UMA ESTRATÉGIA PARA BALANCEAMENTO DE CARGA EM BANCO DE DADOS REPLICADOS

Moisés Ferreira de Souza

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Geraldo Zimbrão da Silva

Rio de Janeiro

Setembro de 2011

UMA ESTRATÉGIA PARA BALANCEAMENTO DE CARGA EM BANCO DE DADOS REPLICADOS

Moisés Ferreira de Souza

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof. Geraldo Zimbrão da Silva, D. Sc.

_____

Prof. Geraldo Bonorino Xexéo, D. Sc.

_____

Prof. Leonardo Guerreiro Azevedo, D. Sc.

RIO DE JANEIRO, RJ – BRASIL

SETEMBRO DE 2011

Souza, Moisés Ferreira de

Uma Estratégia para Balanceamento de Carga em Banco de Dados Replicados / Moisés Ferreira de Souza – Rio de Janeiro: UFRJ/COPPE, 2011.

XIII, 92 p.: il.; 29,7 cm.

Orientador: Geraldo Zimbrão da Silva.

Dissertação (Mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2011.

Referências Bibliográficas: p. 88-91.

1. Banco de dados replicados. 2. Desempenho de algoritmos. 3. Balanceamento de carga. I. Silva, Geraldo Zimbrão da II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha família, minha namorada e meus amigos, pelo carinho e incentivo.

Agradecimentos

Agradeço aos meus pais, Rejane e Cláudio, pela educação, incentivo e carinho que me deram, bem como toda estrutura que me propiciaram para chegar até aqui. Tendo que muitas vezes abdicarem de realizações pessoais e oportunidades de lazer em prol da minha formação.

Agradeço a minha avó, Joana, que apesar de hoje não estar mais entre nós foi uma pessoa de fundamental importância para a construção da pessoa que hoje sou.

Agradeço a minha namorada, Lívia, por todo carinho, amor, apoio e tranqüilidade que foram de grande importância nos últimos três meses de escrita deste trabalho.

Agradeço ao meu orientador, Zimbrão, pela paciência e oportunidade. Por ter acreditado em meu potencial e ter acreditado no término deste trabalho.

Agradeço ao professor Blaschek, por toda oportunidade que me foi dada na COPPETEC. Lugar que tive a oportunidade de ter um grande crescimento profissional, onde pude conhecer pessoas importantes para que este trabalho tenha se iniciado e também concluído. Agradeço por toda confiança que me foi depositada.

Agradeço ao professor Jano pelas oportunidades que me deu desde o final da minha a graduação, passando pela minha aceitação no mestrado e culminando no incentivo para que eu continuasse o mestrado mesmo após meu ingresso na Petrobras.

Agradeço aos professores Xexéo e Leonardo Guerreiro por me concederem seu tempo e paciência, lendo esta dissertação e participando de minha banca.

Agradeço ao Luís Orleans pelo convite de trabalho, por todas as dicas e sugestões durante a elaboração desta dissertação.

Agradeço a Petrobras e em especial a equipe SIPLEX que, mesmo em uma situação delicada do projeto, compreenderam minhas ausências.

Agradeço aos amigos e colegas que fiz durante esse mestrado. Em especial ao Heraldo, Olivério e Rodrigo Mesquita. Um dos grandes benefícios que esse mestrado me propiciou foi conhecer tantas pessoas legais.

Agradeço as pessoas que são responsáveis por manter o PESC funcionando, cuidando de alunos e professores e de toda a infra-estrutura que necessitamos. Sendo assim, fica a minha lembrança ao pessoal da secretaria, do suporte, da segurança e da faxina. Obrigado pela paciência, conversas e sorrisos.

Por fim, agradeço a Deus, por toda saúde, oportunidade e espírito de luta e perseverança que me foi dado para que meus objetivos tanto pessoais quanto profissionais sempre fossem alcançados.

“A alegria está na luta, na tentativa,

no sofrimento envolvido e não na vitória propriamente dita”

Mahatma Gandhi

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UMA ESTRATÉGIA DE BALANCEAMENTO DE CARGA EM BANCO DE DADOS REPLICADOS

Moisés Ferreira de Souza

Setembro/2011

Orientador: Geraldo Zimbrão da Silva

Programa: Engenharia de Sistemas e Computação

Banco de dados replicados tem significativa importância quando se deseja disponibilidade e confiabilidade. Atualmente vem ganhando cada vez mais importância quando se tem por objetivo elasticidade na computação em nuvem. Enquanto transações de consulta são altamente beneficiadas por esta arquitetura, transações de atualização requerem um sincronismo entre as réplicas com o objetivo de manter o banco consistente e ainda performático. Na última década a combinação da comunidade de computação distribuída com a comunidade de banco de dados, através do uso de primitivas de comunicação em grupo, trouxe vários avanços nesta área. Porém um dos grandes problemas reside ainda no fato de que a execução de transações concorrentemente em réplicas distintas causam uma alta taxa de aborto no sistema. Em um cenário de sobrecarga com transações conflitantes pode-se chegar a uma situação de deterioração total do sistema.

Esta dissertação tem por objetivo apresentar um balanceador de carga orientado justamente a evitar tais conflitos. Além disso, o balanceador deve como medida de prevenção, caso o sistema receba uma sobrecarga elevada, impedir a entrada de transações a fim de manter o sistema sobre controle. O objetivo é que o sistema se comporte de maneira previsível ao longo do tempo, sem picos de saturação. Um simulador foi construído para executar os experimentos e validar a proposta deste trabalho.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A STRATEGY FOR LOAD BALANCING IN REPLICATED DATABASES

Moisés Ferreira de Souza

September/2011

Advisor: Geraldo Zimbrão da Silva

Department: Computer and System Engineering

Replicated database is significantly important to achieve availability and reliability. Today is gaining more importance when the goal is elasticity in the cloud. While Query transactions are highly benefited by this architecture, update transactions require synchronization between replicas in order to keep the database consistent and even keep performance. In the last decade the combination of distributed computing community with the community database through the use of group communication primitives, brought several advances in this area. But one major problem still lie in the fact that execution of transactions concurrently on different replicas cause a high abort rate. In a scenario of overload and high conflicting rate the system can get a situation of deterioration.

This paper aims to present a load balancer guided precisely to avoid such conflicts. In addition, the balancer should as a precaution, in the case of the system receives a high overhead, prevent the entry of transactions in order to keep the system under control. The goal is that the system behaves in a predictable manner over time, without peaks of saturation. A simulator was built to run experiments and validate the purpose of this work.

Sumário

Índice de Figuras

Índice de Tabelas

1. Introdução

Neste capítulo introdutório serão apresentados a motivação, o objetivo deste trabalho, a metodologia utilizada e os resultados e contribuições esperados. Neste capítulo também se encontra a organização do texto.

1.1 Motivação

Replicação é o processo de copiar e manter objetos de banco de dados em múltiplas bases de dados configurando, assim, uma especialização de um banco de dados distribuído. Tal replicação é, muitas vezes, necessária por razões de desempenho, confiabilidade e disponibilidade. Recentemente, replicação em banco de dados, tem adquirido uma nova dimensão devido ao papel que desempenha em alcançar elasticidade na camada de banco de dados em ambientes de computação em nuvem (KEMME & ALONSO, 2010). Enquanto as consultas são bastante beneficiadas, podendo ser executadas até em paralelo, vários problemas surgem a respeito à atualização dos dados, pois todas as cópias dos dados (nas diferentes réplicas) devem ser atualizadas de forma adequada. Um dos grandes desafios é a introdução da replicação sem afetar severamente o desempenho quando a carga de trabalho é composta de operações de atualização.

De acordo com (GRAY *et al.*, 1996) os protocolos de replicação são caracterizados usando dois parâmetros. O primeiro se refere a quando a propagação das atualizações acontecem. Neste caso há duas estratégias:

1. **Eager:** as atualizações são propagadas durante a transação. Esta abordagem garante a consistência, porém é custosa em termos de overhead de mensagens e tempo de resposta.
2. **Lazy:** as atualizações são propagadas após a efetivação. Esta abordagem é mais otimizada, porém inconsistências, fatalmente, ocorrerão.

O segundo parâmetro se refere a quem pode executar as atualizações. Neste caso, também, existe duas estratégias:

1. **Primary:** todas as atualizações são enviadas para um único servidor (cópia primária).
2. **Update-everywhere:** as atualizações podem ser enviadas para qualquer cópia.

O uso de cada estratégia está fortemente ligado ao contexto onde o banco de dados está inserido. Quanto maior a flexibilidade, maior o custo envolvido (WIESMANN & SCHIPER, 2005). O principal foco dos algoritmos está em como as atualizações vão ser propagadas em todas as réplicas, levando em consideração desempenho e a flexibilidade e consistência dos dados (FUERTES, 2011). Como será visto nesta dissertação o uso de primitivas de comunicação em grupo, herdadas da computação distribuída (AMIR & TUTU, 2004), possibilitou que protocolos de replicação eager combinados com update-everywhere se tornassem factíveis. É justamente estes algoritmos o foco deste trabalho.

Uma das grandes preocupações ao combinar estas duas estratégias reside no fato de que uma carga muito grande de operações de escrita em combinação com uma taxa elevada de conflitos pode levar a uma degradação considerável do sistema, independente do algoritmo que está sendo utilizado.

