Universidade Federal do Ceará
Centro de Ciências
Departamento de Computação
Mestrado e Doutorado em Ciência da Computação

REPLIC: REPLICAÇÃO ELÁSTICA DE BANCO DE DADOS MULTI-INQUILINO EM NUVEM COM QUALIDADE DE SERVIÇO

Flávio Rubens de Carvalho Sousa

TESE DE DOUTORADO

Fortaleza
Janeiro - 2013

Universidade Federal do Ceará
Centro de Ciências
Departamento de Computação

Flávio Rubens de Carvalho Sousa

REPLIC: REPLICAÇÃO ELÁSTICA DE BANCO DE DADOS MULTI-INQUILINO EM NUVEM COM QUALIDADE DE SERVIÇO

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará como requisito parcial para o grau de Doutor em Ciência da Computação.

Orientador: Prof. Javam de Castro Machado, DSc

Fortaleza
Janeiro - 2013

FLÁVIO RUBENS DE CARVALHO SOUSA

RepliC: Replicação Elástica de Banco de Dados Multi-Inquilino em Nuvem com Qualidade de Serviço

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação na Área de Concentração Sistema da Computação.

Aprovada em: 15 de janeiro 2013

BANCA EXAMINADORA

____

Prof. Dr. Javam de Castro Machado (Orientador)
Universidade Federal do Ceará – UFC

____

Profa. Dra Carmem Satie Hara
Universidade Federal do Paraná – UFPR

____

Profa.Dra. Marta Lima de Queirós Mattoso
Universidade do Rio de Janeiro – UFRJ

____

Prof.Dr. José Antonio Fernandes de Macêdo
Universidade Federal do Ceará – UFC

____

Prof.Dr.José Neuman de Souza
Universidade Federal do Ceará - UFC

Fortaleza, 15 de janeiro de 2013

Aos Meus Pais.

AGRADECIMENTOS

A Deus, por estar sempre ao meu lado, dando-me coragem para enfrentar todos os obstáculos da vida.

Ao meu orientador Prof. Javam Machado. Por sempre ter acreditado no meu potencial e compartilhar seu conhecimento e experiência, fundamentais ao bom desenvolvimento desta tese. Obrigado pela confiança e paciência nos momentos mais delicados.

A Equipe do OLTPBenchmark, especialmente à Carlo Curino (Microsoft Research), pela colaboração e atenção dedicada a esta tese.

Aos Professores Sherif Sakr (National ICT Australia - NICTA), Carlos Fisch (UFC) e Wladimir Tavares (UFC), pela colaboração no desenvolvimento desta tese.

Ao Grupo UFC Cloud, em especial ao prof. Leonardo Moreira e Gustavo Campos, pela ajuda constante na implementação e revisão desta tese.

Aos Profs. Marta Mattoso, Carmem Hara, Neuman Souza e José Antônio de Macedo pelas valiosas sugestões no aprimoramento desta dissertação.

A Heraldo Carneiro pelo desenvolvimento do RepliX e revisão dos artigos elaborados durante esta tese.

A Amazon Web Services in Education pela bolsa de pesquisa concedida sem a qual esse trabalho não teria sido possível.

Aos meus pais, Francisco de Sousa Izidório e Maria Edileusa de C. O. Sousa, que jamais pouparam esforços na abençoada tarefa de me fazer feliz.

A minha namorada, Kaluce Sousa, a cujo amor e dedicação, devo alto percentual de minhas realizações.

Aos meus amigos da UFC Quixadá, por compartilharem comigo seus conhecimentos, experiências e alegrias.

A todos aqueles que me apoiaram durante esse período. Obrigado a todos que, de alguma forma ou de outra, deixaram algo em mim.

If I have seen further it is by
standing on the shoulders of Giants

—ISAAC NEWTON

RESUMO

Fatores econômicos estão levando ao aumento das infraestruturas e instalações de fornecimento de computação como um serviço, conhecido como *Cloud Computing* ou Computação em Nuvem, onde empresas e indivíduos podem alugar capacidade de computação e armazenamento, em vez de fazerem grandes investimentos de capital necessários para a construção e instalação de equipamentos de computação em larga escala.

Na nuvem, o usuário do serviço tem algumas garantias, tais como desempenho e disponibilidade. Essas garantias de qualidade de serviço (QoS) são definidas entre o provedor do serviço e o usuário e expressas por meio de um acordo de nível de serviço (SLA). Este acordo consiste de contratos que especificam um nível de qualidade que deve ser atendido e penalidades em caso de falha. Muitas empresas dependem de um SLA e estas esperam que os provedores de nuvem forneçam serviços baseados em características de desempenho. Contudo, em geral, os provedores baseiam seus SLAs apenas na disponibilidade dos serviços oferecidos.

Sistemas de gerenciamento de banco de dados (SGBDs) para computação em nuvem devem tratar uma grande quantidade de aplicações, *tenants* ou inquilinos. Abordagens multi-inquilino têm sido utilizadas para hospedar vários inquilinos dentro de um único SGBD, favorecendo o compartilhamento eficaz de recursos, além de gerenciar uma grande quantidade de inquilinos com padrões de carga de trabalho irregulares. Por outro lado, os provedores em nuvem devem reduzir os custos operacionais garantindo a qualidade.

Neste contexto, uma característica chave é a replicação de banco de dados, que melhora a disponibilidade, desempenho e, consequentemente, a qualidade do serviço. Técnicas de replicação de dados têm sido usadas para melhorar a disponibilidade, o desempenho e a escalabilidade em diversos ambientes. Contudo, a maior parte das estratégias de replicação de banco de dados têm se concentrado em aspectos de escalabilidade e consistência do sistema com um número estático de réplicas. Aspectos relacionados à elasticidade para banco de dados multi-inquilino têm recebido pouca atenção. Estas questões são importantes em ambientes em nuvem, pois os provedores precisam adicionar réplicas de acordo com a carga de trabalho para evitar violação do SLA e eles precisam remover réplicas quando a carga de trabalho diminui, além de consolidar os inquilinos.

Visando solucionar este problema, este trabalho apresenta RepliC, uma abordagem para a replicação de banco de dados em nuvem com foco na qualidade do serviço, elasticidade e utilização eficiente dos recursos por meio de técnicas multi-inquilino. RepliC utiliza informações dos SGBDs e do provedor para provisionar recursos de forma dinâmica. Com o objetivo de avaliar RepliC, experimentos que medem a qualidade de serviço e elasticidade são apresentados. Os resultados destes experimentos confirmam que RepliC melhora a qualidade com uma pequena quantidade de violação do SLA enquanto utiliza os recursos de forma eficiente.

Palavras-chave: Computação em nuvem, gerenciamento de dados, elasticidade, replicação.

ABSTRACT

Economic factors are causing a significant infrastructure growth for providing computing as a service, a concept known as cloud computing, in which companies and individuals are able to rent processing and storage capacity, instead of making big investments to build and provision a large scale computing platform. These services are typically hosted in data centers, using shared hardware for processing and storage.

In the cloud, the service user has some guarantees, such as performance and availability. Quality of Service (QoS) is defined between the service provider and customer and expressed through a service level agreement (SLA), which specifies a level of performance and availability that must be met and penalties in case of failure. Many companies rely on SLA and they expect cloud providers to guarantee of QoS based on performance aspects. Nevertheless, in general, providers base their SLAs only on the availability of services.

Database systems serving cloud platforms must handle a large number of applications or tenants. Multi-tenancy database has been prevalent for hosting multiple tenants within a single DBMS while enabling effective resource sharing and also to allow you to manage a large amount of tenants with irregular patterns of workload. Providing such performance goals is challenging for providers as they must balance the performance that they can deliver to tenants and the operating costs. In such database systems, a key functionality for service providers is database replication, which is useful for availability, performance, and quality of service.

Database replication techniques have been used to improve availability, performance and scalability in different environments. Solutions for database replication have focused on aspects of scalability and consistency of the system with a static number of replicas. Aspects related to elasticity and multi-tenant database have received little attention. These issues are important in cloud environments, because providers need to add replicas according to the workload to avoid SLA violation, to remove replicas when the workload decreases and also consolidate tenants.

To solve this problem, this thesis presents RepliC, an approach to database replication in the cloud with quality of service, elasticity, and support to multi-tenancy. RepliC uses information from databases and provider's infrastructure to providing resources dynamically. In order to evaluate RepliC, some experiments that measure the quality of service and elasticity are presented. Our experiment results confirm that RepliC improvements the quality of service with small SLA violations, while using resources efficiently.

Keywords: Cloud computing, data management, elasticity, replication.

SUMÁRIO

LISTA DE ABREVIATURAS

AWS - Amazon Web Services

API - Application Programming Interface

ACID - Atomicidade, Consistência, Isolamento e Durabilidade

CG - Comunicação em Grupo

EC2 - Elastic Compute Cloud

FIFO - First In First Out

FSS - Função de Satisfação do SLA

IaaS - Infrastructure as a Service

JDBC - Java Database Connectivity

PaaS - Platform as a Service

QoS - Qualidade de Serviço

QoSDBC - Quality of Service for Database in the Cloud

SaaS - Software as a Service

SLA - Acordo de Nível de Serviço

SLADB - Acordo de Nível de Serviço para Banco de Dados

SLO - Objetivos de Nível de Serviço

VM - Máquina Virtual

XML - eXtensible Markup Language

2PC - Two-Phase Commit

2PL - Two-Phase Locking

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

Esta tese apresenta RepliC, uma abordagem para a replicação de banco de dados em nuvem, cujo propósito é garantir a qualidade do serviço de dados por meio da elasticidade do serviço em ambientes multi-inquilino. Esta abordagem permite a adição e remoção de réplicas de forma dinâmica para manter a qualidade do serviço.

Neste capítulo são apresentadas a justificativa e a motivação para o desenvolvimento deste trabalho, uma descrição do problema tratado, assim como os objetivos e as contribuições. Ao final do capítulo, é descrito como está organizado o restante desta tese.

1.1 MOTIVAÇÃO

Com o avanço da sociedade humana moderna, serviços básicos e essenciais são quase todos entregues de uma forma transparente. Serviços de utilidade pública como água, gás, eletricidade e telefone tornaram-se fundamentais para nossa vida diária e são explorados por meio de um 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 vem sendo realizada com a disseminação de *Cloud Computing* ou Computação em Nuvem [Armbrust et al., 2009].

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 processamento e armazenamento são entregues sob demanda, mas toda a pilha de computação pode ser aproveitada na nuvem.

Sistemas de gerenciamento de banco de dados (SGBDs)[1] 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 *startups* 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].

De forma geral, o gerenciamento de dados em nuvem pode ser organizado em duas classes de sistemas: (i) para apoiar aplicações com muitas atualizações em tempo real (OLTP) e (ii) para analises dos dados e suporte à decisão (OLAP). Esta primeira classe pode ser dividida em duas subclasses: a primeira cujo objetivo do sistema é apoiar uma única aplicação, com grandes quantidades de dados e a segunda, onde o objetivo do sistema é apoiar um grande número de aplicações, cada uma com pequenas quantidades de dados (dezenas de MBs até poucos GB) [Agrawal et al., 2010] [Das et al., 2011]. Estes sistemas devem dar suporte a requisitos não funcionais, por exemplo, desempenho e disponibilidade [Lehner and Sattler, 2010].

De forma a possibilitar a consolidação da computação em nuvem e dos SGBDs neste ambiente, técnicas de virtualização tem 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 as empresas [Minhas et al., 2011].

Associado à virtualização, o compartilhamento de infraestrutura entre inquilinos [Microsoft, 2006] no gerenciamento de dados possibilita uma utilização eficiente dos recursos com baixo custo de operação, além de permitir aos SGBDs em nuvem gerenciar uma grande quantidade de inquilinos com padrões de carga de trabalho irregulares [Agrawal et al., 2011a]. O conceito de multi-inquilino, uma técnica para consolidar aplicações de múltiplos usuários em um único sistema é frequentemente utilizada para

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

eliminar a necessidade de sistemas separados para cada inquilino [Schiller et al., 2011]. Por exemplo, um inquilino pode ser um usuário utilizando uma aplicação que acessa um SGBD ou um SGBD instalado em uma infraestrutura. SGBD 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 [Agrawal et al., 2011b].

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 modelo de máquina virtual compartilhada para melhorar a utilização dos recursos. Entretanto, a maioria dos SGBDs foi projetada para escalar para um ou para poucos bancos de dados grandes e não fornecem suporte multi-inquilino para várias aplicações executadas em hardware compartilhado.

Na nuvem, 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 desempenho elevado, 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 qualidade do serviço [Ferretti et al., 2010]. Assim, estimar métricas de qualidade é um desafio nestes ambientes, pois os recursos gerenciados estão frequentemente sendo expandidos ou realocados com o objetivo de melhorar o desempenho.

Essas garantias de qualidade do serviço são definidas entre o provedor do serviço e o usuário e expressas por meio de um acordo de nível de serviço (SLA) [Cooper et al., 2009], que consiste de contratos que especificam um nível de desempenho que deve ser atendido e penalidades em caso de falha. Questões de qualidade de serviço podem ser abordadas sob vários pontos de vista, tais como prestar um serviço sujeito a restrições de desempenho ou como descobrir e selecionar dinamicamente um serviço com requisitos de desempenho [Entrialgo et al., 2011]. Embora seja menos complexo 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 empresas precisam garantir QoS para as suas aplicações, por exemplo, exibir uma página web dentro de um determinado intervalo de tempo. Essas empresas esperam que os provedores de nuvem garantam a qualidade do serviço 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 importante que os provedores ofereçam SLAs baseados em desempenho para os usuários [Schad et al., 2010]. Para muitos sistemas, a maior parte do tempo de execução está associada ao banco de dados (em vez do servidor web ou servidor de aplicação). Dessa forma, é essencial que a QoS seja aplicada no SGBD para controlar o tempo de processamento das requisições [Schroeder et al., 2006].

Grandes sistemas de armazenamento tais como BigTable, Dynamo, Cassandra e PNUTS foram construídos para escalar diante de um grande número de requisições simultâneas usando infraestrutura compartilhada de milhares de servidores e fornecendo tolerância a falhas. Embora bem sucedidos, estes sistemas utilizam modelos de dados simplificados e não possuem acessos baseados em atributos [Das et al., 2010], o que pode resultar em uma sobrecarga considerável na reestruturação de aplicativos legados que são predominantemente baseados em tecnologia de SGBDs relacionais [Elmore et al., 2011a] [Elmore et al., 2011b].

De acordo com [Vaquero et al., 2011], os provedores devem permitir o acesso a SGBDs relacionais com suporte a transações ACID. Trabalhos recentes mostram que os SGBDs relacionais, tais como o Azure SQL, apresentam bons resultados em diferentes cenários [Kossmann et al., 2010]. Isso indica que SGBDs relacionais podem ser utilizados em nuvem e atender a uma quantidade considerável de aplicações ou podem ser combinados com estratégias para melhorar a escalabilidade e o desempenho destes sistemas [Arnaut et al., 2011]. Além disso, uma aplicação com exigências menores de armazenamento não iria utilizar as vantagens de escala de grandes sistemas de armazenamento.

Diferentemente das abordagens anteriores, nas quais se procurou evitar falhas por meio da utilização de hardware de custo elevado, a infraestrutura para nuvem é construída em cima de hardware de baixo custo e com a suposição de que máquinas e redes podem falhar. Dessa forma, as soluções desenvolvidas para a nuvem devem lidar com falhas, já que estas irão ocorrer em algum momento e o tempo de interrupção do serviço deve ser minimizado [Abadi, 2009].

Técnicas de replicação de dados têm sido usadas para melhorar a disponibilidade, o desempenho e a escalabilidade em diversos ambientes [Bernstein and Newcomer, 2009] [Charron-Bost et al., 2010] [Kemme and Alonso, 2010] [Özsu and Valduriez, 2011]. Contudo, a maior parte destes trabalhos de replicação de banco de dados tem se concentrado em aspectos de desempenho, escalabilidade e consistência do sistema com um número estático de réplicas. Aspectos referentes ao provisionamento da capacidade de forma dinâmica, tais como a adição de réplicas *on-the-fly*, têm recebido pouca atenção. Este problema é importante em ambientes de nuvem, onde as mudanças de carga de trabalho exigem que novas réplicas do banco de dados possam ser inicializadas em prazos curtos de tempo [Cecchet et al., 2011] [Wada et al., 2011]. Para melhorar a qualidade do serviço, novas réplicas podem ser criadas para distribuir a carga de trabalho, o que aumenta o custo e a complexidade do gerenciamento.

De acordo com [Özsu and Valduriez, 2011], o gerenciamento automático da replicação para lidar com mudanças na carga de trabalho é um desafio de pesquisa interessante. Um ponto importante consiste em gerenciar de forma automática os recursos disponíveis e a carga de trabalho do sistema para garantir a qualidade do serviço e melhorar a utilização destes recursos [Sousa et al., 2011a]. Neste contexto, a elasticidade é o ponto chave para desenvolver serviços com QoS para nuvem, pois permite adicionar ou remover recursos, sem interrupções e em tempo de execução para lidar com a variação da carga [Agrawal et al., 2011a]. De acordo com [Kemme et al., 2010], um sistema é elástico se ele ajusta automaticamente o número de réplicas para a carga de trabalho atual. Elasticidade é alcançada por meio de auto provisionamento, que adiciona novas réplicas se o sistema não consegue lidar com a carga de trabalho atual ou remove réplicas desnecessárias.

Existem diversas propostas para replicação de banco de dados em nuvem. Em [Elmore et al., 2011a] [Curino et al., 2011a] não é detalhada a estratégia de replicação e os trabalhos propostos por [Yang et al., 2009] [Savinov and Daudjee, 2010] não implementam elasticidade, multi-inquilino ou QoS. Outras soluções não tratam questões de desempenho com QoS [Azure, 2012], não abordam conceitos de banco de dados multi-inquilino [Minhas et al., 2011] [Cecchet et al., 2011] [Mior and de Lara, 2011] [Sakr et al., 2011] ou tratam apenas o modelo multi-inquilino de máquina compartilhada [Xiong et al., 2011]. Contudo, estas propostas não tratam de diferentes aspectos para a replicação em nuvem, tais como elasticidade, QoS e abordagens multi-inquilino para banco de dados.

Visando a solucionar este problema, esta tese apresenta RepliC, uma abordagem para a replicação de banco de dados em nuvem com foco na qualidade do serviço de dados e elasticidade para ambientes multi-inquilino. RepliC utiliza informações dos bancos de dados e do provedor de infraestrutura para provisionar recursos. O processo de gerenciamento dos dados é automático, visto que réplicas dos bancos de dados são adicionadas ou removidas, auxiliando a gestão e o compartilhamento de recursos.

1.2 DEFINIÇÃO DO PROBLEMA

Em seu sentido mais amplo, este trabalho está relacionado com o problema de replicação de banco de dados no ambiente de computação em nuvem. Tal domínio de problema, claramente extenso, é restringido a partir de um conjunto de características. A Figura 1.1 mostra o cenário do problema abordado neste trabalho.

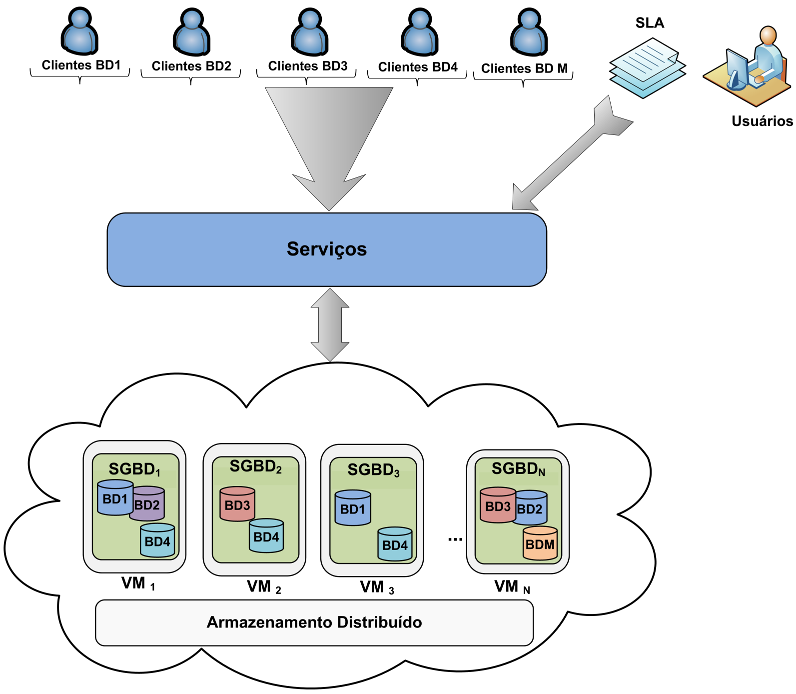

Figura 1.1 Cenário do problema abordado neste trabalho.

Um provedor fornece um conjunto de serviços de banco de dados no ambiente de computação em nuvem. O usuário especifica um SLA por serviço. Cada serviço é caracterizado por um SLA, composto por informações sobre as partes envolvidas, métricas, objetivos e penalidades para as transações que excedam o objetivo definido no SLA. A carga de trabalho associada a cada serviço é definida por uma taxa de transações variável ao longo do tempo enviada por clientes de cada serviço.

Para fornecer estes serviços, o provedor utiliza uma infraestrutura com máquinas virtuais e réplicas de banco de dados para cada serviço de acordo com a carga de trabalho. As máquinas são caracterizadas por CPU, memória, disco e um custo associado por hora de utilização. Cada réplica é caracterizada por CPU, memória e disco utilizado.

O principal problema desta tese consiste em: Como melhorar a qualidade para um conjunto de serviços de banco de dados de acordo com a carga de trabalho corrente enquanto utiliza os recursos eficientemente com pequena violação do SLA?

Hipótese da tese: *Replicação elástica de banco de dados multi-inquilino melhora a qualidade de serviço e a utilização de recursos em ambientes de nuvem.*

1.3 OBJETIVOS

O objetivo geral deste trabalho é desenvolver uma abordagem para a replicação de dados em nuvem com foco em qualidade do serviço e elasticidade para um ambiente de banco de dados multi-inquilino. Este trabalho propõe uma estratégia que monitora continuamente a qualidade de serviço do SGBD e adiciona ou remove réplicas destes bancos em função da variação da carga de trabalho de forma automática.

Para atingir este objetivo geral, foram definidos os seguintes objetivos específicos:

1. Investigar a possibilidade de se utilizar técnicas de gerenciamento de dados em ambientes distribuídos para replicação de banco de dados em nuvem, em particular, apoiar um grande número de aplicações, cada uma com pequenas quantidades de dados [Agrawal et al., 2010].

2. Projetar uma abordagem para a replicação de dados em nuvem com foco em qualidade do serviço e elasticidade para um ambiente de banco de dados multi-inquilino.

3. Analisar a eficácia da solução proposta por meio da realização de um experimento prático utilizando uma nuvem computacional.

1.4 CONTRIBUIÇÕES

As principais contribuições desta tese são:

1. *Uma abordagem para a replicação de banco de dados em nuvem*:
   Esta abordagem utiliza uma arquitetura para banco de dados multi-inquilino que propõe a elasticidade por meio da adição e remoção de réplicas com base em informações coletadas sobre a carga de trabalho. Esta abordagem é independente do sistema de banco de dados e pode ser utilizada em qualquer infraestrutura de nuvem [Sousa and Machado, 2011] [Sousa and Machado, 2012].

2. *Uma estratégia para a qualidade do serviço de dados em nuvem*:
   Esta estratégia define conceitos de SLA para SGBDs e técnicas de monitoramento para garantir a qualidade do serviço por meio do gerenciamento de réplicas. Com isso, a solução proposta mantém o nível do serviço dentro dos limites definidos [Sousa et al., 2011b] [Sousa et al., 2012].

3. *A implementação de uma arquitetura para o gerenciamento de banco de dados baseada na abordagem de replicação e QoS*:
   Este protótipo fornece suporte à infraestrutura Amazon AWS e pode ser facilmente estendido para outras infraestruturas, uma vez que todas as informações necessárias para sua utilização podem ser extraídas dos SGBDs e das infraestruturas por meio de agentes de monitoramento [Sousa et al., 2011b] [Sousa et al., 2012].

4. *Um método de avaliação de serviço de banco de dados multi-inquilino em nuvem*:
   Este método permite construir um ambiente multi-inquilino completo para avaliação e contempla diferentes aspectos, tais como qualidade do serviço e elasticidade [Sousa and Machado, 2012].

Publicações

Esta tese é baseada principalmente nas seguintes publicações:

- SOUSA, F. R. C. ; MACHADO, J. C. "Towards Elastic Multi-Tenant Database Replication with Quality of Service". In: *5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC) 2012.*

- SOUSA, F. R. C. ; MOREIRA, L. O. ; SANTOS, G. A C.; MACHADO, J. C. "Quality of Service for Database in the Cloud". In: *2st International Conference on Cloud Computing and Services Science (CLOSER)*, 2012.

- SOUSA, F. R. C. ; MACHADO, J. C. “An Approach to Database Replication in the Cloud”. In: *26th Brazilian Symposium on Databases (SBBD)*, 2011.

- SOUSA, F. R. C. ; MOREIRA, L. O. ; MACHADO, J. C. “SLADB: Acordo de Nível de Serviço para Banco de Dados em Nuvem”. In: *26th Brazilian Symposium on Databases (SBBD)*, 2011.

- SOUSA, F. R. C. ; MOREIRA, L. O. ; MACHADO, J. C. “Computação em Nuvem Autônoma: Oportunidades e Desafios”. In: *1st Workshop on Autonomic Distributed Systems (WoSiDA), SBRC*, 2011.

- SOUSA, F. R. C. ; MOREIRA, L. O. ; MACEDO, J. A. F,; MACHADO, J. C. “Gerenciamento de Dados em Nuvem: Conceitos, Sistemas e Desafios”. In: *25th Brazilian Symposium on Databases (SBBD)*, 2010.

- SOUSA, F. R. C. ; MOREIRA, L. O. ; MACHADO, J. C. “Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios”. In: *III Escola Regional de Computação Ceará, Maranhão, Piauí (ERCEMAPI)*, 2009.

- SOUSA, F. R. C.; MOREIRA, L. O.; MACHADO, J. C. “Um Middleware para o Gerenciamento de Dados XML Persistentes em Ambientes Distribuídos”. In: *24th Brazilian Symposium on Databases (SBBD)*, 2009.

Embora as publicações abaixo não estejam diretamente relacionadas a esta tese, vários conceitos de arquitetura e melhores práticas para aplicações distribuídas foram desenvolvidas e utilizadas nesta tese:

- SILVA, T. L. C. ; NASCIMENTO, M. A. ; MACEDO, J. A. F. ; SOUSA, F. R. C. ; MACHADO, J. C. “Towards Non-Intrusive Elastic Query Processing in the Cloud”. In: *4th International Workshop on Cloud Data Management (CloudDB)*, 2012.

- MOREIRA, L. O. ; SOUSA, F. R. C. ; MACHADO, J. C. “Analisando o Desempenho de Banco de Dados Multi-Inquilino em Nuvem”. In: *27th Brazilian Symposium on Databases (SBBD)*, 2012.

- REGO, P. A. L. ; COUTINHO, E. F. ; SOUSA, F. R. C. ; SOUZA, J. N. “Alocacão Autonômica de Recursos para Máquinas Virtuais Baseada em Características de Processamento”. In: *2st Workshop on Autonomic Distributed Systems (WoSiDA), SBRC*, 2012.

- MOREIRA, L. O. ; SOUSA, F. R. C. ; MACHADO, J. C. “A Distributed Concurrency Control Mechanism for XML Data”. In: *Journal of Computer and System Sciences (JCSS)*, 2011.

- MOREIRA, E. J. R. ; SOUSA, F. R. C. ; MACHADO, J. C. "RepliXP: Replicação Parcial de Dados XML". In: *23th Brazilian Symposium on Databases (SBBD)*, 2008. *(Best Poster).*

Publicações aceitas, submetidas ou em preparação desenvolvidas em conjunto com esta tese:

- COUTINHO, E. F. ; SOUSA, F. R. C. ; GOMES, D. G. ; SOUZA, J. N. "Elasticidade em Computação na Nuvem: Uma Abordagem Sistemática". In: *31st Brazilian Symposium on Computer Networks and Distributed Systems (SBRC)*, 2013. (Aceito para Publicação).
- SANTOS, G. A C. ; MAIA, G. J. R. ; MOREIRA, L. O. ; SOUSA, F. R. C. ; MACHADO, J. C. "Scale-Space Filtering for Workload Analysis and Forecast". (Submetido para Publicação).
- SILVA, T. L. C. ; NASCIMENTO, M. A. ; MACEDO, J. A. F. ; SOUSA, F. R. C. ; MACHADO, J. C. "Non-Intrusive Elastic Query Processing in the Cloud". (Submetido para Publicação).
- SOUSA, F. R. C. ; MACHADO, J. C. "RepliC: Elastic Multi-Tenant Database Replication with Quality of Service". (Em Preparação).
- SOUSA, F. R. C. ; MACHADO, J. C. "Strong Consistency for Muti-Tenant Database in the Cloud". (Em Preparação).

1.5 ESTRUTURA DA TESE

O restante desta tese está organizado em seis capítulos. As seções seguintes apresentam um sumário de cada um deles.

Capítulo 2: Gerenciamento de Dados em Nuvem

O Capítulo 2 introduz a computação em nuvem e descreve o gerenciamento de dados neste ambiente, destacando diversos aspectos, tais como armazenamento, processamento de consultas, transações e replicação.

Capítulo 3: Trabalhos Relacionados

O Capítulo 3 apresenta trabalhos que tratam da replicação de dados em nuvem. São apresentados os requisitos para a replicação no contexto de computação em nuvem, assim como uma análise comparativa entre os trabalhos relacionados.

