2.1. Requisitos para o Gerenciamento de Dados em Nuvem

A definição dos requisitos é fundamental no gerenciamento de dados como um serviço. [Curino et al. 2010] apresentam uma lista de requisitos de um SGBD como um serviço da perspectiva do usuário, do provedor e requisitos adicionais relacionados à nuvem pública conforme apresenta a Tabela 4.1.

Da perspectiva do usuário, a principal necessidade é um serviço de banco de dados com uma *interface* simples que não necessite de ajuste ou administração. Trata-se de uma melhoria em relação às soluções tradicionais que requerem técnicas para provi-

Requisitos do Usuário	
U1	API simples com pouca configuração e administração (ex. sem tuning)
U2	Alto desempenho (ex. vazão, escalabilidade)
U3	Alta disponibilidade e confiança (ex. hot stand-by, backup)
U4	Acesso fácil à características avançadas (ex. snapshot, evolução de esquema, mineração de dados)
Requisitos do Provedor	
P1	Atender o SLA do usuário (ex. potencialmente sob carga de trabalho dinâmica)
P2	Limitar hardware e custo de energia (ex. multiplexação intensiva)
P3	Limitar custo de administração (ex. custo com pessoal)
Requisitos extra de Nuvem Pública	
P1	Esquema de preço: barato, previsível e proporcional ao uso (elasticidade)
P2	Garantias de segurança e privacidade
P3	Baixa latência (relevante para OLTP e aplicações Web)

Tabela 4.1. Requisitos para SGBD como um Serviço [Curino et al. 2010]

sionar recursos, seleção de SGBDs em nuvem, instalação, configuração e administração. O usuário quer um desempenho satisfatório, expresso em termos de latência e vazão, independente do tamanho da base de dados e das alterações da carga de trabalho. Atualmente, esta é uma tarefa difícil que exige uma ampla análise do pessoal de TI, software caro e atualizações de hardware. Alta disponibilidade é outro requisito fundamental, que normalmente é oferecido pelos SGBDs tradicionais, mas exige cuidados de configuração e manutenção. Finalmente, as características avançadas de gerenciamento do banco de dados, tais como *snapshot*, evolução de esquema e mineração de dados devem estar prontamente disponíveis e simples de utilizar.

Da perspectiva do provedor, é necessário atender aos acordos de nível de serviço, apesar da quantidade de dados e alterações na carga de trabalho. O sistema deve ser eficiente na utilização dos recursos de hardware. O modelo de serviço proporciona a oportunidade de fazer isso, por multiplexação de cargas de trabalho e ajuste dinâmico da alocação de recursos. Finalmente, a quantidade de tarefas de administração deve ser minimizada. Para tanto, ferramentas sofisticadas de análise de carga de trabalho e centralização do gerenciamento de muitos bancos de dados devem ser utilizadas. Para provedores de serviços em nuvem pública, existem requisitos adicionais, tais como esquema de preço, segurança, privacidade e latência. Entretanto, estas questões não são específicas de bancos de dados e podem ser abordadas com técnicas em desenvolvimento pela comunidade de computação em nuvem.

4.2.2. Banco de Dados como um Serviço

Um grande impacto foi causado na indústria de software ao prover um Sistemas de Bancos de Dados como um Serviços (DaaS) e diversos desafios de pesquisa foram impostos à comunidade de banco de dados, incluindo, por exemplo, segurança, gerenciamento de recursos compartilhados e a extensibilidade. Devido a estas necessidades, os DaaS estão surgindo como um novo paradigma para a gestão de dados em ambientes corporativos, em que um prestador hospeda um banco de dados e o fornece como um serviço

[Agrawal et al. 2009].

Neste novo paradigma de gestão de dados, o usuário utiliza o serviço de dados por meio de diversas funcionalidades como por exemplo a configuração das bases de dados, esquemas, carga de dados no serviço de dados e interfaces padronizadas de interação com a base. As atividades de gerenciamento dos aplicativos de banco de dados e dos custos são transferidos do usuário para o provedor de serviços. A Figura 4.4 exibe a organização entre usuários e provedor de DaaS.

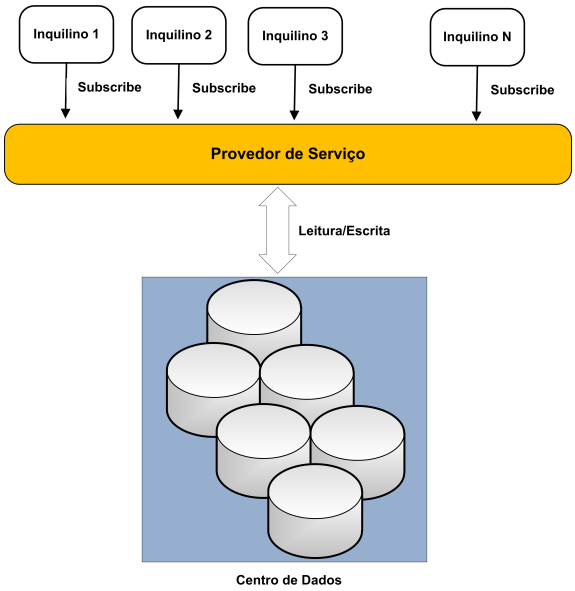

Figura 4.4. Banco de Dados como um Serviço

De acordo com a figura, cada inquilino contrata os serviço fornecido pelo provedor. Este provedor mantem um conjunto de banco de dados hospedados, em geral, em centros de dados.O provedor deve garantir aspectos de disponibilidade, desempenho e qualidade do serviço definidos em SLA.

Do ponto de vista crítico essa organização oferece uma série de benefícios aos usuários, pois estes dispõem de um ambiente altamente escalável, disponível e rápido. Além disso, não há preocupação de manter a infra-estrutura de hardware e software para fornecer o serviço de dados. Um outro ponto importante é que o provedor de serviços garante a QoS requerida pelo usuário. A própria arquitetura distribuída fornece confiabilidade no tocante à replicação de recursos.

Um sistema de banco de dados multi-inquilino deve oferecer esquemas que sejam flexíveis em dois aspectos [Aulbach et al. 2008]. Primeiro, deve ser possível estender o esquema base para suportar múltiplas versões do aplicativo, por exemplo, para regiões geográficas distintas. Uma extensão pode ser privada para um inquilino individualmente ou compartilhada por vários inquilinos. Em segundo lugar, deveria ser possível evoluir de forma dinâmica o esquema base e suas extensões, enquanto o banco de dados está em execução. Evolução de uma extensão deve ser totalmente “self-service”: o provedor de serviço não deve ser envolvido; caso contrário os custos operacionais serão muito altos.

Jacobs e Aulbach [Jacobs and Aulbach 2007] afirmam que um sistema de banco

de dados multi-inquilino pode compartilhar máquinas, processos e tabelas. De acordo com [Hui et al. 2009], existem três abordagens para a construção de um banco de dados multi-inquilino de acordo com o tipo de recurso compartilhado. A seguir, cada uma destas abordagens é apresentada e as vantagens e desvantagens são discutidas.

Banco de Dados Independentes e Instância de Banco de Dados Independente

Nesta abordagem os inquilinos compartilham apenas o hardware. O provedor de serviços executa instâncias do banco de dados independente para servir inquilinos independentes, de forma similar a instâncias de um sistemas de banco de dados convencional. Cada inquilino cria seu próprio banco de dados e armazena seus dados através da interação com uma instância dedicada do banco de dados. A Figura 4.5 ilustra está abordagem.

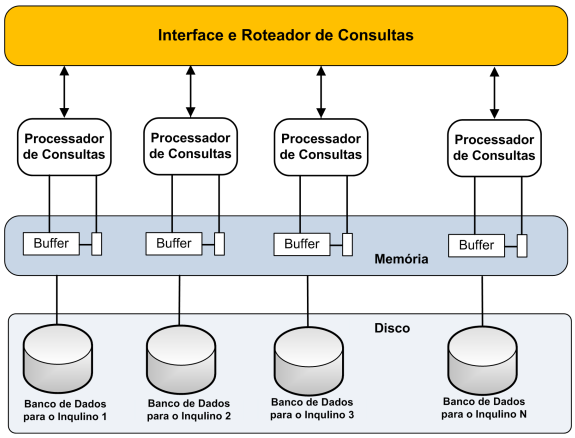

Figura 4.5. Banco de Dados Independentes e Instância de Banco de Dados Independente

Esta abordagem é a mais simples de implementar um banco de dados multi-inquilino, sendo inteiramente construída em cima dos atuais banco de dados sem nenhuma extensão. Ela fornece um bom nível de isolamento dos dados e segurança. No entanto, esta abordagem pode apresentar um alto custo de manutenção. Para gerenciar diversas instâncias de banco de dados, o provedor de serviço precisa realizar muito trabalho de configuração. Outro ponto importante a escalabilidade, já que o número de instâncias de banco de dados cresce linearmente com o número de inquilinos. Técnicas de virtualização podem ajudar a minimizar alguns problemas destas abordagem, visto que estas técnicas otimizam os recursos de hardwares e facilitam a configuração e administração de sistemas de banco de dados.

Tabelas Independentes e Instância de Banco de Dados Compartilhados

Nesta abordagem, os inquilinos compartilham o hardware e instâncias de banco de dados. O provedor de serviços mantém uma grande base de dados compartilhada e serve todos os inquilinos. Cada inquilino utiliza esquemas privados do banco de dados compartilhado. Cada nome de estrutura privada possui um identificador para indicar quem é o proprietário da tabela. Quando uma consulta é submetida, esta é reescrita de forma a

identificar o nome das tabelas modificadas e obter as respostas corretas. Esta reescrita pode ser feita por um roteador de consultas. A Figura 4.6 mostra esta abordagem.

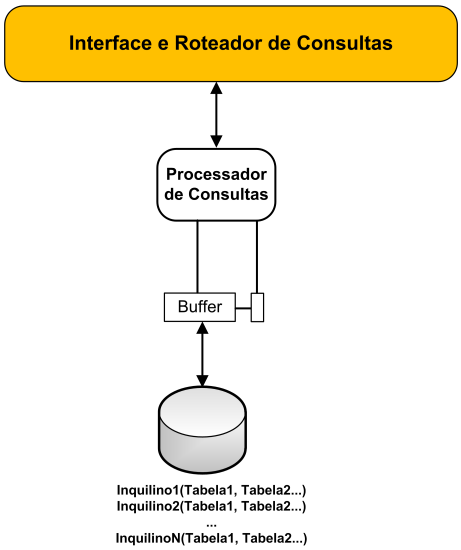

Figura 4.6. Tabelas Independentes e Instância de Banco de Dados Compartilhados

Esta abordagem reduz o custo de manutenção de gerenciamento de diferentes instâncias. Entretanto, o número de estruturas privadas no banco de dados compartilhado cresce linearmente com o número de inquilinos. Assim, a escalabilidade desta abordagem está limitada ao número de estruturas que o SGBD pode manipular e da memória disponível.

Tabelas Compartilhadas e Instância de Banco de Dados Compartilhados

Nesta abordagem os inquilinos compartilham tabelas e instâncias do banco de dados. Na fase de configuração do serviço, o provedor de serviços inicializa o banco de dados compartilhado criando tabelas vazias de acordo com o esquema base. Os inquilinos armazenam suas tuplas em tabelas compartilhadas anexando um identificador a cada tupla, que indica a qual inquilino a tupla pertence e define os atributos não utilizados com NULL. Para recuperar as tuplas na tabela compartilhada, um roteador de consultas é usado para reformular as consultas de acordo com o identificador. A Figura 4.7 ilustra esta abordagem.