O problema está relacionado com a execução de transações conflitantes em réplicas distintas. Nestes algoritmos a execução de transações concorrentes, em servidores distintos, ocorre sem que nenhum deles tenha conhecimento dos potenciais conflitos existentes na execução concorrente. Somente em uma fase posterior é que tais conflitos serão resolvidos. Para cargas leves o problema não é tão relevante, porém para cargas pesadas uma das conseqüências são as altas taxas de aborto (NARVÁEZ, 2009). Além disso, todos os algoritmos acabam tendo seu tempo de resposta aumentado e o throughput reduzido, pois tais conflitos, só serão resolvidos bem depois, na última fase (votação ou certificação).

1.2 Objetivo

O objetivo deste trabalho é justamente prover um mecanismo que seja ciente dos potenciais conflitos envolvidos. Este problema só tende a piorar em cenários onde não só a carga de trabalho é elevada, mas também quando a carga de trabalho possui uma taxa de conflito muito elevada. Neste cenário a taxa de aborto tende a crescer rapidamente com conseqüência direta no tempo de resposta.

O desafio deste trabalho está em buscar um mecanismo que alcance um valor satisfatório para todos os seguintes objetivos, muitas vezes conflitantes, principalmente em um ambiente de banco de dados replicados em cenário de sobrecarga e conflito:

- Maximizar o throughput;
- Minimizar taxa de aborto;
- Minimizar o tempo de resposta.
- Escalabilidade (principalmente em ambientes heterogêneos de hardware)

A meta principal é que o sistema se comporte de forma homogênea e sem oscilações, independente da carga que está sendo submetido. O sistema deve se comportar de maneira a atingir sempre um mínimo satisfatório ao longo do tempo. Segundo (COSTA & FURTADO, 2011) estas características são imprescindíveis quando se deseja objetivos mais ambiciosos como *QoS* (qualidade de serviço) e *QoE* (qualidade de experiência). Qualidade da Experiência é uma medida de satisfação dos usuários e difere das mais tradicionais métricas de Qualidade de Serviço (*QoS*), o último é principalmente focado em tecnologia e desempenho em uma perspectiva técnica e o primeiro é uma abordagem centrada no usuário, que considera os objetivos destes. O foco deste trabalho consiste em criar as condições básicas necessárias para estas duas abordagens. A caracterização, estudo e implementação de níveis de serviço em banco de dados replicados ficam reservados para trabalhos futuros posteriores, não sendo foco nem objetivo deste trabalho.

1.3 Metodologia

Tradicionalmente algoritmos de replicação em banco de dados têm sido desenvolvidos fora do *kernel* do sistema. Vários *middlewares* de replicação foram propostos e desenvolvidos na literatura. Um balanceador de carga em banco de dados replicados se torna essencial para melhoria do desempenho e contornar as falhas essenciais dos algoritmos de replicação (CECCHET *et al.*, 2008). Este trabalho propõe uma abordagem em duas fases de balanceamento de carga.

A primeira fase tem como objetivo a (i) Distribuição justa de carga entre os recursos (réplicas) e (ii) Diminuir número de transações conflitantes em réplicas diferentes. Estas duas características se baseiam em duas observações: (a) Se transações conflitantes são submetidas para o mesmo servidor, o controle de concorrência local será responsável por serializar operações conflitantes adequadamente, diminuindo abortos. (b) Na falta de conflitos, porém, o desempenho é melhorado se as transações executam concorrentemente em diferentes réplicas. Portanto, ao invés de deixar as transações executarem em qualquer réplica, sem critério, as transações serão designadas a servidores preferenciais baseado no tipo das transações, seus parâmetros e conflitos relacionados. Como conseqüência uma diminuição da taxa de aborto é esperado. Porém no cenário a ser estudado, sobrecarga de trabalho e taxas conflitantes elevadas, balanceadores de carga baseados em conflitos tendem a sobrecarregar algumas réplicas enquanto outras ficam ociosas e degeneram os algoritmos *eager update-everywhere* em *eager primary copy*.

A segunda fase do balanceador tenta evitar justamente esta degeneração e limitar a carga de trabalho em cada réplica. O controle de admissão é um recurso muito importante para qualquer algoritmo de balanceamento de carga, onde existe um limite pré-determinado para o número de execuções simultâneas em um sistema de banco de dados particular. Limitando a carga tem o efeito de diminuir as conseqüências causadas por situações de pico, ocasionadas por rajadas de requisições que podem causar uma condição de sobrecarga no banco de dados (AMZA *et al.*, 2005). O número máximo de transações será adaptativo, isto é, sem nenhuma intervenção humana e tem como base um mecanismo de *feedback* (ABOUZOUR *et al.*, 2010). Nesta abordagem apenas duas informações são necessárias: a carga de trabalho em um determinado momento e uma

medida de desempenho do banco dados. A taxa de efetivação foi escolhida como tal medida por já embutir implicitamente os abortos gerados pelo sistema.

A proposta do balanceador de carga será estudada, validada e analisada através de um simulador estocástico determinístico construído ao longo deste trabalho. Este simulador, baseado nos trabalhos (AGRAWAL *et al.*, 1987) e (WIESMANN & SCHIPER, 2005), simula um ambiente completo de banco de dados replicados.

1.4 Resultados esperados e contribuições científicas

Ao término deste trabalho, espera-se uma estratégia de balanceador de carga em banco de dados replicado que seja adequado para situações de sobrecarga e taxas de conflito elevadas, mas que também seja adequado para cenários menos drásticos que estes. Mantendo assim o sistema em um estado de desempenho aceitável.

A conclusão deste trabalho traz como contribuição científica para a comunidade acadêmica, um novo balanceador de carga adaptativo para banco de dados replicados em cenários de sobrecarga e taxa de conflito elevada, capaz de manter o sistema harmônico, balanceado, com baixas taxas de aborto e estável sem picos de saturação, isto é, com um nível mínimo satisfatório de desempenho. Necessidades básicas para alcançar ambições maiores de níveis de serviço. Os conceitos aqui apresentados podem ser utilizados para pesquisas na área de banco de dados, como para sistemas de tempo real.

Como, também, fruto deste trabalho será um simulador completo para banco de dados replicados onde tanto a parte lógica como a física são modeladas. Novas estratégias poderão ser analisadas e avaliadas de maneira mais rápida e controlada através deste simulador minimizando o tempo de pesquisa e experimentação, já que validações e avaliações pré-eliminares podem ser realizadas antes da implementação das estratégias de fato.

1.5 Estrutura da dissertação

O restante da dissertação está estruturado da seguinte forma: o Capítulo 2 faz uma revisão de todos os conceitos necessários para o entendimento deste trabalho, apresentando os algoritmos de replicação de dados e um estudo comparativo de desempenho, ressaltando pontos fortes e fracos. O Capítulo 3 apresenta o problema a ser abordado nesta dissertação, como trabalhos anteriores tentaram resolve-lo e o porquê de suas falhas. Logo em seguida, no Capítulo 4 será apresentada a proposta do trabalho: Um novo balanceador de carga adaptativo para banco de dados replicados em cenários de sobrecarga e taxa de conflito elevada. O Capítulo 5 se destina aos experimentos com o objetivo de avaliar e validar a solução proposta. Neste capítulo será feito um estudo, através de simulação, do balanceador de carga proposto através de várias análises, ao final deste capítulo uma avaliação da proposta será apresentada. As conclusões deste trabalho serão apresentadas no Capítulo 6 juntamente com possíveis pesquisas futuras. Por fim, o Anexo I apresenta o fluxograma do balanceador.

2. Banco de Dados Replicados

Este capítulo apresenta os conceitos e algoritmos fundamentais de replicação em banco de dados, relacionados ao problema abordado por esta dissertação. A seção 2.1 apresenta a replicação em banco de dados em nível conceitual e uma evolução das idéias e dos principais algoritmos e sistemas de replicação, até a combinação de duas áreas aparentemente não complementares: banco de dados e computação distribuída. A seção 2.2 aborda a taxonomia atualmente mais aceita para as abordagens de replicação e a seção 2.3 seus principais algoritmos. As duas últimas seções, 2.4 e 2.5 abordam, respectivamente, a questão do desempenho destes algoritmos e como as cargas de trabalhos podem ser distribuídas entres as réplicas.

2.1 Replicação em Banco de dados

Banco de dados replicados é visto como um modo de aumentar a disponibilidade, desempenho e tolerância a falhas. O objetivo é distribuir a carga de trabalho entre as réplicas com o objetivo de melhorar os tempos de. Isso, geralmente, é possível quando as operações apenas lêem itens de dado. Operações de escrita necessitam de uma coordenação entre as réplicas (WIESSMAN & SCHIPER, 2005).

2.1.1 Modelo conceitual

Um banco de dados é uma coleção de itens de dados controlados por um sistema de gerenciamento. Um banco de dados replicado é, portanto, uma coleção de banco de dados que armazenam cópias dos mesmos itens de dados. Devemos distinguir um item de dado lógico X das suas cópias físicas X_i em diferentes réplicas. A seguir será apresentado um modelo de arquitetura de um banco de dados replicado.

Considere um conjunto de clientes $C = \{C_1, C_2,..., C_n\}$, e um conjunto de servidores $S = \{S_1, S_2,..., S_m\}$. O banco de dados é completamente replicado em cada servidor, isto é, cada servidor possui uma cópia completa do banco de dados. Um cliente conecta-se a um dos servidores, digamos S_i, para executar a transação.