Capítulo 4: Replicação Elástica para Banco de Dados Multi-Inquilino com Qualidade do Serviço

O Capítulo 4 apresenta RepliC, uma abordagem para a replicação de dados em nuvem, destacando uma camada de software para replicar dados. RepliC utiliza técnicas de elasticidade e migração para garantir a qualidade do serviço em ambientes multi-inquilino. Também é apresentada uma solução desenvolvida para monitorar os serviços de gerenciamento de dados em nuvem.

Capítulo 5: Consistência para Banco de Dados Multi-Inquilino

O Capítulo 5 apresenta a estratégia para a consistência de banco de dados multi-inquilino utilizada pelo RepliC. São abordados diferentes algoritmos para tratar a sincronização entre as réplicas em ambientes virtualizados e para o tratamento de falhas.

Capítulo 6: Avaliação Experimental

Este capítulo mostra detalhes sobre a implementação e os experimentos realizados com a solução proposta nos Capítulos 4 e 5 em uma nuvem computacional real. Também apresenta uma análise dos resultados obtidos com o uso desta abordagem para replicar banco de dados em aplicações submetidas para execução na nuvem.

Capítulo 7: Conclusões

Por fim, este capítulo encerra o trabalho com uma análise dos resultados obtidos e com uma discussão sobre direções para possíveis trabalhos futuros.

CAPÍTULO 2

GERENCIAMENTO DE DADOS EM NUVEM

Este capítulo descreve as características do gerenciamento de dados em nuvem. Inicialmente, é apresentada uma introdução sobre computação em nuvem. Em seguida, são apresentados os requisitos do gerenciamento de dados e, por fim, são detalhados os principais aspectos neste contexto.

2.1 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].

Com isso, os usuários estão movendo seus dados e aplicações para a nuvem e assim 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 [Sousa et al., 2009].

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 ficam 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 de base utilizada. Assim, usuários e empresas passaram a acessar os serviços sob demanda e independente de localização, o que aumentou a quantidade de serviços disponíveis [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 custo, conectadas por meio de uma rede como ilustra a Figura 2.1. 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.

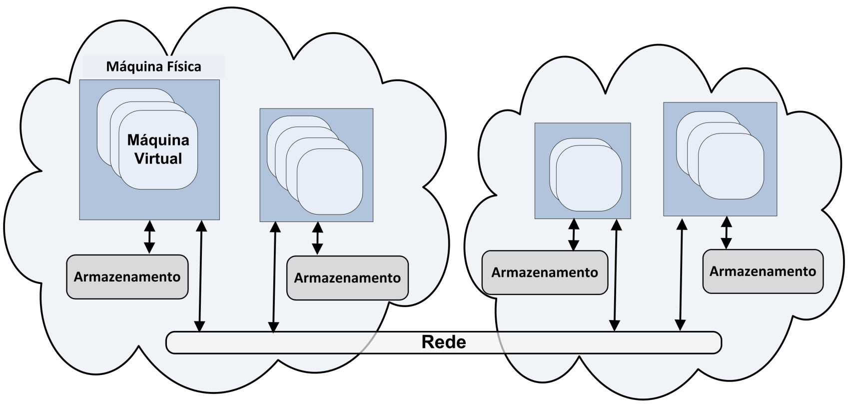

Figura 2.1 Ambiente de computação em nuvem

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 [Binnig et al., 2009].

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 rapidamente adquiridos e liberados com mínimo esforço gerencial ou interação com o provedor de serviços"* [Mell and Grance, 2011]. 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. Informações sobre os principais sistemas de computação em nuvem podem ser encontradas em [Sousa et al., 2009].

2.1.1 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.

2.1.2 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 computação 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, 2012] 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.

2.1.3 Modelos de Implantação

Quanto ao acesso e a 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.

2.2 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 para gerenciar milhares de máquinas e atender 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) elasticidade e escalabilidade, proporcionando a percepção de recursos quase infinitos. Por outro lado, o provedor tem que garantir (i) ilusão de recursos infinitos, sob cargas de trabalho dinâmicas e (ii) minimizar os custos operacionais associados a cada usuário [Curino et al., 2010a].

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] [Dash 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 de carga dinâmico 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 qualidade do serviço.

2.2.1 Requisitos do Gerenciamento de Dados em Nuvem

A definição dos requisitos é fundamental no gerenciamento de dados como um serviço. [Curino et al., 2010a] 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 2.1.

	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 2.1 Requisitos para SGBD como um Serviço [Curino et al., 2010a]

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 provisionar 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.

2.2.2 Características do Gerenciamento de Dados em Nuvem

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 dar suporte a 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 2.2 resume estas características.

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 2.2 Características do Gerenciamento de Dados em Nuvem [Voicu and Schuldt, 2009]

Uma característica essencial no ambiente de nuvem é o gerenciamento autônomo

[Paton et al., 2009]. A computação autônoma é inspirada em sistemas biológicos para lidar com desafios de complexidade, dinamismo e heterogeneidade [Kephart and Chess, 2003], características presentes nos ambientes de computação em nuvem e, assim, fornece uma abordagem promissora neste contexto. Embora a computação em nuvem apresente certas características autônomas como o provisionamento automático de recursos, seu objetivo é reduzir o custo dos recursos ao invés de reduzir a complexidade do sistema [Sousa et al., 2011a].

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 inquilinos.

De forma a possibilitar a consolidação da computação em nuvem e dos SGBDs em nuvem, técnicas de virtualização tem 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%.

Multi-Inquilino

Associado à virtualização, o compartilhamento de infraestruturas entre inquilinos no gerenciamento de dados possibilita uma utilização eficiente dos recursos 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., 2011a]. 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. Um inquilino é definido de acordo com o contexto onde se encontra inserido; por exemplo, um inquilino pode ser usuário de uma aplicação ou pode ser um banco de dados em relação ao SGBD.

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. Algumas características do gerenciamento de dados em nuvem aumentam a relevância de outros modelos de SGBDs multi-inquilino. Para melhorar a compreensão destes modelos, [Elmore et al., 2011a] propõem uma nova classificação, como mostra a Tabela 2.3.

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 2.3 Modelos de banco de dados multi-inquilino e correspondência com a computação em nuvem [Reinwald, 2010] [Elmore et al., 2011a]

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 SO de usuários diferentes e diversas instâncias dos SGBDs. Neste caso, não existe compartilhamento de recursos de banco de dados e as instâncias dos 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 bancos de dados, esquema ou *tablespace* e tupla. 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 tuplas de uma tabela correspondem a um inquilino.

Os modelos das linhas 4-6 são os mais amplamente utilizados, pois permitem um melhor compartilhamento de recursos. Por outro lado, estes três modelos apresentam maior interferência entre os inquilinos do sistema, o que pode interferir no desempenho do sistema. No modelo da linha 4, um inquilino é um banco de dados separado, que melhora o isolamento. Contudo, o modelo 4 está limitado ao número de estruturas que o SGBD pode manipular. Os modelos 5-6 utilizam um número menor de recursos dentre todos os modelos. Entretanto, estes modelos apresentam algumas desvantagens; por exemplo, o modelo 6 necessita de indexação e otimização, já que os inquilinos compartilham as mesmas tabelas, porém apresentam requisitos diferentes. Também não existem estudos detalhados referentes aspectos de isolamento do banco de dados e segurança deste modelo.

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] [Silva et al., 2012]. 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 individual de servidores. Inspirado pelo GFS, o projeto de código livre *Hadoop File System* (HDFS) [Hadoop, 2012] 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. Diferentemente de outras abordagens de sistemas de arquivos distribuídos, o armazenamento e processamento do HDFS são realizados em cada nó do sistema.

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.

O MapReduce gerencia o processamento através de um processo *master*, cuja função é de orquestrar o processamento, gerenciar o processo de agrupamento de registros e distribuir os blocos de forma equilibrada. As tarefas de paralelismo, tolerância a falhas, distribuição dos dados e balanceamento de carga são tratadas pelo MapReduce, que também oferece transparência de replicação, distribuição e sincronização. Existem algumas implementações de código livre, dentre as quais se destaca o *Hadoop MapReduce*. O Hadoop possui vários subprojetos, tais como o HBase, um sistema de banco de dados distribuído e o Pig, uma linguagem de fluxo de dados e execução para computação paralela [Olston et al., 2008] [Thusoo et al., 2010].

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 não normalizada, 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. Na abordagem orientada a 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 de desempenho.

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] [Olston et al., 2008].

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].

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 [Cooper et al., 2009] [Agrawal et al., 2008].

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.

As soluções em nuvem, 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 [Abadi, 2009]. Em [Wei et al., 2009] é apresentado uma solução com suporte às transações ACID e sem comprometer a escalabilidade mesmo na presença de falhas. Contudo, esta solução necessita utilizar técnicas para remover a normalização dos 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.

Escalabilidade

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 [Sousa et al., 2010].

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. Algumas soluções em nuvem utilizam estruturas que auxiliam na distribuição dos dados, tais como DHT, que facilita a fragmentação dos dados. De forma geral, a distribuição dos dados deve ser feita de forma a minimizar o transporte de dados, o qual adiciona um custo relevante ao processamento [Curino et al., 2010b].

Replicação 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]. A replicação de dados é a chave para tratar as falhas e melhorar o desempenho dos sistemas.

Embora replicação seja um conceito intuitivo, sua implementação requer técnicas sofisticadas. Isso ocorre pela dificuldade de manutenção da consistência de réplicas: quando um dado é alterado, suas réplicas também precisam ser atualizadas para manter um estado distribuído consistente [Saito and Shapiro, 2005]. Para manter a consistência das réplicas, são necessários protocolos específicos ou protocolos de replicação. [Gray et al., 1996] classificaram os protocolos de replicação de bancos de dados usando dois parâmetros. O primeiro parâmetro estabelece a forma de propagação das modificações, que pode ser de forma síncrona ou assíncrona. O segundo parâmetro indica quem pode propagar as atualizações: uma réplica específica, chamada de *réplica ou cópia primária* ou qualquer uma das réplicas, onde cada uma destas é denominada *réplica ativa*.

No protocolo de cópia primária, uma das réplicas é escolhida como cópia ou réplica primária e as outras cópias são réplicas secundárias. Essa cópia primária gerencia as demais e envia a resposta da operação para o cliente. Este protocolo apresenta um baixo poder de processamento, já que apenas a cópia primária realiza o processamento das atualizações. Além disso, possui algumas desvantagens como a necessidade de escolha de uma nova cópia primária, no caso de falha, e tempos de respostas inaceitáveis, quando a primária torna-se um gargalo, pois ela centraliza todas as operações de atualização, afetando a escalabilidade [Özsu and Valduriez, 2011].

No protocolo de réplicas ativas, qualquer uma das réplicas pode executar operações de atualização. Essas operações são executadas na mesma sequência por todas as réplicas, produzindo resultados idênticos. Não há uma réplica centralizadora, como no protocolo de cópia primária. Esse protocolo apresenta a vantagem de ser tolerante a falhas, já que não existe uma cópia primária, e de apresentar melhor desempenho, pois várias réplicas podem ser acessadas de forma concorrente. A principal desvantagem desse protocolo é a necessidade de um mecanismo que assegure a consistência entre as réplicas quando atualizações são executadas [Kemme et al., 2010].

Várias estratégias de propagação de atualização entre as réplicas podem ser adotadas. A escolha da melhor estratégia depende da disponibilidade de comunicação, do grau de atualização e do volume das informações requisitadas pelos usuários. As principais formas de propagação são a síncrona e a assíncrona. Na forma síncrona, quando uma réplica é alterada, essa alteração é imediatamente aplicada às demais réplicas dentro de uma transação. Nesse caso, as réplicas cooperam usando estratégias para manter a consistência das réplicas. Sistemas de banco de dados síncronos tradicionalmente utilizam o protocolo de *two-phase commit* (2PC). Neste protocolo, uma réplica é encarregada de coordenar (fase 1) e confirmar (fase 2) a difusão das modificações para as demais réplicas. Esse tipo de consistência é muito dispendioso, principalmente quando o número de participantes é grande, pois o grau de comunicação na rede é alto, e todos os participantes devem estar conectados. Em [Gray et al., 1996] foi provado que o protocolo 2PC é impraticável quando a quantidade de réplicas é grande, pois o número de *aborts*, *deadlocks* e mensagens trocadas cresce de maneira exponencialmente proporcional ao número de réplicas.

Na forma assíncrona, a alteração de uma réplica não é propagada imediatamente, sendo realizada em um momento posterior, dentro de uma transação separada [Bernstein and Newcomer, 2009]. A propagação das atualizações pode ser realizada de forma linear ou constante. A primeira consiste em enviar as atualizações a cada transação. A segunda consiste em definir intervalos de tempo configuráveis para o envio das atualizações. Geralmente, esse tipo de controle de consistência é usado quando não há necessidade de se obter os dados totalmente atualizados.

A replicação apresenta um conjunto de vantagens. Estas vantagens dependem dos padrões de acesso, do sistema em questão e do estado da rede, dos dados replicados e do esquema de replicação utilizada. Um esquema de replicação descreve como um determinado item de dados é replicado, ou seja: número de réplicas, onde essas réplicas estão alocadas e a escolha da técnica de propagação de atualização que rege a consistência. Existem duas técnicas para o desenvolvimento e aplicação destes esquemas de replicação: replicação estática e replicação dinâmica [Doherty and Hurley, 2007].

Replicação estática é o termo usado para descrever a replicação em sistemas onde os esquemas de replicação são desenvolvidos e aplicados em tempo de projeto e permanece praticamente inalterado até que um administrador realize uma intervenção manual. Em um sistema onde os atributos de tráfego e da rede são conhecidos e imutáveis, esquemas estáticos são adequados. Para sistemas em nuvem, que são altamente dinâmicos, uma administração manual da replicação é complexa, pois esta envolve uma grande quantidade de recursos.

A replicação dinâmica utiliza informações do tráfego do usuário para adaptar o esquema de replicação baseada no estado atual da rede, comportamento do usuário e métricas relacionadas ao desempenho. Um sistema que emprega a replicação dinâmica monitora seu ambiente para determinar como e quando alterar esquemas de replicação, e como essas alterações afetam o sistema. Desta forma, os esquemas de replicação são desenvolvidos, ajustados e aplicados de modo a maximizar algum objetivo, por exemplo, o desempenho. A replicação dinâmica permite modificar os esquemas de replicação para lidar com a carga de trabalho e cada réplica que compõe o sistema pode receber parte desta carga.

Além do esquema de replicação dinâmica, utiliza-se uma política, que é definida como um conjunto de regras de alto nível para administrar, gerenciar e controlar a replicação dinâmica e o acesso aos recursos. As políticas são definidas por um administrador e fornecem informações para cada nó do sistema em termos de recursos de processamento e armazenamento e parâmetros de desempenho, por exemplo, capacidades máximas de CPU, memória, armazenamento, limites superiores e inferiores de tempo de reposta e protocolos de replicação. Na nuvem, os administradores podem utilizar SLAs para especificar estas regras.

Em relação ao processo de replicação em nuvem, 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. Dentre os protocolos utilizados na replicação de dados em nuvem pode-se destacar o de cópia primária, réplica ativa, quórum baseado em 2PC, *Paxos* [Chang et al., 2006] e o *Gossip* [DeCandia et al., 2007].

Por outro lado, como o uso de máquinas virtuais auxilia a provisionar recursos [Rogers et al., 2010], pode-se 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].

2.3 CONCLUSÃO

Este capítulo apresentou uma introdução à computação em nuvem e as principais características relacionadas ao gerenciamento de dados em nuvem. Também foram apresentados os requisitos da gestão de dados em nuvem. A compreensão destas características é extremamente importante e possibilita fazer escolhas adequadas no desenvolvimento de soluções para a computação em nuvem. Além disso, acredita-se que a combinação entre características de diferentes abordagens pode conduzir ao surgimento de soluções mais eficazes no gerenciamento de dados. O próximo capítulo apresentado os trabalhos relacionados a esta tese.

CAPÍTULO 3

TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos que tratam da replicação de banco de dados relacional em nuvem. Inicialmente, são apresentados os requisitos para a replicação de banco de dados no contexto de computação em nuvem. Em seguida, descreve-se cada um desses trabalhos considerando os requisitos apresentados. Por fim, um estudo comparativo é apresentado, destacando as limitações destes trabalhos.

3.1 INTRODUÇÃO

Para compreender melhor as características necessárias para uma solução de replicação de banco de dados em nuvem foram identificados os requisitos referentes às aplicações e aos usuários, conforme mostra a Tabela 3.1. Estes requisitos consideram as características da computação em nuvem e do gerenciamento de dados para apoiar diferentes aplicações compartilhando os mesmos recursos.

Elasticidade	Adicionar e remover réplicas
Qualidade do Serviço	Atender o SLA do usuário
Multi-inquilino	Compartilhar recursos
Consistência	Garantir consistência forte

Tabela 3.1 Requisitos para a replicação de banco de dados em nuvem

O provedor deve fornecer elasticidade por meio da adição e remoção de réplicas de acordo com a carga de trabalho. O provedor também deve garantir a qualidade de serviço, definida por meio de um SLA com o usuário, mesmo com alterações na carga de trabalho. Para tanto, deve-se fazer uso do gerenciamento automático, já que os sistemas em nuvem utilizam e compartilham uma grande quantidade de recursos. Neste caso, os provedores de IaaS utilizam técnicas de virtualização para auxiliar o gerenciamento dos recursos e melhorar a flexibilidade do sistema.

Para melhorar a utilização dos recursos, deve-se implementar o conceito de multi-inquilino em algum nível de compartilhamento, tais como SGBD, banco de dados ou tabela. Assim, o provedor reduz os custos, pois utiliza os recursos de forma eficiente. Por fim, as aplicações possuem requisitos de consistência forte, pois muitas destas aplicações estão sendo migradas para a nuvem e foram desenvolvidas considerando este tipo de consistência.

3.2 TRABALHOS RELACIONADOS

Existem muitas soluções para a replicação de banco de dados em nuvem. A seguir, são apresentadas as soluções que estão diretamente relacionadas a esta tese, ou seja, gerenciamento de dados para apoiar diferentes aplicações, cada uma com pequenas quantidades de dados e que utilizam o modelo de dados relacional.

SQL Azure [Mukerjee et al., 2011]

O Microsoft SQL Azure é composto por um conjunto de serviços para o armazenamento e processamento de dados em nuvem [Mukerjee et al., 2011] [Bernstein et al., 2011] [Azure, 2012]. O *SQL Azure Database* (SAD) é o principal componente do SQL Azure e foi construído com base na tecnologia do SGBD relacional SQL Server [Bernstein et al., 2011]. Para permitir uma integração transparente de aplicações com o SQL Azure, o SAD suporta os principais comandos da linguagem *Transact-SQL* (T-SQL). Esta linguagem possui diversas características, tais como manipulação de tabelas, índices, funções, procedimentos e gatilhos.

O SQL Azure implementa alta disponibilidade, tolerância a falhas e o conceito de multi-inquilino no nível de máquina virtual. Cada banco de dados é implementado como uma partição de dados replicados em múltiplas máquinas em um *SQL Azure Datacenter*. Com esse tipo de abordagem, SQL Azure fornece um gerenciamento automático de falhas e balanceamento de carga de trabalho. A Figura 3.1 mostra a arquitetura do SQL Azure. As máquinas são organizadas em um anel lógico, de modo que cada máquina tem dois vizinhos e, assim, é possível detectar falhas de hardware. Um componente para detecção de falhas verifica quando a réplica primária ou secundária torna-se indisponível. O agente de reconfiguração gerencia o restabelecimento de réplicas após uma falha de um nó.

A estratégia de replicação utiliza cópias dos itens de dados para fornecer a dis-

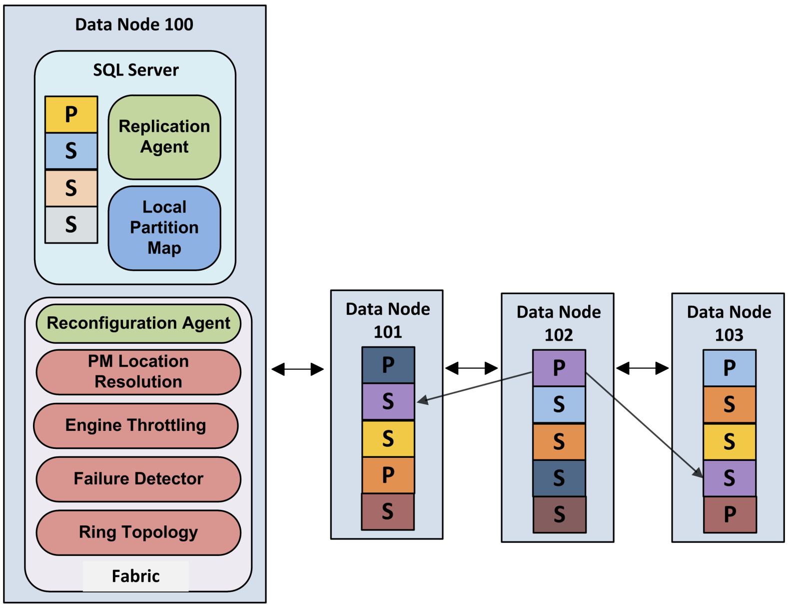

Figura 3.1 Arquitetura do sistema SQL Azure [Azure, 2012]

ponibilidade e implementa consistência forte. No *SQL Azure Database*, um banco de dados individual possui um tamanho limitado de 50 GB. Para criar soluções que armazenem banco de dados maiores do que este tamanho deve-se particionar os dados entre múltiplos bancos de dados e utilizar consultas em paralelo para acessá-los. Entretanto, o particionamento dos dados é realizado de forma manual.

O SQL Azure utiliza o protocolo de cópia primária [Kossmann et al., 2010]. Cada banco de dados possui três réplicas: uma primária e duas secundárias. As operações de escritas são enviadas para a réplica primária e as alterações são propagadas para as réplicas secundárias de forma assíncrona. O SQL Azure utiliza uma estratégia de quórum, onde a réplica primária e pelo menos uma das réplicas secundárias devem confirmar a escrita no *log* antes que uma transação seja considerada consolidada.

Plataforma Proposta por [Yang et al., 2009]

Em [Yang et al., 2009] é apresentada uma plataforma para o gerenciamento de dados que implementa qualidade de serviço para diferentes aplicações. Esta plataforma propõe uma solução completa de gerenciamento com o objetivo de melhorar a vazão e garantir alta disponibilidade. Para tanto, esta plataforma utiliza diversas técnicas, tais como replicação, migração e consistência. A Figura 3.2 mostra a arquitetura da plataforma.

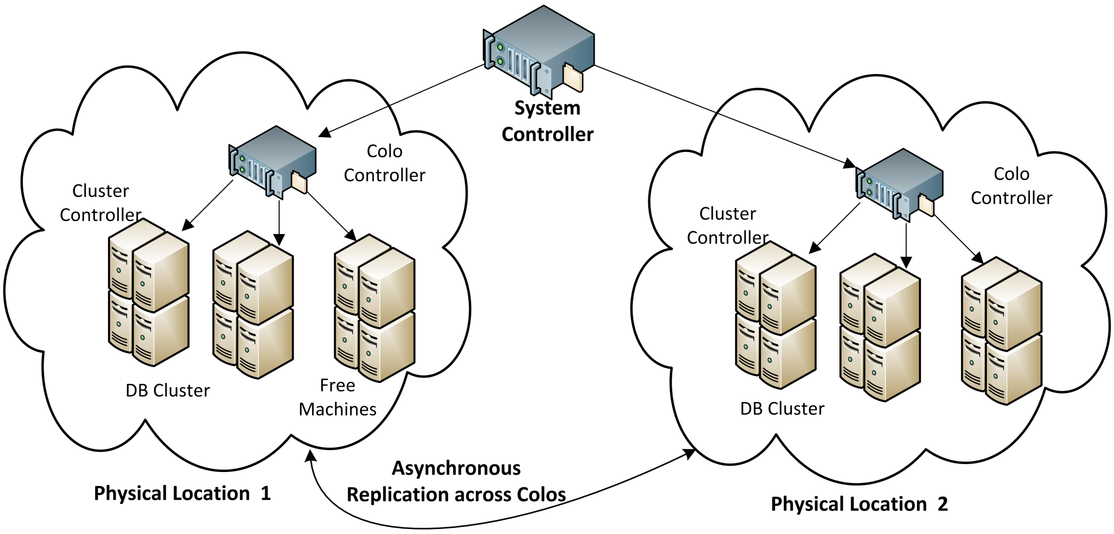

Figura 3.2 Arquitetura da plataforma proposta por [Yang et al., 2009]

O sistema consiste de máquinas em múltiplos colos (similar às zonas de disponibilidade utilizadas em computação em nuvem) geograficamente distribuídos. O banco de dados é replicado em vários colos para prover recuperação de desastres. Os colos são coordenados por um controlador do sistema tolerante à falhas, que encaminha as requisições dos clientes para o colo apropriado, baseado em diferentes aspectos, tais como a configuração da replicação do banco de dados, a carga de trabalho, o estado do colo e a proximidade geográfica entre o cliente e o colo. O controlador de colo gerencia um *poll* de máquinas físicas disponíveis e adiciona estas máquinas para o aglomerado quando a qualidade definida não é atendida.

Cada colo contém um ou mais aglomerados de máquinas. Cada aglomerado contém uma pequena quantidade de máquinas, tipicamente em um mesmo *rack*. Cada máquina executa uma instância de um SGBD e cada banco de dados é mapeado em duas ou mais máquinas no aglomerado. As máquinas do aglomerado são coordenadas por um controlador de aglomerados tolerante à falhas, que executa três tarefas: (a) gerenciar as conexões e garantir que as múltiplas réplicas de cada banco de dados no aglomerado estejam sempre sincronizadas, (b) gerenciar falhas e (c) garantir que o SLA para cada banco de dados individual seja satisfeito.

Neste trabalho também são propostas definições de SLA para banco de dados considerando vazão e disponibilidade. Estas definições tem o objetivo de alocar bancos de dados com o número mínimo de máquinas de tal forma que os SLAs sejam satisfeitos. Em relação à replicação de dados, esta plataforma utiliza protocolos tradicionais, tais como o protocolo baseado em duas fases (2PC), o que pode comprometer o desempenho no ambiente de nuvem.

Re: FRESHiT [Voicu et al., 2010]

Em [Voicu et al., 2010] é apresentado o Re: FRESHiT, um protocolo para a replicação de dados em nuvem. O FRESHiT é baseado no Re:GRIDiT [Voicu and Schuldt, 2009], um protocolo para o gerenciamento de dados em grades computacionais com acesso simultâneo aos dados replicados em diferentes locais, sem um componente global e com controle dinâmico de réplicas.

O Re:FRESHiT permite o acesso a dados atualizados e também com diferentes níveis de consistência. Dependendo do tipo de acesso, os sítios são divididos em sítios de atualização e leitura. As atualizações são propagadas dos sítios de atualização para os de leitura. Os sítios são organizados em estruturas de árvores virtuais de acordo com o nível de consistência. Essas árvores são automaticamente reorganizadas para melhorar o desempenho. Um mecanismo de roteamento é utilizado para garantir o acesso baseado no nível de consistência definido pelo usuário e na carga de trabalho dos sítios. A Figura 3.3 mostra a arquitetura do Re: FRESHiT.

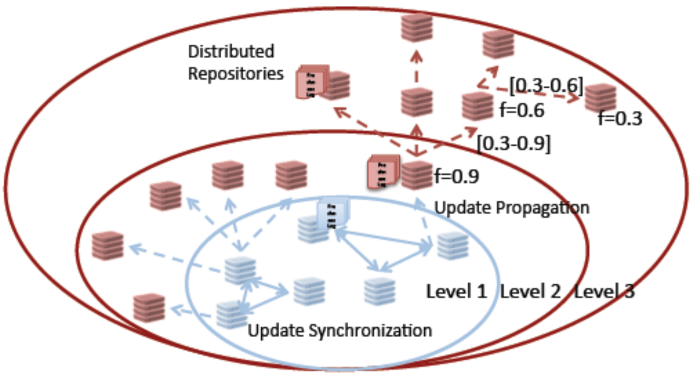

Figura 3.3 Arquitetura do sistema Re: FRESHiT [Voicu et al., 2010]

O primeiro nível contém os sítios de atualização e utiliza o protocolo de cópia primária com propagação síncrona. Este nível possui um relógio de sincronização, necessário para calcular o nível de consistência. No segundo nível, os sítios de leitura mantém os dados atualizados na medida do possível. Existe um relacionamento 1-para-n entre o primeiro e o segundo nível, assim como entre o segundo e o terceiro nível.

No terceiro nível, os sítios de leitura não são atualizados com frequência. Este último nível é opcional, mas pode auxiliar na distribuição da carga de trabalho, principalmente se a quantidade de sítios de leitura no sistema é relativamente maior que a quantidade de sítios de atualização. O terceiro nível pode ser organizado em uma estrutura hierárquica, formando uma estrutura de árvore em profundidade.

Para calcular o nível de consistência é utilizada uma função, onde este nível é decrescente em relação à árvore, da raiz para as folhas. Os sítios de leitura são capazes de determinar o nível de consistência com base no conhecimento local, ou seja, se a versão corrente dos dados pode ser usada para atender uma requisição do usuário ou se esta requisição precisa ser roteada para um sítio predecessor na árvore.

Para tanto, o Re:FRESHiT possui quatro estruturas: (a) um catálogo de réplicas é usado para determinar as réplicas disponíveis na rede (b) um repositório de réplicas é usando para coletar o nível de consistência dos dados de forma periódica ou sob demanda (c) um repositório de carga de trabalho é usado para determinar informações sobre a carga aproximada e (d) filas de propagação são utilizadas para enfileirar as alterações a serem aplicadas nos sítios inferiores das árvores.