Nesta abordagem, o provedor de serviço mantém apenas uma instância de banco de dados simples, o que reduz o custo de manutenção. Além disso, o número de tabelas do banco de dados é determinado pelo esquema base, sendo independente do número de inquilinos, o que fornece melhor escalabilidade. Entretanto, esta abordagem apresenta algumas desvantagens, tais como a necessidade de indexação e otimização, já que os inquilinos compartilham as mesmas tabelas, mas apresentam requisitos diferentes. Também não existem estudos detalhados referentes aos aspectos de isolamento do banco de dados e segurança neste contexto.

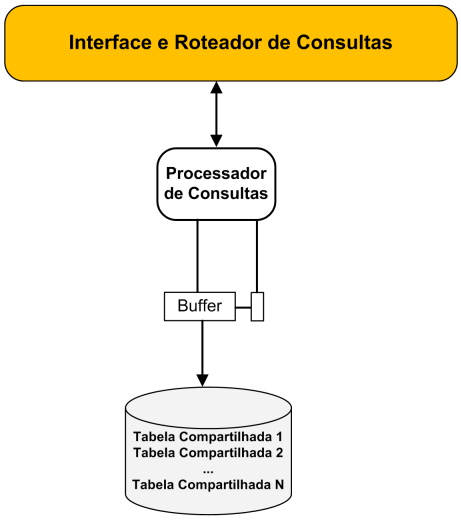

Figura 4.7. Tabelas Compartilhadas e Instância de Banco de Dados Compartilhados

4.2.3. Características do Gerenciamento de Dados em Nuvem

O gerenciamento de dados em nuvem pode ser organizado em duas classes de sistemas: (i) para apoiar aplicações com muitas atualizações e (ii) para análises dos dados e suporte a decisão. Esta primeira classe pode ser dividida em duas subclasses: uma em que o objetivo do sistema é apoiar uma única aplicação, com grandes quantidades de dados e outro no qual o objetivo do sistema é apoiar um grande número de aplicações, cada uma com pequenas quantidades de dados [Agrawal et al. 2010b].

Alguns trabalhos destacam características específicas do gerenciamento de dados em nuvem. Segundo [Voicu and Schuldt 2009], no ambiente em nuvem, os dados são gerenciados em poucos centros de dados, utilizando recursos homogêneos e, mais recentemente, recursos heterogêneos. Estes dados são acessados por meio de APIs simples, SQL ou variações. Os SGBDs em nuvem devem suportar atualizações concorrentes e transações ACID ou variações. Em relação à replicação, esta deve ser transparente para os usuários e fornecer garantias de qualidade de serviço, obtidas pela criação de réplicas dos dados em um mesmo centro de dados ou em centros de dados diferentes, além de permitir granulosidade fina dos dados. Na distribuição de dados, os SGBDs em nuvem podem apresentar um controle global central ou distribuído, devem fornecer escalabilidade e suportar cargas de trabalho inesperadas. A Tabela 4.2 resume estas características.

Uma característica essencial no ambiente de nuvem é o gerenciamento autônomo [Paton et al. 2009]. Hardware e software dentro de nuvens podem ser automaticamente reconfigurados, orquestrados e estas modificações são apresentadas ao usuário como uma imagem única. É possível identificar três diferenças em comparação com os sistemas tradicionais: intervenção humana limitada, alta alternância na carga de trabalho e uma variedade de infraestruturas compartilhadas [Agrawal et al. 2008]. Na maioria dos casos, não haverá administradores de SGBDs em nuvem ou de sistemas para ajudar os desenvolvedores que acessam um banco de dados em nuvem, fazendo com que a solução seja automatizada ao máximo. Os usuários podem variar a carga de trabalho habitual, necessitando de uma infraestrutura eficaz, pois esta será compartilhada por vários usuários ou

Distribuição	Poucos centros de dados
Ambiente	Recursos homogêneos em centros de dados
Operações para acesso aos dados	API simples, SQL ou variações
Atualização	Suporte às atualizações concorrentes
Transações	ACID ou variações
Replicação	Garantia de QoS e transparência
Granulosidade da Replicação	Fina
Controle Global	Central ou Distribuído
Alterações Dinâmicas	Escalabilidade e suporte para cargas de trabalho inesperadas

Tabela 4.2. Características do Gerenciamento de Dados em Nuvem [Voicu and Schuldt 2009]

inquilinos.

De forma a possibilitar a consolidação da computação em nuvem e dos SGBDs em nuvem, técnicas de virtualização têm se tornando populares para a implantação destes sistemas e de outros sistemas de software [Soror et al. 2010]. A virtualização apoia a consolidação dos recursos nas empresas, pois permite que uma variedade de aplicações que funcionam com recursos de computação dedicados sejam movidos para um *pool* de recursos compartilhados, o que ajuda a melhorar a utilização dos recursos físicos, simplificar a administração e reduzir custos para a empresa. Em [Soror et al. 2010] é apresentado um estudo sobre a sobrecarga de executar SGBDs em ambientes com máquinas virtuais. De acordo com o estudo, a execução não traz um alto custo de desempenho, visto que a virtualização introduz pouca sobrecarga para chamadas de sistema, tratamento de falta de páginas e operação de E/S no disco e isto não é traduzido em alta sobrecarga no tempo de execução da consulta. Chamadas de sistema e falta de páginas representam uma pequena fração no tempo de execução de uma consulta. Operações de E/S no disco correspondem a uma fração significativa do tempo, mas o retardo não é muito. Os resultados apresentados mostram que a sobrecarga média é menor do que 10%.

Associado à virtualização, o compartilhamento de infraestruturas entre inquilinos no gerenciamento de dados possibilita uma utilização eficiente dos recursos e com baixo custo de operação, além de permitir que os SGBDs em nuvem possam gerenciar uma grande quantidade de inquilinos com padrões de carga de trabalho irregulares [Elmore et al. 2010]. O conceito de multi-inquilino, uma técnica para consolidar aplicações de múltiplos inquilinos em um único sistema é frequentemente utilizada para eliminar a necessidade de sistemas separados para cada inquilino. Por exemplo, um inquilino pode ser um usuário utilizando uma aplicação que acessa um SGBD ou um SGBD instalado em uma infraestrutura. SGBDs multi-inquilino tem sido utilizado para hospedar múltiplos inquilinos dentro de um único sistema, permitindo o compartilhamento eficaz de recursos em diferentes níveis de abstração e isolamento.

Existem vários modelos de multi-inquilino e estes podem compartilhar desde máquinas até tabelas. Por exemplo, a empresa Salesforce utiliza o modelo de tabela compartilhada [Weissman and Bobrowski 2009] enquanto [Soror et al. 2010] utilizam o mo-

delo de máquina virtual compartilhada para melhorar a utilização dos recursos. Algumas características do gerenciamento de dados em nuvem aumentam a relevância de outros modelos de SGBDs multi-inquilino. Para melhorar a compreensão destes modelos, [Reinwald 2010] propõe uma nova classificação, como mostra a Tabela 4.3.

Modo de Compartilhamento	Isolamento	IaaS	PaaS	SaaS
1. Hardware	VM	*x*		
2. Máquina Virtual	Usuário SO		*x*	
3. Sistema Operacional	Instância do BD		*x*	
4. Instância	BD		*x*	
5. Banco de Dados	Esquema		*x*	
6. Tabela	Linha			*x*

Tabela 4.3. Modelos de banco de dados multi-inquilino e correspondência com a computação em nuvem [Reinwald 2010] [Elmore et al. 2010]

Os modelos correspondentes às linhas 1-3 compartilham recursos nos níveis das mesmas máquinas físicas com diferentes níveis de abstração, por exemplo, múltiplas VMs, contas de usuários diferentes e diferentes instâncias do SGBDs. Neste caso não existe compartilhamento de recursos de banco de dados e as instâncias do SGBDs se mantêm independentes. As linhas 4-6 envolvem o compartilhamento de processos de banco de dados em vários níveis de isolamento, tais como diferentes banco de dados, esquema ou *tablespace* e linha. Nos diferentes modelos, os dados dos inquilinos são armazenados de várias formas. O modelo de hardware compartilhado utiliza a virtualização para chavear várias VMs na mesma máquina. Cada VM possui apenas um processo de banco de dados e uma VM inteira, em geral, corresponde a um inquilino. Já o modelo de tabela compartilhada armazena dados de vários inquilinos em uma mesma tabela e algumas linhas de uma tabela correspondem a um inquilino.

4.2.3.1. Armazenamento e Processamento de Consultas

O armazenamento e processamento de consultas são pontos críticos no contexto da gestão de dados em nuvem [Armbrust et al. 2009]. Existem diversas abordagens para gerenciar dados em nuvem e cada sistema utiliza uma abordagem específica para persistir os dados. Dentre estas abordagens, podemos destacar novos sistemas de arquivos, *frameworks* e propostas para o armazenamento e processamento de dados. *Google File System* (GFS) é um sistema de arquivos distribuídos proprietário desenvolvido pelo Google e projetado especialmente para fornecer acesso eficiente e confiável aos dados usando grandes *clusters* de servidores [Ghemawat et al. 2003]. Em comparação com os sistemas de arquivos tradicionais, o GFS foi projetado e otimizado para ser executado em centros de dados e fornecer elevada vazão, baixa latência e tolerância a falhas individuais de servidores. Inspirado pelo GFS, o projeto de código livre *Hadoop File System* (HDFS) [Hadoop 2010] armazena grandes arquivos em várias servidores e obtém a confiabilidade por meio da replicação de dados. Similar ao GFS, os dados são armazenados em nós geograficamente distribuídos.

Para apoiar o processamento distribuído de grandes conjuntos de dados em *clusters* foi introduzido pelo Google o *framework* MapReduce [Dean and Ghemawat 2004]. No modelo MapReduce cada operação é composta por duas funções: *Map* e *Reduce*. A função *Map* recebe uma porção do arquivo de entrada e, de acordo com a especificação do usuário, emite um conjunto de tuplas intermediárias no formato chave-valor. A função *Reduce* recebe um conjunto de valores associados a cada chave, chamados de blocos. O processamento, definido pelo usuário, é realizado sobre cada bloco. Por fim, cada função *Reduce* emite um conjunto de tuplas que são armazenadas em arquivos de saída. Existem algumas implementações de código livre, dentre as quais se destaca o *Hadoop MapReduce*.

Algumas propostas para o armazenamento e processamento utilizam a estrutura chave-valor em uma *Distributed Hash Table* (DHT) [DeCandia et al. 2007]. Este valor e a chave associada são armazenados de forma desnormalizada, o que facilita a distribuição dos dados entre os nós do sistema, onde cada um destes nós possui parte dos dados. As APIs básicas de acesso são simples com operações tais como *get(key)* e *put(key, value)*. APIs mais sofisticadas permitem a execução de funções definidas pelo usuário no ambiente do servidor tais como *execute(key, operation, parameters)* ou *mapreduce(keyList, mapFunc, reduceFunc)*. Para acessar ou modificar qualquer dado é necessário fornecer uma chave, já que a busca é realizada utilizando a igualdade. É possível realizar pesquisas com outros critérios, tais como maior ou menor ou expressões booleanas. Neste caso, são utilizadas técnicas de busca exaustiva e árvores B+ distribuídas.

Outra abordagem para armazenar e processar dados em nuvem consiste em utilizar uma estrutura de colunas ou *arrays* multidimensionais [Chang et al. 2006]. Os dados são organizados em tabelas e estas possuem diversas colunas. Cada coluna armazena um valor, acessado por meio de uma chave. Nesta abordagem, todos os valores de uma coluna são serializados em conjunto, os valores da coluna seguinte também, e assim por diante. Com isso, os dados semelhantes, de mesmo formato, podem ser agrupados, auxiliando no armazenamento e na recuperação de informação. Além disso, diversas técnicas tais como a fragmentação de dados e o processamento OLAP tornam-se mais eficientes para dados gerenciados com esta abordagem.