Uma transação é uma unidade de trabalho que deve ser executada atomicamente e é composta por um conjunto de operações de leitura e escrita seguidas por uma operação de *commit* ou *abort*. O fim de uma transação deve ser refletido em todas as réplicas independente de um término com sucesso ou falha. Transações que possuem apenas operações de leitura são chamadas de consultas, e transações que possuem ambas as operações são chamadas de transações de atualização. Se duas ou mais transações são executadas concorrentemente, elas precisam ser isoladas uma das outras no caso de conflitarem. Duas operações de diferentes transações conflitam se ambas acessam o mesmo item de dado e pelo menos um deles é uma escrita. O controle de concorrência deve prover um mecanismo, como protocolos de *locking*, de garantir o isolamento destas transações. Todas as definições aqui apresentadas, assim como o modelo descrito podem ser encontradas com mais detalhes em (OZSU & VALDURIEZ, 2011).

O critério de correção baseia-se na seriabilidade de uma cópia (BERNSTEIN *et al.*, 1987). Ela garante que a execução de transações intercaladas é equivalente a execução serial destas transações em uma única cópia de banco de dados.

Uma vez que o cliente se conecta ao servidor de banco de dados S_i, ele envia todas as operações da transação para execução. Este servidor é chamado de servidor de delegação. As transações podem ser submetidas operação por operação ou em uma mensagem única.

Uma vez que a transação termina, o servidor S_i envia o resultado da transação para o cliente e a conexão entre o cliente e o servidor S_i é fechada. Se S_i falhar durante a execução, a transação é abortada e fica a cargo do cliente re-executar a transação, tanto se reconectando a S_i ou a outro servidor S_j.

2.1.2 Estratégias de replicação

Tradicionalmente, protocolos de replicação em banco de dados são categorizados usando dois parâmetros (GRAY *et al.*, 1996): A primeira diz respeito a quando a propagação da atualização ocorre (*eager* ou *lazy replication)*, a segunda diz quem pode executar as atualizações (*primary copy* ou *update everywhere*).

Primary copy: todas as atualizações são realizadas em uma cópia primária e só posteriormente nas outras cópias. Esta técnica simplifica o protocolo de replicação, porém introduz um único ponto de falha e um potencial gargalo.

Update everywhere: permite que qualquer cópia seja atualizada, acelerando o acesso, ao custo de uma coordenação mais complexa.

Eager Replication: garante que as mudanças nas cópias aconteçam dentro da fronteira das transações. Quando uma transação atinge a fase de efetivação, todas as cópias possuem o mesmo valor. Esta abordagem provê consistência, porém é custosa e lenta em termos de overhead de mensagens e tempo de resposta, passível a *deadlock* e baixa escalabilidade.

Lazy Replication: atualiza uma cópia local e propaga as mudanças apenas depois que a transação é efetivada. Este tipo de protocolo é eficiente, mas não garante consistência entre as réplicas quando todos os servidores puderem ser usados para submissão de transações (*update everywhere)*. Se apenas um servidor primário puder ser atualizado, a consistência é alcançada ao preço de introduzir um gargalo e ponto único de falha.

2.1.3 Evolução da Replicação

Na década de 80, a base teórica para banco de dados replicados girava em torno dos conceitos clássicos de seriabilidade e bloqueio (WEIKUM & VOSSEN, 2002). Para garantir a seriabilidade de uma cópia como critério de correção (a história produzirá resultados equivalentes a uma história de um sistema com uma cópia simples), a replicação foi implementada utilizando um protocolo, onde operações de leitura obtêm

locks locais e operações de escrita obterão *locks* distribuídos em todas as cópias. Atomicidade foi assegurada pela utilização de um protocolo de duas fases (2PC). O resultado foi um sistema completo e consistente, que se comportou como um único banco de dados.

Embora elegante e relativamente fácil de entender, tal abordagem de replicação escondia muitas armadilhas e complexos aspectos de engenharia. O trabalho desenvolvido em (GRAY *et al.*, 1996) foi o primeiro a apontar alguns destes problemas. Este enfatizou que tal abordagem era baseada na coordenação de cada operação individualmente. Como resultado, quando o número de cópias aumentasse, o tempo de resposta das transações, a probabilidade de conflito, as taxas de *deadlock* cresceriam exponencialmente. Outro estudo, encontrado em (KEMME, 2000), mostrou como sistemas comerciais da época, baseados nesta abordagem, não eram escaláveis com o aumento da carga de trabalho, degradando consideravelmente o desempenho. Com base nestas observações, concluiu-se que esta abordagem para replicação de banco de dados não poderia escalar e uma série de alternativas *lazy* foram propostas.

Devido a estes resultados e a prevalência de soluções, em sistemas comerciais, baseadas na replicação *lazy*, a pesquisa sobre replicação foi principalmente focada em entender as inconsistências criadas por esta replicação: modelos de fraca consistência, as estratégias de epidemia, imposição de restrições sobre as cópias (usando uma abordagem de cópia primária) e soluções híbridas (provendo consistência nos limites transacionais, mas propagando as atualizações de forma *lazy*) (KEMME & ALONSO, 2000).

2.1.4 Replicação baseada em primitivas de comunicação em grupo

Paralelamente e independente dos trabalhos em banco de dados replicados, uma ampla gama de sistemas de comunicação em grupo foram desenvolvidos e estudados na comunidade de sistemas distribuídos (ATTIYA & WELCH, 2004). Ao contrário de bancos de dados, onde a replicação serve para vários propósitos (de disponibilidade ao desempenho e elasticidade), a finalidade de replicação em computação distribuída é quase exclusivamente a tolerância a falhas.

Sistemas de comunicação em grupo gerenciam a troca de mensagens entre um grupo bem definido de nós, fornecendo primitivas de comunicação com dois propósitos: entrega confiável e ordenação de mensagens (PEDONE *et al*, 2000).

A visão, da comunidade de banco de dados, na época era de que não era possível conseguir desempenho sem sacrificar a consistência. A escolha parecia já ter sido feita: desempenho e escalabilidade tinham precedência sobre consistência (por isso a proliferação de estratégias *lazy*). O fato de que era possível implementar (e comercializar) sistemas com garantias mal definidas de consistência sempre encontrou incredulidade na comunidade de sistemas distribuídos. Na comunidade de banco de dados, os protocolos utilizados em sistemas distribuídos, e especialmente de comunicação em grupo, eram vistos como complexos e caros, sem a capacidade de realmente resolver os reais problemas de banco de dados.

Postgres-R (KEMME & ALONSO, 2000) foi o primeiro sistema de replicação de banco de dados que era tanto escalável quanto capaz de garantir consistência. Ou seja, foi capaz de prover mecanismos que permitiram o uso da abordagem *update-everywhere* em combinação com a propagação *eager* das alterações. Conseguiu alcançar estes dois objetivos, a partir de abordagem completamente diferente dos sistemas comerciais e propostas de pesquisa disponíveis na época.

A inovação mais importante no Postgres-R foi o aproveitamento de idéias de computação distribuída (comunicação em grupo) para resolver o problema da coordenação de atualizações em todas as réplicas, mantendo consistência global. Na época, era uma escolha de projeto contra-intuitivo em banco de dados, bem como na comunidade de sistemas distribuídos. Por um lado, comunicação em grupo foi desenvolvida quase que exclusivamente para fins de tolerância a falhas e Postgres-R o utilizou, principalmente, para razões de desempenho (como uma forma de reduzir o custo de assegurar a consistência). Por outro lado, para entender como primitivas de comunicação em grupo poderiam contribuir para o gerenciamento de transações em um banco de dados replicados era necessário fugir do modelo totalmente síncrono de replicação em bases de dados (2PC) e adotar uma abordagem menos acoplada, baseada na garantia de ordenação.

Tal abordagem extrapolava os limites do pensamento limitado acerca de banco de dados e sistemas distribuídos da época. Tais conceitos sobreviveram ao tempo,

principalmente o resultado de tal sinergia. Em particular o uso de comunicação em grupo, e a quebra da execução em uma fase local e uma fase de coordenação mostraram ser bem eficientes.

Hoje, comunicação em grupo (ou algum forma de protocolo de acordo) é amplamente utilizado como uma forma de implementar replicação em banco de dados tanto em sistemas comerciais quanto em projetos de pesquisa. Isso se deve ao fato de que muito se ganhou com o resultado da combinação de ambas as áreas desde a última década (KEMME & ALONSO, 2010).

Postgres-R trouxe uma abordagem para processamento de transações que até hoje sistemas de banco de dados replicados utilizam. A execução das transações foi dividida em quatro fases distintas: (i) Fase local onde a transação é processada apenas em um banco de dados (réplica local); (ii) Fase de envio onde as atualizações são propagadas; (iii) Fase de sincronização onde uma ordem global de serialização é estabelecida; (iv) Fase de escrita onde réplicas remotas executam a escrita e todas as réplicas efetivam a transação.

Estas quatro fases originaram características chaves que formam a base dos protocolos de replicação e que proporcionaram ganho de desempenho e simplicidade na comunidade de banco de dados replicados (PEDONE *et al.*, 2000).

- **Reduzido overhead de coordenação:** a transação é, primeiramente, completamente executada em uma única réplica e as operações de escrita enviadas apenas no final da transação dentro de uma única mensagem. Assim, evitando a sobrecarga de coordenação. Até a solicitação de confirmação ser submetida, a transação se comporta como se não houvesse replicação.
- **Controle de concorrência local para transações locais:** enquanto as transações estão sendo executadas localmente, elas são isoladas de outras transações locais pelo controle de concorrência tradicional.
- **Entrega em ordem total:** as primitivas de comunicação em grupo são utilizadas para propagar o conjunto de escrita para todas as réplicas em ordem total. Esta ordem é utilizada para determinar a ordem de serialização total durante a fase de sincronização. Como todas as réplicas receberam os conjuntos de escrita na mesma ordem, todas podem serializar as operações

de escrita da mesma forma sem uma maior coordenação com outras réplicas. Isto simplificou o controle de concorrência global significativamente e eliminou *deadlocks* distribuídos inteiramente.