SmartSLA [Xiong et al., 2011]

Em [Xiong et al., 2011] é apresentado o SmartSLA, um sistema inteligente para o gerenciamento de recursos, que considera a carga de trabalho e o custo da infraestrutura. Este trabalho aborda o problema do gerenciamento de recursos virtuais em ambientes de computação em nuvem e utiliza técnicas de aprendizagem de máquina. Estas técnicas são utilizadas para descrever um modelo de desempenho do sistema por meio de uma abordagem orientada a dados. Um modelo preditivo é utilizado para a alocação de recursos de hardware, tais como CPU e memória, assim como para definir o número de réplicas do sistema.

O SmartSLA assume que existem dois tipos de clientes, por exemplo, um ouro e um prata, que compartilham recursos de hardware. Como a demanda de carga de trabalho destes clientes mudam, SmartSLA adiciona ou remove réplicas do banco de dados de forma a atender os requisitos dos clientes. O sistema é monitorado continuamente e os recursos alocados podem mudar periodicamente em cada intervalo de tempo. A Figura 3.4 mostra a arquitetura do SmartSLA.

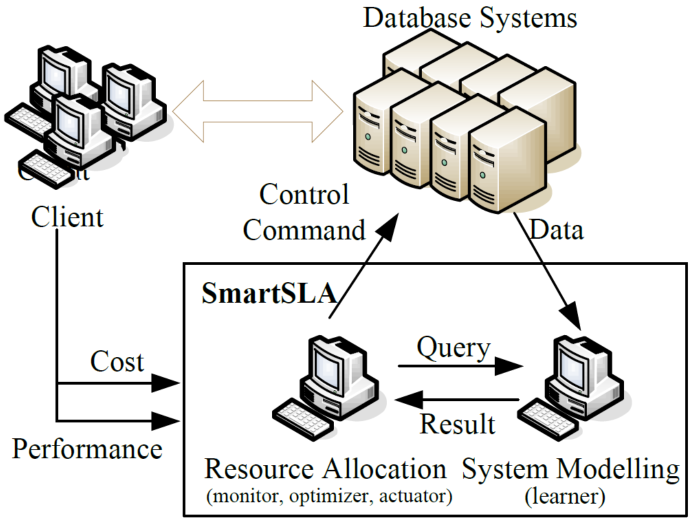

Figura 3.4 Arquitetura do sistema SmartSLA [Xiong et al., 2011]

O SmartSLA consiste de dois componentes principais: módulo de modelagem do sistema e módulo de decisão para alocação de recursos. Os clientes fornecem informações sobre o custo e o desempenho para o SmartSLA. O módulo de modelagem coleta dados sobre o estado dos bancos de dados e o módulo de decisão emite comandos para controlar os recursos. O módulo de modelagem utiliza técnicas de aprendizagem de máquina para construir um modelo que descreva o potencial de margem de lucro para cada cliente com diferentes alocações de recursos.

Este módulo utiliza um modelo baseado no relacionamento entre os recursos alocados e o custo esperado para cada cliente. O módulo de decisão de alocação ajusta dinamicamente os recursos para maximizar os lucros. Em relação à replicação, esta é implementada com o protocolo de cópia primária de forma assíncrona e o SmartSLA trata apenas da questão do número de réplicas necessárias para garantir o SLA.

Amazon Relational Database Service (RDS) [Amazon, 2012]

O Amazon Relational Database Service (RDS) [Amazon, 2012] é um sistema para a criação e acesso a um SGBD relacional em nuvem. Com isso, os usuários não precisam se preocupar em gerenciar a implantação, *patches*, atualizações de software e *backups*. O Amazon RDS possui diferentes configurações, de acordo com o tamanho de instância definida. A Figura 3.5 mostra a arquitetura do Amazon RDS.

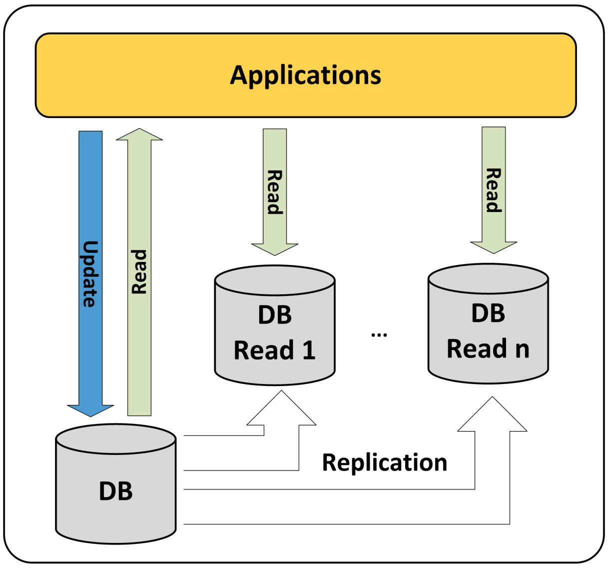

Figura 3.5 Arquitetura do sistema Amazon RDS [Amazon, 2012]

O Amazon RDS utiliza o protocolo de réplica primária, onde uma réplica recebe todas as atualizações e as propaga para as réplicas de leitura. Com isso, o Amazon RDS é capaz de tratar o aumento de atualizações apenas pelo incremento da capacidade da réplica primária. A replicação também é utilizada entre os centros de dados da Amazon para aumentar a disponibilidade.

Relational Cloud [Curino et al., 2011a]

O Relational Cloud é um SGBD como um serviço desenvolvido com o objetivo de consolidar as funcionalidades de gerenciamento de dados em nuvem [Curino et al., 2011a]. O Relational Cloud foca nos conceitos de multi-inquilino, escalabilidade e privacidade dos dados. Para tanto, utiliza uma abordagem orientada a carga de trabalho para tratar a alocação de inquilinos, algoritmos de particionamento dos dados baseados em grafos e um esquema de segurança ajustável para permitir que consultas SQL sejam executadas sobre dados criptografados. O Relational Cloud fornece disponibilidade por meio de replicação transparente, particionamento de dados automático e migração de dados em tempo de execução, além de oferecer transações distribuídas serializáveis.

A Figura 3.6 mostra a arquitetura do Relational Cloud. A comunicação entre as aplicações e o Relational Cloud é realizada por meio de *interfaces* padrão ou protocolos conhecidos, tais como a *interface* JDBC [Curino et al., 2010a]. 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.

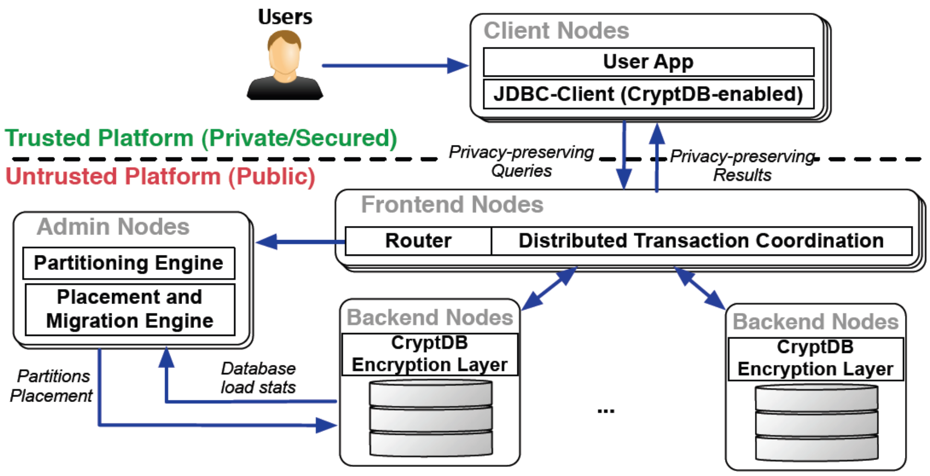

Figura 3.6 Arquitetura do sistema Relational Cloud [Curino et al., 2011a]

Por meio do monitoramento do desempenho e da carga de trabalho, o Relational Cloud ajusta o particionamento dos dados e as opções de localização em tempo de execução [Curino et al., 2011b]. Falhas no sistema e alterações de carga de trabalho exigem a evolução do esquema de particionamento e alocação em tempo de execução. Para tanto, utiliza-se a migração dos dados baseada nas instâncias do SGBD, o que melhora o desempenho do sistema.

No Relational Cloud, cada máquina executa várias instâncias de um SGBD. Cada banco de dados é dividido em partições lógicas por meio de técnicas de particionamento. Estas 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. Em relação ao particionamento, este sistema propõe um novo algoritmo com o objetivo de minimizar a probabilidade de uma determinada operação ter que acessar múltiplos sítios para responder uma requisição. O Relational Cloud não detalha a estratégia de replicação adotada, apenas destaca sua utilização em conjunto com o particionamento de dados [Curino et al., 2010b].

Trabalho Proposto por [Savinov and Daudjee, 2010]

Em [Savinov and Daudjee, 2010] é apresentado um estudo sobre a replicação de dados em ambientes virtualizados com foco no provisionamento quando a cópia primária está sobrecarregada ou falha. Este trabalho utiliza um balanceador de cargas, uma cópia primária e várias cópias secundárias. A Figura 3.7 mostra a arquitetura do trabalho, destacando a estrutura e o relacionamento entre os componentes. Linhas sólidas mostram a transferência de consultas e seus respectivos resultados, enquanto linhas tracejadas representam o caminho para as atualizações.

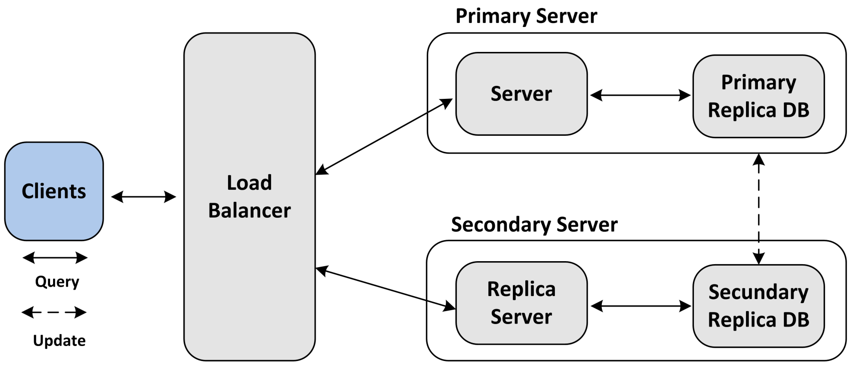

Figura 3.7 Arquitetura da abordagem proposta por [Savinov and Daudjee, 2010]

Transações submetidas pelos clientes para o sistema são recebidas pelo balanceador de cargas. Se a transação é de atualização, esta é encaminhada para a cópia primária, que processa todas as atualizações do sistema. Se a transação é de leitura, esta é encaminhada para as cópias secundárias. Um processo de propagação na cópia primária coleta um lote de atualizações periodicamente dos arquivos de *log* e envia este lote para as cópias secundárias. Transações de atualizações são mantidas em um arquivo de *log* com identificador único e incrementado monotonicaamente, representando a ordem na qual as atualizações foram aplicadas na cópia primária. O balanceador de cargas mantem uma cópia das últimas transações submetidas ao sistema para tratar os casos de falhas da cópia primária.

As réplicas são adicionadas no sistema a qualquer momento. Um *dump* é usado para recuperar as informações da cópia primária e atualizar as cópias secundárias. Para distribuir a carga de trabalho, o balanceador considera o tempo de resposta das requisições. No caso de falha na cópia primária, o balanceador de cargas seleciona a réplica mais atualizada e aplica todas as transações contidas na fila, mantidas pelo balanceador, enquanto a cópia primária não está operacional.

CloudDB AutoAdmin [Sakr et al., 2011]

CloudDB AutoAdmin é um *framework* para o gerenciamento automático da camada de banco de dados [Sakr et al., 2011] [Sakr and Liu, 2012]. Este *framework* permite ajustar e adaptar dinamicamente os recursos para garantir a qualidade do serviço. A Figura 3.8 mostra a arquitetura do CloudDB AutoAdmin. O módulo do controle recebe as configurações de métricas de SLA definidas para a aplicação. Estas configurações são especificadas na linguagem XML e permitem definir as condições para satisfazer o SLA, tais como tempo de resposta e vazão. Este módulo verifica continuamente o SLA de acordo com as informações dos bancos de dados. O módulo de monitoramento coleta informações sobre a execução dos bancos de dados e envia informações para o módulo de controle. O módulo de ação executa as ações para garantir as regras definidas no SLA. Estas ações incluem o balanceamento da carga de trabalho e a adição/remoção de réplicas.

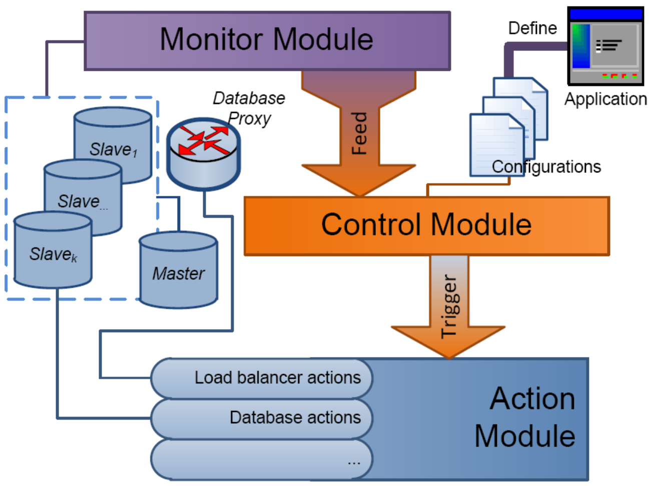

Figura 3.8 Arquitetura do sistema CloudDB AutoAdmin [Sakr et al., 2011]

Em relação à replicação, o CloudDB AutoAdmin utiliza a estratégia de cópia primária com a replicação total do banco de dados [Zhao et al., 2012]. Apesar de permitir a elasticidade pela adição e remoção de réplicas, os experimentos realizados consideram apenas cenários com a adição de réplicas [Liang Zhao and Liu, 2012]. Por fim, o CloudDB AutoAdmin não utiliza nenhum modelo multi-inquilino, fundamental para compartilhar recursos e reduzir os custos.

Dolly [Cecchet et al., 2011]

Em [Cecchet et al., 2011] é apresentado Dolly, um sistema para provisionamento dinâmico de réplicas de banco de dados em nuvem. Este sistema foca na virtualização e tem como base utilizar clonagem e *snapshots* de máquinas virtuais de forma inteligente. Neste sistema, cada réplica do banco de dados é executada em uma máquina virtual isolada. Em vez de utilizar mecanismos tradicionais de banco de dados para criar novas réplicas, é clonada toda a máquina virtual de uma réplica existente, incluindo o ambiente operacional, o SGBD com todas as suas configurações, ajustes e o próprio banco de dados. A máquina virtual é iniciada em um novo servidor, resultando em uma nova réplica, que sincroniza o estado com outras réplicas antes de processar solicitações do usuário.

Este trabalho se concentra na camada de banco de dados e aborda o problema do provisionamento como duas tarefas: (i) quando alterar a capacidade baseada na carga de trabalho corrente e tendências futuras e (ii) como iniciar e remover as réplicas. Para tratar estas tarefas, Dolly estima o tempo de latência para iniciar uma nova réplica de banco de dados e usa este tempo para realizar o provisionamento de forma antecipada. É utilizado um algoritmo inteligente que considera as funções de custo especificadas pelo usuário para otimizar a sobrecarga de adicionar ou remover réplicas.

A Figura 3.9 mostra a arquitetura do sistema Dolly. Sistemas de predição observam o comportamento do sistema e auxiliam na demanda de capacidade futura. Dolly possui quatro componentes principais: provisionamento de capacidade, escalonador de *snapshot*, escalonador geral e desalocador de recursos. Para calcular a capacidade por um determinado prazo, é necessário executar ações de provisionamento que considerem o tempo necessário para replicar o estado do banco de dados. Como as réplicas têm de ser geradas a partir de um *snapshot* do banco, o escalonador decide quando novos *snapshots* do banco devem ser obtidos. Alguns recursos, tais como *backups* podem se tornar obsoletos com o tempo e são desalocados. O escalonador orquestra e executa as requisições dos outros componentes.

O processo de execução do sistema consiste em um conjunto de etapas. Primeiro, um comando para adicionar uma nova réplica é emitido a partir do console de gerenciamento para a camada de replicação. Um ponto de controle é criado no *log* transacional

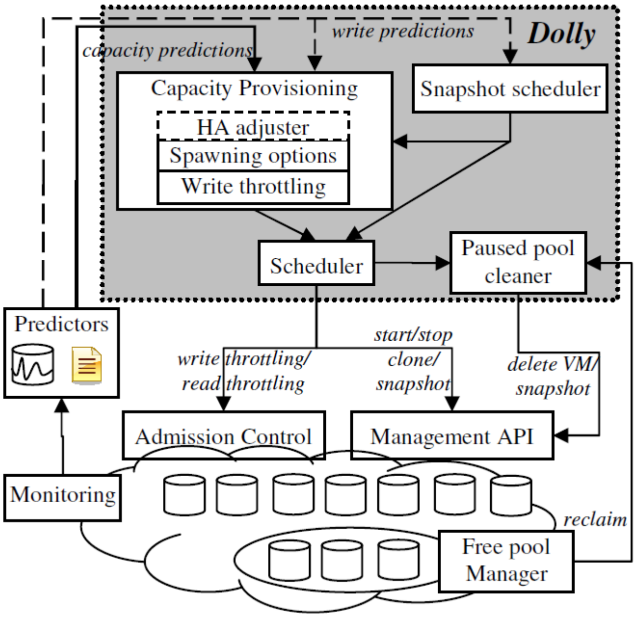

Figura 3.9 Arquitetura do sistema Dolly [Cecchet et al., 2011]

e uma réplica é removida temporariamente do aglomerado de banco de dados para realizar um *snapshot* do banco de dados. Assim que o *snapshot* é realizado, esta réplica é sincronizada novamente, reexecutando as operações de escrita no *log* transacional desde o ponto de controle e adicionada ao aglomerado de banco de dados.

Uma nova réplica é iniciada em uma máquina separada e um *snapshot* é aplicado a esta nova réplica usando uma operação de restauração. Por fim, as atualizações que ocorreram desde que o *snapshot* foi realizado são reexecutados a partir do *log* transacional para sincronizar a nova réplica e torná-la atualizada em relação as demais réplicas do sistema. Dolly utiliza a estratégia de cópia primária e trata a elasticidade, mas não aborda a questão multi-inquilino.

FlurryDB [Mior and de Lara, 2011]

FlurryDB é uma abordagem para replicação de banco de dados baseada em técnicas de clonagem de máquinas virtuais [Mior and de Lara, 2011]. Esta abordagem cria uma cópia da máquina virtual e utiliza um roteador para replicar as operações do banco de dados executadas durante a clonagem da máquina com o objetivo de manter a consistência do banco de dados. Os clientes se conectam a um roteador que realiza o balanceamento da carga de trabalho e o gerenciamento das operações do banco de dados. Estas operações são envidas para um *proxy* presente em cada máquina. Para replicar uma máquina, o FlurryDB cria um clone da máquina primária e aplica as operações realizadas durante a clonagem. A Figura 3.10 mostra a arquitetura do FlurryDB.

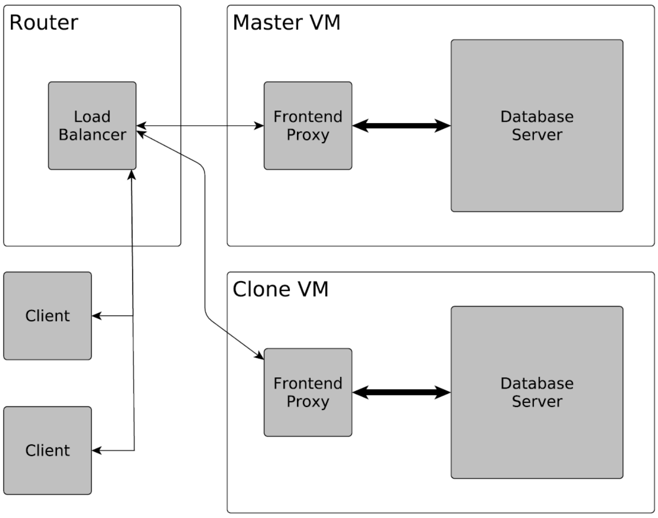

Figura 3.10 Arquitetura do sistema FlurryDB [Mior and de Lara, 2011]

FlurryDB utiliza a estratégia de cópia primária para replicar a máquina. Apesar de replicar o banco de dados rapidamente, esta abordagem requer um *hypervisor* ou monitor de máquina virtual específico para tratar do processo de clonagem, o que dificulta sua utilização. Além disso, o FlurryDB não trata questões de elasticidade e multi-inquilino.

RemusDB [Minhas et al., 2011]

RemusDB é um sistema para fornecer alta disponibilidade para SGBDs em ambientes virtualizados [Minhas et al., 2011]. RemusDB utiliza pontos de verificação (*checkpoint*) para enviar as alterações do servidor primário para o servidor de *backup*. A Figura 3.11 mostra a arquitetura do RemusDB. O servidor ativo trata todas as requisições dos clientes durante a operação normal. Pontos de verificação da máquina, incluindo memória, disco e conexões de rede ativas são continuamente salvos no servidor em espera. As requisições são executadas e consolidadas no servidor ativo antes do envio para o servidor em espera. Para concluir o processo de replicação, estes pontos são executados no servidor em espera.

Em caso de falha do servidor ativo, o servidor em espera recebe as requisições.

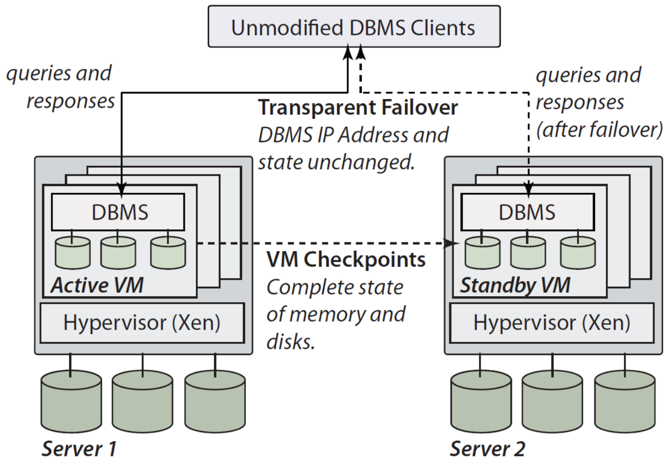

Figura 3.11 Arquitetura do sistema RemusDB [Minhas et al., 2011]

Este processo utiliza técnicas de *failover* para garantir as mesmas configurações de rede e é transparente para o usuário. Além disso, o RemusDB garante a consistência, pois transfere o estado completo da máquina, incluindo o estado do banco de dados e o estado interno do SGBD, que contém informações de *buffer* e bloqueios. Esta estratégia garante que o servidor em espera possui exatamente o mesmo estado do servidor ativo, não afetando as conexões dos clientes.

RemusDB utiliza a estratégia de cópia primária para a replicação e fornece alta disponibilidade para SGBDs, o que melhora a qualidade do serviço. Contudo, esta abordagem requer modificação no *hypervisor* da máquina virtual para suportar a replicação do SGBD. Em relação à elasticidade e técnicas multi-inquilino, o RemusDB não aborda estes aspectos.

3.3 ANÁLISE COMPARATIVA ENTRE OS TRABALHOS RELACIONADOS

Esta análise comparativa considerou os requisitos definidos para a replicação de banco de dados em nuvem. Os sistemas SQL Azure, Dolly e Amazon RDS implementam as características de elasticidade completamente, pois adicionam e removem réplicas de acordo com a carga de trabalho. Já os sistemas Relational Cloud, Flurry e CloudDB AutoAdmin tratam apenas da adição de novas réplicas.

A plataforma proposta por [Yang et al., 2009] compartilha recursos, mas não especifica o modelo multi-inquilino utilizado. O Relational Cloud implementa diferentes instâncias de SGBD, mas as técnicas desenvolvidas por este sistema consideram apenas máquinas físicas, o que inviabiliza sua utilização em ambientes virtualizados. Já o SQL Azure compartilha recursos apenas por meio de partição de dados. O smartSLA trabalha com um nível de multi-inquilino de máquina virtual, o que não contribui para uma utilização eficiente de recursos, aumentando os custos. Nenhum dos trabalhos considera a replicação utilizando o modelo multi-inquilino no nível de banco de dados.

Alguns trabalhos, tais como Dolly, smartSLA e CloudDB AutoAdmin implementam qualidade de serviço considerando características de desempenho, tais como tempo de resposta e vazão. O SQL Azure e Amazon RDS abordam parcialmente a qualidade de serviço, pois contemplam somente a disponibilidade. Os sistemas Dolly, RemusDB e FlurryDB apresentam estratégias eficientes para replicar o banco de dados, pois utilizam técnicas para clonar as VMs juntamente com o SGBD. Contudo, estes sistemas não podem ser usados em provedores públicos, como o da Amazon, pois necessitam de alterações no ambiente de virtualização.

Estes trabalhos implementam consistência forte, mas utilizam protocolos de replicação tradicionais, como o de réplica primária e o 2PC, o que pode comprometer a disponibilidade e o desempenho. Nenhum dos trabalhos relacionados estudados atende totalmente os requisitos definidos neste trabalho no que se refere à replicação de dados, requisitos estes contemplados pelo RepliC. A Tabela 3.2 apresenta um resumo deste comparativo.

Requisitos / Trabalhos	Elasticidade	Multi-inquilino	Qualidade de Serviço	Consistência
[Yang et al., 2009]		VM	sim	sim
Re: FRESHiT [Voicu et al., 2010]				sim
[Savinov and Daudjee, 2010]				sim
SQL Azure [Mukerjee et al., 2011]	sim	VM	apenas disp.	sim
Dolly [Cecchet et al., 2011]	sim		sim	sim
smartSLA [Xiong et al., 2011]	apenas adição	VM	sim	sim
Rel. Cloud [Curino et al., 2011a]	apenas adição	sim		sim
Flurrydb [Mior and de Lara, 2011]	apenas adição			sim
CloudDB [Sakr et al., 2011]	apenas adição		sim	sim
RemusDB [Minhas et al., 2011]				sim
AmazonRDS [Amazon, 2012]	sim		apenas disp.	sim

Tabela 3.2 Análise comparativa entre os trabalhos relacionados

3.4 CONCLUSÃO

Neste capítulo foram discutidas as principais abordagens encontradas na literatura para a replicação de banco de dados relacional em nuvem. Foram destacados os requisitos para a replicação neste ambiente e uma análise comparativa detalhada destas abordagens também foi apresentada. Apesar da grande quantidade de trabalhos relacionados, nenhum destes contempla os diversos aspectos da replicação de dados em nuvem, pontos estes previsto neste trabalho. O próximo capítulo apresenta a abordagem RepliC e são explanadas suas características, sua especificação e os algoritmos desenvolvidos.

CAPÍTULO 4

REPLICAÇÃO ELÁSTICA PARA BANCO DE DADOS MULTI-INQUILINO COM QUALIDADE DO SERVIÇO

Este capítulo descreve RepliC, uma abordagem para a replicação de banco de dados multi-inquilino em nuvem, cujo propósito é garantir a qualidade de serviço com a utilização eficiente de recursos por meio da adição e remoção de réplicas. Também são apresentados o modelo multi-inquilino utilizado neste trabalho, um modelo para qualidade de serviço de banco de dados e os algoritmos para replicação que tratam da elasticidade.

4.1 INTRODUÇÃO

RepliC é uma abordagem para a replicação de dados em nuvem com foco na garantia da qualidade de serviço, elasticidade e na utilização eficiente dos recursos sob uma carga de trabalho variável. Baseado em técnicas de monitoramento, RepliC modifica o estado do sistema e são realizadas modificações na estratégia de replicação quando o desempenho do sistema não está de acordo com o SLA definido.

A elasticidade permite ajustar dinamicamente a capacidade do sistema pela adição e remoção de réplicas de acordo com a carga de trabalho. Para melhorar a utilização dos recursos, RepliC utiliza o modelo multi-inquilino de instância. O gerenciamento da infra-estrutura é automático, auxiliando na gestão da estratégia de replicação. Esta abordagem implementa consistência forte, pois muitas aplicações trabalham com esse tipo de consistência. O modelo de dados relacional é utilizado pelo RepliC, visto que este modelo é amplamente utilizado em diferentes aplicações em nuvem [Kossmann et al., 2010].

4.2 MODELO DE BANCO DE DADOS MULTI-INQUILINO

Existem diversos modelos de banco de dados multi-inquilino, cada um com vantagens e desvantagens, conforme apresentado no Capítulo 2. Nesta tese, optou-se por utilizar o modelo multi-inquilino de instância, pois é o mais utilizado e apresenta a melhor relação entre a utilização de recursos, desempenho e segurança [Barker et al., 2012]. De acordo com o modelo utilizado, um banco de dados corresponde a um inquilino no sistema, conforme mostra a Figura 4.1. Neste modelo, cada VM possui uma instância de um SGBD. Por sua vez, cada SGBD contém uma quantidade variável de inquilinos, de acordo com a capacidade dos recursos na VM e da carga de trabalho. Por exemplo, o SGBD da VM_4 possui os inquilinos T_1, T_4 e T_6. Neste trabalho, um inquilino é representado por uma réplica do banco de dados no sistema.