Outras abordagens para o armazenamento e processamento de consultas gerenciam os dados de tal sorte a refletir a estrutura de um documento ou tratam os dados na forma de grafos. A abordagem orientada à documento, as técnicas são desenvolvidas para o armazenamento, modelagem, consulta e apresentação de dados de forma a refletir a estrutura de um documento, que, em geral, não possui um esquema pré-definido. Neste contexto, o formato *JavaScript Object Notation* (JSON) tem sido bastante utilizado, já que é simples criar e manipular dados neste formato, facilitando a utilização desta abordagem.

Na abordagem baseada em grafo [Rodriguez and Neubauer 2010], os dados são gerenciados da seguinte forma: (i) *nó (vértice)*: possui o mesmo conceito de uma instância de um objeto com identificador único; (ii) *relacionamento (arestas)*: fornece uma ligação entre dois nós, sendo que estes nós possuem uma direção e um tipo de relacionamento e (iii) *propriedade (atributo)*: são pares de *strings* chave-valor do objeto que podem existir tanto em um nó quanto em um relacionamento. Esta abordagem auxilia na modelagem de estruturas complexas e por meio da representação utilizando grafos, a navegação entre

os nós e os relacionamentos apresenta bons resultados.

4.2.3.2. Transações

Os conceitos de processamento de transações são de extrema importância em ambientes centralizados e distribuídos, sendo que nestes últimos é comum haver dados replicados e alocados em locais geograficamente distantes. Estes conceitos definem o nível de consistência e integridade dos dados nestes ambientes. A utilização de transações distribuídas define o controle do processamento de dados em todo ambiente de computação em nuvem e tem a responsabilidade de garantir as propriedades ACID ou variações destas no ambiente. Neste sentido, necessita-se utilizar protocolos de replicação de dados, terminação distribuída e sincronização de acesso devido à natureza compartilhada dos recursos.

Um ponto fundamental na construção de sistemas distribuídos e considerado por todos os sistemas em nuvem é o teorema *Consistency*, *Availability*, *Partition Tolerance* (CAP) [Brewer 2000] [Gilbert and Lynch 2002]. Este teorema mostra que os sistemas distribuídos não podem assegurar as seguintes propriedades simultaneamente:

- *Consistência*: todos os nós tem a mesma visão dos dados ao mesmo tempo.
- *Disponibilidade*: falhas em nós não impedem os demais nós de continuar a operar.
- *Tolerância a partições*: o sistema continua a operar mesmo com a perda arbitrária de mensagens.

Um sistema distribuído pode suportar apenas duas dessas três propriedades ao mesmo tempo. Como uma forma simples de entender um assunto complexo, o teorema CAP tornou-se um modelo popular para compreender aspectos de sistemas distribuídos. No entanto, esta simplicidade pode conduzir a interpretações incorretas do teorema. Estas propriedades não devem ser interpretadas no sentido de que o sistema é disponível ou consistente, mas que quando ocorre uma falha de rede, é necessário escolher qual propriedade torna-se mais importante para o sistema.

Em decorrência deste teorema, algumas abordagens para o gerenciamento de dados em nuvem têm utilizado diferentes formas de consistência. Uma alternativa é utilizar a abordagem *Basically Available, Soft state, Eventually consistent* (BASE) [Pritchett 2008], que é caracterizada pelo sistema ser basicamente disponível, pois este parece estar em funcionamento todo o tempo; em estado leve, já que o sistema não precisa estar sempre consistente; e eventualmente consistente, que define que o sistema torna-se consistente em um determinado momento.

De acordo com [Vogels 2009], existem duas formas de caracterizar a consistência: *do ponto de vista do programador/cliente* - como os dados são observados e atualizados e *do ponto de vista do servidor* - como é processado o fluxo das atualizações por meio do sistema e que garantias este pode fornecer, no que diz respeito às atualizações. Por exemplo, podem-se definir alguns tipos de consistência. A *consistência forte* garante que após uma atualização ser concluída, qualquer acesso subsequente fornecerá o valor atualizado.

Por outro lado, na *consistência fraca*, o sistema não garante que os acessos subsequentes irão retornar o valor atualizado. O período entre uma atualização e o momento no qual é garantido ao observador que este irá sempre visualizar o valor atualizado é denominado *janela de inconsistência*.

A *consistência eventual* é uma forma específica de consistência fraca, na qual o sistema garante que, se nenhuma atualização for feita ao dado, todos os acessos irão devolver o último valor atualizado. Se não ocorrer nenhum erro, o tamanho máximo da janela de inconsistência pode ser determinado com base em fatores tais como os atrasos de comunicação, a carga do sistema e o número de réplicas envolvidas do esquema de replicação. O sistema mais popular que implementa consistência eventual é o *Domain Name System* (DNS).

O modelo de consistência eventual apresenta um número de variações tais como consistência causal, consistência leitura/escrita, consistência de sessão, consistência de leitura/escrita monótona. Estas variações podem ser combinadas com o objetivo de tornar mais simples a construção de aplicações e permitir aos sistemas melhorarem a consistência e fornecer maior disponibilidade. Sob o ponto de vista do servidor, o nível de consistência depende de como as atualizações são propagadas entre as réplicas dos dados. Isto permite melhorar a taxa de transferência e fornecer escalabilidade. Contudo, o desenvolvedor deve estar consciente sobre quais garantias de consistência são fornecidas pelo sistema na construção de aplicações.

4.2.3.3. Escalabilidade e Desempenho

A nuvem é composta por uma enorme rede de máquinas que necessita ser escalável. A escalabilidade deve ser transparente para os usuários, podendo estes armazenar seus dados na nuvem sem a necessidade de saber a localização dos dados ou a forma de acesso. Na nuvem, os desenvolvedores dispõem de ambientes escaláveis, mas eles têm que aceitar algumas restrições sobre o tipo de software que se pode desenvolver, desde limitações que o ambiente impõe na concepção das aplicações até a utilização de SGBDs em nuvem do tipo chave-valor, ao invés de SGBDs relacionais.

De forma geral, é possível identificar duas dimensões de escalabilidade: vertical e horizontal. Na escalabilidade vertical melhora-se a capacidade do hardware, incrementando individualmente os nós existentes, como por exemplo, por meio da disponibilização de um servidor com mais memória física ou da melhoria da largura de banda que conecta dois nós. Isto funciona razoavelmente bem para os dados, mas tem várias limitações tais como a aquisição constante de hardware de maior capacidade, o que pode criar dependência de fornecedores, aumentando os custos. A escalabilidade horizontal consiste em adicionar mais máquinas à solução atual de tal modo que seja possível distribuir as requisições entre estas máquinas. No caso dos dados, pode-se agrupá-los por funções e distribuí-los por vários SGBDs. Dessa forma, ocorre a fragmentação dos dados em SGBDs em nuvem e cada um destes sistemas pode ser dimensionado de forma independente. Este tipo de escalabilidade oferece maior flexibilidade, mas necessita de um planejamento específico.

A escalabilidade envolvendo dados é uma tarefa complexa, já que a maioria dos SGBDs em nuvem utiliza arquiteturas não compartilhadas, tornando a disposição dos

dados um ponto chave. Pode-se pensar que adicionar dinamicamente um novo servidor de banco de dados é tão simples como dividir os dados em mais um servidor. Por exemplo, se há dois servidores, cada um com 50% do total dos dados e se adiciona um terceiro servidor, poder-se-ia colocar um terço dos dados em cada servidor, fazendo com que cada um dos três servidores ficasse com 33% dos dados. Entretanto, não é tão simples assim, pois as consultas dos usuários envolvem dados relacionados em diferentes servidores, o que exige o transporte dos dados, diminuindo o desempenho do sistema. Por esta razão, a fragmentação dos dados deve ser feita de forma a minimizar o transporte de dados, o qual adiciona um custo relevante ao processamento.

Em relação ao desempenho, as soluções de gerenciamento de dados em nuvem devem lidar com problemas de tempo de resposta, em virtude das diferentes tecnologias e heterogeneidade do hardware utilizado, o que pode influenciar o desempenho. Por exemplo, uma falha de hardware, tais como problemas no acesso ao núcleo de uma máquina com múltiplos *cores* ou a disputa por recursos não virtualizados, causa degradação do desempenho de uma máquina do sistema. Se a carga de trabalho necessária para executar uma consulta é dividida igualmente entre as máquinas com configurações diferentes, ocasiona um atraso no processamento, visto que o tempo para completar a consulta será aproximadamente igual ao tempo para a execução da máquina com menor configuração para completar a tarefa atribuída.

4.2.3.4. Tolerância a Falhas e Distribuição de Dados

Diferentemente das abordagens anteriores, onde se procurou evitar falhas por meio da utilização de hardware de custo elevado, as infraestruturas para nuvem são construídas em cima de hardware de baixo custo e com a suposição de que máquinas e redes podem falhar. Dessa forma, as soluções desenvolvidas devem lidar com falhas, já que estas irão ocorrer em algum momento [Abadi 2009]. Para tratar as falhas, as soluções em nuvem utilizam técnicas que auxiliam a distribuição dos dados, tais como DHT, que facilita a fragmentação dos dados. Por meio da fragmentação dos dados é possível melhorar a disponibilidade e distribuir a carga, tanto para operações de escrita quanto de leitura. Se apenas uma máquina falhar, os dados pertencentes a essa máquina são afetados, mas não o armazenamento de dados como um todo.

Em relação ao processo de replicação, pode-se criar e iniciar novas réplicas rapidamente por meio de máquinas virtuais [Soror et al. 2010]. Isso permite manter muitas réplicas por máquina, o que reduz a quantidade total de armazenamento e, portanto, os custos associados. O processo de replicação pode ser realizado de forma síncrona ou assíncrona ou uma combinação destas. Isso determina o nível de consistência, confiabilidade e desempenho do sistema. Dentre os protocolos utilizados na replicação de dados pode-se destacar o de cópia primária, réplica ativa, quorum baseado em 2PC, *Paxos* [Chang et al. 2006] e o *Gossip* [DeCandia et al. 2007].

4.2.4. Classificação dos Sistemas de Gerenciamento de Dados em Nuvem

Existem diversos SGBDs em nuvem, cada um destes com características e propósitos específicos. Com o objetivo de auxiliar o estudo destes sistemas e realizar um compara-

tivo entre os mesmos, propomos uma classificação com base nos seguintes parâmetros: *modelo relacional* e *nativo para nuvem*. O primeiro parâmetro se refere à utilização do modelo relacional pelo sistema e o segundo parâmetro está relacionado ao processo de construção do sistema, ou seja, se o sistema foi concebido para atender as necessidades da computação em nuvem. Os sistemas com modelos de dados em comum foram agrupados de forma a facilitar a compreensão. Vale ressaltar que alguns destes sistemas possuem características de mais de um modelo de dados. A Figura 4.8 apresenta essa classificação e destaca alguns sistemas.