- **Fases de escrita Independentes:** para evitar que o tempo de resposta aumente por ter que esperar até que a transação seja efetivada em todas as réplicas, a réplica local efetivará a transação, logo que a sua posição na ordem total for conhecida. A fase de escrita é executada de forma independente nas réplicas remotas.
- **Desbloqueio cedo:** A fase de escrita em uma réplica remota deve ser mais rápida devido ao fato de que todas as operações de escrita já serem conhecidas e não ser necessário coordenação com outras réplicas. Assim, os *locks* e os recursos necessários são mantidos apenas por um curto período de tempo, muito menor do que durante a fase local na réplica local. Portanto, aplicar operações de escrita de transações remotas tem pouco impacto na execução de transações locais.
- **Entrega confiável:** através do uso de entrega confiável, mesmo que falhas ocorram, todas as réplicas disponíveis recebem o mesmo conjunto escrita. Na prática, isto provê atomicidade: todas as réplicas (disponíveis) efetivam o mesmo conjunto de transações sem o uso do 2PC.

Um dos aspectos, principais, que o Postgres-R difere dos seus sucessores está relacionado justamente à implementação dentro de um sistema de banco de dados real. Enquanto se tem uma grande oportunidade de otimizações e acoplamento com o controle de concorrência e replicação, esta é uma abordagem invasiva e intrusiva e dependente da implementação de banco de dados.

Muitos dos sistemas posteriores optaram por implementar replicação fora do banco de dados em uma camada de *middleware*. Uma camada *middleware* tem a grande vantagem de que o banco de dados subjacente não precisa ser modificado. Ela também leva a uma boa separação de responsabilidades, permitindo ambientes heterogêneos, e permitindo o uso de sistemas de banco de dados cujo código fonte não está disponível. Em (CECCHET *et al.*, 2008), uma análise detalhada de abordagens baseadas em *middleware* é fornecido. *Middlewares* de replicação podem ser encontrados vastamente

na literatura: (CORREIA *et al.*, 2010), (MISHIMA & NAKAMURA, 2009), (PATIÑO-MARTÍNEZ *et al.*, 2005) e (PLATTNER *et al.*, 2006).

2.2 Classificação dos algoritmos de Replicação

Algoritmos *eager Replication* podem ser organizados de acordo com três parâmetros (WIESMANN *et al.*, 2000) que determinam a natureza e as propriedades de cada protocolo. Estes parâmetros são: (i) arquitetura do servidor, (ii) como alterações ou operações são propagadas través dos servidores e (iii) o protocolo de término da transação.

O primeiro parâmetro diz respeito em qual servidor, as transações poderão ser iniciadas e já foi brevemente explicado anteriormente. Existem duas possibilidades:

- ***Primary Copy*:** deve existir uma réplica específica, a cópia primária. Qualquer atualização ao item de dado precisa ser executada, primeiro, na cópia primária (executada ou pelo menos analisada para estabelecer a ordem de serialização). A cópia primária então propaga as atualizações (ou seus resultados) para todas as outras réplicas. Esta arquitetura tem principal desvantagem, a introdução de um único ponto de falha e um gargalo ao sistema.

- ***Update Everywhere:*** Atualizações nos itens de dados podem ser executadas em qualquer servidor do sistema, portanto, atualizações do mesmo item de dado podem chegar concorrentemente para duas réplicas distintas. Esta arquitetura é uma solução mais elegante, e que em teoria, não introduz nenhum gargalo, portanto é solução mais robusta a falhas e que facilita a distribuição de cargas através das réplicas.

O segundo parâmetro a considerar envolve o grau de comunicação entre os servidores de banco de dados durante a execução da transação. Isto determina a quantidade de tráfego de rede gerado pelo algoritmo de replicação e de toda sobrecarga no processamento das transações. Este parâmetro é expresso como função do número de mensagens necessárias para executar as operações da transação. Dois casos são considerados:

- **Iteração Linear:** técnicas onde o servidor de banco de dados propaga cada operação da transação. Enviar uma mensagem por operação aumenta muito a carga e leva a taxas de tráfego inaceitáveis. Além disso, estas mensagens chegam a diferentes momentos. Coordenar a execução se torna muito mais complexo (GRAY *et al.*, 1996).

- **Iteração Constante:** Corresponde a técnicas onde um número constante de mensagens é utilizado para sincronizar os servidores para uma determinada transação. Tipicamente protocolos nesta categoria trocam uma única mensagem por transação, agrupando todas as operações de uma operação em uma única mensagem. Enviar uma mensagem por transação, porém, tem suas desvantagens. Alguma forma de otimismo deve ser utilizada para primeiro executar a transação no servidor de delegação e então determinar a ordem de serialização. Se as taxas de conflito forem altas, esse otimismo pode resultar em altas taxas de aborto.

O último parâmetro considera a forma como as transações terminam e como a atomicidade é garantida. Existem dois casos:

- **Término com Votação:** exige uma rodada extra de mensagens para coordenar as diferentes réplicas. Existem duas formas para indicar o término: 2PC (BERNSTEIN *et al.*, 1987) ou mensagens de confirmação. Na mensagem de confirmação o servidor de delegação da transação decide de forma unilateral o resultado da transação. Porém as outras réplicas trabalham de forma determinística, já que deve obedecer a decisão do servidor de delegação.

- **Término Sem Votação:** implica que cada réplica pode decidir por si só se deve finalizar com sucesso ou falha (*commit* ou *abort*). Este tipo de técnica requer que as réplicas se comportem de forma determinística com a garantia, de se obter a mesma serialização para todos os servidores (serialização global).

Comparando todas estas características, conclui-se que: 1) *Update everywhere* tem um bom potencial para distribuir a carga através das réplicas. 2) Iteração linear leva a um grande *overhead*, tornando apenas iteração constante viável. Portanto *update everywhere* e iteração constante são as opções mais promissoras para *eager replication*.

2.3 Algoritmos

Combinado todos os critérios de classificação apresentados, chega-se a um total de oito categorias possíveis, como pode ser observado na Tabela 1. Porém como discutido, anteriormente, algoritmos que utilizam iteração linear são inviáveis e algoritmos baseados em *primary copy* não fazem balanceamento de carga. Portanto neste trabalho só serão considerados os algoritmos do quarto quadrante da tabela. Algoritmos *update everywhere* com iteração constante e que podem usar votação ou não.

Um estudo completo de todas as categorias de algoritmos pode ser encontrado em (FUERTES, 2011). A seguir serão apresentadas três técnicas de replicação mais relevantes baseadas em comunicação em grupo: ativa, certificação e votação fraca (MUÑOZ-ESCOÍ *et al.*, 2007).

Tabela 1 - Categorização e agrupamento dos algoritmos de replicação

Iteração entre os Servidores	*Término da transação*	*Arquitetura do Servidor*	
		Update Everywhere	**Primary Copy**
Constante	**Votação**	**Update Everywhere Iteração Constante Votação**	Primary Copy Iteração Constante Votação
	Sem Votação	**Update Everywhere Iteração Constante Sem Votação**	Primary Copy Iteração Constante Sem Votação
Linear	**Sem Votação**	Update Everywhere Iteração Linear Sem Votação	Primary Copy Iteração Linear Sem Votação
	Votação	Update Everywhere Iteração Linear Votação	Primary Copy Iteração Linear Votação

2.3.1 Replicação ativa

Também chamada de *state-machine replication*. Nesta técnica, todas as réplicas são inicializadas com o mesmo estado inicial e, sempre, recebem e processam a mesma seqüência de requisições dos clientes.

A principal vantagem desta abordagem é a simplicidade e a transparência a falhas, já que se uma réplica falhar, as requisições serão processadas por outras. Por outro lado, a corretude desta abordagem requer que as requisições sejam processadas de forma determinística, isto é, dado uma mesma seqüência de requisições, as réplicas produzem a mesma seqüência de saída e possuem o mesmo estado final.

Porém a execução concorrente de transações conflitantes não é determinística. Especificamente, a ordem na qual os *locks* são adquiridos é difícil de predizer. Para contornar este problema existem duas abordagens:

- Um escalonador global que gerencia quais comandos SQL podem ser processados concorrentemente a fim de garantir o determinismo. Esta abordagem introduz uma complexidade adicional e pode limitar a concorrência em cargas de trabalho com atualizações intensivas.