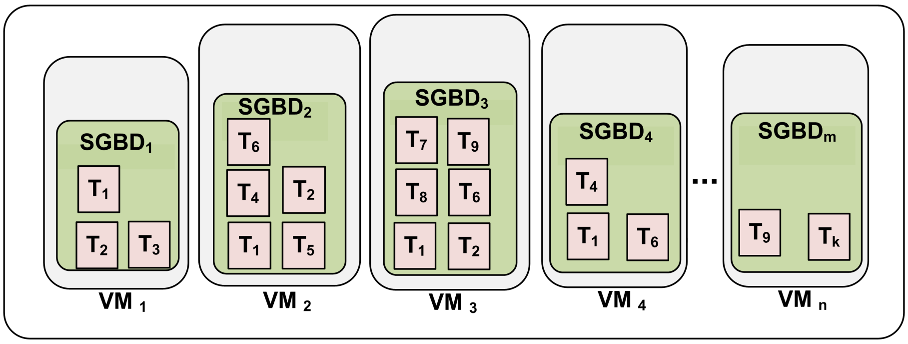

Figura 4.1 Modelo multi-inquilino utilizado pelo RepliC.

4.2.1 Interferência entre Inquilinos

O modelo multi-inquilino utilizado no RepliC pode apresentar interferência entre os inquilinos, conforme apresentado em nosso estudo [Moreira et al., 2012]. Uma interferência ocorre quando o aumento na carga de trabalho de um inquilino altera o desempenho de outros inquilinos que estão sendo executados no mesmo SGBD. Com isso, a escolha do SGBD ajuda a diminuir as interferências. Além disso, para diminuir tais interferências, é necessário identificar o nível de interação entre os inquilinos e, consequentemente, melhorar a alocação dos inquilinos de acordo com suas interferências. Para tanto, é necessária uma análise do perfil do inquilino. Também é importante isolar inquilinos suscetíveis a este problema.

Alguns trabalhos utilizam um modelo analítico ou executam experimentos reais para avaliar a alocação de recursos e diminuir as interferências [Ahmad and Bowman, 2011] [Lang et al., 2012]. Contudo, estas abordagens ocasionam uma sobrecarga significativa com a alteração da carga de trabalho. Nestas abordagens, faz-se necessário recalibrar e revalidar o modelo ou executar um novo conjunto de experimentos para selecionar uma nova alocação de recursos. Durante o período de adaptação, o sistema pode ser executado com uma configuração ineficiente e não garantir a qualidade [Vasić et al., 2012].

Além disso, estas abordagens consideram o conhecimento prévio de informações sobre a carga de trabalho, componentes de hardware, software e as interações entre todos estes componentes [Tsakalozos et al., 2011]. Contudo, o ambiente de computação em nuvem é dinâmico e estas informações são obtidas apenas em tempo de execução.

Para tratar este problema, RepliC utiliza uma estratégia baseada em DejaVu [Vasić et al., 2012], um *framework* para a aprendizagem e reutilização de alocação de recursos. Este *framework* coleta informações sobre o estado do sistema durante a execução e reutiliza estas informações para garantir uma alocação aproximada para futuras cargas de trabalho semelhantes às cargas anteriores. Esta estratégia é independente de componentes de hardware, software e pode ser utilizada por qualquer infraestrutura.

RepliC monitora os recursos e armazena informações sobre os inquilinos, suas cargas de trabalho e os limites dentro dos quais eles podem ser executados sem violação do SLA. A interferência entre inquilinos é ocasionada pela violação do SLA de qualquer um destes inquilinos. Essa interferência entre os inquilinos é modelada por um conjunto de regras. Uma regra de interferência entre dois inquilinos P e Q, denotada por $P \rightarrow Q$, define a carga de trabalho limite dentro da qual a violação do SLA não ocorre, como mostra a seguinte regra:

$$P \rightarrow Q \;\; if \; carga\_atual_{P,Q} > carga\_SLA_{P,Q} \tag{4.1}$$

Esta definição pode ser aplicada para k inquilinos, por exemplo, $P \rightarrow Q, R, k$. No caso de violação do SLA, uma nova regra com o valor anterior à violação é armazenada pelo RepliC. Quando ocorrem alterações na carga de trabalho, RepliC verifica estas regras e limita as requisições enviadas para os inquilinos que apresentaram interferência. Para garantir a qualidade de serviço, a carga de trabalho excedente é enviada para um inquilino do sistema com os recursos disponíveis. Caso contrário, uma nova réplica é inicializada para lidar com esse aumento da carga. Estas regras são reutilizadas com o objetivo de evitar violações do SLA similares às execuções anteriores.

Para ilustrar a interferência, considere o seguinte exemplo. Suponha que os inquilinos P e Q executem as cargas de trabalho atuais $carga_P = 800$, $carga_Q = 1000$ e que com estes valores o SLA para cada inquilino seja satisfeito. Agora considere que a $carga_P$ continua constante e a $carga_Q$ foi alterada para 1200, mas o SLA não foi violado. Estes valores $carga_P = 800$ e $carga_Q = 1200$ representam a nova $carga_{SLA}$ e são armazenados, pois indicam que estes inquilinos podem ser executados com estas cargas sem violar o SLA. Por fim, suponha que a carga do inquilino Q foi incrementada para $carga_Q = 1700$, aumentando o tempo de resposta dos dois inquilinos e ocasionando violação do SLA, ou seja, interferência entre os mesmos. RepliC armazena e reutiliza estes valores em futuras execuções para garantir a qualidade.

4.3 MODELO DE QUALIDADE DE SERVIÇO PARA BD EM NUVEM

Existem muitos modelos para SLA e qualidade de serviço em nuvem [Fito et al., 2010], [Malkowski et al., 2010] [Schnjakin et al., 2010]. Entretanto, estes modelos são muito gerais e não tratam características do gerenciamento de dados em nuvem. Modelos para SLAs e qualidade específicos para serviços de banco de dados são descritos em [Yang et al., 2009] [Xiong et al., 2011] [Chi et al., 2011] [LSCR, 2012]. Contudo, estes modelos não contemplam alguns aspectos do gerenciamento de dados em nuvem, tais como métricas específicas para serviços de banco de dados em nuvem e propõem apenas parte de uma solução para qualidade, por exemplo, a definição de um SLA ou abordagens para penalidades. Além disso, estes trabalhos não utilizam técnicas eficazes de monitoramento específicas para SGBDs, fundamentais para tratar a elasticidade do ambiente de computação em nuvem.

RepliC utiliza o *Service-level Agreement for Database* (SLADB), um modelo de qualidade proposto em nossos trabalhos anteriores [Sousa et al., 2011b] [Sousa et al., 2012]. Este modelo define uma solução para auxiliar a qualidade de serviço, pois aborda diferentes questões, tais como definição do SLA, técnicas de monitoramento e penalidades. Ele combina diferentes técnicas de monitoramento, permitindo a melhoria dos serviços de banco de dados em nuvem e contempla aspectos do gerenciamento de dados, tais como tempo de resposta, vazão, disponibilidade e consistência.

4.3.1 Especificação

Um SLA contém informações sobre o modelo de receita do provedor de serviço em nuvem, a determinação do valor recebido pelo provedor para cumprir o SLA, bem como penalidades ou multas em caso de falhas. A definição de um SLA é uma tarefa não trivial, pois estes acordos devem refletir o valor econômico, bem como as exigências de serviço ao usuário. Adicionalmente, SLAs têm que descrever as condições comuns de negócios, tais como parâmetros de desempenho e disponibilidade, métricas de avaliação, contabilidade e questões jurídicas, bem como prazos de contrato [Malkowski et al., 2010]. Esta tese aborda apenas aspectos técnicos relacionados aos parâmetros de desempenho e métricas de avaliação.

As propostas para SLAs apresentam muitas diferenças, mas é possível identificar uma estrutura geral comum: informações sobre as partes envolvidas, parâmetros do SLA, métricas utilizadas para calcular os parâmetros do SLA, algoritmo para calcular os parâmetros do SLA, objetivo de nível de serviço (SLO) e ações a serem realizadas em caso de violação do acordo [Schnjakin et al., 2010]. O modelo SLADB utiliza a seguinte definição:

Definição 1.1: *Um SLA para Serviço de Banco de Dados em Nuvem é composto por informações das partes envolvidas, métricas do SLA, SLOs, algoritmos para calcular as métricas do SLA e penalidades.*

Informações sobre as partes envolvidas referem-se ao contrato entre o provedor e o cliente. As métricas do SLA estão relacionadas aos itens a serem monitorados (ex. tempo de resposta e vazão) e o SLO contém os limites pré-definidos para o parâmetro (ex. tempo de reposta menor do que 5 ms). Para cada parâmetro é definido uma forma de calculá-lo (ex. tempo médio) e penalidades referentes às ações em caso de não conformidade dos SLOs (ex. multa).

De acordo com [Chi et al., 2011], as métricas de SLA para banco de dados em nuvem devem otimizar o sistema, tratar aspectos relevantes para o gerenciamento de dados e contemplar as características do modelo de computação em nuvem. O SLADB utiliza as métricas de tempo de resposta, vazão, disponibilidade e consistência[1] e as considera fundamentais para o gerenciamento de serviços de banco de dados em nuvem. Entretanto, é importante ressaltar que o provedor de serviço pode optar por fornecer apenas algumas destas métricas no seu SLA, visto que existem métricas relacionadas, por exemplo, tempo de resposta e vazão. Um SLO é associado para cada métrica, conforme destacado a seguir:

- *Tempo de reposta*: o tempo de resposta máximo, em segundos, para cada transação, durante um período de tempo t.

[1]Está métrica é importante para serviços de banco de dados em nuvem, mas não se aplica a todos os serviços. Assim, esta métrica é opcional.

- *Vazão*: o rendimento mínimo, em transações por segundo, durante um período de tempo t.
- *Disponibilidade*: a fração máxima de transações rejeitadas ao longo de um período de tempo t.
- *Consistência*: o acesso a dados atualizados de acordo com o tipo de consistência: forte ou fraca.

4.3.2 Monitoramento das métricas do SLA

Do ponto de vista do usuário, um serviço de banco de dados executa bem se os requisitos de desempenho e disponibilidade que o usuário contrata são cumpridos. Um primeiro ponto é traduzir os requisitos de desempenho definidos pelo usuário em um conjunto comum de métricas que podem ser obtidos por meio do monitoramento. Exemplos de tais métricas incluem o tempo de resposta e vazão. O tempo de resposta médio é uma das formas mais comuns para verificar a qualidade do serviço.

Em diferentes contextos, é importante estabelecer metas mais gerais para o QoS [Schroeder et al., 2006], tais como o percentil, onde $x\%$ dos tempos de resposta são inferiores a um valor y. O percentil é solicitado pelos usuários como componente de um SLA, por exemplo, para garantir que pelo menos 90% das transações dos clientes tenha um tempo de resposta abaixo de um limite especificado [Entrialgo et al., 2011]. Para cada métrica do SLA, pode-se utilizar um algoritmo para calcular as métricas do SLA. O SLADB utiliza a seguinte estratégia:

- *Tempo de reposta*: percentil $x\%$ dos tempos de resposta inferiores a um valor y durante um período de tempo t.
- *Vazão*: percentil $z\%$ de vazão maiores a um valor k durante um período de tempo t.
- *Disponibilidade*: função atendido/não-atendido
- *Consistência*: função atendido/não-atendido.

O SLADB utiliza o intervalo de tempo de uma hora para verificar as penalidades do SLA, visto que este valor é utilizado pela maioria dos provedores para a tarifação de recursos. Para definir os limites de monitoramento, adotou-se estados para o SLA, conforme mostra a Figura 4.2. Utilizou-se uma escala para o tempo de resposta/vazão e estados atendido e não atendido para a disponibilidade e consistência.

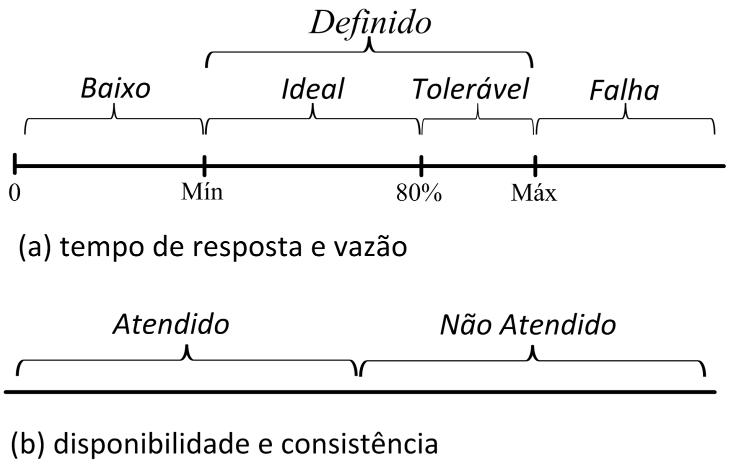

Figura 4.2 Estados do SLA.

- *Baixo*: O SLA é menor do que o definido para a aplicação. Neste estado, recursos podem ser removidos do sistema.
- *Definido*: O nível definido é dividido em ideal e tolerável. Na faixa ideal, o SLA é mantido dentro um intervalo aceitável (valor de 80%), mas pode-se configurar outros valores de acordo com os requisitos da aplicação. Na faixa tolerável, o SLADB intensifica o monitoramento do sistema de forma a preparar a adição de recursos.
- *Falha*: Neste nível, ocorreu uma falha em relação ao SLA. Neste caso, o provedor é penalizado de acordo com a quantidade de consultas no nível de falha e novos recursos devem ser adicionadas rapidamente para retornar ao nível definido.

A disponibilidade de um sistema pode ser obtida a partir de indicadores de falhas de seus componentes, tais como o *Mean Time Between Failures* (MTBF) e *Mean Time To Repair* (MTTR). Define-se MTBF como o tempo médio (normalmente medido em horas) entre duas falhas consecutivas de um componente ou serviço. O MTTR representa o tempo médio para se reparar uma falha identificada. Esta medida inclui o período de tempo para detectar o problema, diagnosticar e solucionar o mesmo. Assim, a disponibilidade de um sistema é dada pela fórmula MTBF/(MTBF + MTTR) [Tanenbaum and Steen, 2006].

A consistência da aplicação pode ser verificada por meio de técnicas propostas por [Zellag and Kemme, 2012], que permitem detectar anomalias de consistência para aplicações e não requerem nenhum conhecimento sobre a lógica das aplicações. A detecção de anomalias consiste na construção e verificação de gráfico de dependência durante a execução, permitindo identificar os itens de dados que estão inconsistentes.

Devido à sua representatividade, o tempo de resposta e a vazão são métricas de desempenho de alto nível que precisam ser obtidas e analisadas. Os valores dessas métricas são dependentes do estado do SGBD. Quando ele não está sobrecarregado, os valores são quase constantes. Entretanto, quando o sistema de banco de dados está sobrecarregado, os valores crescem linearmente e depois, exponencialmente. Assim sendo, é necessário ter mecanismos eficazes para detectar a aumento ou diminuição destes valores [Malkowski et al., 2010].

Existem vários métodos para avaliar o desempenho de serviços em nuvem. Contudo, a natureza aleatória da demanda do usuário e as mudanças constantes na carga de trabalho ao longo do tempo tornam complexo o planejamento de capacidade em um curto período de tempo. Isso ocasiona alguns problemas: (i) a carga de trabalho muda constantemente, implicando em uma atualização contínua da configuração do sistema e este pode ficar sobrecarregado devido à execução dos procedimentos de adição e remoção de recursos; (ii) a quantidade e a precisão de dados coletados deve refletir o estado atual do sistema.

O tempo de resposta e vazão do serviço podem variar muito em curtos períodos de tempo e é necessário filtrar essa variabilidade para obter um padrão regular e evitar adição ou remoção de recursos. O SLADB utiliza uma estratégia similar a [Fito et al., 2010] para calcular os dados coletados. A coleta é realizada seis vezes com o intervalo de 10 segundos. Para cada coleta, o SLADB calcula a mediana e desvio-padrão do tempo de resposta. As duas medianas com menor desvio são selecionadas para obter os valores finais a serem armazenados.

Com os valores de tempo de resposta coletados, aplica-se uma média ponderada exponencial $X'_t = \alpha\ X_t + (1 \text{ - } \alpha)\ X_{t-1}$, onde α é o fator de ponderação ($0 \leq \alpha \leq 1$). Nesta média, os valores mais recentes têm maior peso, com o peso declinando exponencialmente à medida que esses valores se tornam ultrapassados. O fator de ponderação pode ser determinado experimentalmente, utilizando uma combinação de *benchmarks* com cargas artificiais e aplicações com cargas reais. Esta técnica de monitoramento é semelhante à adaptação de sobrecarga de serviços de internet [Welsh and Culler, 2003].

As métricas do SLA são calculadas diretamente no provedor de serviço, pois é complexo realizar medições no cliente em virtude das variações na qualidade da conexão. Contudo, muitos provedores de Internet fornecem soluções para melhorar a qualidade da conexão, por exemplo, *Virtual Private Network*(VPN), que podem ajudar na qualidade geral do sistema [Pierre and Stratan, 2012].

4.3.3 Penalidades

Como o custo é importante no ambiente de computação em nuvem, um modelo para representar SLA para banco de dados em nuvem deve contemplá-lo. Da perspectiva do provedor de serviço, o lucro é o objetivo final. Assim, um SLA orientado ao lucro é caracterizado por duas partes: receita e custo operacional. A receita é o valor pago pelos clientes ao provedor do serviço de acordo com o SLA. A receita não é fixa e pode mudar com uma potencial redução do pagamento ou mesmo penalidades, dependendo da qualidade de serviço. Custo operacional é o custo dos recursos usados para executar o serviço, por exemplo, pagamento da infraestrutura [Malkowski et al., 2010].

SLADB utiliza uma estratégia de SLA orientado ao lucro. Este tipo de SLA apresenta um funcionamento confiável dos sistemas, pois o provedor está motivado a prestar o serviço com uma qualidade elevada. Nos casos onde o provedor não é capaz de atender o SLA, ele é incentivado a continuar a prestar o serviço até obter o lucro. A receita é o valor pago pelo cliente ao provedor para cumprir um SLA S_i e o custo operacional é o gasto do provedor para a execução de um serviço com um SLA S_i especificado. Dessa forma, o lucro consiste na receita subtraída do custo operacional e das penalidades, conforme mostra a fórmula a seguir.

$$Lucro = Receita - (Custo + Penalidades) \tag{4.2}$$

A penalidade ou multa é um valor que o provedor deve pagar ao cliente, se o SLA S_i não é cumprido. Por exemplo, no Google AppEngine[2] ou Amazon S3[3], se a disponibilidade ficar abaixo de 99,9%, em seguida, os clientes recebem um crédito no

[2]http://code.google.com/appengine/sla.html
[3]http://aws.amazon.com/s3-sla/

serviço de acordo com a qualidade de serviço e proporcional a receita. Da mesma forma, o tempo de resposta é fundamental para garantir a qualidade de serviço e pode incorrer em penalidades em alguns modelos de serviços [Xiong et al., 2011]. No SLADB, o custo de penalidade é definido pela razão entre a soma de todas as consultas violadas pelo total de consultas multiplicado pela receita do sistema, de acordo com a fórmula abaixo.

$$CP = \frac{\sum consultas\_violadas}{\sum consulta} * Receita \tag{4.3}$$

Com isso, pode-se definir uma função de satisfação para o SLA, conforme apresentado a seguir. A função é atendida, caso o SLA S_i seja satisfeito, ou seja, todos os SLOs do SLA S_i sejam satisfeitos e, não é atendida, caso contrário.

$$FSS(Si) = \begin{cases} 1 & \text{se SLA Si é satisfeito} \\ 0 & \text{se SLA Si é violado} \end{cases} \tag{4.4}$$

4.4 ELASTICIDADE NA REPLICAÇÃO DE BANCO DE DADOS EM NUVEM

Elasticidade é a capacidade de aumentar e diminuir o número de réplicas para se adaptar à carga de trabalho [Perez-Sorrosal et al., 2011]. Um sistema elástico para replicação de dados deve realizar os seguintes passos: adição, remoção e migração dos dados para uma nova réplica. De acordo com [Soundararajan and Amza, 2006], a adição de uma nova réplica consiste de duas fases: migração de dados e estabilização do sistema. A migração é o processo de mover os dados e executar as transações pendentes para tornar uma réplica atualizada. A estabilização envolve o gerenciamento durante a adição de uma réplica ao sistema. Como a variação no número de réplicas altera os valores monitorados, deve-se garantir a estabilidade e o desempenho do sistema durante a adição de novas réplicas.

RepliC utiliza um processo de decisão para adição e remoção de réplicas baseado na função de satisfação do SLA (FSS), descrita na seção anterior. RepliC monitora o sistema e executa a decisão para modificar o estado de acordo com as métricas coletadas. A Figura 4.3 mostra a lógica para a adição e remoção de réplicas e as condições sob as quais uma réplica do banco de dados é adicionada ou removida.

No estado estável, isto é, onde o SLA está sendo atendido (ideal ou tolerável), RepliC monitora o SLA constantemente durante um intervalo de tempo por meio da

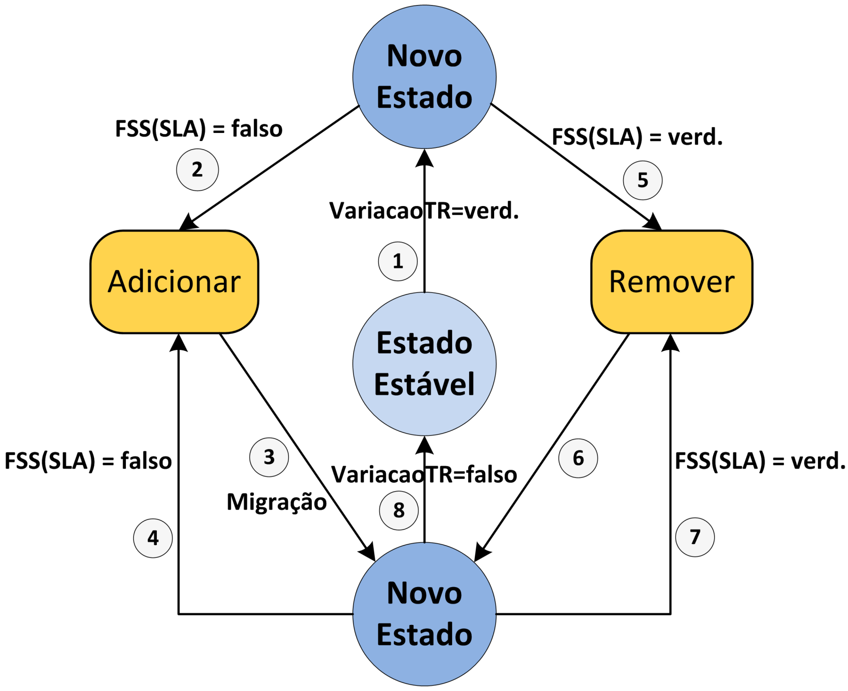

Figura 4.3 Lógica para adição e remoção de réplicas.

função FSS. Caso ocorra uma variação no tempo de resposta (1), o sistema passa para um novo estado. Se o SLA definido não for atendido, RepliC inicia o processo de adição de uma nova réplica (2). Em seguida, a migração dos dados é realizada para atualizar a réplica (3), conduzindo o sistema para um novo estado com a nova réplica. No novo estado, RepliC verifica o resultado das alterações realizadas e compara os valores coletados com o objetivo de determinar quanto a nova réplica agregou ao desempenho do sistema. Caso o SLA ainda não seja atendido (4), RepliC executa novamente os passos (2) e (3). Caso contrário, o sistema encontra-se no estado estável (8).

Se a função FSS for atendida por um determinado intervalo de tempo (tempo abaixo do ideal), uma réplica pode ser removida (5), conduzindo o sistema para um novo estado (6). Se a função FSS continuar sendo atendida (7), outra réplica pode ser removida. Se o SLA torna-se ideal, o sistema é conduzido para o estado estável (8).

Para melhorar o processo de adição, pode-se utilizar informações sobre as réplicas recém-adicionadas como uma heurística para determinar a contribuição agregada por uma nova réplica, visto que a estabilização pode demorar e comprometer o desempenho do sistema. Para tanto, o processo de adição e remoção deve monitorar a variação do SLA e modificar a quantidade de recursos somente se o sistema apresentar alterações constantes no aumento ou diminuição da carga de trabalho. A estratégia adotada na função FSS, que utiliza a média ponderada exponencial para o processo de monitoramento, evita a adição ou remoção de réplicas desnecessárias, principalmente quando ocorrem alterações pontuais da carga de trabalho.

4.4.1 Adição de Réplicas

Para realizar a adição de uma nova réplica, RepliC verifica a possibilidade de adicionar uma nova réplica do banco de dados em uma das VMs em execução, observando os recursos de CPU, memória e disco disponíveis. Caso a adição seja possível, a primeira VM com recursos disponíveis é selecionada. Caso não exista uma VM com recursos disponíveis, uma nova VM é iniciada. Em seguida, a migração de dados para a nova réplica é realizada. Por fim, a réplica recebe as transações executadas desde o início do processo de migração. Durante o processo de adição, RepliC suspende as decisões de adição até que o processo seja concluído. Após a conclusão, o SLA é monitorado considerando a nova réplica.

4.4.2 Remoção de Réplicas

Com a diminuição da carga de trabalho de um determinado banco de dados e, consequentemente, a satisfação da função FSS, pode-se realizar a remoção de uma quantidade de réplicas associadas a este banco de dados, desde que a qualidade do serviço seja mantida. Quando uma réplica precisa ser removida do sistema, RepliC seleciona a réplica com menor carga de trabalho e esta réplica não recebe novas requisições. Quando todos os clientes atuais da réplica forem desconectados, ela é removida. Se a remoção precisa ser rápida, os clientes são redirecionados para outra réplica deste banco de dados em execução.

Com a remoção de réplicas dos bancos de dados, muitas VMs podem ficar com poucas réplicas, implicando em recursos ociosos. Para tratar este problema, podem-se reorganizar as réplicas de tal forma a reduzir a quantidade de VMs em uso. Este processo de reorganizar consiste na adição de uma nova réplica na VM de destino e a remoção da réplica na VM atual ou VM de origem. Esta reorganização não deve interferir no SLA dos bancos de dados e deve ser realizada de forma supervisionada por meio técnicas que permitam aos desenvolvedores acompanharem o processo para garantir a qualidade de serviço. Este processo de reorganização não é abordado neste trabalho, mas pode ser adaptado à partir das ações de adicionar e remover réplicas presentes no RepliC.

4.4.3 Migração de Dados entre as Réplicas

A migração de dados trata da atualização de uma réplica nova. Durante o processo de adição, a nova réplica recupera os dados persistidos no serviço de armazenamento, tais como *snapshot* e *logs*, visto que as instâncias dos SGBDs estão sendo executados em um ambiente virtualizado. Associado a estes dados, as transações executadas nas outras réplicas durante a adição da nova réplica são recuperadas. Este passo é realizado armazenando as transações executadas e, em seguida, enviando-as para a nova réplica. A migração dos dados deve minimizar as interrupções e sobrecarga no processamento das transações. Para auxiliar este processo, RepliC armazena e gerencia as transações a serem executadas. Isso melhora o desempenho deste processo, pois diminui a sobrecarga sobre as réplicas em execução.

De forma similar a [Savinov and Daudjee, 2010], RepliC mantém *logs* das transações de atualizações. Os *logs* são armazenados até que as transações sejam efetivadas ou abortadas pelas réplicas e são reexecutados durante a migração de dados para atualizar uma nova réplica. Para detectar o conjunto de atualizações que ainda não foram aplicados, utiliza-se um identificador para cada transação. Para melhorar o desempenho na execução das transações em uma nova réplica, estas transações são executadas em bloco. Quando os blocos são executados, as réplicas tornam-se atualizadas e podem receber requisições enviadas ao sistema.

A migração dos dados deve conduzir o sistema entre estados consistentes. Para detalhar o processo de migração, utiliza-se a notação da Tabela 4.1, que mostra os componentes utilizados na migração e envolve um conjunto de fases. A Figura 4.4 apresenta as fases realizadas durante a migração dos dados entre as réplicas.

Fase de Preparação: Nesta fase, um ponto de controle é criado e uma réplica atualizada R_{Atl} é selecionada do sistema. Em seguida, uma imagem (snapshot) I_{Snap} do banco de dados da réplica R_{Atl} é armazenado no serviço de armazenamento S_{Arm}. A partir deste ponto, o serviço de escalonamento S_{Esc} salva o *log* das transações confirmadas.

Fase de Transferência: O processo de atualização da réplica R_{Nova} é iniciada nesta fase. A imagem I_{Snap} é recuperada do S_{Arm} e restaurada na réplica R_{Nova}.

Parâmetro	Valor
R_{Atl}	Réplica atualizada
R_{Nova}	Réplica nova
S_{Esc}	Serviço de escalonamento
S_{Arm}	Serviço de armazenamento
I_{Snap}	Imagem (snapshot) do banco de dados

Tabela 4.1 Convenções de notação para migração de dados entre as réplicas.

Fase de Conclusão: Nesta fase, o *log* armazenado pelo S_{Esc} é enviado e executado na réplica R_{Nova}. Assim, a R_{Nova} torna-se atualizada em relação as outras réplicas e começa a receber as transações enviadas ao sistema.

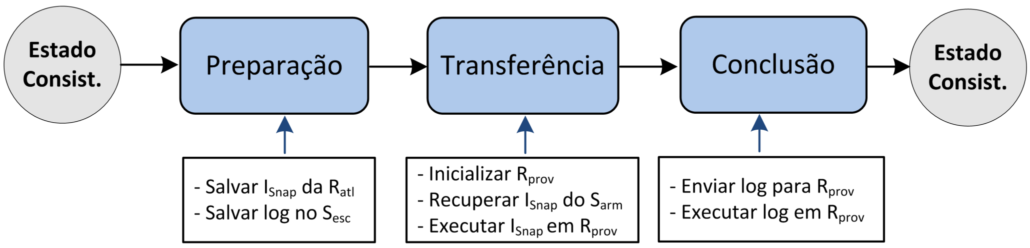

Figura 4.4 Migração dos dados entre as réplicas.