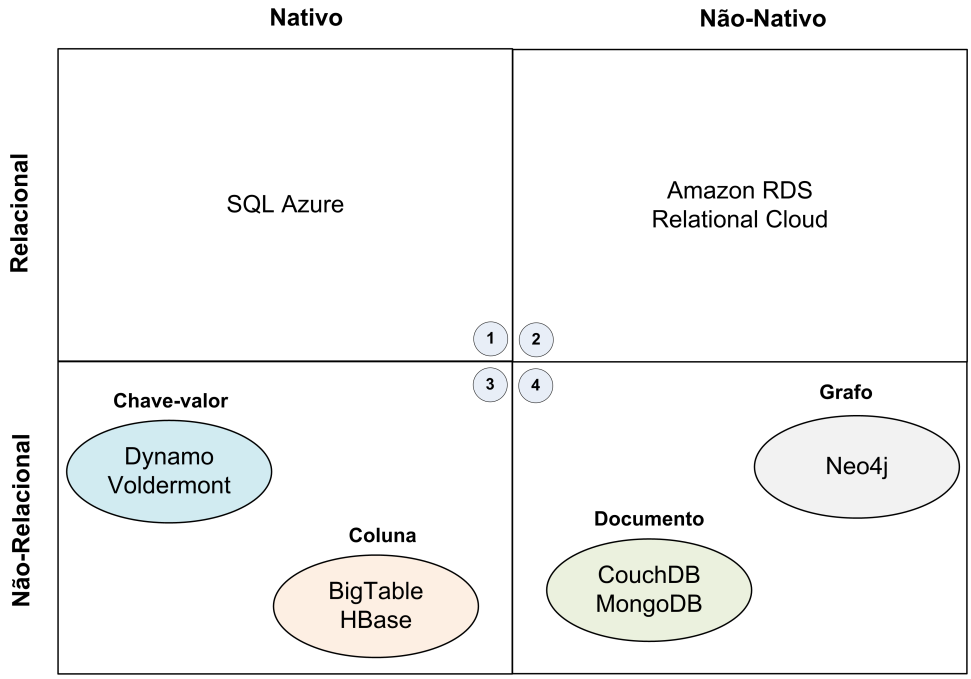

Figura 4.8. Classificação dos Sistemas de Gerenciamento de Dados em Nuvem

No primeiro quadrante estão os SGBDs relacionais desenvolvidos nativamente para nuvem. Estes sistemas estão sendo concebidos considerando as características da computação em nuvem juntamente com aspectos do modelo relacional. O exemplo mais conhecido destes sistemas é o Microsoft SQL Azure. No segundo quadrante estão SGBDs relacionais que não foram concebidos para nuvem, mas que podem ser executados em uma infraestrutura baseada em nuvem. Como exemplo destes sistemas destacam-se o Amazon *Relational Database Service* (Amazon RDS) [Amazon 2010] e o Relational Cloud [Curino et al. 2010].

No terceiro quadrante estão os sistemas considerados nativos para nuvem que não utilizam o modelo relacional, tais como os sistemas que utilizam o modelo chave-valor e baseado em coluna. Dentre os sistemas que utilizam o modelo chave-valor pode-se destacar o Amazon Dynamo [DeCandia et al. 2007] e o Voldemort. BigTable, HBase e Cassandra são exemplos de sistemas baseados em coluna [NoSQL 2010]. No quarto quadrante estão os sistemas não-nativos que não utilizam o modelo relacional, tais como grafo, documento ou XML. Estes sistemas estão sendo desenvolvidos para propósitos específicos e utilizados em ambientes em nuvem. O Neo4j é um exemplo dos sistemas baseados em grafo e CouchDB e MongoDB são exemplos de sistemas orientados a documento.

Os sistemas não-relacionais, coletivamente, iniciaram um movimento denominado *Not only SQL* (NoSQL). NoSQL é um termo genérico para uma classe definida de SGBDs não-relacionais que foram projetados para gerenciar grandes volumes de dados e, em geral, fornecem garantias de consistência fraca, como consistência eventual e utilizam estruturas e *interfaces* simples. Estes sistemas estão sendo utilizado principalmente em serviços de hospedagem de sítios, redes sociais e em outros serviços específicos.

Segundo [Stonebraker 2010], o desempenho e a flexibilidade são duas razões para utilizar estes novos sistemas. Tratando-se do desempenho, o tempo total do *Online Transaction Processing* (OLTP) está relacionado com quatro componentes: *logging*, bloqueio, *latching* e gerenciamento de *buffer*. Eliminando-se qualquer destes componentes de sobrecarga, pode-se melhorar o desempenho em 25%. Os sistemas NoSQL abordam alguns destes componentes utilizando gerenciamento eficiente de *buffer*, novas arquitetura de *thread*, suporte para transações em um único registro e consistência eventual. Dessa forma, o desempenho depende da remoção das sobrecargas e tem pouco a ver com o SQL, razão pela qual o termo NoSQL é um equívoco, já que o desempenho está relacionando com as implementações tradicionais de transações ACID, *multithreading* e o gerenciamento de disco. Para melhorar o desempenho, as sobrecargas devem ser removidas e isso é possível no contexto SQL ou em qualquer outro contexto.

4.3. Sistemas para o Gerenciamento de Dados em Nuvem

Existe uma grande quantidade de SGBDs em nuvem e, assim, não é possível destacar cada um destes sistemas [Agrawal et al. 2010a]. A seguir destacamos alguns destes sistemas e apresentamos uma análise comparativa considerando diversas características do gerenciamento de dados.

4.3.1. Microsoft SQL Azure

O Microsoft SQL Azure é composto por um conjunto de serviços para o armazenamento e processamento de dados em nuvem [Azure 2011]. O SQL Azure juntamente com o Windows Azure Storage compõem a solução de gerenciamento de dados em nuvem da Microsoft. O Windows Azure Storage inclui armazenamento persistente por meio de *blobs*, *tables* e filas. Um *blob* é um par nome, objeto que permite armazenar objetos de até 50 GB. *Tables* são diferentes das tabelas relacionais e são compostas de entidades. Elas não são acessadas usando a linguagem SQL, mas por serviços de dados. Já as filas fornecem um serviço de troca de mensagens persistentes e confiável.

O *SQL Azure Database* (SAD) é o principal componente do SQL Azure e foi construído com base na tecnologia do SGBD relacional SQL Server. Para permitir uma integração transparente de aplicações com SQL Azure, o SAD suporta os principais comandos da linguagem *Transact-SQL* (T-SQL). Esta linguagem possui diversas características, dentre os quais podemos destacar tabelas, índices, funções, procedimentos e gatilhos, entre outros. O gerenciamento do SQL Azure pode ser realizado por meio da ferramenta denominada *Houston*.

O SQL Azure implementa alta disponibilidade, tolerância a falhas e o conceito de multi-inquilino. Cada banco de dados é implementado como uma partição de dados replicados em múltiplas máquinas em um *SQL Azure Datacenter*. Com esse tipo de abor-

dagem, SQL Azure fornece um gerenciamento automático de falhas e balanceamento de carga de trabalho. A estratégia de replicação utiliza três cópias dos itens de dados para fornecer a disponibilidade e implementa consistência forte. No *SQL Azure Database*, um banco de dados individual possui um tamanho limitado a 50 GB. Para criar soluções que armazenem dados maiores do que este tamanho deve-se fragmentar grandes conjuntos de dados entre múltiplos bancos de dados e utilizar consultas em paralelo para acessá-los. Entretanto, a fragmentação dos dados é realizada de forma manual.

4.3.2. Amazon S3/SimpleDB

O Amazon *Simple Storage Service* (S3) é um sistema de armazenamento distribuído desenvolvido com base no Dynamo [DeCandia et al. 2007]. O Dynamo utiliza o modelo par chave-valor armazenados em uma DHT e não possui suporte a associações ou esquemas. Para garantir um nível de escalabilidade e disponibilidade, de acordo com o teorema CAP, o Dynamo relaxa a consistência em alguns cenários de falhas e faz uso de versões de objetos e resolução de conflitos assistida por meio de uma *interface* para os desenvolvedores. No Dynamo, as operações de leitura e escrita utilizam identificadores únicos, sendo realizadas sobre objetos binários de até 5GB e não existe suporte às operações sobre múltiplos objetos. As propriedades das transações ACID apresentam as seguintes características: atomicidade e isolamento são garantidos pela escrita de um único objeto; durabilidade é obtida por meio de escrita replicada e apenas a consistência não é forte. A API do Dynamo possui as operações *get()* e *put()*.

Escalabilidade, balanceamento de carga e suporte à heterogeneidade foram aspectos enfatizados na concepção do Dynamo e tornam relativamente simples adicionar nós, distribuir as requisições e suportar nós com características diferentes. As propriedades de simetria e descentralização são utilizadas de tal forma que todos os pontos são igualmente responsáveis e com o objetivo de evitar pontos únicos de falhas. O Dynamo usa a técnica de *hashing consistente* para prover a fragmentação e a replicação. Para tratar o problema de balanceamento de carga, decorrente da utilização de poucos nós ou nós heterogêneos, o Dynamo utiliza a estratégia de nós virtuais, onde cada nó físico pode ter vários nós virtuais associados. Dessa forma, máquinas com maior capacidade de processamento e armazenamento podem ter uma maior quantidade de nós, sendo estes distribuídos aleatoriamente. O Dynamo utiliza o protocolo *Gossip* para verificar se um nó pertence ao sistema e para detecção de falhas.

O Amazon S3 utiliza o conceito de objeto, entidade fundamental que consiste de dados, metadados e um identificador único. Os objetos são organizados em *buckets* e estes servem para agrupar objetos ou espaço de nomes. Para tratar a ocorrência de falhas, os dados são propagados entre os centros de dados de forma postergada e o usuário pode especificar a localização geográfica de um *buckets*. No S3, as operações de escrita são atômicas, mas podem ser executadas sobre múltiplas chaves. Entretanto, este sistema não fornece mecanismos de bloqueio.

O SimpleDB é um sistema de armazenamento hierárquico de dados construído para superar as limitações do S3. O SimpleDB possui atributos múltiplos, indexação e suporte a consultas. O armazenamento de dados é livre de esquema, escalável e eficiente para cenários onde as operações de leituras são predominantes, tais como fóruns de dis-

cussão e *backup*. O modelo de dados do SimpleDB utiliza o conceito de domínios, que podem ser comparados a uma tabela em SGBDs relacionais, com a diferença que um domínio pode conter um conjunto diferente de atributos para cada item. Cada item pode ter vários valores para cada atributo. Os itens são identificados por domínio e os atributos são organizados na forma de par chave-valor, sem definição de tipo, ou seja, uma cadeia de caracteres indexados automaticamente. Para criar relacionamentos no SimpleDB, o desenvolvedor pode desnormalizar seus dados ou tratar as relações na camada de aplicação. O SimpleDB possui um linguagem de consultas semelhante ao SQL e suporta seleção, operadores, ordenação e contagens, mas uma consulta pode ser processada apenas em um domínio. O SimpleDB tem algumas restrições em relação ao tamanho e número de domínios, tempo da consulta, não permite junções e comparações de itens dentro de uma consulta, suporte a transações, entre outros.

4.3.3. Bigtable/Google App Engine datastore

O BigTable é um sistema de armazenamento de dados distribuído em larga escala. Ele funciona como um SGBDs orientado a colunas e utiliza o GFS para gerenciar os dados, podendo ser utilizado juntamente com o *MapReduce* para distribuir o processamento dos dados [Chang et al. 2006]. O BigTable não suporta um modelo relacional, mas fornece um modelo simples e flexível. No modelo de dados do BigTable, uma tabela é um mapa esparso, distribuído, persistente e multidimensional, organizado em três dimensões: linha, coluna e *timestamp*, formando uma célula. Linhas com chaves consecutivas são agrupadas em *tablets*, que são unidades de distribuição e balanceamento de carga. As linhas são a unidade básica de atomicidade e alterações de uma única linha são sempre transacionais. As colunas são agrupadas em famílias que formam a unidade de controle. Várias versões de uma mesma célula podem ser armazenadas e indexadas pelo *timestamp*. BigTable não suporta nenhum tipo de junção e relações são gerenciadas na camada de aplicação.