- A segunda abordagem consiste em fixar o determinismo no início da transação. No inicio do processamento, a transação completa é enviada via primitivas de comunicação em grupo para todas as réplicas. Se um servidor abortar, então todos os servidores também abortarão a transação. Este modelo exclui transações iterativas: todas as operações precisam ser conhecidas desde o inicio. Outro ponto negativo é que a transação completa é enviada para todas as réplicas, incluindo leitura e escrita – isto significa que as leituras são executadas por todos os servidores, inviabilizando o balanceamento de carga das operações de leitura. A Figura 1 ilustra o esquema para esta abordagem. Um exemplo de implementação deste algoritmo é o *Database State Machine Protocol* (ZUIKEVICIUTE & PEDONE, 2005)

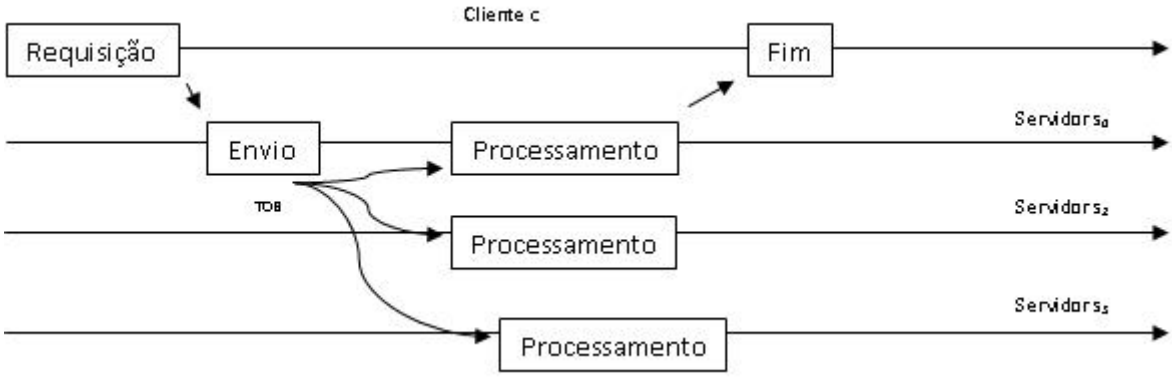

Figura 1 - Esquematização do algoritmo de Replicação Ativa (WIESMANN & SCHIPER, 2005)

2.3.2 Replicação baseada em certificação

Nesta técnica, a abordagem consiste em permitir que várias transações executem otimistamente em réplicas diferentes e, em tempo de efetivação, executar uma fase de certificação de modo a garantir a consistência global. Tipicamente, a coordenação global é alcançada com o uso de primitivas de comunicação em grupo, que estabelece uma ordem global entre transações concorrentes (THOMSON & ABADI, 2010).

Muitas variantes desta abordagem foram propostas. Uma delas consiste em usar o isolamento *snapshot* (JUNG *et al.*, 2011). No momento em que a transação é iniciada, uma réplica é escolhida para executar a transação (geralmente, a mais perto do cliente, isto é, o servidor de delegação). Quando é alcançado o ponto de efetivação, a identificação da transação, a versão de leitura do banco de dados e o conjunto de escrita são enviados usando primitivas de comunicação em grupo. Todas as réplicas verificam se a transação possui a mesma versão do banco de dados. Em caso afirmativo, a efetivação pode ser realizada. Caso contrário, é preciso verificar se nenhuma transação previamente efetivada conflita com transação em questão. Um conflito é detectado se houver atualização, concorrentemente, de um determinado item de dado em réplicas distintas. Se um conflito é detectado, a transação é abortada. Como o procedimento é determinístico e todas as réplicas, inclusive o servidor de delegação, recebem as transações na mesma ordem, todas as réplicas chegam à mesma decisão acerca do

resultado da transação. O servidor de delegação pode, portanto, informar ao cliente da aplicação o resultado da transação.

Outra proposta pode ser obtida através da serialização de transações. Para isto, o conjunto de leitura deve ser utilizado a fim de detectar conflitos de leitura-escrita durante a certificação. Um dos pontos negativos desta abordagem são os impactos negativos no desempenho (CORREIA *et al.*, 2005).

A abordagem baseada em certificação não requer que o banco de dados atue totalmente de forma determinística: somente a fase de certificação necessita deste requisito. Além disso, é permitido que diferentes transações de atualização sejam executadas ao mesmo tempo em réplicas distintas. Se o número de conflitos é relativamente pequeno, abordagens baseadas em certificação podem prover tolerância a falhas e escalabilidade. Caso contrário, o número de transações abortadas após a sua execução no servidor de delegação será muito alto. A Figura 2 ilustra o funcionamento do algoritmo.

Esta técnica possui os mesmo requisitos de comunicação da replicação ativa: apenas uma mensagem é necessária por transação. Porém não tem as desvantagens: ela aceita transações iterativas.

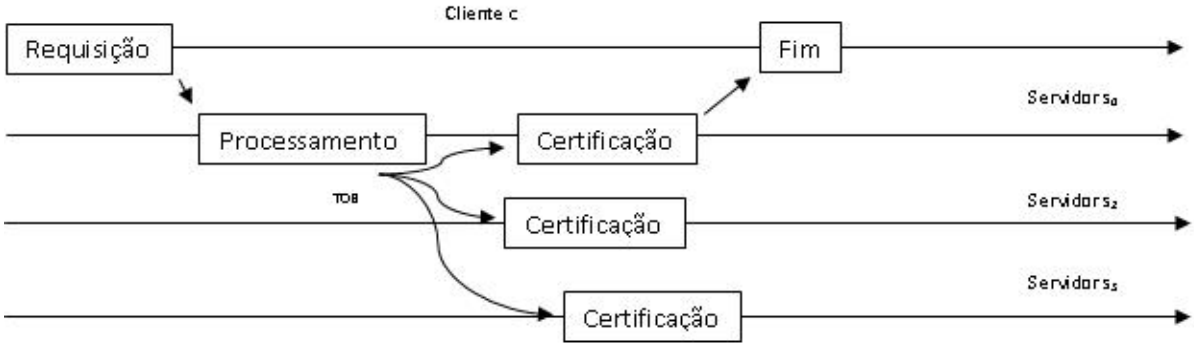

Figura 2 - Esquematização do algoritmo de Replicação baseado em certificação (WIESMANN & SCHIPER, 2005)

2.3.3 Weak Voting Replication

Nesta técnica, mostrada na Figura 3, o servidor de delegação inicializa e executa a transação completa e quando o ponto de efetivação é alcançado, o conjunto de escrita é enviado para todas as réplicas via primitivas de comunicação (inclusive o servidor de delegação). Uma vez que o conjunto de escrita é enviado, o protocolo de replicação, no servidor de delegação, verifica se existe conflito com alguma transação previamente efetivada. Caso exista, a transação é abortada. Caso não exista conflito, a transação será efetivada. Uma vez que a decisão de término seja tomada, uma segunda mensagem é enviada para as demais réplicas informando o resultado da transação. Os outros servidores, não podem decidir por si próprios, e precisam esperar pela segunda para completar a transação. As vantagens desta técnica são:

- Nenhum conjunto de leitura é enviado para as outras réplicas. Os conflitos de leitura-escrita são verificados no servidor de delegação. Isto é, o conjunto de leitura de transações locais contra o conjunto de escrita de transações remotas.
- Transações somente de leitura podem ser executadas localmente, sem a necessidade de envio para outras réplicas.

Por outro lado a principal desvantagem é a necessidade de envio de duas mensagens para completar a transação.

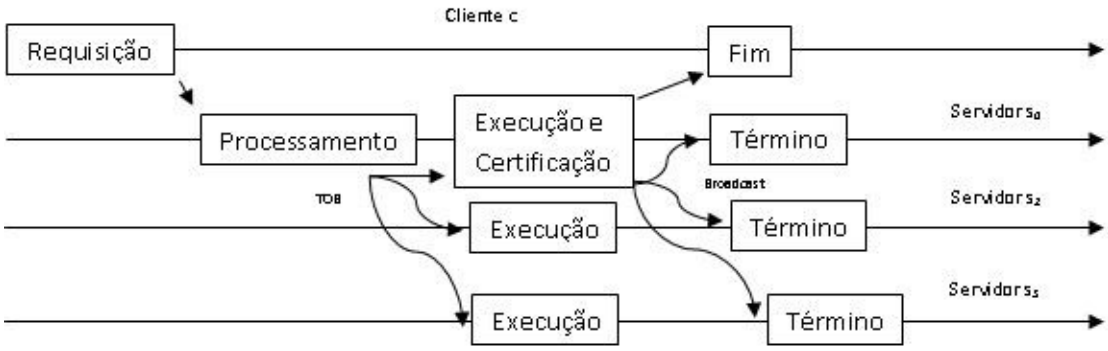

Figura 3 - Esquematização do algoritmo de Replicação baseado em Votação (WIESMANN & SCHIPER, 2005)

Esta técnica é parecida com a anterior, sendo a principal diferença o fato de que a certificação determinística é substituída por uma fase de votação fraca, isto é, o servidor de delegação decide entre *commit* ou *abort*. Note que a replicação baseada em certificação requer apenas uma mensagem por transação. A votação é dita fraca, pois apenas o servidor de delegação decide o resultado da transação. Os outros servidores não influenciam esta decisão. Esta técnica se encaixa perfeitamente no isolamento serializável. O trabalho (JUÁREZ *et al.*, 2007a) é um bom exemplo da aplicação desta abordagem de replicação.

2.4 Desempenho dos algoritmos de replicação

Em (WIESMANN & SCHIPER, 2005) é apresentado uma comparação das três técnicas descritas anteriormente juntamente com outras abordagens que se não contam com grupo de comunicação (*Primary Copy*, *Lazy* e 2PC). O trabalho conclui que as classes de protocolos de replicação com o melhor desempenho e que garantam consistência são aqueles que utilizam a estratégia *eager* e que combinam o uso de um único servidor de delegação por transação com a propagação de atualização utilizando primitivas de comunicação em grupo. Replicação *lazy* está incluso no estudo somente para fins de comparação (que requer um mínimo de sincronização, mas mais tarde necessidades técnicas de reconciliação para assegurar consistência, o que torna esta técnica não prática). A Figura 4 ilustra os resultados obtidos neste trabalho, nos quais as principais conclusões são:

- *Primary copy* possui um problema de sobrecarga devido ao fato de toda requisição se concentrar em um servidor.
- *Active replication* possui um limite assintótico, isto se deve ao fato da necessidade em se obter *locks* na ordem definida pelas primitivas de comunicação em grupo no início da transação.
- *Weak-voting replication* e do *certification-base replication* tiveram tempos de resposta muito próximos. O algoritmo baseado em replicação é ligeiramente inferior devido justamente à necessidade de envio de uma mensagem para decidir o resultado da transação.