4.4.4 Provisionamento de Banco de Dados em Nuvem

O provisionamento no ambiente de nuvem é fundamental para garantir a qualidade do serviço. As estratégias podem ser reativas ou proativas. Estratégias de provisionamento reativas são mais adequadas para a camada web, visto que nesta camada pode-se adicionar maior capacidade sempre que necessário e a latência refere-se apenas à inicialização da máquina virtual.

Diferentemente, o provisionamento de uma nova réplica de banco de dados envolve os seguintes passos: (i) extrair o conteúdo do banco de dados de uma réplica existente, se este ainda não estiver disponível e (ii) restaurar esse conteúdo em uma nova réplica. Estas operações podem demorar, dependendo do tamanho do banco de dados, configurações do sistema de armazenamento e ferramentas de *backup*/restauração [Cecchet et al., 2011]. Dessa forma, para a camada de gerenciamento de dados, deve-se utilizar estratégias proativas.

Uma estratégia para provisionamento deve considerar o tempo para replicar o estado do banco de dados e a sobrecarga de sincronização. Caso estes pontos não sejam considerados, a nova réplica pode demorar um intervalo grande de tempo para estar operacional e não poderá lidar com um aumento da carga de trabalho em tempo hábil. Assim, é necessário estimar o tempo necessário para replicar os dados. Contudo, não é trivial estimar o tempo exato necessário para gerar uma nova réplica.

Modelos de determinação de capacidade podem ajudar a predizer a carga de trabalho futura e técnicas de provisionamento podem ser utilizadas para estimar o número de réplicas necessárias para atender o serviço. A técnica de provisionamento do RepliC é simples e ela define apenas quando uma nova réplica deve ser adicionada de acordo com a carga de trabalho. RepliC não implementa técnicas específicas de predição e provisionamento de recursos, pois estas são características ortogonais ao processo de replicação. Aspectos relativos à predição e provisionamento são abordados em nosso trabalho [Santos et al., 2012] e podem ser adicionados ao RepliC com pequenos ajustes.

Em geral, existe um equilíbrio entre o tempo para realizar um *snapshot* do banco de dados, o tamanho do *log* transacional e a quantidade de operações de atualização na carga de trabalho [Cecchet et al., 2011]. Por exemplo, uma nova réplica pode ser iniciada com um *snapshot* antigo, que evita a sobrecarga da fase de *backup*. Entretanto, a utilização de um *snapshot* antigo força o sistema a manter um *log* transacional grande e aumenta o tempo para reexecutar as atualizações deste *log* durante a nova execução. Por outro lado, utilizar um novo *snapshot* para cada nova réplica pode incorrer em sobrecargas significativas durante a fase de *backup*, especialmente se o banco de dados é grande. Como este trabalho considera o contexto de aplicações que manipulam pequenas quantidades de dados, esta questão do tamanho do banco de dados não se apresenta como um desafio considerável.

No ambiente virtualizado, onde os sistemas de bancos de dados estão sendo executados, os dados são persistidos por meio de serviços de armazenamento, visto que as máquinas virtuais possuem apenas armazenamento em memória principal. Dessa forma, RepliC utiliza uma estratégia incremental para armazenar os dados, conforme apresenta a Figura 4.5.

Quando um novo banco de dados é adicionado ao sistema, é realizada uma cópia inicial completa deste banco no serviço de armazenamento. Com o processamento das transações, estas transações são organizadas em blocos de dados com um identificador único. Estes blocos são persistidos por meio do serviço de armazenamento à medida que são preenchidos com um conjunto de transações. Quando a quantidade de blocos aumenta, RepliC salva uma cópia atual completa do banco de dados e a estratégia é reiniciada.

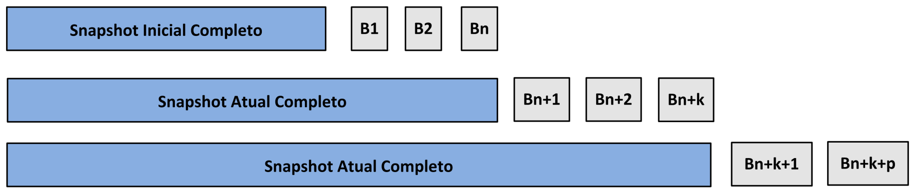

Figura 4.5 Estatégia incremental para armazenamento dos dados.

Os *logs* persistidos pelo RepliC são removidos por meio de um processo de coleta de lixo quando se tornam obsoletos, isto é, quando um novo *snapshot* do banco de dados é persistido. Para minimizar o tempo de reintegração, por exemplo, após uma falha, e para facilitar a coleta de lixo periódica do *log*, utilizam-se pontos de verificação em intervalos de tempo regulares.

4.5 ALGORITMOS PARA REPLICAÇÃO DE BANCO DE DADOS EM NUVEM

Esta seção descreve os principais algoritmos do RepliC referentes a elasticidade. Estes algoritmos tratam do monitoramento, adição/remoção de réplicas e a migração de dados entre as réplicas. A Tabela 4.2 mostra a notação utilizada para descrever os recursos do sistema.

Com o objetivo de melhorar a disponibilidade e desempenho, este trabalho considera as seguintes restrições: a) cada banco de dados possui pelo menos duas réplicas; b) as réplicas de um serviço não estão alocadas em uma mesma máquina virtual; c) o somatório dos recursos das réplicas deve ser menor do que os recursos da máquina.

O Algoritmo 1 refere-se ao monitoramento do SLA. Este procedimento é executado continuamente (linha 2) e verifica o SLA monitorado para cada banco de dados (linhas 3 e 4). Caso o SLA definido não seja atendido pela função FSS (linha 5), uma nova réplica deste banco de dados é criada (linha 6). Em seguida, a migração de dados é realizada para copiar os dados para a nova réplica (linha 7) e a mesma é adicionada ao sistema, recebendo as requisições (linha 7). Por fim, o sistema é estabilizado para refletir o estado atual considerando a nova réplica.

Caso o SLA seja atendido, uma réplica é removida do sistema. Neste caso, as

Parâmetro	**Valor**
VM	Conjunto de m máquinas virtuais
VME	Conjunto de máquinas virtuais em execução
vm	Máquina virtual
BD	Conjunto de k banco de dados
bd	Banco de dados - representa um inquilino/réplica
cap	Quantidade de recursos
res	Recurso residual - representa os recursos disponíveis

Tabela 4.2 Notação utiliza para descrever os recursos do sistema.

requisições são redirecionadas para um conjunto de réplicas de tal forma que a réplica com menor carga de trabalho não receba requisições (linha 10). A réplica é removida (linha 11) e o sistema é estabilizado para refletir o estado atual.

Algoritmo 1 - Procedimento de Monitoramento do SLA

```
1: procedure Monitoramento(SLA)
2:     loop
3:         for bd ∈ BD do
4:             for i ∈ range(SLA(bd)).length) do
5:                 if FSS(SLA(bd)) = false then
6:                     CriarReplica(bd_i);
7:                     MigrarDados(bd_i);
8:                     AdicionarSistema(bd_i);
9:                 else
10:                    RedirecionarRequisicoes();
11:                    RemoverSistema(bd_i);
12:                end if
13:            end for
14:        end for
15:    end loop
16: end procedure
```

O Algoritmo 2 descreve a função de busca utilizada durante a adição de uma réplica. Esta função recebe como entrada um banco de dados (linha 1) e verifica se é possível adicionar uma réplica deste banco em uma máquina virtual em execução. Inicialmente, as máquinas virtuais são ordenadas de acordo com sua capacidade residual (linha 2), isto é, a capacidade de recursos disponível. Em seguida, as máquinas são percorridas para verificar se a réplica do banco de dados pode ser adicionada (linhas 3 e 5). Se a quantidade de recursos necessários para o banco de dados for maior do que os recursos disponíveis nas máquinas (linha 6), a função é abortada e uma nova máquina é retornada (linha 17).

Se existir uma máquina com recursos para comportar a réplica do banco de dados, esta função verifica se já existe uma réplica deste banco de dados nesta máquina (linha 12), visto que esta é uma restrição do problema abordado neste trabalho. Caso não exista, esta máquina é retornada (linha 13). Caso contrário, uma nova máquina é retornada (linha 17).

Algoritmo 2 - Função para Buscar um Banco de Dados

```
1: function Busca(bd )
2:     Ordenar VME por capacidade residual res;
3:     for vm ∈ VME do
4:         flag = true;
5:         for i ∈ range(res.length) do
6:             if cap^i(bd) > res^i(vm) then
7:                 flag = false;
8:                 break;
9:             end if
10:        end for
11:        if (flag = true) then
12:            if (bd ∉ VME(vm)) then
13:                return  vm;
14:            end if
15:        end if
16:    end for
17:    return  New(vm);
18: end function
```

O Algoritmo 3 mostra o procedimento para adicionar uma nova réplica ao sistema. Este procedimento consiste em realizar uma busca utilizando a função definida anteriormente (linha 2), que retorna uma máquina na qual a réplica será adicionada. Após a adição (linha 3), a capacidade da máquina virtual é decrementada (linha 5) e a réplica é adicionada ao conjunto máquinas em execução (linha 7). Este procedimento é executado duas vezes quando um novo banco de dados é adicionado à infraestrutura, visto que, considera-se que cada banco possui duas réplicas em máquinas virtuais diferentes.

O Algoritmo 4 remove uma réplica de banco de dados do sistema. Este procedimento consiste em selecionar a réplica com a menor carga de trabalho (linha 2) baseada em uma variação da função descrita para adicionar novas réplicas. Em seguida, a réplica é removida (linha 3), liberando os recursos na máquina onde a mesma estava sendo executada (linha 5). Após a remoção, as VMs que não possuírem réplicas também podem ser removidas do sistema.

Algoritmo 3 - Procedimento para Adicionar uma Nova Réplica

procedure $Adiciona(bd)$
    $vm = Busca(bd);$
    $vm.Adiciona(bd);$
    for $i \in range(cap.length)$ **do**
        $cap^i(vm) = cap^i(vm) - cap^i(bd);$
    end for
    $VME(vm) = VME(vm) + bd;$
end procedure

Algoritmo 4 - Procedimento para Remover uma Réplica

procedure $Remove(bd)$
    $vm = BuscaMenorCarga(bd);$
    $vm.Remove(bd);$
    for $i \in range(cap.length)$ **do**
        $cap^i(vm) = cap^i(vm) + cap^i(bd);$
    end for
    $VME(vm) = VME(vm) - bd;$
end procedure

O Algoritmo 5 mostra o procedimento para migrar dados para uma réplica. Primeiramente, um *snapshot* ou imagem da réplica é persistido no serviço de armazenamento (linha 2). O serviço de escalonamento persiste um *log* com as operações de atualização executadas pelas demais réplicas (linha 3). Em seguida, a imagem persistida é recuperada na nova réplica (linha 4) e as operações de atualizações pendentes em uma fila são executadas (linha 5). Finalmente, a réplica é adicionada ao sistema (linha 6).

Algoritmo 5 - Procedimento para Migração de Dados para uma Réplica

1: **procedure** $Migra$
2:     $ServicoArmazenamento(imagem);$
3:     $ServicoEscalonamento(log);$
4:     $Restore(imagem);$
5:     $Execute(log);$
6:     $AdicionaSistema(bd);$
7: **end procedure**

4.5.1 Exemplo de Execução do RepliC

Para ilustrar a execução do RepliC, considere três serviços de banco de dados, $YCSB$, $Wiki$ e $Twitter$, que representam bancos de dados do *Yahoo! Cloud Serving Benchmark (YCSB)*, Wikipedia e Twitter, respectivamente. Cada serviço recebe uma carga de trabalho/taxa de transações variável ao longo do tempo. Estes serviços possuem os seguintes SLAs de tempo de resposta $YCSB_{SLA}$= 0.1s, $Wiki_{SLA}$= 0.2s, $Twitter_{SLA}$= 0.5s. A penalidade é definida pela soma das transações que não atenderam ao SLA, com valor de \$ 0.001 por transação não atendida.

Considere a infraestrutura da Amazon com máquinas do tipo *small*, com custo de \$ 0.08. A alocação inicial considera que duas réplicas de cada banco de dados são utilizadas, com a seguinte alocação VM_1={$YCSB$, $Wiki$, $Twitter$}, VM_2={$YCSB$, $Wiki$, $Twitter$}. Para simplificar o exemplo, suponha que não ocorre interferência entre as réplicas ou inquilinos.

Suponha, no momento inicial de execução dos serviços, as seguintes taxas de transações para os serviços $YCSB_{taxa}$= 4t/s, $Wiki_{taxa}$= 1t/s, $Twitter_{taxa}$= 3t/s. Neste cenário, os tempos de respostas (TR) calculados são $YCSB_{TR}$= 0.03s, $Wiki_{TR}$= 0.09s, $Twitter_{TR}$= 0.23s e os SLAs estão sendo atendidos para os três serviços.

No $Tempo_{crono}$ = 3hs, suponha que a taxa de transações do $Twitter$ aumentou para $Twitter_{taxa}$= 12t/s e o $Twitter_{TR}$= 1.2s. Isso implica em violação do $Twitter_{SLA}$. Para tratar isso, RepliC adiciona uma nova máquina e uma réplica do $Twitter$, visto que as máquinas em uso já possuem uma réplica do $Twitter$. Assim sendo, a VM_3={$Twitter$} é adicionada. Após a adição, que demorou 35s desde a violação, o sistema é monitorado e verifica-se $Twitter_{TR}$= 0.34s.

No $Tempo_{crono}$ = 4hs, suponha que a taxa de transações do $Wiki$ foi alterada para $Wiki_{taxa}$= 6t/s e o $Wiki_{TR}$= 0.32s. O sistema verifica a disponibilidade de recursos nas máquinas em uso para alocar uma réplica do $Wiki$. A VM_3 não possui recursos. As VM_1 e VM_2 possuem recursos, mas já possuem uma réplica do $Wiki$. Dessa forma, uma nova máquina e uma réplica são adicionadas VM_4={$Wiki$}, levando 42s para estar em execução desde a violação e tornando $Wiki_{TR}$= 0.13s.

Considere que o $Twitter$ encontra-se com $Twitter_{TR}$= 0.35s após o $Tempo_{crono}$ = 6hs e a carga de trabalho não apresenta alterações, tornando a função de satisfação do SLA verdadeira. Neste caso, o sistema redireciona a carga de trabalho da VM_3={$Twitter$} para a VM_1= {$YCSB$, $Wiki$, $Twitter$}, alterando $Twitter_{TR}$= 0.42s, mas não viola o SLA, que é $Twitter_{SLA}$= 0.5s.

Com isso, a réplica do $Twitter$ da VM_3 é removida do sistema. Como a VM_3 continha apenas a réplica removida, essa máquina também é removida do sistema, diminuindo os custos. Neste exemplo, os serviços foram encerrados no $Tempo_{crono} = 7$ hs. O custo do provedor com as VMs é calculado pela quantidade de VMs em cada intervalo de tempo: (1-3hs x 2 VM) + (3-4hs x 3 VM) + (4-6hs x 4 VM) + (6-7hs x 3 VM), totalizando \$ 1.12.

Durante a execução, ocorreu violação do SLA com o *Twitter* e *Wiki*. Para calcular o total de transações que não atenderam o SLA, multiplica-se a taxa de transações no momento da violação pelo tempo decorrido desde a violação até a adição da nova réplica. Para o *Twitter* tem-se 12t/s x s = 420 transações e para o *Wiki* 6t/s x 42s = 252 transações, totalizando 672 transações. Como o custo por transação violada é \$ 0.001, o valor final da penalidade paga pelo provedor é \$ 0.672. O custo total do provedor é calculado pela soma do custo com as VMs \$ 1.12 e as penalidades \$ 0.672, totalizando \$ 1.792.

Embora este exemplo seja simples, pode-se observar a execução da abordagem proposta. RepliC melhora a utilização dos recursos por meio do modelo multi-inquilino adotado, reduzindo as penalidades e melhorando a qualidade por meio da elasticidade.

4.6 CONCLUSÃO

Este capítulo apresentou RepliC, uma abordagem para a replicação de banco de dados em nuvem, cujo propósito é garantir a qualidade de serviço com a utilização eficiente dos recursos. Foram elencados quais os pressupostos considerados pelo RepliC e um modelo de qualidade de serviço foi descrito. Os algoritmos desenvolvidos referentes à elasticidade também foram apresentados, assim como um exemplo de execução. O modelo multi-inquilino utilizado pelo RepliC permite compartilhar os recursos de forma eficiente. A estratégia de elasticidade associada às características da infraestrutura em nuvem garante a qualidade do serviço por meio da adição e remoção de réplicas. O próximo capítulo trata de questões relativas à consistência dos dados e tolerância a falhas implementadas pelo RepliC.

CAPÍTULO 5

CONSISTÊNCIA PARA BANCO DE DADOS MULTI-INQUILINO

Este capítulo descreve a estratégia para consistência de banco de dados multi-inquilino utilizada pelo RepliC. Inicialmente, são destacados aspectos relevantes à consistência de dados em nuvem. Em seguida, são abordados algoritmos para tratar a sincronização entre as réplicas em ambientes virtualizados de tal forma a garantir a consistência. Finalmente, são apresentadas estratégias para tratar falhas neste ambiente.

5.1 INTRODUÇÃO

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] [Baker et al., 2011]. Devido à grande quantidade de aplicações que utilizam consistência forte, RepliC implementa este tipo de consistência, onde as aplicações sempre acessam dados atualizados.

Aspectos Relevantes

Para melhorar o desempenho, o processo de replicação deve estar completamente sintonizada com a aplicação e a carga de trabalho. Além disso, alterações no padrão de acesso podem conduzir a mudanças nas estratégias de replicação, o que torna a concepção de uma estratégia ideal muito complexa [Sivasubramanian et al., 2005].

Uma questão importante em qualquer sistema replicado é a consistência. O gerenciamento das réplicas tem dois aspectos principais: a propagação de atualizações e controle de concorrência [Kemme et al., 2010]. As estratégias de propagação de atualizações podem ser classificadas em *push* e *pull*. Estratégias de atualização de dados baseadas em *push* asseguram a consistência das réplicas de forma síncrona, pois todas as atualizações são enviadas para as réplicas imediatamente. Estratégias baseadas em *pull* propagam as atualizações de forma assíncrona e são mais apropriadas para evitar atualizações desnecessárias quando não ocorre acesso de dados entre duas transações de atualizações subsequentes.

A propagação de atualizações não é suficiente para manter a consistência dos dados, pois deve-se ter mecanismos para sincronização entre as diferentes réplicas. O sistema deve lidar com atualizações simultâneas dos dados em várias réplicas. Soluções tradicionais como o protocolo de duas fases (2PC) são muitas limitadas, pois exigem bloqueio global de todas as réplicas, reduzindo assim as melhorias de desempenho proporcionadas pela replicação [Özsu and Valduriez, 2011].

Outro ponto importante refere-se à alocação das máquinas virtuais pelos provedores de infraestrutura. Os provedores criam as máquinas virtuais de forma automática com o objetivo de melhorar a utilização dos recursos, minimizando os custos. Com isso, as máquinas virtuais correspondentes a uma mesma aplicação ou banco de dados podem estar em estruturas físicas diferentes, por exemplo, *racks* diferentes, o que pode aumentar a latência [Wada et al., 2011].

RepliC utiliza o modelo de replicação transparente combinando as estratégias *push* e *pull*, onde o cliente da aplicação que acessa o banco de dados não precisa se preocupar com problemas de replicação e pode se concentrar apenas nos requisitos funcionais da aplicação. Em relação a alocação das máquinas virtuais na infraestrutura, RepliC não interfere ou modifica esta alocação. O ajuste na quantidade de réplicas é realizado de acordo com o monitoramento do ambiente, verificando parâmetros de desempenho.

Dentre vários tipos de protocolos de replicação, a abstração de comunicação em grupo (CG) é uma tecnologia eficiente para implementar estes protocolos, pois provê garantias de confiabilidade que simplificam a aplicação de técnicas de tolerância a falhas [Birman, 2012]. Estas primitivas de comunicação em grupo têm sido aplicadas com eficiência nesses protocolos, tanto em abordagens síncronas como assíncronas. Dessa forma, comunicação em grupos é uma solução viável para implementar protocolos de replicação.

5.2 COMUNICAÇÃO EM GRUPOS

A abstração de grupos [Birman, 2012] tem provado ser um mecanismo eficiente para implementar protocolos de consistência de réplicas, facilitando o desenvolvimento de aplicações distribuídas confiáveis. Essa abstração tem por objetivo resolver problemas básicos de inconsistências na comunicação entre processos distribuídos que cooperam para a execução de uma tarefa. Nesse sentido, um grupo é apenas um conjunto de processos que cooperam. Na forma mais geral de comunicação em grupos, um determinado processo pode pertencer a diferentes grupos, ou seja, os grupos podem se sobrepor.

A existência de múltiplos membros em um grupo é invisível para o cliente. Um cliente que deseja se comunicar com um grupo, ao invés de enviar uma mensagem individual para cada membro, envia apenas uma mensagem para o endereço do grupo. A abstração de grupos evita que um processo externo tenha que conhecer individualmente cada um dos membros.

O serviço baseado em grupos é composto de duas partes: *group membership* e *group communication* [Birman, 2012]. O serviço de *membership* provê aos processos a composição do grupo. Tal composição evolui de acordo com o interesse dos processos de se unirem (*join*) ou sairem (*leave*) do grupo e com a verificação da ocorrência de falhas, sejam de processos pertencentes ao grupo ou de canais de comunicação. Esse serviço abstrai os eventos que provocam mudanças na composição do grupo, através da entrega a cada membro de uma "visão", ou seja, a composição do grupo em um determinado instante, atualizada e composta por todos os processos considerados ativos. Essas visões são atualizadas de forma coerente de tal maneira que os processos que as instalam concordam com a sua composição.

O objetivo do serviço de *communication* é prover aos processos primitivas de comunicação necessárias à troca eficiente de mensagens [Tanenbaum and Steen, 2006]. A principal primitiva oferecida por esse serviço é a de *multicast* ou difusão. Quando uma entidade externa invoca essa operação a um grupo, essa primitiva garante a entrega da mensagens a todos os membros do grupo. Uma mensagem enviada pode ser confiável ou não-confiável. Quando o envio é confiável, garante-se que a mensagem será recebida por todos os processos não-falhos ou por nenhum deles, garantindo a propriedade da atomicidade. Com essa propriedade, o processo que envia a mensagem para um grupo irá receber uma mensagem de erro se um ou mais integrantes do grupo tiver problema no recebimento. Já quando o envio é não-confiável, o serviço de CG não garante a atomicidade.

O serviço de grupos também devem garantir alguma forma de integração entre a entrega de invocações regulares (*multicast*) e a entrega de eventos de mudança de visão.

Dessa forma, tais serviços fazem uso do paradigma de *View Synchronous Communication* [Birman, 2012]. Esse paradigma garante que as mensagens de *multicast* sejam entregues na mesma ordem para todos os processos do grupo.

Classificação

Grupos podem ser classificados segundo suas características. A principal classificação é quanto à facilidade em refletir ou não as alterações do sistema distribuído [Birman, 2012]. Em grupos estáticos, não ocorre qualquer alteração na quantidade de membros durante a vida útil do sistema, não sendo necessário um serviço para controlar a entrada e saída de membros. Grupos dinâmicos sofrem alterações, refletindo entradas e saídas de membros, ou seja, qualquer alteração no número de membros de um grupo é indicada na visão do grupo.

Os grupos também podem ser classificados quanto à forma na qual o *multicast* é implementado [Birman, 2012]. Se todos os membros estão aptos a fazer o *multicast*, sem necessitar de um membro coordenador, o grupo é não-hierárquico ou simétrico. Quando o grupo precisa de um membro coordenador para realizar o sequenciamento de mensagens, o grupo é assimétrico ou hierárquico. Neste caso, apenas o coordenador precisa estar apto a fazer o *multicast*. Os demais membros que desejam realizar um *multicast* enviam uma mensagem ponto-a-ponto para o coordenador. Além dessas classificações, os grupos podem ser fechados, onde os membros recebem somente mensagens de outros membros, ou abertos, no qual os membros recebem mensagens de qualquer processo.

Ordenação de Mensagens

Uma mensagem recebida por um membro do grupo que está funcionando corretamente passa por um processo de entrega que eventualmente precisa ser controlado para que a sequência de entrega seja a mesma sequência de envio. Os tipos de ordenação que normalmente estão implementadas nos sistemas de CG são:

- FIFO: mensagens enviadas a partir de um único membro são entregues na ordem em que são enviadas a todos os membros do grupo. Por exemplo, para todas as mensagens M_1 e M_2 e todos os processos P_i e P_j, se P_i envia M_1 antes de enviar M_2, então M_2 não é recebida em P_j antes que M_1 seja recebida.
- Causal: mensagens enviadas a partir de múltiplos membros que possuam uma

relação de causalidade são entregues a todos os membros do grupo respeitando sua ordem de causalidade. Duas mensagens possuem relação de causalidade caso uma delas tenha sido gerada depois do recebimento da outra. Pode ser necessário preservar essa ordem em todos os participantes, já que o conteúdo da segunda mensagem pode ser afetado pelo processamento da primeira.

- Total: todas as mensagens enviadas ao grupo são entregues a todos os membros na mesma ordem. Por exemplo, para todas as mensagens M_1 e M_2 e todos os processos P_i e P_j, se M_1 é recebida em P_i antes que $M2$ seja, então M_2 não é recebida em P_j antes que M_1 seja.

5.3 PROTOCOLO PARA REPLICAÇÃO

RepliC combina protocolos síncronos, assíncronos e primitivas de comunicação em grupo, visando obter uma estrutura eficiente para a replicação de banco de dados em nuvem. O critério *one-copy serializability* é usado neste trabalho como modelo de corretude [Bernstein and Newcomer, 2009]. Esse modelo garante que a execução de transações concorrentes produza resultados equivalentes a uma execução sequencial do mesmo conjunto de transações em uma instância do banco de dados. Isso significa que clientes de um sistema replicado o enxergam como um banco de dados com apenas uma instância e que operações submetidas sempre obtêm dados atualizados.

Como no ambiente de computação em nuvem os recursos são compartilhados, dificultando o gerenciamento, este trabalho realiza uma estratégia baseada nos trabalhos propostos por [Vo et al., 2010] [Cao et al., 2011], que consiste em ter diferentes camadas ou grupos virtuais de replicação. Também foi estendido a estratégia proposta em nosso trabalho anterior [Sousa et al., 2007] para contemplar os seguintes aspectos:

1. Execução em ambientes virtualizados: permite a execução em máquina virtual e utilização de serviços de armazenamento em nuvem.

2. Modelo multi-inquilino: possui suporte ao modelo multi-inquilino de instância de banco de dados.

3. Modelo relacional: permite o gerenciamento de sistemas que utilizam o modelo de dados relacional.

Para auxiliar o gerenciamento das réplicas, RepliC utiliza *grupos virtuais*. Cada grupo virtual corrresponde a um conjunto de réplicas de banco de dados de uma determinada aplicação e estes grupos podem ser de dois tipos: grupo de atualização e grupo de leitura, que tratam transações de atualização e transações de leitura, respectivamente.

As réplicas de cada banco de dados da aplicação e suas respectivas transações possuem um identificador único que permite gerenciá-las. RepliC realiza a distinção entre as transações, classificando-as de acordo com o conteúdo de suas operações. Transações que contenham apenas operações de leitura são consideradas de leitura. Caso a transação contenha pelo menos uma operação de modificação (inserção, atualização ou remoção), ela é classificada como de atualização.

Para reduzir os custos, apenas a quantidade mínima de réplicas é alocada para cada banco de dados. RepliC utiliza duas réplicas de cada banco de dados da aplicação, sendo que uma réplica está contida no grupo de atualização e outra no grupo de leitura. Contudo, pode-se configurar RepliC para iniciar com uma quantidade maior de réplicas. Estes grupos são alterados pela adição e remoção de réplicas de acordo com a carga de trabalho e a satisfação do SLA.

A estratégia de utilizar grupos tem como objetivo melhorar o desempenho do sistema, visto que a carga de trabalho pode ser distribuída e apenas uma parte das réplicas necessita ser atualizada a cada modificação, diminuindo conflitos durante as operações de atualização. RepliC implementa replicação total, pois esta tese considera o contexto de inquilinos com requisito de gerenciamento de pequenas quantidades de dados.

5.3.1 Grupo de Atualização

A réplica do grupo de atualização que recebe uma transação de atualização é denominada de *réplica primária*. Esta réplica é responsável por verificar os conflitos com as demais réplicas deste grupo, denominadas de *réplicas secundárias* em relação a réplica primária, por meio do protocolo proposto por [Sousa et al., 2007]. Em seguida, os dados são persistidos no serviço de armazenamento da nuvem, que utiliza múltiplas cópias dos dados persistidos, garantindo a alta disponibilidade dos dados.

Os serviços de armazenamento em nuvem disponibilizados pelos provedores utilizam uma estrutura de replicação em nível de arquivos, por exemplo, Amazon EBS e S3. Com isso, quando uma das réplicas do grupo de atualização executa e salva os dados, estes serviços garantem a durabilidade dos mesmos. As demais réplicas do grupo de atualização executam as transações e confirmam a execução, mas não é necessário que elas persistam os dados no serviço de armazenamento, apenas localmente, para a conclusão de uma transação. Em caso de falha em algumas réplicas, os dados podem ser recuperados por meio destes serviços.