O Bigtable tem três componentes principais: uma biblioteca comum aos clientes, um servidor mestre e vários servidores de *tablets*. O servidor mestre é responsável por designar *tablets* para os servidores de *tablets*, balanceamento de cargas e por gerenciar alterações nos esquemas dos dados. Cada servidor de *tablet* responde às requisições de leituras e escritas nas tabelas e o serviço de bloqueio denominado *Chubby* é usado para administrar os servidores. O serviço *Chubby* utiliza o protocolo baseado em quorum *Paxos* para resolver o problema de consenso distribuído e fornecer bloqueios exclusivos para os arquivos gerenciados. O BigTable fornece consistência forte, mas não replica o banco de dados diretamente, pois um *tablet* pode ser atribuído para apenas um servidor de *tablet* por vez. A replicação dos dados é realizada por meio do GFS, que gerencia o armazenamento dos *tablets* e dos arquivos de *log*.

O Google App Engine (GAE) datastore é uma solução de gerenciamento de dados baseado no BigTable. O GAE *datastore* utiliza um modelo de dados livre de esquema, suporta tipos de dados primitivos e armazena os objetos de forma serializada no BigTable. O GAE *datastore* suporta transações realizadas sobre várias entidades, mas exige que todos os objetos em uma transação estejam dentro do mesmo grupo de entidade. O GAE *datastore* utiliza consistência forte, similar ao BigTable. Muitos tipos de dados comuns são suportados no armazenamento de dados, incluindo *String*, *int*, *float*, e *DateTime*. O GAE *datastore* utiliza uma chave como identificador único para atributos como linha,

links, entre outros e pode ser acessado com a linguagem de consulta *Google Query Language* (GQL), um subconjunto da linguagem SQL. A linguagem GQL possui suporte a seleção, ordenação e alguns operadores. A instrução de seleção pode ser realizada em uma única tabela, ou seja, não suporta junções, assim como associações ou chaves compostas em decorrência do modelo de dados utilizado. Dessa forma, associações devem ser implementadas na camada de aplicação.

4.3.4. CouchDB

O CouchDB é um SGBD orientado a documento e livre de esquema [Apache 2010]. O CouchDB possui uma série de características que torna sua utilização viável em servidores que possuem hardware de baixo desempenho e utiliza técnicas de armazenamento e controle de concorrência baseadas na estrutura do documento. Este sistema foi implementado utilizando a plataforma *Erlang OTP* devido à ênfase desta plataforma em tolerância a falhas e oferecer escalabilidade, disponibilidade e confiabilidade, mesmo ao executar em hardware que está sujeito à falhas.

O CouchDB organiza os dados em documentos e cada um destes possui um identificador único. Cada documento pode ter qualquer quantidade de atributos e cada atributo pode conter listas ou objetos. Os documentos são armazenados e acessados como objetos *JavaScript Object Notation* (JSON). Operações de atualização são executadas sobre todo o documento e o CouchDB gerencia as alterações por meio de um identificador de revisão contido em cada documento. O CouchDB utiliza uma estrutura de arquivo baseada em árvores B+ para persistir os dados.

O CouchDB não utiliza os conceitos de tabela, chave, relacionamento ou junção. O modelo de consulta do CouchDB consiste nos conceitos de *visões*, que são construídas utilizando funções *MapReduce* e de uma API de consultas via *http*. Uma *visão* é basicamente uma coleção de pares chave-valor que, juntamente com o *MapReduce* permite ao usuário criar um programa de mapeamento e outro de redução para filtrar e agregar dados em uma base de documentos. O CouchDB implementa o protocolo de controle de concorrência multi-versão (MVCC) adaptado para documento e utiliza o modelo de consistência eventual. Adicionalmente, o CouchDB possui um mecanismo incremental de replicação com detecção e gerenciamento bi-direcional de conflitos.

4.3.5. Cassandra

Cassandra é um sistema de armazenamento distribuído para o gerenciamento de grandes quantidades de dados espalhados por centenas de máquinas [Lakshman and Malik 2010]. Este sistema foi desenvolvido para ter uma alta disponibilidade, consistência eventual, escalabilidade incremental, replicação otimista e com baixo custo de administração. O sistema Cassandra pode funcionar em hardware de baixo custo e lida com alta taxa de escrita sem sacrificar a eficiência na leitura. Por utilizar uma arquitetura distribuída, Cassandra não possui um ponto único de falha. Este sistema é tolerante a falhas e, com isso, aumenta a confiabilidade do sistema, já que os dados são automaticamente replicados em vários nós de um ou mais centros de dados e o protocolo *Gossip* é utilizado para tratar às falhas. A vazão de operações de leitura e escrita no sistema pode crescer linearmente e novas máquinas são adicionadas sem nenhum custo ou interrupção para as aplicações.

Cassandra é um híbrido que combina a arquitetura descentralizada do Dynamo com o modelo de dados do BigTable e possui uma API simples como meio de acesso aos dados. Cassandra possui um modelo de dados na forma de um *hash* de quatro ou cinco dimensões: espaço de chave, família de colunas, coluna, chave e super-coluna. O espaço de chave é um local para as famílias de colunas e, tipicamente, é definido uma por aplicação. Uma família de colunas é o local para agrupar as colunas, assim como uma tabela em um SGBD relacional e é acessada por meio de uma chave. Uma chave é similar aos outros sistemas de armazenamento chave-valor. Uma super-coluna pode ser comparada a uma coluna que possui sub-colunas e é utilizada para modelar tipos de dados complexos.

A coluna é a menor unidade de dados existente no modelo de dados do Cassandra e é formada por uma tripla que contém um nome, um valor e um *timestamp*. Todos os valores são fornecidos pelo usuário, incluindo o *timestamp*. Neste caso, os relógios dos usuários devem ser sincronizados, pois estes *timestamps* são usados para resolução de conflitos. Uma família de colunas contém uma lista ordenada de colunas, que o usuário pode fazer referência ao nome da coluna. Cada família de coluna é armazenada em um arquivo separado e este arquivo é classificado por uma chave, que determina qual dado é armazenado na máquina. Cassandra determina os nós responsáveis pelos dados e, quando um usuário submete uma solicitação de escrita para um nó aleatório, as operações de escrita são registradas e então aplicadas a uma versão na máquina. Um *log* com as operações consolidadas é armazenado em um disco dedicado para a máquina. Para operações de escrita não existem bloqueios e a atomicidade é garantida pelo acesso por meio de uma chave. Além disso, as escritas são aceitas durante cenários de falhas, o que aumenta a disponibilidade.

Cassandra não fornece índices automaticamente e o usuário precisa definir soluções para melhorar o desempenho. Uma alternativa é utilizar estratégias que envolvam família de colunas por consulta. Com isso, é possível melhorar a organização dos dados e minimizar o esforço no acesso e no processamento de consultas. Cassandra também possui algumas limitações, como por exemplo, todos os dados de um único registro devem pertencer ao disco de uma máquina do cluster. Isso porque as chaves dos registros são utilizadas para determinar os nós responsáveis por replicar dados. Outra limitação é que o valor de uma coluna não pode ultrapassar 2GB.

4.3.6. Relational Cloud

Relational Cloud é um SGBD como um serviço concebido com o objetivo de consolidar as funcionalidades de gestão de dados para grandes empresas e *outsourcing* do gerenciamento de dados em nuvem para pequenas e médias empresas [Curino et al. 2010]. O Relational Cloud prove disponibilidade por meio de replicação transparente, gerenciamento automático das cargas de trabalho e migração de dados em tempo de execução, além de oferecer suporte a transações distribuídas serializáveis.

O Relational Cloud foca na fragmentação e alocação dos dados, migração de dados em tempo de execução e análise de carga de trabalho. Em relação à fragmentação, o Relational Cloud propõe um novo algoritmo com o objetivo de minimizar a probabilidade de uma determinada operação ter que acessar múltiplos nós para fornecer uma resposta.

A migração em tempo de execução é obtida por meio de uma estratégia que prevê quando uma adaptação será necessária antes que algum nó do sistema seja sobrecarregado. Para tanto, o deslocamento de pequenos conjuntos de dados é realizada durante a execução. A alocação e análise de carga de trabalho são realizadas por meio de técnicas estáticas e dinâmicas de caracterização da carga de trabalho, seleção de SGBDs, atribuição de cargas de trabalho para as instâncias de banco de dados e atribuição de instâncias de banco de dados para nós físicos.

O Relational Cloud foi projetado para ser executado em máquinas em um único centro de dados. Cada máquina física executa várias instâncias de um SGBDs. Estas instâncias podem utilizar sistemas de armazenamento diferentes, visto que sistemas especializados, em geral, são eficientes para tarefas específicas. Cada banco de dados é dividido em partições lógicas por meio de técnicas de fragmentação. Essas partições são armazenadas em grupos de réplicas com o objetivo de garantir a disponibilidade e tolerância a falhas. Um grupo de réplica consiste de várias instâncias do banco de dados sendo que cada uma armazena cópia dos dados de uma partição lógica. A fragmentação dos dados e alocação dos grupos de réplicas nas máquinas são controladas pelo analisador de carga de trabalho.

A comunicação entre as aplicações e o Relational Cloud é realizada por meio de *interfaces* padrão ou protocolos conhecidos, tais como a *interface* JDBC. As consultas SQL recebidas são enviadas para um roteador, responsável por analisar e verificar os metadados do banco de dados e determinar o plano de execução. Por fim, o sistema de transações distribuídas distribui a carga de trabalho, assegurando a serializabilidade e o tratamento de falhas. Por meio do monitoramento constante de desempenho e carga de trabalho, o Relational Cloud ajusta a fragmentação dos dados e as opções de localização em tempo de execução. Falhas no sistema e alterações de carga de trabalho exigem a evolução do esquema de fragmentação e alocação em tempo de execução. Para tanto, é necessária a migração dos dados baseada nas instâncias do mecanismo de armazenamento, o que melhora o desempenho do sistema.

4.3.7. Outros Sistemas de Gerenciamento de Dados em Nuvem

4.3.7.1. Yahoo PNUTS

PNUTS é um SGBD distribuído para aplicações web desenvolvido pelo Yahoo. Este sistema apresenta diferentes níveis de consistência, replicação para áreas geograficamente distantes, escalabilidade, tolerância a falhas, alta disponibilidade e baixa latência para um grande número de requisições simultâneas [Cooper et al. 2008]. Este sistema utiliza um modelo de dados relacional simplificado, onde os dados são organizados em tabelas de registros com atributos e permite uma estrutura arbitraria dentro de um registro, tais como *blobs*. O esquema de dados é flexível, já que um novo atributo pode ser adicionado sem suspender consultas ou atualizações ativas e ter atributos vazios em um registro. PNUTS possui integridade referencial, junções ou agregação. A linguagem de consulta suporta seleção e projeção em uma tabela simples e atualizações e exclusões são realizadas apenas pela chave primária.

As réplicas são geograficamente distribuídas em diversas regiões. As tabelas são

fragmentadas horizontalmente, criando *tablets*, que é apenas um grupo de registros de uma determinada tabela. Uma cópia de um tablet é alocada em cada região. As unidades de armazenamento podem utilizar um sistema de arquivo ou MySQL. Roteadores mantem copia em cache dos mapeamentos internos para os tablets e os controladores de tablets gerenciam o intervalo de mapeamento, tabelas fragmentadas e o balanceamento de carga. PNUTS fornece novas garantias de consistência por registro, denominada consistência *timeline*, um tipo de consistência entre serializabilidade e consistência eventual. Neste caso, todas as réplicas são atualizadas para um registro na mesma ordem. Uma réplica é sempre escolhida, por registro, como primária. As atualizações são executadas na réplica primária e propagadas de forma assíncrona para as outras réplicas utilizando um mecanismo *publish-subscribe*. Operações de leitura podem ser executadas em qualquer uma das réplicas ou na réplica primária, que contem informações sobre as versões.