- O algoritmo de certificação é otimista por natureza, em cenários de carga elevada, a taxa de aborto sofre um acréscimo considerável.

- 2PC tem tempos de resposta similares ao algoritmo baseado em certificação e votação, porém estes últimos possuem tempos de reposta melhores. Um dos problemas apontado pelo 2PC é a falta de escalabilidade, um aumento no número de servidores piora o desempenho deste algoritmo. O outro problema se refere às altas taxas de aborto com o aumento da carga e número de servidores.

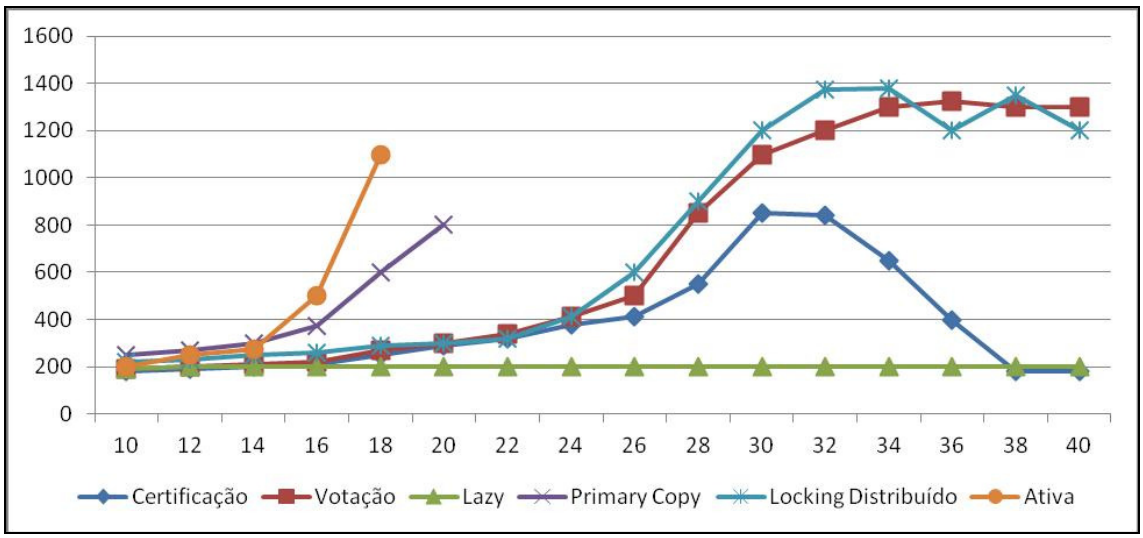

Figura 4 - Comparação de desempenho de algoritmos de replicação. Adaptado de (WIESMANN & SCHIPER, 2005)

Pesquisas recentes têm seu foco voltado para a melhoria do desempenho e melhoria dos efeitos colaterais negativos destes algoritmos. Três tendências principais podem ser observadas: (i) balanceamento de carga, (ii) níveis de isolamento e (iii) escalabilidade (MUÑOZ-ESCOÍ *et al.*, 2007).

Balanceamento de carga consiste em distribuir a carga recebida por cada réplica, de modo que nenhum servidor fique nem tão ocioso nem tão sobrecarregado.

Pesquisas em níveis de isolamento procuram dar suporte a múltiplos níveis de isolamento em banco de dados replicados. Transações que usem níveis de isolamento mais relaxados podem ter um ganho em seus tempos de resposta.

Escalabilidade consiste em aumentar o número de nós com o objetivo de melhorar o desempenho do sistema. Um bom algoritmo de replicação deve suportar esta característica.

Como visto pelo gráfico acima, o algoritmo de certificação possui desempenho superior ao de votação, porém possui algumas conseqüências negativas. Um balanceador de carga independente do nível de isolamento que se deseja utilizar será proposto. Na verdade são abordagens complementares no objetivo de melhoria do desempenho. O foco deste trabalho está em melhorar o desempenho destes algoritmos em cenários de sobrecarga e taxas de conflito elevadas. Busca-se, na verdade, controlar o sistema de banco de dados replicado em situações de estresse, mantendo um nível confiável de utilização sem picos de forma que estratégias mais ambiciosas como QoS e QoE tenham um terreno estável para que sejam desenvolvidas em trabalhos futuros.

2.5 Balanceamento de Carga

Um banco de dados replicado construído para alta disponibilidade deve eliminar todos os pontos únicos de falha. Muitas vezes, projetos com foco no desempenho negligenciam as necessidades dos componentes principais de replicação, como balanceadores de carga. Um balanceador de carga em banco de dados replicados se torna essencial para melhoria do desempenho e contornar as falhas essenciais dos algoritmos de replicação (CECCHET *et al.*, 2008).

Replicação em banco de dados nos últimos anos foi implementada em nível de *middleware*, isto é, o *kernel* dos bancos de dados não foram alterados para dar tal suporte (KEMME & ALONSO, 2010). Uma abordagem baseada em *middleware* carrega consigo uma infra-estrutura adequada para abordagens de balanceamento de carga. Um balanceador de carga perfeitamente ajustado às necessidades do sistema mantém todas as réplicas ocupadas, deixando em níveis equivalentes de carga e deve sempre considerar a heterogeneidade de hardware existente nas distintas réplicas. Um balanceador atendendo a estas características é essencial para tornar um sistema escalável e estável.

Balanceamento de carga pode ser implementado em nível de conexão, nível de transação ou nível de consulta. Em nível de conexão o balanceamento de carga distribui novas conexões dos clientes a réplicas de acordo com uma política específica; todas as transações e pedidos desta conexão irão para a mesma réplica até a conexão ser fechada. Esta abordagem é simples, mas não oferece equilíbrio quando os clientes usam *pools* de conexão ou conexões persistentes. Balanceamento de carga de nível de transação ou consulta realiza o balanceamento de carga em granularidade mais fina, direcionando as requisições baseado na transação ou consulta, respectivamente.

O foco deste trabalho é propor um balanceador de carga para banco de dados replicados de tal maneira que o sistema se comporte de maneira homogênea, confiável e sem oscilações ao longo do tempo, independente da carga a ser submetida.

3. Replicação em cenários de sobrecarga e taxa de conflito elevada

Este capítulo apresenta o problema principal que este trabalho pretender resolver em banco de dados replicados: manter o sistema sobre controle em um cenário de sobrecarga e taxa de conflito elevada. A seção 3.1 descreve o que acontece em um banco de dados replicado em tal cenário e a seção 3.2 apresenta trabalhos relacionados que poderiam ajudar a resolver tal problema.

3.1 Problemas em um cenário de sobrecarga de taxa de conflito elevada

O estudo em (WIESMANN & SCHIPER, 2005), (JUÁREZ *et al.*, 2007b) e (JUÁREZ *et al.*, 2008) sugerem que apenas dois algoritmos devam ser considerados quando se trata de replicação em banco de dados: certificação e votação. Devido à própria natureza de replicação, estes algoritmos mostram queda de desempenho quando submetidas a cargas de trabalho elevadas. Para analisar e entender as conseqüências deste cenário, duas medidas de desempenho devem ser monitoradas e estudadas:

- Tempo de Reposta
- Taxa de Aborto

A Figura 5 ilustra o que acontece com o tempo de resposta de um banco de dados replicado quando submetido a este cenário.

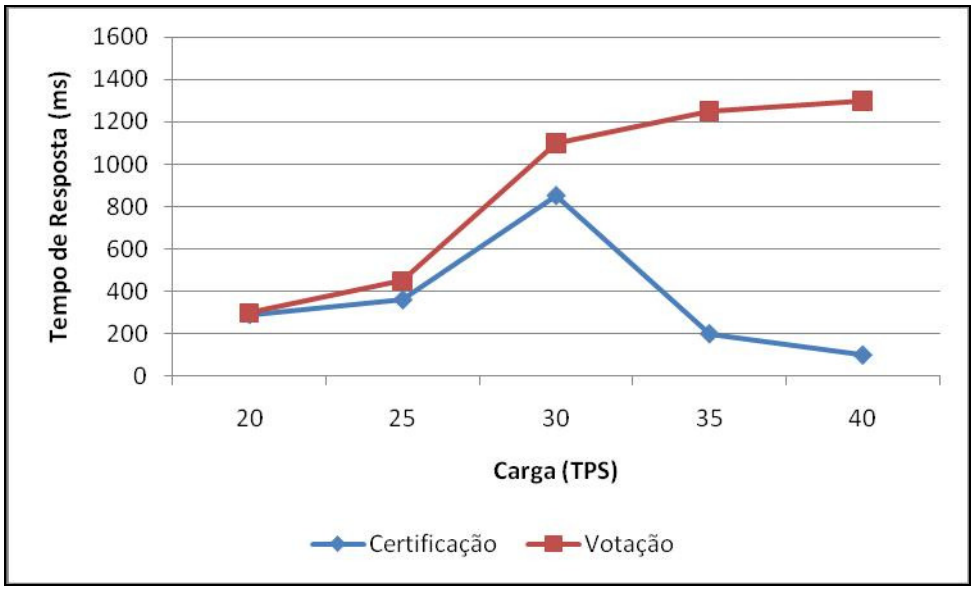

Figura 5 - Comparação do tempo de resposta de algoritmos de replicação. Adaptado de (WIESMANN & SCHIPER, 2005)

A curva do algoritmo baseado em votação, para uma carga de trabalho baixa, tem desempenho ligeiramente inferior que o algoritmo de replicação (como discutido anteriormente). Tudo se comporta como intuitivamente esperado (o tempo de resposta cresce com o aumento da carga de trabalho devido ao compartilhamento de recursos, contenção do sistema entre outros fatores que levam o sistema a um desempenho inferior). Porém a curva do algoritmo de certificação sofre uma queda abrupta e passa a ter tempos de resposta bem inferiores (por volta de 30 transações por segundo). A primeira vista o comportamento do algoritmo de certificação é muito melhor que o algoritmo de votação: o tempo de resposta fica significativamente melhor mesmo quando o sistema começa a ficar sobrecarregado.