As modificações do grupo de atualização são serializadas e enviadas continuamente pela réplica primária para cada réplica do grupo de leitura, denominada *slaves* por meio de um *multicast* com a propriedade de ordenação FIFO. Essas modificações são adicionadas em filas locais de cada réplica do grupo de leitura e executadas na mesma sequência do grupo de atualização.

A Figura 5.1 mostra um exemplo do grupo de atualização. Na VM_1, a réplica do banco de dados BD_1 é a réplica primária em relação ao cliente c_x e a réplica do banco de dados BD_1 na VM_2 e VM_M são réplicas secundárias. Por outro lado, a réplica do banco de dados BD_1 na VM_2 é a réplica primária para o cliente c_y e as outras réplicas deste banco de dados são secundárias. Neste exemplo, o banco de dados BD_1 na máquina VM_M é sempre uma réplica secundária.

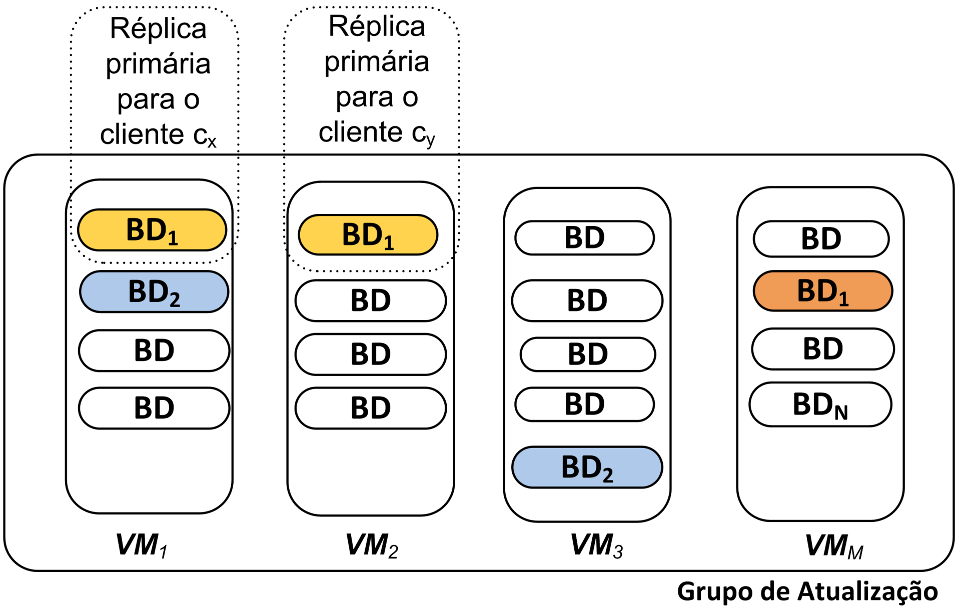

Figura 5.1 Grupo de atualização

5.3.2 Grupo de Leitura

O grupo de leitura utiliza múltiplas filas para gerenciar os dados enviados pelo grupo de atualização, onde cada réplica da máquina virtual possui uma fila responsável por gerenciar as transações desta réplica. Para cada réplica, é criada uma fila persistente com as transações correspondentes. Cada máquina virtual possui um conjunto de banco de dados e para cada um destes existe uma fila associada. Estas filas contêm as transações referentes às modificações enviadas pelo grupo de atualização. Para garantir a consistência, as transações contidas nas filas são aplicadas as réplicas.

O grupo de leitura executa dois tipos de transações: propagação e *refresh*. Transações de propagação são executadas durante o tempo ocioso de uma réplica, ou seja, quando não estão sendo executadas transações de leitura ou transações de *refresh*, com o objetivo de efetivar as atualizações. Transações de *refresh* são aplicadas para adicionar as transações contidas na fila local a uma réplica do grupo de leitura.

Durante a execução das transações de leitura em uma determinada réplica, RepliC gerencia as réplicas por meio da execução de transações de propagação e de *refresh*. Caso novas modificações sejam adicionadas na fila local, RepliC continua a execução da consulta nesta réplica e posteriormente executa uma transação de propagação, adicionando o conteúdo da fila ao banco de dados local. Quando uma nova transação é direcionada para esta réplica, RepliC realiza as seguintes verificações: (i) se a nova transação requisita dados que foram atualizados, uma transação de *refresh* é executada, (ii) caso contrário, a transação é executada sem a necessidade de transações de *refresh* ou propagação.

A Figura 5.2 apresenta um exemplo do grupo de leitura. Cada replica *slave* possui uma fila com transações a serem executadas. Replicas primárias e secundárias estão alocadas na mesma VM, por exemplo, a réplica primária BD_1 e a secundária BD_2 na VM_1. O banco de dados BD_2 já executou as transações da fila e está completamente atualizado. Os bancos de dados BD_3 e BD_{11} precisam executar as transações de *refresh* para se tornarem atualizados.

5.4 CONSISTÊNCIA PARA BANCO DE DADOS MULTI-INQUILINO

RepliC implementa um protocolo para consistência forte que permite a execução de diferentes cargas de trabalho. Quando se utiliza apenas uma réplica no grupo de atualização,

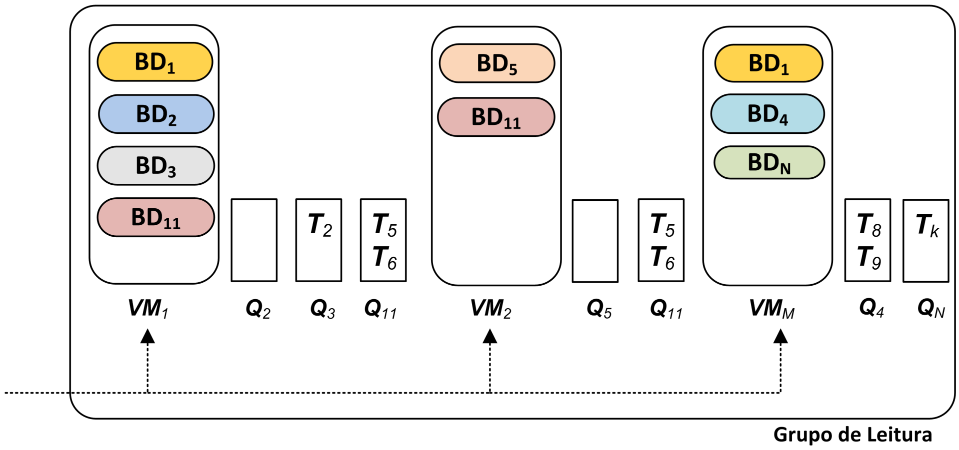

Figura 5.2 Grupo de leitura

este protocolo comporta-se de forma similar ao protocolo de cópia primária, mas com melhor desempenho, pois combina estratégias síncronas e assíncronas.

5.4.1 Algoritmos para Consistência

No RepliC, as atualizações são executadas de forma síncrona, primeiramente, na réplica primária do grupo de atualização, e, caso exista, nas demais réplicas deste grupo. Em seguida, as atualizações são enviadas de forma assíncrona para os grupos de leitura.

O Algoritmo 6 descreve as operações realizadas pela réplica primária. Esta réplica recebe continuamente as mensagens enviadas pelos clientes e pelo sistema de comunicação em grupo. Se a mensagem indica a existência de uma nova visão (linha 3), esta é prontamente instalada, substituindo a versão anterior (linha 4). Essa mensagem é proveniente do sistema de CG, enquanto as mensagens de *begin* e *commit* de transações são originadas pelos clientes da réplica primária. Caso a mensagem seja um *begin* (linha 5), a transação é iniciada no banco de dados local (linha 6), sendo executadas suas operações e os resultados retornados ao cliente (linha 7). Quando o cliente requisita o *commit* da transação (linha 8), as operações de atualização (*write set*) da transação são enviados para as outras réplicas do grupo de atualização (réplicas *secundárias*) utilizando uma primitiva de ordenação total (linha 9).

Em seguida, a réplica *primária* aguarda as confirmações do *teste de certificação* executado nas réplicas *secundárias* da visão atual (linha 10). Se ocorrerem conflitos no *teste de certificação* (linha 12), a réplica primária envia um *multicast* com a mensagem

Algoritmo 6 - Procedimento executado pela Réplica Primária

```
1: loop
2:    msg ← replica_primaria.recebe_mensagem();
3:    if msn.tipo=nova_visao then
4:       replica_primaria.instala_nova_visao;
5:    else if msn.tipo=begin then
6:       replica_primaria.begin;
7:       resultados ← replica_primaria.execute(msg.transacao, msg.operacoes);
8:    else if msn.tipo=commit then
9:       replica_primaria.multicast(grupo_atualizacao, msg.transacao, write_set);
10:      replica_primaria.espera_confirmacao_certificacao;
11:      if encontrou conflito nas certificacoes then
12:         replica_primaria.multicast(grupo_atualizacao, msg.transacao, abort);
13:      else
14:         replica_primaria.multicast(grupo_atualizacao, msg.transacao, commit);
15:      end if
16:   end if
17:   replica_primaria.commit;
18:   servico_armazenamento_nuvem(msg.operacoes);
19:   replica_primaria.multicast(grupo_leitura, msg.transacao, write_set);
20: end loop
```

de *abort* para as réplicas *secundárias* (linha 12). Caso não ocorram conflitos, é feito um *multicast* para as réplicas *secundárias* para que estas realizem o *commit* da transação nos bancos de dados locais (linha 14). Na sequência, a réplica *primária* faz o *commit* localmente (linha 17), persiste as operações no serviço de armazenamento em nuvem (linha 18) e envia o *write set* da transação as réplicas do grupo de leitura (linha 19). Posteriormente, as réplicas *slave* efetivaram essas modificações em seus bancos de dados locais.

O Algoritmo 7 descreve o comportamento das réplicas *secundárias.* Ao receber uma mensagem enviada pela réplica primária (linha 2), caso a mensagem contenha um *write set* (linha 4), é executado um *teste de certificação* (linha 4) para verificar se as atualizações conflitam com alguma transação em execução e, em caso negativo, é enviada uma mensagem de confirmação à réplica primária (linha 5). As réplicas que não responderem, por motivos de falha na réplica ou de comunicação são excluídas do grupo por meio de uma nova visão. Essas réplicas podem ser reintegradas posteriormente ao grupo. Se a mensagem recebida for um *commit* (linha 6), uma transação é iniciada no banco de dados local, e as operações do *write set* recebidas anteriormente são executadas (linha 8). Em seguida, o *commit* da transação é realizado (linha 9). Se a mensagem recebida for um *abort*, a réplica *secundária* aborta a transação (linha 11).

O Algoritmo 8 mostra o *teste de certificação* utilizado na réplica *secundária.* Esse

Algoritmo 7 - Procedimento executado pelas Réplicas Secundárias

```
loop
    msg ← replica_secundaria.recebe_mensagem();
    if msg.tipo =write_set then
        resultado ← replica_secundario.teste_de_certificao(msg.operacoes);
        return  resultado;
    else if msg =commit then
        replica_secundaria.begin;
        replica_secundaria.execute(msg.transacao, msg.write_set);
        replica_secundaria.commit;
    else if msg =abort then
        replica_secundaria.abort(msg.transacao);
    end if
end loop
```

teste verifica conflitos entre as transações, comparando seus *read sets* e *write sets* (linhas 2 a 4). Depois do *teste de certificação*, a transação é confirmada, abortando transações locais (nas réplicas *secundárias*) em execução que estão em conflito com a transação enviada pela réplica *primária*.

Algoritmo 8 - Procedimento do Teste de Certificação

```
1: for transacoes em replica_secundaria do
2:     if (operacoes.read_set= transacoes.operacoes.write_set) or
3: (operacoes.write_set= transacoes.operacoes.read_set) or
4: (operacoes.write_set= transacoes.operacoes.write_set) then
5:         return  falso;
6:     end if
7: end for
```

O Algoritmo 9 descreve o comportamento das réplicas *slaves*. Cada réplica *slave* recebe mensagens de clientes ou do grupo de atualização (linha 2). Ao receber mensagens de *begin* (linha 3), provenientes de clientes, a fila é inspecionada, verificando se existem modificações pendentes recebidas do grupo de atualização a serem efetivadas (linha 4). Em caso positivo, os itens de dados da réplica a serem atualizados são bloqueados (linha 5), ou seja, estes itens de dados não podem ser acessados por outras transações até que sejam desbloqueados, para a execução de uma transação de *refresh* (linha 6) com as atualizações presentes na fila. Este passo é denominado bloqueio de *refresh*.

Ao final da execução, os dados da réplica são desbloqueados (linha 7). Em seguida, a transação é iniciada no banco de dados local, suas operações são executadas dentro da transação iniciada (linha 10) e os resultados retornados ao cliente (linha 11). Caso a mensagem seja de atualização (linha 12), originada por uma das réplicas do grupo de atualização, o *write set* recebido é colocado no final da fila da réplica *slave* (linha 13) para, posteriormente, ser efetivado por meio de uma transação de *refresh* ou de propagação.

Nos momentos em que a réplica está ociosa (linha 15), isto é, não está executando nenhuma transação, é verificado se existem atualizações pendentes na fila (linha 16). Caso existam, os itens de dados a serem atualizados são bloqueados (linha 17), e uma transação de propagação com as alterações contidas na fila é executada (linha 18). Os itens de dados são bloqueados (linha 17) para impedir que novas transações sejam executadas antes da transação de propagação terminar, impedindo que aquelas vejam dados desatualizados. Ao final da execução, os itens de dados da réplica são desbloqueados (linha 19) e este fica disponível para executar as transações pendentes.

Algoritmo 9 - Procedimento executado pelas Réplicas Slave

```
1: loop
2:     msg ← replica_leitura.recebe_mensagem;
3:     if msg.tipo=begin then
4:         if fila.vazia()=false then
5:             replica_leitura.bloquear(msg.item_de_dados);
6:             replica_leitura.executa_transacao_refresh(fila);
7:             replica_leitura.desbloquear(msg.item_de_dados);
8:         end if
9:         replica_leitura.begin;
10:        resultado ← replica_leitura.execute(msg.operacoes);
11:        return resultados;
12:    else if msg.tipo=atualizacao then
13:        fila.mensagem.(msg.write_set);
14:    end if
15:    while replica_leitura.ocioso()=true do
16:        if fila.vazio=false then
17:            replica_leitura.bloquear(msg.item_de_dados);
18:            replica_leitura.executar(transacao_propagacao_fila);
19:            replica_leitura.desbloquear(msg.item_de_dados);
20:        end if
21:    end while
22: end loop
```

É importante ressaltar que todas as réplicas são executadas em máquinas virtuais, ou seja, os dados estão em memória principal. Na nossa estratégia, as réplicas do grupo de atualização persistem os dados por meio do serviço de armazenamento em nuvem.

5.5 TOLERÂNCIA A FALHAS

Existem diferentes tipos de falhas, dentre as quais podem ser destacadas falhas de hardware, rede, armazenamento, sistema operacional e instância do SGBD. Os provedores de

IaaS tratam a maioria destas falhas, principalmente pela utilização de hardware redundante, que manter a disponibilidade da infraestrutura em 99%. Entretanto, o desenvolvedor deve ser capaz de manter o sistema funcionando corretamente durante estas falhas da perspectiva do cliente.

O intervalo de indisponibilidade do serviço deve ser minimizado, pois isso viola a regra do SLA, resultando em uma penalidade. Por exemplo, no Google AppEngine, se a disponibilidade fica abaixo de 99,9%, em seguida, os inquilinos recebem um crédito no serviço. Da mesma forma, o tempo de resposta é fundamental para garantir a qualidade do serviço e pode incorrer em penalidades em alguns modelos de serviços [Xiong et al., 2011]. Assim, técnicas para replicação em nuvem devem gerar um impacto mínimo sobre os SLAs definidos para os inquilinos.

Em geral, estes provedores permitem apenas que as máquinas virtuais de uma determinada aplicação sejam criadas em estruturas físicas diferentes ou em diferentes zonas de disponibilidade, onde cada uma destas zonas descreve o local físico dos recursos [Bonvin et al., 2009]. Cada zona de disponibilidade é projetada para estar isolada das falhas das demais zonas de disponibilidade. Dentro de uma mesma zona de disponibilidade a latência é baixa, mas podem apresentar variações de acordo com a disposição dos recursos. Este ponto é relevante para sistemas replicados, visto que estes sistemas trocam uma quantidade elevada de mensagens.

Para melhorar a disponibilidade, cada serviço possui pelo menos duas réplicas e as réplicas de um determinado serviço não estão alocadas em uma mesma máquina, por exemplo, duas réplicas do banco de dados Wiki não são alocados na mesma VM. RepliC gerencia as réplicas dos bancos de dados de tal forma que as falhas não sejam percebidas para o cliente. RepliC implementa um agente para verificar o estado da VM e do SGBD com o objetivo de detectar falhas. As máquinas virtuais são organizadas para comportar as réplicas dos grupos de atualização e leitura. As réplicas de cada grupo são organizadas logicamente em uma topologia em anel. Com isso, as falhas podem ser detectadas pelas réplicas adjacentes.

A persistência dos dados é realizada por meio de um serviço de armazenamento em nuvem, por exemplo, Amazon S3. Assim, quando uma transação é confirmada e persistida, pode-se garantir que na presença de falhas, os dados podem ser recuperados por meio deste serviço. Para as transações de atualização, durante o processo de consolidação, o *log* e os registros consolidados são armazenados em um serviço de armazenamento distribuído para garantir a durabilidade.

Similar a [Savinov and Daudjee, 2010], se a réplica primária falhar, a transação pode ser recuperada por meio de uma arquivo de *log* que armazena as transações a serem executadas. No caso de falha em uma réplica secundária ou *slave*, novas transações não são enviadas transações para estas réplicas e as mesmas são removidas do sistema. Caso esta réplica se torne novamente operacional, um processo para verificar o estado da réplica é executado e a adição da réplica é realizada. Caso contrário, uma nova réplica secundária ou *slave* é criada. Neste trabalho, o processo de recuperação das réplicas não é abordado. Contudo, o processo de recuperação proposto por [Perez-Sorrosal et al., 2011] pode ser utilizado com pequenos ajustes.

5.6 CONCLUSÃO

Este capítulo apresentou a estratégia para consistência de banco de dados multi-inquilino utilizada pelo RepliC. Esta estratégia não possui os problemas dos protocolos tradicionais, tais como cópia primária e réplica ativa. A especificação e os algoritmos desenvolvidos também foram apresentados. Por fim, foram discutidos questões de tolerância a falhas. A abordagem para tratar falhas utilizada pelo RepliC não apresenta um ponto de falha único, melhorando a disponibilidade. O próximo capítulo apresenta os experimentos realizados e uma análise dos resultados obtidos com o uso do RepliC.

CAPÍTULO 6

AVALIAÇÃO EXPERIMENTAL

Este capítulo descreve a implementação do RepliC e a avaliação experimental realizada. São apresentadas as diferenças no processo de avaliação de serviços de dados em nuvem e, ao final, os resultados obtidos considerando diversas características de um ambiente de banco de dados replicado em nuvem.

6.1 INTRODUÇÃO

A abordagem RepliC foi desenvolvida em conjunto com nossa camada de software *Quality of Service for Database in the Cloud* (QoSDBC) [Sousa et al., 2012]. O QoSDBC implementa o modelo de qualidade descrito nesta tese e simplifica a utilização de SGBDs em nuvem. Esta camada é desenvolvida como uma PaaS e pode ser facilmente integrada às diferentes infraestruturas existentes.

6.1.1 Arquitetura do QoSDBC

A arquitetura do QoSDBC é dividida em três partes: *QoSDBCDriver*, *QoSDBCCoordinator* e *Agente*. O *QosDBCDriver* é o componente que fornece acesso ao sistema. Este *driver* substitui o *driver* JDBC específico do banco de dados, mas oferece a mesma *interface* sem a necessidade de modificações no SGBD. O *QoSDBCCoordinator* consiste de um conjunto de serviços que tratam do gerenciamento das réplicas. O *Agente* é o componente adicionado em cada VM, sendo responsável por coletar, monitorar e interagir com a VM e o SGBD, bem como verificar o estado dos itens monitorados, por exemplo, se os mesmos estão operacionais ou indisponíveis. Uma visão geral da arquitetura do QoSDBC pode ser observada na Figura 6.1.

O Serviço de Monitoramento é responsável por gerenciar as informações coletadas pelo *Agente* sobre o estado das VMs e dos SGBDs. Um catálogo armazena as informações coletadas e as restrições sobre os recursos necessários para a execução. O Serviço de SLA

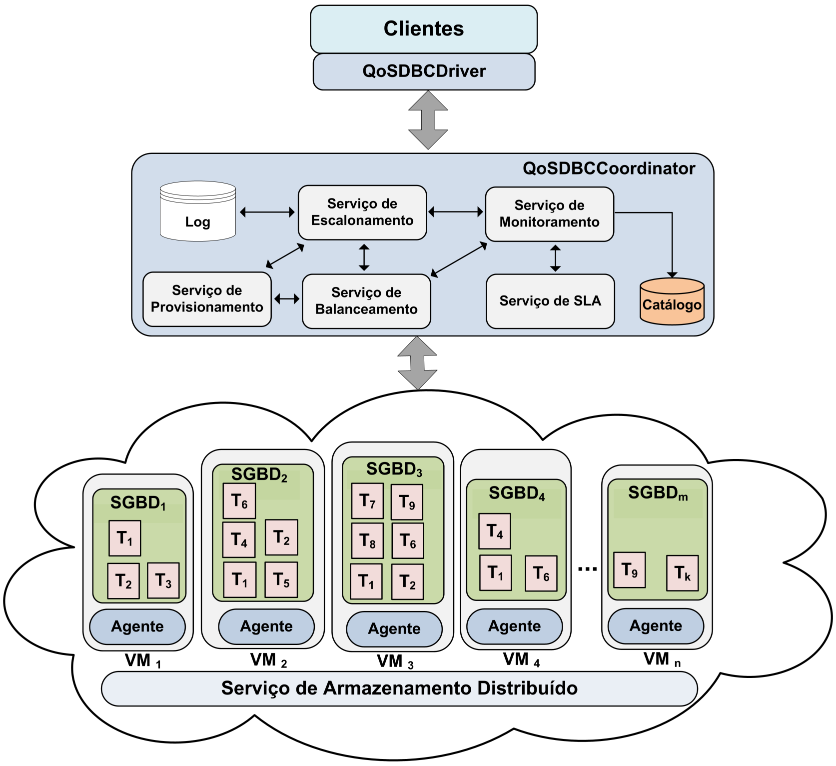

Figura 6.1 Arquitetura do QoSDBC.

gerencia o acordo de nível de serviço acertado entre o usuário e o provedor do serviço. O Serviço de Balanceamento realiza a distribuição das requisições entre as réplicas. O Serviço de Provisionamento utiliza as informações dos demais módulos para definir os recursos necessários de forma a garantir a qualidade do serviço e o gerenciamento das réplicas. Por fim, o Serviço de Escalonamento enumera e seleciona os recursos a serem utilizados para a execução das requisições e o *log* armazena as transações a serem executadas.

Serviço de Monitoramento

O Serviço de Monitoramento gerencia as informações coletadas pelo *Agente*. Este serviço monitora informações sobre cada VM e SGBD, estatísticas da carga de trabalho, consulta informações sobre os recursos necessários para os SGBDs e sobre os recursos disponíveis na infraestrutura. Essas informações são utilizadas para definir a adição ou remoção dos SGBDs das VM e provisionar recursos de forma a garantir a qualidade do serviço.

As informações são armazenadas no catálogo e continuamente atualizadas para ajustar os outros serviços. As seguintes informações são monitoradas para cada VM: estado da VM, recursos de CPU, memória e disco. Para cada SGBD são monitoradas as informações sobre os recursos de CPU, memória e tamanho das bases de dados utilizadas, assim como as métricas definidas no SLA.

Serviço de SLA

O Serviço de SLA contém informações sobre o acordo de nível de serviço acertado entre o usuário e o provedor do serviço. Estas informações estão relacionadas às definições do SLA, tais como o tempo de resposta, vazão, disponibilidade e consistência. Este serviço fornece parâmetros para o Serviço de Monitoramento verificar os valores definidos e ajustar os demais serviços.

Serviço de Balanceamento

O Serviço de Balanceamento de carga recebe as transações enviadas ao sistema e as direciona para as réplicas de acordo com uma estratégia de fila circular ou baseada na utilização de recursos. Este serviço evita que as VMs e réplicas tornem-se sobrecarregadas enquanto outras estão subutilizadas. O equilíbrio de carga é realizado direcionando as transações por meio do *QoSDBCDriver*.

Serviço de Provisionamento

O Serviço de Provisionamento utiliza informações dos outros serviços para provisionar os recursos. RepliC define como e quando uma réplica deve ser adicionada ou removida. A carga de trabalho é distribuída entre as réplicas com recursos suficientes para executá-las de forma que o SLA seja satisfeito. Este serviço também decide como os recursos devem ser compartilhados entre os vários inquilinos. A infraestrutura em nuvem permite um rápido provisionamento de recursos por meio de técnicas de virtualização, o que melhora o impacto de cargas de trabalho com grandes variações.

Serviço de Escalonamento

Este serviço enumera e seleciona os recursos provisionados. Para tanto, o escalonador gerencia as réplicas, garantindo o acesso ao SGBD durante e após o processo de replicação. Este processo consiste em gerenciar as transações e direcioná-las para os SGBDs em execução. As operações são analisadas e a transação é direcionada a uma réplica do grupo de atualização ou de leitura dependendo do seu conteúdo. Este serviço mantém uma cópia das últimas transações submetidas para auxiliar na replicação. Cada SGBD é responsável pelo processamento interno e otimização local.

6.1.2 Implementação

A implementação do RepliC baseou-se no nosso sistema anterior [Sousa et al., 2007], uma vez que este sistema possui uma solução para a replicação de banco de dados. Por decisões de projeto, optou-se por desenvolver RepliC servindo-se da linguagem Java. Características como portabilidade, reusabilidade, processamento distribuído e *multithreading* são propriedades desta linguagem que favorecem um desenvolvimento confiável. Em relação ao sistema de comunicação em grupo, que implementa as primitivas FIFO e de ordenação total, optou-se pela utilização do sistema Spread [Spread, 2012], que possui código aberto, o que facilita seu uso principalmente no ambiente acadêmico.

A solução desenvolvida foi implementada na infraestrutura da Amazon. O monitoramento dos recursos é realizado por meio de consultas utilizando a *API* do *Amazon EC2 CloudWatch*. Os dados monitorados são armazenados no sistema de banco de dados *Amazon SimpleDB*. Foi utilizada a *API* Java fornecida pela *Amazon EC2* para iniciar, terminar ou reconfigurar as VMs. Uma *Amazon Machine Image* (AMI) contendo o SGBD e o agente foi criada, permitindo iniciar uma nova VM rapidamente. Para armazenar e compartilhar os *snapshots* do banco de dados foi utilizado o *Amazon Elastic Block Store* (EBS).

Para o processo de migração de dados, foi utilizada a ferramenta de *backup* Xtrabackup [Percona, 2012], que permite *backup* "quente" sem interromper o processamento de transações [Barker et al., 2012]. RepliC aproveita a função de *backup* "quente" para obter uma cópia consistente do banco de dados e para iniciar uma nova réplica. Foi utilizada a ferramenta Linux *pv*, que permite limitar a quantidade de dados enviados por meio do *pipe* Unix, para evitar a interferência entre o SGBD e o processo de *backup/restore*. Esta abordagem limita efetivamente o uso de recursos para a CPU e I/O.

Em relação às definições do QoS, existem algumas linguagens para definir SLA, principalmente para serviços web, tais como WSLA, WSOL ou SLAang [Schnjakin et al., 2010]. Estas linguagens permitem o processamento e automação de tal forma a expressar as necessidades dos clientes e informações dos provedores. O RepliC utiliza uma versão simplificada da linguagem WSLA [Keller and Ludwig, 2003] para lidar com a gestão dos SLAs. A linguagem WSLA consiste de uma linguagem extensível baseado em XML Schema e permite a definição de SLAs individuais e personalizados. Mais detalhes sobre a implementação podem sem obtidos em [Sousa et al., 2007] [QoSDBC, 2012].

6.2 AVALIAÇÃO

A avaliação de SGBDs em nuvem apresenta diferenças significativas em relação à avaliação dos SGBDs executados em infraestruturas convencionais. Estes SGBDs pressupõem a existência de configurações fixas de recursos, tratam exclusivamente da otimização de desempenho e tem como objetivo minimizar 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 os 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 qualidade para cada requisição e, com isso, o custo operacional é um parâmetro importante [Florescu and Kossmann, 2009].

Existem muitos *benchmarks* para SGBDs relacionais executados em infraestruturas tradicionais, tais como os *benckmarks* do tipo TPC [TPC, 2012], que são amplamente utilizados para avaliar o desempenho de diferentes SGBDs. SGBDs em nuvem apresentam características que não fazem parte dos parâmetros avaliados pelos *benchmarks* TPC, tais como a elasticidade. Estes sistemas são elásticos e, sendo assim, têm a capacidade de melhorar a sua configuração durante a execução da carga de trabalho. Além disso, os *benchmarks* TPC consideram que o sistema a ser avaliado é baseado em transações com as propriedades ACID, que não são utilizadas em muitos sistemas em nuvem. Estes pontos indicam que os atuais *benchmarks* padrão para os SGBDs devem ser repensados [Florescu and Kossmann, 2009].

O problema de padronização de *benchmarks* para avaliar SGBDs em nuvem é um desafio. Isso se deve principalmente à diversidade de SGBDs em nuvem e a forma como estes sistemas são construídos, visto que as implementações variam em termos de modelo de dados, níveis de consistência, expressividade da linguagem de acesso aos dados, entre outros [Sousa et al., 2010]. Devido à elasticidade e ao tamanho destes sistemas, que podem possuir centenas ou milhares de máquinas, alguns problemas podem não ocorrer até que um determinado cenário seja alcançado. Com isso, o desenvolvimento de um *benchmark* que aborde todos os possíveis casos de uso do sistema é muito difícil e inviável.