4.3.7.2. Neo4j

SGBD baseado na estrutura em grafo, com transações ACID, índices e integração com linguagens de programação. Este sistema permite gerenciar dados complexos e navegar entre os dados de forma simples e eficiente [Neo 2010].

4.3.7.3. Amazon Relational Database Service (RDS)

SGBD como um serviço desenvolvido pela Amazon que disponibiliza todas as funcionalidades de um SGBD relacional. Com o Amazon RDS, não é necessário instalar, configurar e manter o sistema e pode-se escolher diferentes tamanhos de instâncias [Amazon 2010].

4.3.8. Análise Comparativa

Novos serviços de dados em nuvem, como o GAE datastore, Amazon S3, SimpleDB e Cassandra utilizam modelos simplificados de dados baseados em par chave-valor ou orientados a coluna. Os dados são armazenados em estruturas otimizadas e acessados por meio de APIs simples. GAE datastore, S3/SimpleDB, CouchDB não possuem suporte a transações ACID. Cassandra utiliza um conceito simplificado de transações para linhas [Candan et al. 2009]. Estes sistemas suportam apenas consistência eventual, exceto o GAE datastore e todos apresentam alta escalabilidade e alta disponibilidade, menos o CouchDB que possui média escalabilidade, em decorrência do modelo de dados utilizado.

GAE datastore, Amazon S3, SimpleDB e Cassandra apresentam bons resultados na manipulação de grandes volumes de dados, pois os modelos de dados utilizados por estes sistemas favorecem a fragmentação dos dados. Contudo, em geral, estes sistemas não suportam operações como junções, possuem apenas garantias de consistência fraca e fornecem suporte limitado a transações. Além disso, utilizando um modelo de dados mais simples, a complexidade é empurrada para as demais camadas da aplicação, exigindo dos desenvolvedores a adição de funcionalidades mais complexas nas camadas superiores.

CouchDB e sistemas baseados em grafo optaram por modelos mais ricos de dados. Com isso, eles apresentam abstrações poderosas que tornam mais fácil modelar domínios complexos. Estes modelos de dados mais ricos introduzem maior quantidade de ligação

entre os dados, o que prejudica a escalabilidade. Vale ressaltar que estes sistemas ainda são muito recentes e existem poucos estudos detalhados sobre os mesmos, assim como ferramentas e tecnologias de apoio para sua ampla utilização.

Característica	S3 SimpleDB	BigTable GAE	Cassandra	CouchDB	SQL Azure	Relational Cloud
Modelo	Chave-valor	Coluna	Coluna	Documento	Relacional	Relacional
Armazenamento	Hash consistente	Índices	Índices	Árvore B+	Tabela	Tabela
Linguagem de Consulta	API simples	API simples	API simples	API simples	SQL	SQL
Transações	Não	Não	Sim simplificada	Não	Sim	Sim
Consistência	Eventual	Forte	Eventual	Eventual	Forte	Forte
Escalabilidade	Alta	Alta	Alta	Média	Média	Baixa
Disponibilidade	Alta	Alta	Alta	Alta	Alta	Média

Tabela 4.4. Comparativo entre os Sistemas de Ger. de Dados em Nuvem

Os sistemas SQL Azure e Relational Cloud utilizam o modelo de dados relacional, que fornece grande flexibilidade no acesso aos dados, tais como agregação e consultas com junções. Eles implementam transações ACID e garantia de consistência forte, o que facilita no desenvolvimento de aplicações. Em relação à escalabilidade e disponibilidade, SQL Azure e Relational Cloud apresentam características diferentes. O SQL Azure possui média escalabilidade, já que não oferece suporte a transações distribuídas ou consultas através de múltiplas partições de dados localizados em diferentes nós, mas apresenta alta disponibilidade. O sistema Relational Cloud apresenta baixa escalabilidade e média disponibilidade, pois se trata de um sistema em desenvolvimento e que não considera aspectos de elasticidade e ambientes virtualizados, características essenciais da computação em nuvem. O SQL Azure não possui suporte a fragmentação automática de dados, essencial para melhorar a escalabilidade, mas o Relational Cloud apresenta iniciativas neste sentido. A Tabela 4.4 mostra um resumo deste comparativo.

É importante ressaltar que os modelos de dados utilizados pelos sistemas de gerenciamento de dados em nuvem são compatíveis, ou seja, pode-se expressar um conjunto de dados em qualquer um deles. Por exemplo, é possível decompor todos os dados de um banco de dados relacional em uma coleção de par chave-valor. Dessa forma, existem contextos onde cada um destes modelos de dados é mais apropriado. De acordo com [Kossmann et al. 2010], os principais provedores tem adotado diferentes arquiteturas para seus serviços e o custo e o desempenho destes serviços variam significativamente dependendo da carga de trabalho.

4.4. Desafios e Oportunidades de Pesquisa

A computação em nuvem apresenta diversas vantagens, mas também possui uma série de desafios que devem ser superados para sua ampla utilização. A seguir destacamos alguns destes desafios no contexto do gerenciamento de dados. Outros desafios envolvendo aspectos gerais da computação em nuvem podem ser encontrados em [Armbrust et al. 2009]

[Sousa et al. 2009] [Zhang et al. 2010] .

4.4.1. Armazenamento e Processamento de Consultas

Com o aumento no volume de dados e dos requisitos para extrair valores a partir destes dados, empresas necessitam gerenciar e analisar uma grande quantidade de dados e fornecer alto desempenho. Além disso, os dados são gerenciados em várias partições, o que dificulta garantias transacionais, tais como atomicidade e isolamento. Para tratar estes problemas, diferentes soluções tem sido desenvolvidas combinando tecnologias como o MapReduce ou SGBDs paralelos [Abouzeid et al. 2009]. Um desafio é definir uma arquitetura para paralelizar o processamento de consultas com ênfase em consultas *On Line Analytical Processing* (OLAP) expressas em SQL ou em novas linguagens [Olston et al. 2008]. Esta arquitetura deve focar na interação entre o mecanismo de processamento de consultas e os sistemas de arquivos paralelos, tais como o HDFS e proporcionar diferentes níveis de isolamento.

O *framework* MapReduce e suas diversas implementações como o Hadoop e o Drayd foram projetados para o processamento distribuído de tarefas e geralmente utilizam sistemas de arquivos como o GFS e o HDFS. Estes sistemas de arquivos são diferentes dos sistemas de arquivos distribuídos tradicionais em sua estrutura de armazenamento, padrão de acesso e *interface* de programação. Em particular, eles não implementam a *interface* padrão POSIX, e, portanto, apresentam problemas de compatibilidade com aplicações e sistemas de arquivos legados [Zhang et al. 2010]. Um ponto importante é desenvolver soluções para tratar a compatibilidade com os sistemas de arquivos distribuídos tradicionais, tais como métodos para suportar o *framework* MapReduce usando sistemas de arquivos para cluster e técnicas para o acesso concorrente aos dados.

Em relação ao acesso aos serviços de dados em nuvem, os provedores fornecem linguagens com restrições, APIs simples e estas apresentam muitas limitações, tais como a ausência de consultas com junções, permitem apenas consultas por uma chave e não suportam múltiplas chaves. Considerando a grande quantidade de serviços de dados em nuvem, os desenvolvedores necessitam utilizar APIs diferentes para cada serviço. Isso exige mais esforço dos desenvolvedores e aumenta a complexidade na camada de aplicação. Um desafio é fornecer uma API comum para vários serviços, assim como uma linguagem de consulta robusta e com diversas características dos SGBDs relacionais [Cooper et al. 2009]. Outros aspectos relevantes neste contexto estão relacionados à otimização e *tuning* de SGBDs em nuvem [Agrawal et al. 2008].

4.4.2. Escalabilidade e Consistência

As soluções em nuvem focam em escalabilidade e, em geral, oferecem consistência fraca de dados, por exemplo, consistência eventual. Este tipo de consistência não permite a construção de uma ampla gama de aplicações, tais como serviços de pagamento e leilões *online*, que não podem trabalhar com dados inconsistentes [Wei et al. 2009]. Neste contexto, aspectos de armazenamento de dados, processamento de consultas e controle transacional têm sido flexibilizados por algumas abordagens para garantir a escalabilidade, mas ainda não existem soluções que combinem estes aspectos de forma a melhorar o desempenho sem comprometer a consistência dos dados [Abadi 2009]. De acordo com

[Armbrust et al. 2009], o desenvolvimento de um sistema de armazenamento que combine os diversos aspectos de computação em nuvem, de forma a aumentar a escalabilidade, a disponibilidade e a consistência dos dados é um problema de pesquisa em aberto. Dessa forma, um desafio é desenvolver soluções escaláveis e com consistência forte.

Em [Wei et al. 2009] é apresentado uma solução com suporte a transações ACID e sem comprometer a escalabilidade mesmo na presença de falhas. Contudo, esta solução necessita utilizar técnicas de desnormalização de esquemas, o que pode dificultar sua utilização. [Yang et al. 2009] propõem uma plataforma para o gerenciamento de dados que pode escalar para uma grande quantidade de pequenas aplicações e fornecer consistência forte. Para tanto, várias técnicas de SGBDs, tais como replicação, migração e gerenciamento de SLA para garantir vazão e disponibilidade foram desenvolvidas.

4.4.3. SGBDs Virtualizados

A tecnologia de virtualização tornou-se comum em modernos centros de dados e sistemas de *clusters* [Soror et al. 2010]. Enquanto muitos SGBDs já estão sendo utilizados em ambientes virtualizados, questões relacionadas à utilização destes sistemas ainda permanecem em aberto [Rogers et al. 2010]. Uma vez que o uso de máquinas virtuais pode auxiliar a provisionar recursos, um desafio é explorar a possibilidade de escalar SGBDs para tratar com cargas de trabalho inesperadas por meio da utilização de novas réplicas em máquinas virtuais recém-provisionadas [Soror et al. 2010]. Com isso, é necessário garantir o acesso consistente ao SGBD durante e após o processo de replicação, coordenar solicitações de roteamento para as máquinas virtuais antigas e novas, desenvolver políticas para a provisão de novas réplicas e modelos mais abrangentes para o planejamento da capacidade necessária para a nuvem [Aboulnaga et al. 2009]. Também é importante desenvolver soluções para otimizar as consultas em SGBDs executados em ambientes virtualizados e ajustar os parâmetros do banco de dados com a máquina virtual visando melhorar a interação entre estes sistemas [Soror et al. 2010].

Outro ponto fundamental é a alocação de recursos, já que as máquinas físicas trabalham com apenas parte de sua capacidade. Estas máquinas poderiam ser melhor utilizadas, permitindo a redução de custos. Uma questão relevante é determinar a melhor alocação das aplicações para as máquinas virtuais de acordo com métricas que maximizem a utilização dos recursos. Em [Rogers et al. 2010] é proposto uma solução para a alocação de SGBDs em ambientes virtualizados. Entretanto, não são abordados aspectos relacionados à replicação dos dados.

4.4.4. Qualidade dos Serviços de Dados

Em ambientes de computação em nuvem, a qualidade do serviço é uma característica definida entre o provedor e o usuário e expressa por meio de SLA. Com isso, o usuário do serviço tem algumas garantias, tais como desempenho e disponibilidade. Apesar das limitações de rede e segurança, as soluções em nuvem devem fornecer elevado desempenho, além de serem flexíveis para se adaptar diante de uma determinada quantidade de requisições. Como, em geral, os ambientes de computação em nuvem possuem acesso público, torna-se imprevisível e variável a quantidade de requisições realizadas, dificultando fazer estimativas e fornecer garantias de QoS. As soluções em nuvem necessitam estimar

métricas de qualidade tendo como base o estado global do sistema e o histórico de processamentos do mesmo. Entretanto, estimar métricas de qualidade é um fator desafiador nestes ambientes, pois os recursos gerenciados estão, freqüentemente, sendo expandidos ou realocados com o intuito de melhorar o desempenho.