A diferença está na taxa de aborto entre as duas técnicas. A Figura 6 mostra a taxa de aborto para os dois algoritmos. Os parâmetros utilizados foram os mesmos que os da Figura 5. Pode-se ver que enquanto o tempo de resposta do algoritmo baseado em certificação é baixo, sua taxa de aborto é significativamente alta. A sobrecarga do sistema gerou uma taxa de conflito muito alta na fase de certificação; levando ao aborto de várias transações de atualização. É interessante notar que enquanto o tempo de resposta do algoritmo baseado em votação aumenta, a taxa de aborto se mantém estável. Abaixo de 2%.

Este fato pode ser explicado devido ao fato de que o algoritmo baseado em certificação necessita manter um *log*, ordenado de transações efetivadas, que será utilizado na fase de certificação com objetivo de abortar ou efetivar transações que foram entregues pelo mecanismo de comunicação em grupo. A alta taxa de aborto se deve ao fato de que as transações podem ficar muito tempo no *log* (causando conflitos com novas transações que necessitam ser certificadas) até serem removidas após ter sido efetivada em todas as réplicas.

O algoritmo baseado em votação não sofre deste problema devido ao fato de que não necessita de um *log*, pois uma vez que um conjunto de escrita é enviado, este é aplicado atomicamente em cada réplica (muitas vezes causando o aborto de transações locais) e uma mensagem posterior do servidor de delegação informa se a transação deve ser abortada ou efetivada.

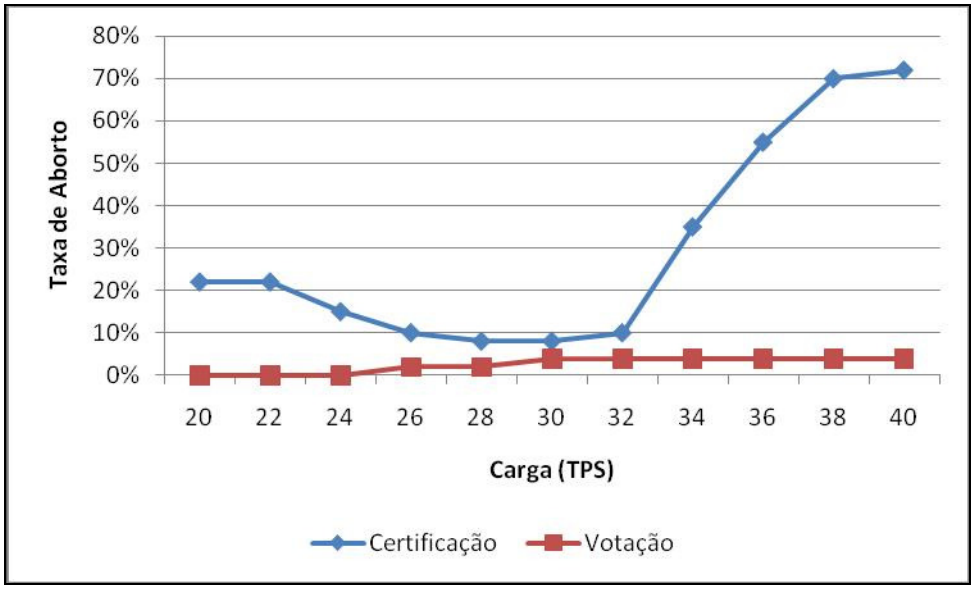

Figura 6 - Comparação da Taxa de Aborto. Adaptado de (WIESMANN & SCHIPER, 2005)

Todas estas observações estão relacionadas à mesma causa. Um dos grandes problemas em protocolos de replicação em banco de dados é a execução de transações conflitantes em réplicas distintas. Para cargas leves o problema não é tão relevante, porém para cargas pesadas, o algoritmo baseado em certificação acaba sofrendo altas taxas de aborto. Além disso, os dois algoritmos acabam tendo seu tempo de resposta

aumentado e o throughput reduzido devido ao fato de que os conflitos de leitura-escrita, só serão resolvidos bem depois, na última fase (votação ou certificação). Este problema só tende a piorar não só onde a carga de trabalho é elevada, mas também quando a carga de trabalho possui uma taxa de conflito muito elevada. Neste cenário taxa de aborto e tempo de resposta tenderão a crescer rapidamente.

Portanto a ocorrência destes dois fatores: carga e taxa de conflito elevada tendem a degradar um sistema de banco de dados replicados. Sozinhos ambos os fatores já trazem conseqüências indesejáveis, porém quando surgem ao mesmo tempo levam a uma perda de desempenho consideravelmente alta. A seguir este cenário será analisado em duas etapas: (i) taxa de conflito elevada e (ii) carga de trabalho elevada.

Em um cenário de taxa de conflito elevada, as transações não podem ser designadas aos servidores sem nenhuma política, servidores preferenciais devem ser escolhidos de acordo com a carga de trabalho submetida com o objetivo de maximizar o desempenho como um todo. Como o problema reside no fato da execução de transações conflitantes em servidores distintos, uma abordagem natural seria designar transações conflitantes para o mesmo servidor. Assim o próprio banco de dados local serializa as transações, evitando conflitos na fase posterior. Portanto, um balanceador de carga seria necessário para realização de tal tarefa. Estratégias para este balanceamento fará parte da proposta desta dissertação e será analisada mais adiante.

Porém, com o aumento da carga de trabalho, atinge-se um determinado ponto onde o balanceador de carga não produzirá mais resultados. Todos os servidores estarão sobrecarregados e o balanceador não será suficiente e uma alternativa será obrigatória.

A única saída para contornar este problema, é limitar o número de transações ativas (multiprogramação - MPL). Fácil de entender, esta abordagem limitando o número de transações que podem entrar no sistema, impede que este entre em estado de sobrecarga. Existem duas abordagens para a escolha do MPL: estática ou dinâmica. A primeira requer que um administrador de banco de dados, através de várias simulações, escolha um MPL adequado. Porém as características da carga de trabalho podem e inevitavelmente mudam ao longo do tempo. É necessário que o MPL se adapte, caso contrário, não corresponderá mais à realidade levando o sistema à sobrecarga ou a um throughput ineficiente, deixando o sistema ocioso (servidores ociosos). O ideal é o que o sistema possa se adaptar dinamicamente às mudanças tanto da carga de trabalho

quanto da configuração do sistema. Estratégias de MPL adaptativo serão analisadas ainda neste capítulo e farão parte da proposta a ser apresentada.

Em resumo, as seguintes características são desejadas:

- Distribuição justa de carga entre os recursos (réplicas);
- Respeitar o limite de carga de cada réplica (MPL);
- Diminuir número de transações conflitantes em réplicas diferentes.

O desafio deste trabalho está em buscar um mecanismo que alcance um valor satisfatório para todos os seguintes objetivos, muitas vezes conflitantes, principalmente em um ambiente de banco de dados replicados em cenário de sobrecarga e conflito:

- Maximizar o throughput;
- Minimizar taxa de aborto;
- Minimizar o tempo de resposta.
- Alcançar escalabilidade (principalmente em ambientes heterogêneos de hardware)

3.2 Trabalhos relacionados

Existem vários trabalhos relacionados a balanceamento de carga em banco de dados replicados, porém estes trabalhos consideram um ambiente mais controlado no que se refere às características da carga de trabalho submetida. Além disso, a maioria destes trabalhos não leva em consideração os conflitos existentes entre as transações, apenas alguns podem ser destacados como será abordado adiante, porém sem o desempenho esperado quando submetidos ao cenário estudado.

Em relação ao controle do MPL vários trabalhos relacionados podem ser encontrados na literatura. Recentemente o foco de pesquisa nesta área tem se voltado para ao controle adaptativo baseado em um mecanismo de *feedback*. Adiante será

analisado as abordagens que trouxeram resultados mais atraentes. Logo em seguida, uma discussão comparativa das abordagens será apresentada.

A combinação destes dois temas é uma área com carência de pesquisa e com poucos resultados, principalmente quando se trata de um cenário de sobrecarga e taxa de conflito elevada (AMZA *et al.*, 2005), (CECCHET *et al.*, 2008) e (ABOUZOUR *et al.*, 2010).

3.2.1 Balanceamento baseado em conflito

Esta técnica de balanceamento de carga (MUÑOZ-ESCOÍ *et al.*, 2006), (ZUIKEVICIUTE & PEDONE, 2008) se baseia em duas observações: (a) Se transações conflitantes são submetidas para o mesmo servidor, o controle de concorrência local será responsável por serializar operações conflitantes adequadamente, diminuindo abortos. (b) Na falta de conflitos, porém, o desempenho é melhorado se as transações executam concorrentemente em diferentes réplicas.