[Florescu and Kossmann, 2009] destacam que sistemas de *benchmarks* para o 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 por que *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 o *Yahoo! Cloud Serving Benchmark* (YCSB), *framewok* com o objetivo de facilitar a avaliação de diferentes SGBDs chave/valor em nuvem. Este *framewok* trata características de desempenho e escalabilidade, mas ainda não aborda aspectos de disponibilidade e replicação. O trabalho [Folkerts et al., 2012] discute requisitos, tais como elasticidade e variabilidade da carga de trabalho que devem ser tratados na concepção de um novo *benchmark* para nuvem.

[Huppler, 2011] discute que os *benchmarks* TPC não refletem os aspectos atuais de custo da indústria e propõe sugestões sobre custo, desempenho e disponibilidade que devem ser contemplados pelos novos *benchmarks*. Em [Sethuraman and Taheri, 2011] é proposto o TPC-V, um *benchmark* para avaliar o desempenho de SGBDs em ambientes virtualizados. [Kossmann et al., 2010] apresentam uma extensão do *benchmark* TPC-W com métricas que contemplam diferentes níveis de consistência, escalabilidade, custo, carga de trabalho dinâmica e tolerância a falhas. Essas novas métricas permitem medir aspectos dinâmicos dos SGBDs em nuvem.

Para ambientes multi-inquilino, é essencial utilizar um *benchmark* para avaliar SGBDs multi-inquilino. De acordo com [Hui et al., 2009], não existe um *benchmark* padrão para esta tarefa e eles propõem uma estratégia para gerar um esquema de banco de dados TPC-C extensível. Contudo, esta estratégia foca no modelo multi-inquilino de tabela e não permite trabalhar com outros modelos. [Ahmad and Bowman, 2011] [Kiefer et al., 2012] utilizam múltiplas instâncias dos *benchmarks* TPC-C e TPC-H para simular um ambiente multi-inquilino. Essas estratégias não modelam um ambiente multi-inquilino completo, pois utilizam uma única aplicação.

6.2.1 Benchmark

Para fornecer um ambiente de banco de dados multi-inquilino completo para a avaliação do RepliC, este trabalho utilizou uma nova estratégia com diferentes bancos de dados. Essa estratégia modela um ambiente similar a um cenário real de nuvem. Estes bancos são fornecidos pelo *framework* OLTPBenchmark [Curino et al., 2012]. O OLTPBenchmark é um *framework* para avaliar o desempenho de diferentes SGBDs relacionais diante de configurações de cargas de trabalho OLTP. O *framework* possui diversos *benchmarks* que representam diferentes aplicações, tais como TPC-C, Twitter, YCSB, AuctionMark, Wikipedia.

O OLTPBenchmark possibilita configurar a taxa de transações, definir o percentual de cada tipo de transação por tempo de experimento, obter informações sobre vazão, média do tempo de resposta e informações sobre utilização dos recursos de SGBD e do sistema operacional. Este *framework* utiliza o padrão JDBC para a conexão com o SGBD.

Ambiente de Avaliação

A infraestrutura da Amazon foi utilizada neste trabalho. Esta infraestrutura oferece VMs nas quais pode-se instalar e executar os SGBDs. Assim, uma imagem de VM com um SGBD pré-instalado e pré-configurado está disponível para a implantação de forma simples e rápida [Aboulnaga et al., 2009]. As VMs são diferenciadas pelos recursos que elas oferecem, tais como CPU, memória e espaço em disco.

Neste trabalho utilizou-se apenas instâncias de VM do tipo *small* na mesma zona de disponibilidade (us-west-1c). Cada instância possui um processador Xeon 2.4 GHz, 1.7 GB de memória, 160 GB de disco e utiliza o sistema operacional Ubuntu 12.04, SGBD MySQL 5.5 com *engine* InnoDB e 128 MB de *buffer*.

As IaaS fornecem serviços de armazenamento e transferência de dados, por exemplo o *Amazon EBS*, normalmente com custos adicionais referentes a quantidade de dados armazenados e de requisições de E/S. Estes custos dos serviços de armazenamento são relativamente menores do que os custos associados às VMs e alguns provedores não cobram por estes serviços. RepliC utiliza os recursos de forma eficiente por meio da adição e remoção de réplicas e do compartilhamento de recursos, o que auxilia na redução dos custos operacionais e reflete nos custos monetários. Entretanto, neste trabalho, não são considerados os custos monetários, visto que cada provedor utiliza uma estratégia diferente de tarifação (apenas os valores relativos às máquinas virtuais são contabilizados).

Os SGBDs persistem os dados, que consistem de *logs* e *snapshots*, em um sistema de armazenamento distribuído ou em um *Network-Attached Storage* (NAS) [Das et al., 2011]. Por exemplo, um NAS oferece um armazenamento escalável, altamente disponível e tolerante à falhas para persistir os dados dos clientes. Esta arquitetura é diferente de sistemas de disco compartilhado que utiliza um controle entre operações concorrentes [Bernstein and Newcomer, 2009]. Esta dissociação da propriedade de armazenamento auxilia na migração dos dados entre as réplicas e é utilizada por sistemas como o *Amazon EBS*.

Para estes experimentos, utilizou-se uma versão simplicada do modelo de qualidade proposto neste trabalho. Esta versão considerou a métrica de tempo de reposta. Foram definidos três serviços, onde cada um destes possui um banco de dados (réplicas são adicionadas e removidas de acordo com a qualidade) e um SLA (tempo de resposta) associado a este serviço. De acordo com o OLTPBenchmark, os seguintes bancos de dados foram gerados: AuctionMark (450 MB), Wikipedia (600 MB) e YCSB (600 MB). Foram usadas todas as transações com os pesos padrões definidos pelo OLTPBenchmark. Os seguintes valores de tempo de resposta foram definidos para cada serviço: AM_{SLA} = 1s, $Wiki_{SLA}$ = 0.1s , $YCSB_{SLA}$ = 0.2s e percentil de 95% em relação ao tempo de resposta. Experimentos utilizando a versão completa do modelo de qualidade proposta neste trabalho podem ser encontrados em [Sousa et al., 2011b].

6.2.2 Experimentos

Elaborar e construir experimentos que representam aplicações reais é complexo, assim como comparar diferentes abordagens para o gerenciamento de dados em nuvem. A comparação direta com os trabalhos relacionados é inviável de ser realizada, visto que cada trabalho elenca um conjunto de pressupostos próprios. Dessa forma, não foram realizados experimentos diretos comparando RepliC e os trabalhos relacionados.

Dessa forma, a avaliação deste trabalho buscou analisar as diversas características contempladas pelo RepliC. Cada experimento explora um aspecto diferente de um SGBD replicado em nuvem, tais como elasticidade e qualidade do serviço, incluindo questões de penalidades. Estes experimentos foram baseados na avaliação dos trabalhos propostos por [Cecchet et al., 2011], [Xiong et al., 2011] e [Bonvin et al., 2011]. A variação da taxa de transações foi utilizada para avaliar a elasticidade e qualidade dos serviços considerados.

RepliC foi comparado com uma estratégia de provisionamento reativo similar a um sistema de auto escalonamento, por exemplo, o *Amazon Auto Scaling*. Este tipo de sistema utiliza uma abordagem orientada a recursos e seu comportamento segue um conjunto de regras pré-definidas. Uma regra neste sistema é formado por um par de informações contendo um antecedente e um consequente. Neste trabalho, a seguinte regra foi definida: uma VM é adicionada quando a média de utilização da CPU excede 80% por um período contínuo de 2 minutos; uma VM é removida quando a utilização da CPU é menor do que 20% pelo mesmo período [Islam et al., 2012].

A estratégia de provisionamento reativo implementa o protocolo de cópia primária. Para RepliC e para esta estratégia, inicialmente, duas réplicas foram utilizadas, cada uma destas em um dos grupos. Contudo, para atender um SLA mais restrito, uma quantidade maior de réplicas pode ser inicializada. Na configuração de provisionamento reativo, todas as réplicas primárias de cada serviço foram alocadas em uma mesma VM, visto que seria complexo adaptar o protocolo de cópia primária para trabalhar com réplicas primárias em diferentes VMs. Na configuração do RepliC, a alocação foi a seguinte: VM_1=$\{AM_{Primario}$, $Wiki_{Slave}$, $YCSB_{Slave}\}$ e VM_2=$\{AM_{Slave}$, $Wiki_{Primario}$, $YCSB_{Primario}\}$.

6.2.3 Resultados

Qualidade de serviço

No primeiro experimento para avaliar a qualidade, a taxa de transações do serviço YCSB foi incrementada e a taxa dos serviços AM e Wiki foram fixadas durante toda a avaliação. Este experimento consistiu em aumentar a taxa de transações a cada 20 minutos. O intervalo para aumentar a taxa é similar ao trabalho proposto por [Cecchet et al., 2011]. Para os serviços AuctionMark (AM) e Wiki, a taxa foi fixada em 1000. Todos os serviços foram executados com 100 clientes/terminais.

A Figura 6.2 (a) mostra a variação da métrica de tempo de resposta do SLA com o provisionamento reativo/cópia primária quando a taxa do YCSB foi incrementada. Pode-se observar que o tempo de resposta do YCSB aumentou quando a taxa foi aumentada. A estratégia reativa demora um longo período para identificar o aumento do tempo de resposta. Isso ocorre porque esta estratégia é orientada a recursos e o limite para tomar a decisão para adicionar uma nova réplica não foi atingido. Somente quando a taxa atingiu o valor de 3000, uma nova réplica *slave* do YCSB foi adicionada ao sistema.

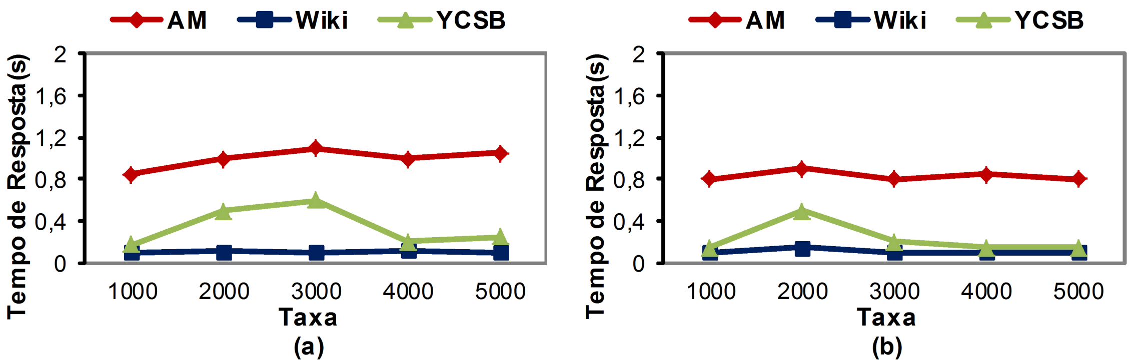

Figura 6.2 Aumento da taxa do serviço YCSB - (a) Tempo de resposta com o Provisionamento Reativo - (b) Tempo de resposta com RepliC

A Figura 6.2(b) apresenta o resultado com RepliC. Com o aumento da taxa do YCSB, a estratégia de monitoramento do RepliC identificou a variação do tempo de resposta decorrente do aumento da taxa de transações e uma nova VM com uma réplica primária do YCSB é adicionada. Neste experimento, não ocorreu interferência entre o YCSB e os serviços.

A Tabela 6.1 mostra a quantidade de VMs e a alocação das réplicas para o primeiro experimento. As réplicas primárias são destacadas, por exemplo, *A'*, *W'* e *Y'* representam réplicas primárias dos serviços AM, Wikipedia e YCSB, respectivamente. Neste experimentos, as duas estratégias utilizaram a mesma quantidade de VMs, sendo que RepliC iniciou uma nova VM primeiramente, quando a taxa alcançou o valor de 2000 e instanciou duas réplicas primárias do serviço YCSB.

No segundo experimento de qualidade, as taxas de transações de todos os serviços foram aumentadas simultaneamente. A Figura 6.3(a) mostra a variação da métrica de tempo de resposta do SLA com o provisionamento reativo quando a taxa dos serviços AM, Wiki e YCSB foram incrementados. Neste experimento, inicialmente, o tempo de resposta aumentou rapidamente para todos os serviços.

Duas VMs com réplicas *slave* de todos os bancos de dados são adicionados quando a taxa aumenta para 2000. A cópia primária utilizada pelo provisionamento reativo tem um limite para gerenciar as transações. Assim, o sistema tornou-se sobrecarregado e a

Taxa	**Provisionamento Reativo**
1000	VM_1={A',W',Y'}, VM_2={A,W,Y}
2000	VM_1={A',W',Y'}, VM_2={A,W,Y}
3000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y}
4000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y}
5000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y}
Taxa	**RepliC**
1000	VM_1={A',W',Y'}, VM_2={A,W,Y}
2000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y'}
3000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y'}
4000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y'}
5000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y'}

Tabela 6.1 Alocação das réplicas durante o primeiro experimento de QoS.

cópia primária teve problemas para processar e enviar as atualizações para as réplicas *slaves*. Em adição, ocorreram muitas interferências entre os bancos de dados executados na mesma VM, visto que esta contém todas as cópias primárias. Isso ocasiona um aumento no tempo de resposta e na violação do SLA.

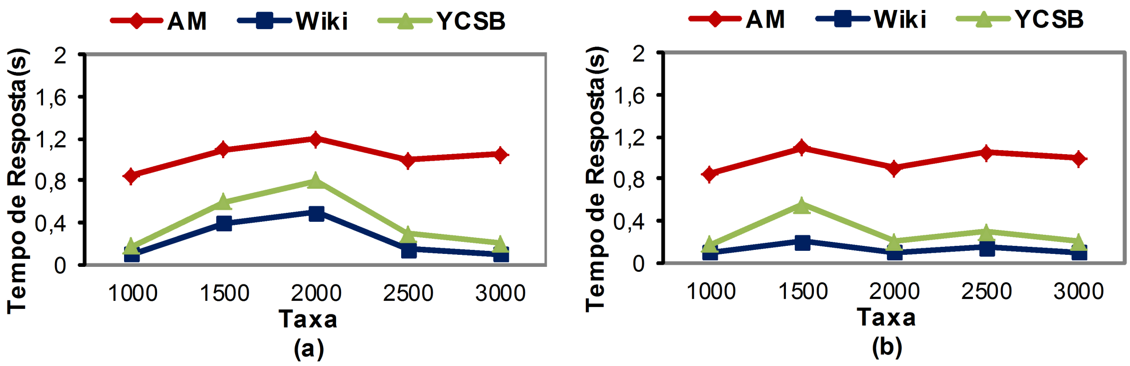

Figura 6.3 Aumento da taxa de todos os serviços - (a) Tempo de resposta com o Provisionamento Reativo - (b) Tempo de resposta com RepliC

O resultado do RepliC é apresentado na Figura 6.3(b). O tempo de reposta aumentou, mas RepliC adicionou duas VMs com réplicas primárias e *slaves* para cada serviço. Com a estratégia de monitoramento e a verificação das interferências entre as réplicas, RepliC identificou a condição em que as réplicas podem ser executadas com menor violação do SLA. RepliC melhorou o tempo de resposta distribuindo as transações de atualização entre as réplicas primárias. Em adição, a alocação eficiente das réplicas primárias em diferentes VMs foi um ponto fundamental para melhorar o desempenho do sistema. Neste experimento, RepliC armazenou as regras sobre a interferência entre os bancos de dados, principalmente entre o AM e YCSB. Essas informações podem ser utilizadas em execuções posteriores.

A Tabela 6.2 mostra a quantidade de VMs e a alocação das réplicas para o segundo experimento de QoS. Neste experimento foram utilizadas a mesma quantidade de VMs. Contudo, é importante observar que RepliC inicializou primeiramente a VM_3 quando a taxa atingiu o valor de 1500 e a réplica do YCSB na VM_4 é primária, contribuindo para a melhoria da qualidade.

Taxa	**Provisionamento Reativo**
1000	VM_1={A',W',Y'}, VM_2={A,W,Y}
1500	VM_1={A',W',Y'}, VM_2={A,W,Y}
2000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={A,W,Y}, VM_4={A,W,Y}
2500	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={A,W,Y}, VM_4={A,W,Y}
3000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={A,W,Y}, VM_4={A,W,Y}
Taxa	**RepliC**
1000	VM_1={A',W',Y'}, VM_2={A,W,Y}
1500	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={A',W,Y}, VM_4={A,W',Y'}
2000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={A',W,Y}, VM_4={A,W',Y'}
2500	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={A',W,Y}, VM_4={A,W',Y'}
3000	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={A',W,Y}, VM_4={A,W',Y'}

Tabela 6.2 Alocação das réplicas durante o segundo experimento de QoS.

Elasticidade

Para analisar a elasticidade, um experimento foi realizado variando a taxa de transações ao longo do tempo. No primeiro experimento de elasticidade, a taxa de transações do serviço YCSB foi aumentada/decrementada de acordo com a sequencia apresentada na Tabela 6.3. Os serviços AM e Wiki utilizam valores fixos com as mesmas taxas do primeiro experimento de QoS.

A Figura 6.4(a) mostra a variação da métrica de tempo de resposta do SLA para a estratégia de provisionamento reativo. Com o aumento da taxa de transações do

	Taxa
Tempo	YCSB
20	1000
40	5000
60	1000
80	1000
100	3000

Tabela 6.3 Taxa de transações para o serviço YCSB durante o primeiro experimento de elasticidade.

YCSB, uma nova VM com uma réplica *slave* do YCSB foi adicionada, que recebeu parte das transações e o tempo de resposta do YCSB diminui. Contudo, no tempo 60, com a diminuição da carga de trabalho, o limite da CPU não é atingido (isto é, 20%) e a VM não foi removida. O provisionamento reativo utiliza uma estratégia orientada a recursos e não existe uma relação linear entre os recursos alocados e o SLA. Assim sendo, esta estratégia não trabalha bem com a elasticidade para sistemas que mantém o estado, por exemplo, SGBDs.

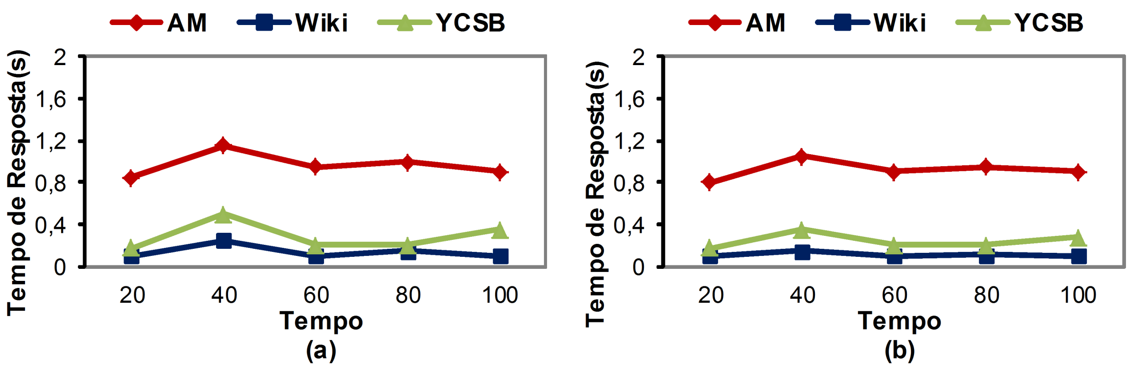

Figura 6.4 Elasticidade do serviço YCSB - (a) Tempo de resposta com o Provisionamento Reativo - (b) Tempo de resposta com RepliC

O resultado do RepliC neste experimento é apresentado na Figura 6.4(b). Com o aumento na carga do YCSB, uma nova VM com uma réplica *slave* do YCSB foi adicionada e o SLA foi garantido. Com a diminuição da carga de trabalho, a réplica foi removida, mesmo com o SGBD mantendo os recursos atuais alocados, uma vez que o tempo de resposta melhorou. Por fim, uma nova réplica *slave* do YCSB é adicionada no tempo 100, quando a carga aumentou para 3000. Isso está relacionado à estratégia de monitoramento do RepliC, que identifica a variação da carga de trabalho e do tempo de resposta. Assim, RepliC alterou a quantidade de réplicas rapidamente.

A Tabela 6.4 mostra a quantidade de VMs e a alocação das réplicas para o primeiro experimento de elasticidade. Neste experimento, RepliC utilizou uma quantidade menor de VMs, visto que uma réplica do YCSB foi adicionada no instante de tempo 40 e removida no tempo 60. Já a estratégia de provisionamento reativo mantém a quantidade de VMs constante após a adição da réplica do YCSB no tempo 40.

Tempo	**Provisionamento Reativo**
20	VM_1={A',W',Y'}, VM_2={A,W,Y}
40	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y}
60	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y}
80	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y}
100	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y}
Tempo	**RepliC**
20	VM_1={A',W',Y'}, VM_2={A,W,Y}
40	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y'}
60	VM_1={A',W',Y'}, VM_2={A,W,Y}
80	VM_1={A',W',Y'}, VM_2={A,W,Y}
100	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={Y'}

Tabela 6.4 Alocação das réplicas durante o primeiro experimento de elasticidade.

No segundo experimento de elasticidade, a taxa de transações foi alterada para todos os serviços simultaneamente. A Tabela 6.5 mostra a taxa de cada serviço durante este experimento.

	Taxa		
Tempo	AM	Wiki	YCSB
20	1000	1000	1000
40	5000	5000	2000
60	5000	1000	1000
80	1000	5000	2000
100	1000	1000	1000

Tabela 6.5 Taxa de transações para cada serviço durante o segundo experimento de elasticidade.

A Figura 6.5(a) mostra a variação da métrica do tempo de resposta com o provisionamento reativo de acordo com a Tabela 6.5. No tempo 40, com o aumento da taxa

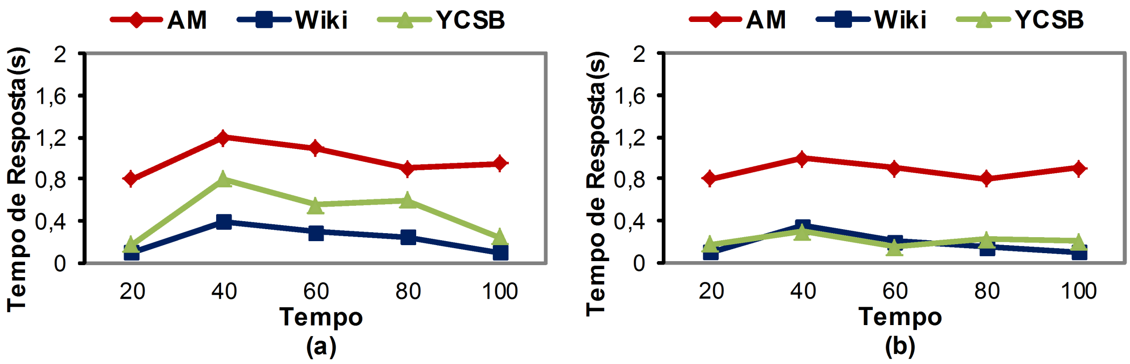

Figura 6.5 Elasticidade para todos os serviços - (a) Tempo de resposta com o Provisionamento Reativo - (b) Tempo de resposta com RepliC

dos serviços AM e Wiki, o tempo de resposta aumentou para todos os serviços. Assim, duas novas réplica *slave* do banco de dados foram adicionadas. No tempo 60, com a diminuição da carga do Wiki, o tempo de resposta continuou elevado. O tempo de resposta melhorou no final do experimento, quando a taxa de transações de todos os serviços diminuiu. Este comportamento está relacionado com a dificuldade de gerenciar o aumento e diminuição da carga de trabalho, principalmente devido à interferência entre os banco de dados executando na mesma VM e a sobrecarga da cópia primária.

A Figura 6.5(b) apresenta o resultado com RepliC. Com o aumento da carga de trabalho, RepliC adicionou duas novas VMs com réplicas primárias e *slaves* dos serviços AM e Wiki. Assim, o tempo de resposta diminuiu. No tempo 60, a carga de trabalho do Wiki diminuiu e RepliC removeu uma réplica deste serviço. No tempo 80, RepliC adicionou uma nova réplica *slave* do Wiki. RepliC reusou as regras armazenadas sobre a interferência entre o AM e o YCSB ocorrida durante o segundo experimento de QoS, reduzindo a violação do SLA.

No experimento de elasticidade, RepliC também utilizou uma quantidade menor de VMs, já que réplicas dos serviços Wiki e AM são adicionadas e removidas durante o experimento, conforme mostra a Tabela 6.6.

Violação do SLA

A Tabela 6.7 mostra a violação do SLA para os experimentos anteriores. No primeiro experimento de QoS, o provisionamento reativo apresentou 39% de violação do SLA para o serviço YCSB. RepliC garante melhor qualidade com a mesma quantidade de recursos.

Tempo	**Provisionamento Reativo**
20	VM_1={A',W',Y'}, VM_2={A,W,Y}
40	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={W, AM}, VM_4={W, AM}
60	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={W, AM}, VM_4={W, AM}
80	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={W, AM}, VM_4={W, AM}
100	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={W, AM}, VM_4={W, AM}
Tempo	**RepliC**
20	VM_1={A',W',Y'}, VM_2={A,W,Y}
40	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={A, W'}, VM_4={A',W}
60	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={A, W'}, VM_4={A'}
80	VM_1={A',W',Y'}, VM_2={A,W,Y}, VM_3={A, W'}, VM_4={A',W}
100	VM_1={A',W',Y'}, VM_2={A,W,Y}

Tabela 6.6 Alocação das réplicas durante o segundo experimento de elasticidade.

Uma razão para isso é que as transações foram distribuídas entre as réplicas dos grupos de atualização e leitura, melhorando o processamento. No segundo experimento de QoS, a estratégia reativa apresentou um resultado de desempenho inferior comparado com RepliC. Contudo, os valores de violação do RepliC aumentaram consideralmente. Isto é devido ao fato de que mais mensagens foram enviadas do grupo de atualização para o grupo de leitura. Nesses experimentos, RepliC teve menor violação do SLA do que a estratégia reativa.

	Provisionamento Reativo			RepliC		
Experimento	AM	Wiki	YCSB	AM	Wiki	YCSB
1º Experimento QoS	12%	7%	39%	1%	3%	14%
2º Experimento QoS	20%	35%	47%	14%	10%	20%
1º Experimento Elastic	14%	16%	27%	6%	5%	15%
2º Experimento Elastic	26%	25%	51%	12%	14%	16%

Tabela 6.7 Violações de SLA.

No primeiro experimento de elasticidade, RepliC apresentou menor violação SLA, especialmente para o serviço YCSB, onde ocorreu a variação na taxa de transações. Para o segundo experimento, a estratégia reativa teve um alto valor de violação para todos os serviços. Por sua vez, RepliC gerenciou melhor a variação da carga de trabalho e a interferência entre os bancos de dados.

A estratégia de sincronização do RepliC melhorou o desempenho, visto que esta estratégia atualiza inicialmente somente as réplicas do grupo de atualização e não possui apenas uma réplica primária. A alocação eficiente da réplica primária foi um ponto determinante para melhorar o tempo, pois estas réplicas foram distribuídas entre as VMs. Em adição, a abordagem RepliC reduz a interferência entre as réplicas *slave* e primária, sendo estas últimas responsáveis por enviar atualizações para as réplicas *slaves*.

Durante a execução destes experimentos, foi observada uma pequena quantidade de *aborts* (menos de 3%) no grupo de atualização, devido ao teste de certificação. A média do tempo de resposta para adicionar uma nova VM foi 55 segundos usando a AMI criada. De uma forma geral, RepliC utilizou a mesma quantidade de VMs em todos os experimentos, exceto no primeiro experimento de QoS e no segundo experimento de elasticidade. Contudo, nestes casos, RepliC utilizou apenas uma VM adicional, mas garantindo o QoS, que confirmaram que essa abordagem utilizou os recursos eficientemente.

6.3 CONCLUSÃO

Este capítulo descreveu a implementação e avaliação do RepliC. Foi justificada a escolha dos sistemas utilizados, os detalhes da implementação foram descritos, além do modelo e do ambiente de avaliação. Ao final, foram apresentados os resultados obtidos na avaliação do RepliC. Os resultados demostraram que a abordagem RepliC garante a qualidade do serviço em diferentes cenários com pequena violação do SLA. Além disso, RepliC utiliza os recursos de forma eficiente por meio da estratégia multi-inquiino utilizada. O capítulo a seguir apresenta as conclusões deste trabalho.

CAPÍTULO 7

CONCLUSÃO

Este capítulo apresenta as conclusões desta tese e uma discussão sobre direções para possíveis trabalhos futuros.

7.1 CONCLUSÕES

As principais conclusões desta tese são:

- A abordagem RepliC melhora a qualidade de serviço para banco de dados multi-inquilino em nuvem. O modelo multi-inquilino adotado permite compartilhar os recursos de forma eficiente. As estratégias de replicação, elasticidade e migração, associadas às características da infraestrutura em nuvem garantem a qualidade de serviço por meio da adição e remoção de réplicas. O processo de gerenciamento dos dados é automático, melhorando a administração das réplicas e da infraestrutura.

- O modelo de qualidade SLADB contempla os conceitos de SLA para banco de dados em nuvem, tais como tempo de resposta, vazão, disponibilidade e consistência. Este modelo é uma solução para auxiliar a qualidade de serviço, pois aborda diferentes questões, como a definição do SLA, técnicas de monitoramento e penalidades. As técnicas de monitoramento utilizadas pelo SLADB evitam que variações pontuais na carga de trabalho interfiram na qualidade de serviço.