Muitas empresas dependem de SLA, por exemplo, para exibir uma página web dentro de um determinado intervalo de tempo. Essas empresas esperam que os provedores de nuvem forneçam garantias de qualidade utilizando SLAs com base em características de desempenho. Contudo, em geral, os provedores baseiam seus SLAs apenas na disponibilidade dos serviços oferecidos, ao passo que os serviços em nuvem apresentam uma variabilidade de desempenho bastante elevada. Portanto, é essencial que os provedores ofereçam SLAs baseados em desempenho para os usuários [Schad et al. 2010]. Para muitos sistemas, a maior parte do tempo de processamento das requisições está associada ao SGBD (em vez do servidor web ou servidor de aplicação). Dessa forma, é importante que a qualidade seja aplicada no SGBD para controlar o tempo de processamento [Schroeder et al. 2006].

Uma questão relevante para garantir a qualidade em qualquer infraestrutura compartilhada é isolar o desempenho de aplicações diferentes. Aplicações podem adicionar uma carga variável sobre a nuvem e é necessário verificar como esta carga de trabalho irá afetar as outras aplicações que compartilham o mesmo hardware. Algumas abordagens para este problema utilizam quotas de recursos ou compartilhamento ponderado [Cooper et al. 2009]. Independente da abordagem utilizada, esta terá que garantir a qualidade de serviço, mesmo em condições extremas e com diferentes componentes da nuvem. Também é importante garantir a qualidade do serviço de dados independente da forma de acesso. Por exemplo, o provedor deve fornecer um serviço com garantias para um usuário que acessa este serviço por meio de um *desktop* ou de um dispositivo móvel. Para tanto, é necessário identificar a requisição e, caso necessário, alocar mais recursos de forma a garantir o SLA com o usuário. Técnicas adaptativas e dinâmicas deverão ser desenvolvidas para tornar os SGBDs em nuvem viáveis [Aboulnaga et al. 2009], além de ferramentas e processos que automatizem tarefas tais como a alocação de recursos [Agrawal et al. 2008].

4.4.5. SGBDs Multi-Inquilino

No SGBD como um serviço, os inquilinos acessam o sistema e compartilham recursos, o que pode interferir no desempenho do sistema. Dessa forma, o provisionamento de recursos deve ser eficiente, já que as cargas de trabalho dos SGBDs como um serviço são muito variáveis, visto que os inquilinos podem acessar o sistema com maior freqüência em determinados momentos. Com a ampla utilização de ambientes virtualizados, altamente compartilhados, a questão do provisionamento torna-se determinante. A distribuição dos dados é realizada por meio da fragmentação e da replicação e deve garantir que os dados dos inquilinos não sejam perdidos e que a recuperação seja simples. Para tratar a recuperação, pode-se estender a linguagem de consulta de forma a permitir o acesso considerando a distribuição dos dados dos inquilinos.

Outro aspecto importante é a falta de suporte para a arquitetura multi-inquilino. A maioria dos SGBDs foi projetada para escalar para um ou para poucos bancos de dados

grandes e eles não fornecem suporte multi-inquilino para várias aplicações executadas em hardware compartilhado. Embora seja relativamente fácil de controlar o SLA para um ou poucos bancos de dados pela adição de recursos, isto se torna um problema complexo de gerenciamento quando se necessita cumprir SLAs para milhares de aplicações que utilizam recursos compartilhados, visto que utilizar recursos dedicados não é uma opção economicamente viável devido à grande quantidade de aplicações [Yang et al. 2009].

Muitas soluções para construir SGBDs multi-inquilino necessitam de reescrita da consulta [Hui et al. 2009]. Isso ocasiona um esforço adicional para o sistema, o que pode comprometer a escalabilidade. Novas propostas tais como o BigTable permitem fornecer alternativas para a construção de SGBDs multi-inquilino e podem ser incorporadas aos sistemas existentes, melhorando o suporte multi-inquilino. Dessa forma, um desafio é desenvolver uma arquitetura multi-inquilino para SGBDs e técnicas para maximizar a quantidade de inquilinos, com baixo custo e garantias de desempenho.

4.4.6. Segurança dos Serviços de Dados

A computação em nuvem é um modelo que utiliza a Internet para disponibilizar seus serviços. Isso se torna complexo, visto que os recursos computacionais utilizam diferentes domínios de redes, sistemas operacionais, software, criptografia, políticas de segurança, entre outros. Questões de segurança devem ser consideradas para prover a autenticidade, confidencialidade e integridade. No que diz respeito à confiabilidade e responsabilidade, o provedor deve fornecer recursos confiáveis, especialmente se a computação a ser realizada é crítica e deve existir uma delimitação de responsabilidade entre o provedor e o usuário. Dessa forma, devem-se ter meios para impedir o acesso não autorizado a informações e que os dados sensíveis permaneçam privados, pois estes podem ser processados fora das empresas [Agrawal et al. 2009]. Em geral, cada sistema tem seu próprio modelo de dados e política de privacidade destes dados [Cooper et al. 2009]. Quando ocorre a movimentação de dados entre sistemas, deve-se garantir a privacidade dos dados mesmo com a mudança entre modelo de dados diferente e que aplicações multi-inquilino acessem dados de outras aplicações apenas de acordo com as políticas definidas.

Técnicas de criptografia podem ser utilizadas para garantir a privacidade dos dados. No entanto, estas técnicas têm implicações significativas de desempenho de consultas em SGBDs. Dessa forma, alternativas para a integração de técnicas de criptografia com SGBDs devem ser investigadas e desenvolvidas, já que a complexidade computacional da criptografia de dados aumenta o tempo de resposta da consulta. Em [Agrawal et al. 2009] é apresentado uma abordagem segura e escalonável para preservar a privacidade. Em vez de utilizar a criptografia, que é computacionalmente caro, é utilizada uma estratégia de distribuição dos dados em vários sítios do provedor e técnicas para acessar as informações de forma secreta e compartilhada.

4.4.7. Descrição, Descoberta e Integração de Serviços de Dados

Na computação em nuvem, vários modelos evoluíram rapidamente para aproveitar as tecnologias de software, plataformas de programação, armazenamento de dados e infraestrutura de hardware como serviços [Youseff et al. 2008]. Enquanto estes modelos se referem ao núcleo dos serviços de computação em nuvem, suas inter-relações têm sido ambíguas

e a viabilidade de sua interoperabilidade é questionável. Além disso, cada serviço da nuvem tem *interfaces* e protocolos diferentes e é complexo para os usuários encontrar e compor serviços, visto que os diversos serviços estão dispersos na Internet e possuem características distintas. Dessa forma, um desafio é desenvolver técnicas eficazes para descrever, descobrir e compor serviços na nuvem de forma a auxiliar os usuários em suas tarefas. Ontologias podem ser utilizadas para a organização do domínio de conhecimento de computação em nuvem, seus componentes e suas relações, ajudando na descrição e descoberta de serviços em nuvem [Youseff et al. 2008], assim como na composição de novos serviços a partir dos serviços existentes. Isso ajudará no projeto de serviços com interoperabilidade entre diferentes provedores, proporcionando melhorias na qualidade dos serviços.

Com a evolução da computação em nuvem, as empresas necessitam integrar os diferentes ambientes de TI, pois estas empresas utilizam modelos híbridos, nos quais os sistemas instalados possam interagir com diversos provedores. Contudo, não existem padrões de integração de sistemas de computação em nuvem [OpenCloud 2010]. O formato XML pode ser uma alternativa para mover dados entre ambientes em nuvem, mas os sistemas também precisam gerenciar dados localmente. A utilização de APIs pode auxiliar neste processo de integração. Por exemplo, as APIs da Amazon estão se tornando um padrão de fato para serviços sob demanda. Contudo, a quantidade de tecnologias envolvidas é muito grande, tornando-se um desafio padronizar as diversas *interfaces* e serviços, bem como fornecer interoperabilidade entre recursos heterogêneos. Desempenho e a evolução dos serviços são aspectos importantes na integração de nuvem, pois as aplicações possuem requisitos de QoS e as evoluções são constantes. Dessa forma, o uso de tecnologias de integração de dados, serviços e linguagens devem ser utilizadas e adaptadas no contexto da computação em nuvem.

4.4.8. Avaliação de Serviços de Dados em Nuvem

A avaliação de serviços de dados em nuvem apresenta diferenças significativas em relação aos SGBDs tradicionais. Sistemas tradicionais pressupõem a existência de configurações fixas de recursos, tratam exclusivamente da otimização de desempenho e tem como objetivo minimizarem o tempo de resposta para cada requisição. Essa visão não considera os custos operacionais do sistema. Entretanto, o modelo de pagamento baseado no uso da computação em nuvem requer que custos operacionais sejam considerados juntamente com o desempenho. No ambiente em nuvem, o objetivo é minimizar a quantidade de recursos necessários para garantir uma meta de tempo de resposta para cada requisição [Florescu and Kossmann 2009].

Para garantir a disponibilidade e a tolerância a falhas, os serviços de dados em nuvem fornecem diferentes garantias de consistência. Consistência forte implica em alto custo por transação e, em algumas situações, reduz a disponibilidade. Consistência fraca apresenta menor custo, mas resulta em alto custo operacional, por exemplo, *overselling* de um produto em uma loja virtual. Em [Kraska et al. 2009] é apresentado um novo paradigma que permite aos desenvolvedores definir garantias de consistência e o chaveamento automático destas garantias em tempo de execução com o objetivo de minimizar os custos.

Existem algumas iniciativas para a avaliação de serviços de dados em nuvem.

[Florescu and Kossmann 2009] destacam que sistemas de *benchmarks* para gerenciamento de dados devem tratar de aspectos de custo, tempo de resposta, vazão, escalabilidade, consistência, flexibilidade e discutem como estes aspectos podem impactar na arquitetura dos SGBDs. [Binnig et al. 2009] discutem porque *benchmarks* tradicionais não são suficientes para analisar os novos serviços em nuvem e apresentam idéias para o desenvolvimento de um novo *benchmark*. Em [Cooper et al. 2010] é apresentado um *framewok* com o objetivo de facilitar a avaliação de diferentes SGBDs em nuvem. Este *framewok* trata características de desempenho e escalabilidade, mas ainda não aborda aspectos de disponibilidade e replicação.

4.5. Conclusão

A computação como um serviço está finalmente emergindo e as empresas podem prestar serviços diretamente aos usuários por meio da Internet de acordo com as suas necessidades. Neste contexto, a computação em nuvem é um paradigma que está cada vez mais popular. Diversas empresas apresentaram suas iniciativas na promoção da computação em nuvem. A comunidade científica também tem apresentado algumas iniciativas, principalmente com foco em suas necessidades. Este trabalho apresentou os principais aspectos de computação em nuvem, destacando o gerenciamento de dados.

Foi possível perceber que a computação em nuvem ainda não tem uma definição clara e completa na literatura, mas existe um grande esforço neste sentido. No gerenciamento de dados, SGBDs estão evoluindo rapidamente e os desenvolvedores podem escolher o modelo de dados mais adequado para suas aplicações. No futuro, as infraestruturas serão compostas tanto por SGBDs relacionais como também por sistemas baseados nos modelos chave-valor, coluna, documento, grafo, entre outros.

Por fim, foram discutidos alguns desafios importantes no gerenciamento de dados, tais como escalabilidade, segurança, qualidade do serviço de dados, entre outros. É importante ressaltar que, várias soluções, existentes em outros modelos computacionais, que solucionem ou atenuem estes desafios, podem ser aplicadas em ambientes de computação em nuvem. Estes desafios geram oportunidades de pesquisa que devem ser superados, de forma que computação em nuvem seja amplamente aceita e utilizada por todos.

Referências

[Abadi 2009] Abadi, D. J. (2009). Data management in the cloud: Limitations and opportunities. *IEEE Data Eng. Bull.*, 32:3–12.

[Aboulnaga et al. 2009] Aboulnaga, A., Salem, K., Soror, A. A., Minhas, U. F., Kokosielis, P., and Kamath, S. (2009). Deploying database appliances in the cloud. *IEEE Data Eng. Bull.*, 32(1):13–20.

[Abouzeid et al. 2009] Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D. J., Rasin, A., and Silberschatz, A. (2009). Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *PVLDB*, 2(1):922–933.

[Agrawal et al. 2010a] Agrawal, D., Abbadi, A. E., Antony, S., and Das, S. (2010a). Data management challenges in cloud computing infrastructures. In *Databases in Networ-*

ked Information Systems, 6th International Workshop, DNIS 2010, volume 5999 of *Lecture Notes in Computer Science*, pages 1–10. Springer.

[Agrawal et al. 2009] Agrawal, D., Abbadi, A. E., Emekci, F., and Metwally, A. (2009). Database management as a service: Challenges and opportunities. *Data Engineering, International Conference on*, 0:1709–1716.

[Agrawal et al. 2010b] Agrawal, D., Das, S., and Abbadi, A. E. (2010b). Big data and cloud computing: New wine or just new bottles? *PVLDB*, 3(2):1647–1648.

[Agrawal et al. 2008] Agrawal, R., Garcia-Molina, H., Gehrke, J., Gruenwald, L., Haas, L. M., Halevy, A. Y., Hellerstein, J. M., Ioannidis, Y. E., Korth, H. F., Kossmann, D., Madden, S., Ailamaki, A., Magoulas, R., Ooi, B. C., O'Reilly, T., Ramakrishnan, R., Sarawagi, S., Stonebraker, M., Szalay, A. S., Weikum, G., Bernstein, P. A., Brewer, E. A., Carey, M. J., Chaudhuri, S., Doan, A., Florescu, D., and Franklin, M. J. (2008). The claremont report on database research. *ACM SIGMOD Record*, 37:9.

[Amazon 2010] Amazon (2010). *Amazon Relational Database Service (Amazon RDS)*. http://aws.amazon.com/rds/.

[Apache 2010] Apache (2010). *Apache CouchDB*. http://couchdb.apache.org.

[Armbrust et al. 2009] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley.

[Aulbach et al. 2008] Aulbach, S., Grust, T., Jacobs, D., Kemper, A., and Rittinger, J. (2008). Multi-tenant databases for software as a service: schema-mapping techniques. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1195–1206, New York, NY, USA. ACM.

[Azure 2011] Azure (2011). *Microsoft Azure*. http://www.microsoft.com/azure/.

[Binnig et al. 2009] Binnig, C., Kossmann, D., Kraska, T., and Loesing, S. (2009). How is the weather tomorrow?: towards a benchmark for the cloud. In *DBTest '09: Proceedings of the Second International Workshop on Testing Database Systems*, pages 1–6, New York, NY, USA. ACM.

[Brantner et al. 2008] Brantner, M., Florescu, D., Graf, D., Kossmann, D., and Kraska, T. (2008). Building a database on s3. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, page 251, New York. ACM Press.

[Brewer 2000] Brewer, E. A. (2000). Towards robust distributed systems (abstract). In *PODC*, page 7. ACM.

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

[Candan et al. 2009] Candan, K. S., Li, W.-S., Phan, T., and Zhou, M. (2009). Frontiers in information and software as services. In *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 1761–1768, Washington, DC, USA. IEEE Computer Society.

[Chang et al. 2006] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2006). Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 15–15, Berkeley, CA, USA. USENIX Association.

[Ciurana 2009] Ciurana, E. (2009). *Developing with Google App Engine*. Apress, Berkely, CA, USA.

[Cooper et al. 2009] Cooper, B. F., Baldeschwieler, E., Fonseca, R., Kistler, J. J., Narayan, P. P. S., Neerdaels, C., Negrin, T., Ramakrishnan, R., Silberstein, A., Srivastava, U., and Stata, R. (2009). Building a cloud for yahoo! *IEEE Data Eng. Bull.*, 32(1):36–43.

[Cooper et al. 2008] Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D., and Yerneni, R. (2008). Pnuts: Yahoo!'s hosted data serving platform. *PVLDB*, 1(2):1277–1288.

[Cooper et al. 2010] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, New York, NY, USA. ACM.

[Curino et al. 2010] Curino, C., Jones, E., Zhang, Y., Wu, E., and Madden, S. (2010). Relational cloud: The case for a database service. Technical report, MIT-CSAIL-TR-2010-014. Computer Science and Artificial Intelligence Laboratory, MIT, USA.

[Dean and Ghemawat 2004] Dean, J. and Ghemawat, S. (2004). Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, Berkeley, CA, USA. USENIX Association.

[DeCandia et al. 2007] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220.

[Elmore et al. 2010] Elmore, A., Das, S., Agrawal, D., and Abbadi, A. E. (2010). Who's driving this cloud? towards efficient migration for elastic and autonomic multitenant databases. Technical Report CS 2010-05, University of California, Santa Barbara, CA, USA.

[Embarcadero 2010] Embarcadero (2010). *Database Trends Survey Report*. http://www.embarcadero.com/images/dm/technical-papers/database-survey-report.pdf .

[Florescu and Kossmann 2009] Florescu, D. and Kossmann, D. (2009). Rethinking cost and performance of database systems. *SIGMOD Rec.*, 38(1):43–48.

[Ghemawat et al. 2003] Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43.

[Gilbert and Lynch 2002] Gilbert, S. and Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59.

[Hadoop 2010] Hadoop (2010). *Apache Hadoop*. http://hadoop.apache.org.

[Hui et al. 2009] Hui, M., Jiang, D., Li, G., and Zhou, Y. (2009). Supporting database applications as a service. In *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 832–843, Washington, DC, USA. IEEE Computer Society.

[Jacobs and Aulbach 2007] Jacobs, D. and Aulbach, S. (2007). Ruminations on multi-tenant databases. In *BTW*, volume 103 of *LNI*, pages 514–521. GI.

[Kossmann et al. 2010] Kossmann, D., Kraska, T., and Loesing, S. (2010). An evaluation of alternative architectures for transaction processing in the cloud. In *SIGMOD '10: Proceedings of the 2010 international conference on Management of data*, pages 579–590, New York, NY, USA. ACM.

[Kraska et al. 2009] Kraska, T., Hentschel, M., Alonso, G., and Kossmann, D. (2009). Consistency rationing in the cloud: Pay only when it matters. *PVLDB*, 2(1):253–264.

[Lakshman and Malik 2010] Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40.

[Liu et al. 2007] Liu, S., Liang, Y., and Brooks, M. (2007). Eucalyptus: a web service-enabled e-infrastructure. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 1–11, New York, NY, USA. ACM.

[Mell and Grance 2009] Mell, P. and Grance, T. (2009). *Draft NIST Working Definition of Cloud Computing*. National Institute of Standards and Technology. http://csrc.nist.gov/groups/SNS/cloud-computing.

[Microsoft 2010] Microsoft (2010). *Architecture Strategies for Catching the Long Tail*. http://msdn.microsoft.com/en-us/library/aa479069.aspx.

[Neo 2010] Neo (2010). *Neo4j - Graph Database*. http://neo4j.org.

[NoSQL 2010] NoSQL (2010). *NoSQL East*. http://nosqleast.com.

[Olston et al. 2008] Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008). Pig latin: a not-so-foreign language for data processing. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110, New York, NY, USA. ACM.

[OpenCloud 2010] OpenCloud (2010). *The Open Could Manifesto*. http://www.opencloudmanifesto.org.

[Paton et al. 2009] Paton, N. W., Aragão, M. A. T., Lee, K., Fernandes, A. A. A., and Sakellariou, R. (2009). Optimizing utility in cloud computing through autonomic workload execution. *IEEE Data Eng. Bull.*, 32(1):51–58.

[Pritchett 2008] Pritchett, D. (2008). Base: An acid alternative. *Queue*, 6(3):48–55.

[Reinwald 2010] Reinwald, B. (2010). Database support for multi-tenant applications. *In IEEE Workshop on Information and Software as Services (WISS). Co-located with ICDE*.

[Robinson 2008] Robinson, D. (2008). *Amazon Web Services Made Simple: Learn how Amazon EC2, S3, SimpleDB and SQS Web Services enables you to reach business goals faster*. Emereo Pty Ltd, London, UK, UK.

[Rodriguez and Neubauer 2010] Rodriguez, M. A. and Neubauer, P. (2010). Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41.

[Rogers et al. 2010] Rogers, J., Papaemmanouil, O., and Cetintemel, U. (2010). A generic auto-provisioning framework for cloud databases. In *IEEE 26th International Conference on Data Engineering Workshops (ICDEW)*, pages 63 –68.

[Schad et al. 2010] Schad, J., Dittrich, J., and Quiané-Ruiz, J.-A. (2010). Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *PVLDB*, 3(1):460–471.

[Schroeder et al. 2006] Schroeder, B., Harchol-Balter, M., Iyengar, A., and Nahum, E. (2006). Achieving class-based qos for transactional workloads. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 153–, Washington, DC, USA. IEEE Computer Society.

[Soror et al. 2010] Soror, A. A., Minhas, U. F., Aboulnaga, A., Salem, K., Kokosielis, P., and Kamath, S. (2010). Automatic virtual machine configuration for database workloads. *ACM Trans. Database Syst.*, 35(1):1–47.

[Sousa et al. 2009] Sousa, F. R. C., Moreira, L. O., and Machado, J. C. (2009). *Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios*. In: MOURA, R. S. (Org.) ; SOUZA, F. V. (Org.) ; OLIVEIRA, A. C. (Org.). Escola Regional de Computação (Ceará, Maranhão e Piauí, ERCEMAPI 2009, 1. ed. EDUFPI, Piauí.

[Stonebraker 2010] Stonebraker, M. (2010). Sql databases v. nosql databases. *Commun. ACM*, 53(4):10–11.

[Vaquero et al. 2009] Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2009). A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55.

[Vecchiola et al. 2009] Vecchiola, C., Chu, X., and Buyya, R. (2009). *Aneka: A Software Platform for .NET-based Cloud Computing*, pages 267–295. In: W. Gentzsch, L. Grandinetti, G. Joubert (Eds.). High Speed and Large Scale Scientific Computing. IOS Press, Amsterdam, Netherlands.

[Vogels 2009] Vogels, W. (2009). Eventually consistent. *Commun. ACM*, 52(1):40–44.

[Voicu and Schuldt 2009] Voicu, L. C. and Schuldt, H. (2009). How replicated data management in the cloud can benefit from a data grid protocol: the re:gridit approach. In *CloudDB '09: Proceedings of the First International Workshop on Cloud Data Management*, pages 45–48, New York, NY, USA. ACM.

[Wei et al. 2009] Wei, Z., Pierre, G., and Chi, C.-H. (2009). Scalable transactions for web applications in the cloud. In *Euro-Par*, pages 442–453.

[Weissman and Bobrowski 2009] Weissman, C. D. and Bobrowski, S. (2009). The design of the force.com multitenant internet application development platform. In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pages 889–896, New York, NY, USA. ACM.

[Yang et al. 2009] Yang, F., Shanmugasundaram, J., and Yerneni, R. (2009). A scalable data platform for a large number of small applications. In *CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, Online Proceedings*, pages 1–10.

[Youseff et al. 2008] Youseff, L., Butrico, M., and Da Silva, D. (2008). Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10.

[Zhang et al. 2010] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1:7–18.