- A arquitetura QoSDBC é modular e flexível, o que facilita sua integração a qualquer infraestrutura, uma vez que todas as informações necessárias para sua utilização são extraídas dos SGBDs e das infraestruturas por meio de agentes de monitoramento. Além disso, QoSDBC pode ser estendida, adicionando novas características.

- A estratégia para consistência dos dados baseada em uma extensão do nosso trabalho anterior para a replicação de dados XML implementa consistência forte e não possui os problemas dos protocolos tradicionais, tais como cópia primária e réplica ativa. As extensões introduzidas permitiram sua execução em ambientes virtualizados com sistemas de banco de dados multi-inquilinos, melhorando o desempenho e a disponibilidade.

- O método de avaliação de serviço de banco de dados multi-inquilino proposto nesta tese modela um ambiente similar a um cenário real de nuvem. Para isso, foi utilizado o *OLTPBenchmark*, que permite construir um ambiente de banco de dados multi-inquilino e variar a carga de trabalho em função do tempo para implementar a elasticidade.

- Os resultados dos experimentos realizados confirmaram que RepliC garantiu a qualidade com uma pequena quantidade de violação do SLA, enquanto utiliza os recursos de forma eficiente, mesmo em cenários com diferentes variações da carga de trabalho. Estes resultados comprovam a nossa hipótese de que replicação elástica de banco de dados multi-inquilino melhora a qualidade de serviço e utilização de recursos em ambientes de nuvem.

7.2 TRABALHOS FUTUROS

Os trabalhos propostos como atividades a serem feitas posteriormente para dar continuidade a este trabalho são os seguintes:

1. *Diferentes Níveis de Consistência*

RepliC foi desenvolvido para trabalhar com consistência forte. Por outro lado, existe uma grande quantidade de aplicações que não necessita de consistência forte. De acordo com [Kraska et al., 2009], consistência e custo estão relacionados e devem ser considerados no desenvolvimento de soluções em nuvem. Consistência forte implica em alto custo por transação e, em algumas situações, reduz a disponibilidade. Consistência fraca apresenta menor custo.

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. Para garantir a disponibilidade e a tolerância à falhas, os serviços de dados em nuvem devem fornecem diferentes garantias de consistência.

Devido à diversidade de aplicações, pretende-se estender RepliC para implementar diferentes níveis, tais como consistência fraca ou consistência eventual. De acordo com os requisitos da aplicação em relação à consistência, um nível de consistência para o banco de dados desta aplicação poderia ser utilizado, adequando o custo à consistência utilizada. Outro direcionamento seria adaptar novos protocolos de replicação e consistência, tais como *Paxos* e *Gossip* ao RepliC, melhorando ainda mais o desempenho e disponibilidade.

2. *Novos Modelos Multi-Inquilinos*

RepliC utiliza o modelo multi-inquilino de instância, que permite um bom compartilhamento de recursos. Por outro lado, este modelo apresenta interferências entre os inquilinos. Nosso estudo [Moreira et al., 2012] apresenta uma análise de um conjunto de experimentos para medir a variação do desempenho de SGBDs multi-inquilino em nuvem. Esta análise apresenta em detalhes as interferências e as suas possíveis causas, tais como a carga de trabalho e o tipo de SGBD utilizado.

Neste contexto, as soluções em nuvem devem alocar inquilinos com pouca interferência entre si. Para tanto, é necessária uma análise do perfil do inquilino para identificar o nível de interferência. Também é importante isolar inquilinos suscetíveis à interferência. Neste caso, outros modelos multi-inquilino podem ser utilizados, por exemplo, SGBD privado para inquilinos suscetíveis à interferência. Além disso, para diminuir tais interferências, é necessário identificar o nível de interação entre os inquilinos e, consequentemente, melhorar a alocação dos inquilinos de acordo com suas interferências.

Em trabalhos futuros, pretende-se realizar experimentos com outros modelos multi-inquilinos não contemplados neste trabalho, assim como verificar o nível de interferências em outros SGBDs e os recursos utilizados por cada inquilino. Outro aspecto importante é desenvolver técnicas para migrar inquilinos com muita interferência, melhorando o desempenho do sistema. Pretende-se também realizar um estudo para elaborar uma estratégia de alocação para inquilinos na nuvem, visando diminuir as interferências.

3. *Novos Experimentos de QoS*

Os experimentos realizados neste trabalho abordaram aspectos específicos de elasticidade e QoS do sistema avaliado. Pretende-se realizar experimentos com cargas de trabalho maiores e com variações em um menor intervalo de tempo. Tais experimentos permitirão avaliar o comportamento do RepliC nestes cenários e, se for o caso, aprimorar seus algoritmos. Em relação a experimentos para avaliar a disponibilidade, pretende-se desenvolver experimentos para validar a dependabilidade total do RepliC, englobando também a confiabilidade. Para tanto, é necessário desenvolver experimentos detalhados com injeção de falhas.

Também é proposta a avaliação do RepliC em diferentes zonas de disponibilidade com o intuito de identificar a variação no desempenho adicionado em decorrência da latência da rede. Para tanto, são necessários identificar parâmetros e desenvolver cenários que contemplem um ambiente mais geral do que o utilizado nos experimentos aqui apresentados.

RepliC foi avaliado com a infraestrutura da Amazon. Entretanto, ele pode ser utilizado por qualquer infraestrutura. Em trabalhos futuros, poder-se-ia avaliá-lo com outras infraestruturas, tais como OpenNebula, OpenStack e CloudStack, verificando a melhoria proporcionada ao serviços executados em cada uma dessas infraestruturas.

4. *Gerenciamento Autônomo*

A nuvem é um sistema autônomo onde hardware e software podem ser automaticamente reconfigurados, orquestrados e estas modificações são apresentadas de forma transparente para os usuários. Com isso, tem-se um ambiente dinâmico, elástico e distribuído, o que dificulta o desenvolvimento de soluções para o gerenciamento destes ambientes [Sousa et al., 2011a].

Um sistema autônomo para o gerenciamento de dados em nuvem deve monitorar o comportamento e desempenho do ambiente, tratar questões de tolerância a falhas, elasticidade e balanceamento da carga de trabalho, modelar e prever o comportamento para as cargas de trabalho e realizar ações para lidar com as variações destas cargas. Baseados em uma estimativa de carga de trabalho esperada, os clientes/desenvolvedores podem monitorar e, consequentemente, ajustar a capacidade do sistema.

O gerenciamento autônomo pode permitir ao RepliC acompanhar a execução e aplicar soluções a eventos, por exemplo, alterações na carga de trabalho ou falhas, assim como garantir a qualidade do serviço e evitar desperdício de recursos. Para tanto, técnicas de aprendizagem de máquina podem ser utilizadas para classificar a carga de trabalho e prever o custo de operações, melhorando o provisionamento [Rogers et al., 2010] [Agrawal et al., 2011a].

5. *Processo de Tomada de Decisão para Adicão e Remoção de Réplicas*

RepliC utiliza uma heurística baseada no tempo de resposta para determinar a adição e remoção de réplica. Apesar desta heurística apresentar resultados satisfatórios, ela não combina as diferentes informações do ambiente para o processo de decisão, tais como carga de trabalho, quantidade de réplicas e interferência entre inquilinos.

De acordo com uma investigação inicial, percebeu-se que o problema abordado pelo RepliC pode ser modelado como um problema de decisão. RepliC monitora o estado do ambiente e executa ações baseadas na qualidade do serviço e nas penalidades. Assim sendo, um direcionamento para trabalhos futuros é utilizar o *Markov Decision Process* (MDP) para adicionar e remover máquinas e inquilinos do ambiente.

6. *Modelo de Custo para SGBDs em Nuvem*

Os provedores de serviços em nuvem devem tratar aspectos de custo baseados na utilização dos SGBDs. As IaaS fornecem serviços de armazenamento e transferência de dados, normalmente com custos adicionais referentes a quantidade de dados armazenados e de requisições de E/S. No contexto deste trabalho, estes custos dos serviços de armazenamento foram relativamente menores do que os custos associados às VM. Além disso, cada provedor utiliza uma estratégia diferente de tarifação. Dessa forma, este trabalho considerou apenas os custos referentes às máquinas virtuais.

Trabalhos recentes tem propostos modelos de custo abordando diversos aspectos do gerenciamento de dados, tais como armazenamento, processamento e E/S aos dados [Viana et al., 2011] [Floratou et al., 2012] [Upadhyaya et al., 2012] [Nguyen et al., 2012]. Assim, outro direcionamento consiste em utilizar ou estender estes modelos de forma a permitir ao RepliC contabilizar todos os custos associados ao SGBD e, consequentemente, fornecer uma solução com contabilização total dos custos para os usuários.

REFERÊNCIAS BIBLIOGRÁFICAS

[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., 2011a] Agrawal, D., Abbadi, A. E., Das, S., and Elmore, A. J. (2011a). Database scalability, elasticity, and autonomy in the cloud - (extended abstract). In *Database Systems for Advanced Applications - 16th International Conference, DASFAA 2011*, pages 2–15.

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

[Agrawal et al., 2011b] Agrawal, D., Das, S., and Abbadi, A. E. (2011b). Big data and cloud computing: Current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT '11, New York, NY, USA. ACM.

[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.

[Ahmad and Bowman, 2011] Ahmad, M. and Bowman, I. T. (2011). Predicting system performance for multi-tenant database workloads. In *Proceedings of the Fourth International Workshop on Testing Database Systems*, DBTest '11, pages 6:1–6:6, New York, NY, USA. ACM.

[Amazon, 2012] Amazon (2012). *Amazon Relational Database Service (Amazon RDS)*. `http://aws.amazon.com/rds/`.

[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.

[Arnaut et al., 2011] Arnaut, D. E. M., Schroeder, R., and Hara, C. S. (2011). Phoenix: A relational storage component for the cloud. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, pages 684–691, Washington, DC, USA. IEEE Computer Society.

[Azure, 2012] Azure (2012). *Microsoft Azure*. `http://www.microsoft.com/azure/`.

[Baker et al., 2011] Baker, J., Bond, C., Corbett, J. C., Furman, J., Khorlin, A., Larson, J., Léon, J. M., Li, Y., Lloyd, A., and Yushprakh, V. (2011). Megastore: Providing scalable, highly available storage for interactive services. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, Online Proceedings*, pages 223–234.

[Barker et al., 2012] Barker, S., Chi, Y., Moon, H. J., Hacigümüş, H., and Shenoy, P. (2012). "cut me some slack": latency-aware live migration for databases. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 432–443, New York, NY, USA. ACM.

[Bernstein and Newcomer, 2009] Bernstein, P. and Newcomer, E. (2009). *Principles of transaction processing: for the systems professional.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, second edition.

[Bernstein et al., 2011] Bernstein, P. A., Cseri, I., Dani, N., Ellis, N., Kalhan, A., Kakivaya, G., Lomet, D. B., Manne, R., Novik, L., and Talius, T. (2011). Adapting microsoft sql server for cloud computing. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011*, pages 1255–1263. IEEE Computer Society.

[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.

[Birman, 2012] Birman, K. P. (2012). *Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services.* Springer.

[Bonvin et al., 2009] Bonvin, N., Papaioannou, T. G., and Aberer, K. (2009). Dynamic cost-efficient replication in data clouds. In *ACDC '09: Proceedings of the 1st workshop on Automated control for datacenters and clouds*, pages 49–56, New York, NY, USA. ACM.

[Bonvin et al., 2011] Bonvin, N., Papaioannou, T. G., and Aberer, K. (2011). Autonomic sla-driven provisioning for cloud applications. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 434–443, Washington, DC, USA. IEEE Computer Society.

[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.

[Cao et al., 2011] Cao, Y., Chen, C., Guo, F., Jiang, D., Lin, Y., Ooi, B. C., Vo, H. T., Wu, S., and Xu, Q. (2011). Es^2: A cloud data storage system for supporting both oltp and olap. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011*, pages 291–302. IEEE Computer Society.

[Cecchet et al., 2011] Cecchet, E., Singh, R., Sharma, U., and Shenoy, P. (2011). Dolly: virtualization-driven database provisioning for the cloud. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, pages 51–62, New York, NY, USA. ACM.

[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.

[Charron-Bost et al., 2010] Charron-Bost, B., Pedone, F., and Schiper, A., editors (2010). *Replication: Theory and Practice*, volume 5959 of *Lecture Notes in Computer Science*. Springer.

[Chi et al., 2011] Chi, Y., Moon, H. J., Hacigümüş, H., and Tatemura, J. (2011). Sla-tree: a framework for efficiently supporting sla-based decisions in cloud computing. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 129–140, New York, NY, USA. ACM.

[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., 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., 2011a] Curino, C., Jones, E., Popa, R., Malviya, N., Wu, E., Madden, S., Balakrishnan, H., and Zeldovich, N. (2011a). Relational cloud: a database service for the cloud. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, Online Proceedings*, pages 235–240.

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

[Curino et al., 2011b] Curino, C., Jones, E. P., Madden, S., and Balakrishnan, H. (2011b). Workload-aware database monitoring and consolidation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 313–324, New York, NY, USA. ACM.

[Curino et al., 2010b] Curino, C., Zhang, Y., Jones, E. P. C., and Madden, S. (2010b). Schism: a workload-driven approach to database replication and partitioning. *PVLDB*, 3(1):48–57.

[Curino et al., 2012] Curino, C. A., Difallah, D. E., Pavlo, A., and Cudre-Mauroux, P. (2012). Benchmarking oltp/web databases in the cloud: the oltp-bench framework. In *Proceedings of the Fourth International Workshop on Cloud Data Management (CloudDB '12)*, pages 17–20, New York, NY, USA. ACM.

[Das et al., 2010] Das, S., Agrawal, D., and El Abbadi, A. (2010). G-store: a scalable data store for transactional multi key access in the cloud. In *SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing*, pages 163–174, New York, NY, USA. ACM.

[Das et al., 2011] Das, S., Nishimura, S., Agrawal, D., and El Abbadi, A. (2011). Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration. *Proc. VLDB Endow.*, 4:494–505.

[Dash et al., 2009] Dash, D., Kantere, V., and Ailamaki, A. (2009). An economic model for self-tuned cloud caching. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 1687–1693, Washington, DC, USA. IEEE Computer Society.

[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.

[Doherty and Hurley, 2007] Doherty, C. and Hurley, N. (2007). Autonomic distributed data management with update accesses. In *Proceedings of the 1st international conference on Autonomic computing and communication systems*, Autonomics '07, pages 10:1–10:8. ICST.

[Elmore et al., 2011a] Elmore, A., Das, S., Agrawal, D., and Abbadi, A. E. (2011a). Towards an elastic and autonomic multitenant database. In *NetDB 2011 - 6th International Workshop on Networking Meets Databases Co-located with SIGMOD 2011.*

[Elmore et al., 2011b] Elmore, A. J., Das, S., Agrawal, D., and El Abbadi, A. (2011b). Zephyr: live migration in shared nothing databases for elastic cloud platforms. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 301–312, New York, NY, USA. ACM.

[Entrialgo et al., 2011] Entrialgo, J., García, D. F., García, J., García, M., Valledor, P., and Obaidat, M. S. (2011). Dynamic adaptation of response-time models for qos management in autonomic systems. *J. Syst. Softw.*, 84:810–820.

[Ferretti et al., 2010] Ferretti, S., Ghini, V., Panzieri, F., Pellegrini, M., and Turrini, E. (2010). Qos-aware clouds. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 321–328, Washington, DC, USA. IEEE Computer Society.

[Fito et al., 2010] Fito, J. O., Presa, I. G., and Guitart, J. (2010). Sla-driven elastic cloud hosting provider. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on*, 0:111–118.

[Floratou et al., 2012] Floratou, A., Patel, J. M., Lang, W., and Halverson, A. (2012). When free is not really free: what does it cost to run a database workload in the cloud? In *Proceedings of the Third TPC Technology conference on Topics in Performance Evaluation, Measurement and Characterization*, TPCTC'11, pages 163–179, Berlin, Heidelberg. Springer-Verlag.

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

[Folkerts et al., 2012] Folkerts, E., Alexandrov, A., Sachs, K., Iosup, A., Markl, V., and Tosun, C. (2012). Benchmarking in the cloud: What it should, can, and cannot be. In *TPC Technology Conference, TPCTC 2012*.

[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.

[Gray et al., 1996] Gray, J., Helland, P., O'Neil, P., and Shasha, D. (1996). The dangers of replication and a solution. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 173–182, New York, NY, USA. ACM Press.

[Hadoop, 2012] Hadoop (2012). *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.

[Huppler, 2011] Huppler, K. (2011). Price and the tpc. In *Proceedings of the Second TPC technology conference on Performance evaluation, measurement and characterization of complex systems*, TPCTC'10, pages 73–84, Berlin, Heidelberg. Springer-Verlag.

[Islam et al., 2012] Islam, S., Lee, K., Fekete, A., and Liu, A. (2012). How a consumer can measure elasticity for cloud platforms. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, ICPE '12, pages 85–96, New York, NY, USA. ACM.

[Keller and Ludwig, 2003] Keller, A. and Ludwig, H. (2003). The wsla framework: Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.*, 11:57–81.

[Kemme and Alonso, 2010] Kemme, B. and Alonso, G. (2010). Database replication: a tale of research across communities. *PVLDB*, 3(1):5–12.

[Kemme et al., 2010] Kemme, B., Jiménez-Peris, R., and Patiño-Martínez, M. (2010). *Database Replication*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

[Kephart and Chess, 2003] Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

[Kiefer et al., 2012] Kiefer, T., Schlegel, B., and Lehner, W. (2012). Multe: A multi-tenancy database benchmark framework. In *TPC Technology Conference, TPCTC 2012*.

[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.

[Lang et al., 2012] Lang, W., Shankar, S., Patel, J. M., and Kalhan, A. (2012). Towards multi-tenant performance slos. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, pages 702–713, Washington, DC, USA. IEEE Computer Society.

[Lehner and Sattler, 2010] Lehner, W. and Sattler, K.-U. (2010). Database as a service (dbaas). In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010*, pages 1216–1217, Long Beach, California, USA. IEEE.

[Liang Zhao and Liu, 2012] Liang Zhao, S. S. and Liu, A. (2012). Application-managed replication controller for cloud-hosted databases. In *IEEE CLOUD 2012 - Proceedings of the 5th IEEE International Conference on Cloud Computing*, pages 922–929.

[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.

[LSCR, 2012] LSCR (2012). *SLA for database projects*. `http://lscr.berkeley.edu/rates/sla/database.php`.

[Malkowski et al., 2010] Malkowski, S., Hedwig, M., Jayasinghe, D., Pu, C., and Neumann, D. (2010). Cloudxplor: a tool for configuration planning in clouds based on empirical data. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 391–398, New York, NY, USA. ACM.

[Mell and Grance, 2011] Mell, P. and Grance, T. (2011). *NIST Working Definition of Cloud Computing (Draft)*. National Institute of Standards and Technology. `http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf`.

[Microsoft, 2006] Microsoft (2006). *Architecture Strategies for Catching the Long Tail*. `http://msdn.microsoft.com/en-us/library/aa479069.aspx`.

[Minhas et al., 2011] Minhas, U. F., Rajagopalan, S., Cully, B., Aboulnaga, A., Salem, K., and Warfield, A. (2011). Remusdb: Transparent high availability for database systems. *PVLDB*, 4(11):738–748.

[Mior and de Lara, 2011] Mior, M. J. and de Lara, E. (2011). Flurrydb: a dynamically scalable relational database with virtual machine cloning. In *Proceedings of the 4th Annual International Conference on Systems and Storage*, SYSTOR '11, pages 1:1–1:9, New York, NY, USA. ACM.

[Moreira et al., 2012] Moreira, L. O., Sousa, F. R. C., and Machado, J. C. (2012). Analisando o desempenho de banco de dados multi-inquilino em nuvem. In *XXVII Simpósio Brasileiro de Banco de Dados, SBBD 2012*, pages 161–168.

[Mukerjee et al., 2011] Mukerjee, K., Talius, T., Kalhan, A., Ellis, N., and Cunningham, C. (2011). Sql azure as a self-managing database service: Lessons learned and challenges ahead. *IEEE Data Eng. Bull.*, 34(4):61–70.

[Nguyen et al., 2012] Nguyen, T.-V.-A., Bimonte, S., d'Orazio, L., and Darmont, J. (2012). Cost models for view materialization in the cloud. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, EDBT-ICDT '12, pages 47–54, New York, NY, USA. ACM.

[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.

[Özsu and Valduriez, 2011] Özsu, M. T. and Valduriez, P. (2011). *Principles of Distributed Database Systems, 3rd Edition.* Springer.

[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.

[Percona, 2012] Percona (2012). *XtraBackup.* `http://www.percona.com/software/percona-xtrabackup`.

[Perez-Sorrosal et al., 2011] Perez-Sorrosal, F., Patiño-Martinez, M., Jimenez-Peris, R., and Kemme, B. (2011). Elastic si-cache: consistent and scalable caching in multi-tier architectures. *The VLDB Journal*, pages 1–25.

[Pierre and Stratan, 2012] Pierre, G. and Stratan, C. (2012). Conpaas: A platform for hosting elastic cloud applications. *IEEE Internet Computing*, 16:88–92.

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

[QoSDBC, 2012] QoSDBC (2012). *QoSDBC.* `http://code.google.com/p/qosdbc/`.

[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.

[Saito and Shapiro, 2005] Saito, Y. and Shapiro, M. (2005). Optimistic replication. *ACM Comput. Surv.*, 37:42–81.

[Sakr and Liu, 2012] Sakr, S. and Liu, A. (2012). Sla-based and consumer-centric dynamic provisioning for cloud databases. *2012 IEEE Fifth International Conference on Cloud Computing*, 0:360–367.

[Sakr et al., 2011] Sakr, S., Zhao, L., Wada, H., and Liu, A. (2011). Clouddb autoadmin: Towards a truly elastic cloud-based data store. In *Proceedings of the 2011 IEEE International Conference on Web Services*, ICWS '11, pages 732–733, Washington, DC, USA. IEEE Computer Society.

[Santos et al., 2012] Santos, G. A. C., Maia, J. G. R., Moreira, L. O., Sousa, F. R. C., and Machado, J. C. (2012). Scale-space filtering for workload analysis and forecast. In *Submetido para Publicação.* `http://replic.sf.net/ccgrid2013.pdf`.

[Savinov and Daudjee, 2010] Savinov, S. and Daudjee, K. (2010). Dynamic database replica provisioning through virtualization. In *Proceedings of the second international workshop on Cloud data management*, CloudDB '10, pages 41–46, New York, NY, USA. ACM.

[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.

[Schiller et al., 2011] Schiller, O., Schiller, B., Brodt, A., and Mitschang, B. (2011). Native support of multi-tenancy in rdbms for software as a service. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 117–128, New York, NY, USA. ACM.

[Schnjakin et al., 2010] Schnjakin, M., Alnemr, R., and Meinel, C. (2010). Contract-based cloud architecture. In *Proceedings of the second international workshop on Cloud data management*, CloudDB '10, pages 33–40, New York, NY, USA. ACM.

[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.

[Sethuraman and Taheri, 2011] Sethuraman, P. and Taheri, H. R. (2011). Tpc-v: a benchmark for evaluating the performance of database applications in virtual environments. In *Proceedings of the Second TPC technology conference on Performance evaluation, measurement and characterization of complex systems*, TPCTC'10, pages 121–135, Berlin, Heidelberg. Springer-Verlag.

[Silva et al., 2012] Silva, T. L. C., Nascimento, M. A., Macêdo, J. A. F., Sousa, F. R. C., and Machado, J. C. (2012). Towards non-intrusive elastic query processing in the cloud. In *Proceedings of the Fourth International Workshop on Cloud Data Management (CloudDB '12)*, pages 9–16, New York, NY, USA. ACM.

[Sivasubramanian et al., 2005] Sivasubramanian, S., Alonso, G., Pierre, G., and van Steen, M. (2005). Globedb: autonomic data replication for web applications. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 33–42, New York, NY, USA. ACM.

[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.

[Soundararajan and Amza, 2006] Soundararajan, G. and Amza, C. (2006). Reactive provisioning of backend databases in shared dynamic content server clusters. *ACM Trans. Auton. Adapt. Syst.*, 1:151–188.

[Sousa et al., 2007] Sousa, F. R. C., Filho, H. J. A. C., and Machado, J. C. (2007). A new approach to replication of xml data. In *Database and Expert Systems Applications, 18th International Conference, DEXA 2007, Regensburg, Germany*, volume 4653 of *Lecture Notes in Computer Science*, pages 141–150. Springer.

[Sousa and Machado, 2011] Sousa, F. R. C. and Machado, J. C. (2011). An approach to database replication in the cloud. In *XXVI Proceedings of Brazilian Symposium on Databases, SBBD 2011.*

[Sousa and Machado, 2012] Sousa, F. R. C. and Machado, J. C. (2012). Towards elastic multitenant database replication with quality of service. In *Proceedings of the 5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '12).*

[Sousa et al., 2010] Sousa, F. R. C., Moreira, L. O., Macêdo, J. A. F., and Machado, J. C. (2010). *Gerenciamento de Dados em Nuvem: Conceitos, Sistemas e Desafios*, pages 101–130. In: PEREIRA, A. C. M.; PAPPA, G. L.; WINCKLER, M.; GOMES, R. L. (Org.). Tópicos em Sistemas Colaborativos, Interativos, Multimídia, Web e Bancos de Dados, SIWB 2010, 1. ed. SBC, Belo Horizonte.

[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.

[Sousa et al., 2011a] Sousa, F. R. C., Moreira, L. O., and Machado, J. C. (2011a). Computação em nuvem autônoma: Oportunidades e desafios. In *Proceedings of the I Workshop on Autonomic Distributed Systems, WoSiDA 2011, collocated with SBRC 2011, Campo Grande, MS.*

[Sousa et al., 2011b] Sousa, F. R. C., Moreira, L. O., and Machado, J. C. (2011b). Sladb: Acordo de nível de serviço para banco de dados em nuvem. In *XXVI Simpósio Brasileiro de Banco de Dados, SBBD 2012, São Paulo.*

[Sousa et al., 2012] Sousa, F. R. C., Moreira, L. O., Santos, G. A. C., and Machado, J. C. (2012). Quality of service for database in the cloud. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science - CLOSER '12*, pages 595–601.

[Spread, 2012] Spread (2012). *Spread.* `http://www.spread.org`.

[Tanenbaum and Steen, 2006] Tanenbaum, A. S. and Steen, M. v. (2006). *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Thusoo et al., 2010] Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sen Sarma, J., Murthy, R., and Liu, H. (2010). Data warehousing and analytics infrastructure at facebook. In *SIGMOD '10: Proceedings of the 2010 international conference on Management of data*, pages 1013–1020, New York, NY, USA. ACM.

[TPC, 2012] TPC (2012). *Transaction Processing Performance Council*. `http://www.tpc.org/`.

[Tsakalozos et al., 2011] Tsakalozos, K., Kllapi, H., Sitaridi, E., Roussopoulos, M., Paparas, D., and Delis, A. (2011). Flexible use of cloud resources through profit maximization and price discrimination. In *Proc. of the 27th IEEE Int. Conf. on Data Engineering (ICDE'11)*, pages 75–86.

[Upadhyaya et al., 2012] Upadhyaya, P., Balazinska, M., and Suciu, D. (2012). How to price shared optimizations in the cloud. *Proc. VLDB Endow.*, 5(6):562–573.

[Vaquero et al., 2011] Vaquero, L. M., Rodero-Merino, L., and Buyya, R. (2011). Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.*, 41:45–52.

[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.

[Vasić et al., 2012] Vasić, N., Novaković, D., Miučin, S., Kostić, D., and Bianchini, R. (2012). Dejavu: accelerating resource allocation in virtualized environments. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '12, pages 423–436, New York, NY, USA. ACM.

[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.

[Viana et al., 2011] Viana, V., de Oliveira, D., and Mattoso, M. (2011). Towards a cost model for scheduling scientific workflows activities in cloud environments. *Services, IEEE Congress on*, 0:216–219.

[Vo et al., 2010] Vo, H. T., Chen, C., and Ooi, B. C. (2010). Towards elastic transactional cloud storage with range query support. *PVLDB*, 3(1):506–517.

[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.

[Voicu et al., 2010] Voicu, L. C., Schuldt, H., Breitbart, Y., and Schek, H.-J. (2010). Flexible data access in a cloud based on freshness requirements. In *IEEE International Conference on Cloud Computing*, pages 180–187, Los Alamitos, CA, USA. IEEE Computer Society.

[Wada et al., 2011] Wada, H., Fekete, A., Zhao, L., Lee, K., and Liu, A. (2011). Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, Online Proceedings*, pages 134–143.

[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.

[Welsh and Culler, 2003] Welsh, M. and Culler, D. (2003). Adaptive overload control for busy internet servers. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 4–4, Berkeley, CA, USA. USENIX Association.

[Xiong et al., 2011] Xiong, P., Chi, Y., Zhu, S., Moon, H. J., Pu, C., and Hacigümüs, H. (2011). Intelligent management of virtualized resources for database systems in cloud environment. In *Proc. of the 27th IEEE Int. Conf. on Data Engineering (ICDE'11)*, pages 87–98.

[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.

[Zellag and Kemme, 2012] Zellag, K. and Kemme, B. (2012). How consistent is your cloud application? In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 6:1–6:14, New York, NY, USA. ACM.

[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.

[Zhao et al., 2012] Zhao, L., Sakr, S., Fekete, A., Wada, H., and Liu, A. (2012). Application-managed database replication on virtualized cloud environments. In *Data Management in the Cloud (DMC '12), ICDE Workshops*, pages 127–134, Los Alamitos, CA, USA. IEEE Computer Society.