Portanto, ao invés de deixar as transações executarem em qualquer réplica, sem critério, as transações serão designadas a servidores preferenciais baseado no tipo das transações, seus parâmetros e conflitos relacionados. Como conseqüência uma diminuição do número de abortos é esperado.

Idealmente, o balanceador de carga deveria minimizar o número de transações conflitantes executando em réplicas distintas e maximizar o paralelismo entre transações, mas infelizmente estes são requisitos opostos. Designar transações a servidores preferenciais é um problema de otimização. Por exemplo, concentrar transações conflitantes em poucas réplicas reduzirá a taxa de aborto, mas poderá sobrecarregar algumas replicas e deixar outras ociosas. Este problema será atacado por um balanceador de carga baseado em conflito. Este balanceador busca encontrar um equilíbrio entre os dois requisitos.

Ambos os algoritmos (certificação e votação) se beneficiam do balanceador pela melhor distribuição da carga, diminuindo a taxa de aborto, os tempos de resposta e aumentando o throughput do sistema.

Os resultados alcançados em (ZUIKEVICIUTE & PEDONE, 2008) mostraram que balancear transações com o único objetivo de maximizar o paralelismo já dobra o throughput obtido comparado ao deixar as transações serem livremente designadas aos servidores. Balancear transações com o objetivo de minimizar conflitos pode reduzir abortos na falta de sincronização, mas melhorias são obtidas ao custo de um aumento no tempo de resposta já que o balanceamento será injusto.

Este mesmo trabalho concluiu que uma abordagem híbrida permite trocar uma distribuição de carga equilibrada por baixas taxas de aborto com o objetivo de aumentar o throughput e sem nenhuma degradação no tempo de resposta.

Apesar desta abordagem, ter conseguido bons resultados, ela não foi aplicada a realidade do cenário sendo estudado. Quando o número de transações conflitantes aumenta, alguns servidores ficarão sobrecarregados em detrimento a outros, isto é, o balanceamento de carga falhará. Com o aumento do número de requisições, esta situação se agrava até o ponto onde o seguinte ocorrerá:

- Taxa de aborto ficará em limites inaceitáveis;
- Tempo de resposta aumentará consideravelmente;
- Throughput diminuirá drasticamente;
- Alguns servidores ficarão ociosos e outros sobrecarregados.

Uma análise das observações acima sugere que ocorrerá uma degeneração do algoritmo *everywhere* para uma abordagem baseada em *primary copy*. Ou seja, o balanceador de carga não fará o que foi proposto a fazer, na verdade, o balanceador será responsável por degradar o sistema mais ainda. Em um cenário como este, uma atribuição randômica do sistema seria uma alternativa melhor apesar do acréscimo da taxa de aborto.

A abordagem proposta, também, possuirá um balanceador de carga baseado em conflito, mas sem estas características indesejáveis. Todas estas afirmações e análise qualitativa foram obtidas através dos experimentos quantitativos realizados no Capítulo 5.

3.2.2 Controle adaptativo da Multiprogramação (MPL)

Controle de admissão é um componente muito importante para qualquer algoritmo de balanceamento de carga, onde existe um limite pré-determinado para o número de execuções simultâneas em um repositório em particular. Limitando a carga tem-se como conseqüência, diminuir os efeitos causados por situações de pico, ocasionadas por rajadas de requisições que poderia causar uma condição de sobrecarga no banco de dados (AMZA *et al.*, 2005).

Normalmente, as tarefas que encontram, no momento de suas chegadas, o sistema saturado, são colocadas em uma fila de espera, normalmente com uma política de retirada baseada na ordem de chegada (primeiro a chegar, primeiro a ser servido, isto é, FCFS). Uma segunda abordagem, utilizada por servidores de bancos de dados é a rejeição da requisição, com o conseqüente envio de uma notificação de erro para o cliente. A Figura 7 ilustra o uso do controle de admissão. Um número variável de clientes requer os recursos de conexões do servidor de banco de dados, caso o número de requisições seja maior que o número de conexões disponíveis, estas requisições serão enfileiradas à espera de conexões liberadas por outras requisições em execução.

A grande dificuldade é como definir o tamanho do *pool* de conexões para alcançar bons níveis de throughput (HARIZOPOULOS, 2005). Este parâmetro efetivamente controla o nível de multiprogramação do servidor. DBAs tem duas opções ao definir o valor deste parâmetro:

- **Grande número de conexões:** Usando um valor grande para o número de conexões permite que o servidor processe um grande número de pedidos em simultâneo. No entanto, a desvantagem é que há um risco substancial de que um grande número de conexões coloque o servidor em um estado de sobrecarga (CHEN & GORTON, 2002). Outro problema está relacionado à natureza da replicação. Muitas transações concorrentes aumentam substancialmente a taxa de aborto e o tempo de resposta.

- **Pequeno número de conexões:** O benefício imediato desta abordagem é que as taxas de aborto e tempos de resposta diminuem significativamente. No entanto, a desvantagem é que o nível de multiprogramação do servidor é reduzido e os recursos de hardware, talvez subutilizados. Isto pode ser constatado, observando o baixo valor de throughput.

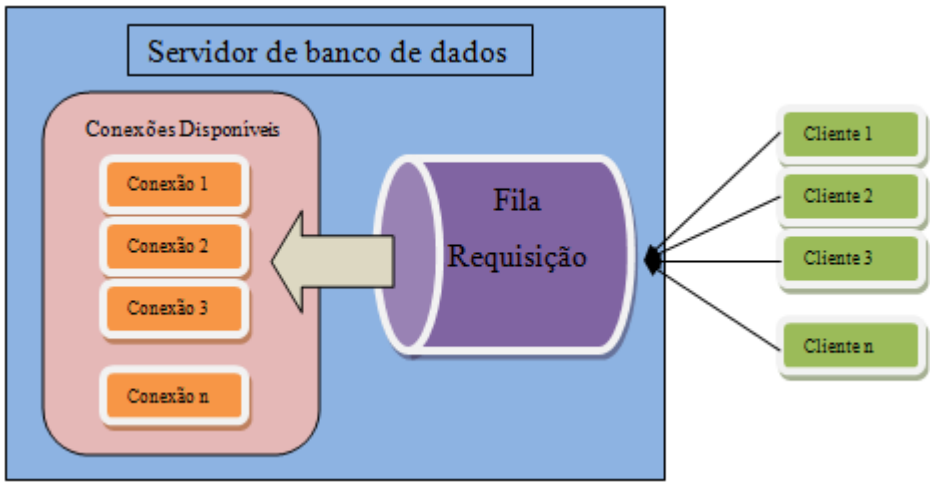

Figura 7 - Esquematização do Controle de admissão (HARIZOPOULOS, 2005)

A fim de decidir o melhor valor a ser usado para o número de conexões, DBAs devem executar testes de carga para chegar a um valor que possa atingir o tempo de resposta mínimo necessário pela aplicação de banco de dados. Embora a abordagem experimental possa ser efetiva na configuração da multiprogramação do servidor de banco de dados, ela sofre de alguns inconvenientes. Um dos principais inconvenientes é que esta parametrização só é adequada sob as condições testadas e não pode lidar com aumento ou diminuição da carga de trabalho e diferentes cargas no que se refere às suas características. Devido a isto, várias abordagens para ajustar a multiprogramação dinamicamente foram propostas. Estas abordagens se baseiam em um mecanismo de *feedback* de alguma medida de desempenho do banco de dados. Três principais abordagens foram propostas: (i) Maximizar o throughput, (ii) minimizar a taxa de conflito e (iii) minimizar o tempo de resposta. A seguir estas três abordagens serão apresentas.

3.2.2.1 Maximizar o throughput

Um dos primeiros trabalhos que expôs o problema de controle de carga adaptativa em sistema de banco de dados foi desenvolvido em (HEISS & WAGNER, 1991). Nesse artigo, os autores descreveram diferentes fontes de contenção e sugeriram um mecanismo de controle baseado em *feedback* que limita o nível de concorrência do servidor. Dois algoritmos diferentes foram propostos: Passos Incrementais (IS) e uma

abordagem de Aproximação por Parábola (PA). Na abordagem IS, o controlador monitora o throughput do servidor e o nível de concorrência atual. O controlador, então, aumenta o nível de concorrência e monitora seu um efeito sobre o throughput do servidor. Se o resultado for um aumento do throughput, o controlador continua a incrementar o nível de concorrência e, inversamente, se o throughput diminuir, o controlador reduz a multiprogramação até o ponto em que o desempenho começa a declinar. Na abordagem PA, o desempenho do servidor é aproximado usando uma função parabólica, esta estratégia será analisada em detalhes adiante, pois fará parte da proposta deste trabalho.

3.2.2.2 Minimizar a taxa de conflito

Em (MONKEBERG & WEIKUM, 1992) foi proposto um método de controle adaptativo da carga baseado na taxa de conflito como métrica de desempenho. A taxa de conflito é a razão do número de *locks* que são realizados por transações no sistema, dividido pelo número de *locks* mantidos por transações ativas (não bloqueadas). Se a proporção exceder um valor determinado, então o sistema estará em sobrecarga. O controle da carga é feito pelo controle de admissão e cancelamento das transações que não podem ser executadas no momento da requisição (por falta de conexões disponíveis). O controle de carga adaptável é executado antes do início de uma transação ou após o final de uma transação. Além disso, o trabalho também sugeriu o uso de informações sobre transações para decidir qual política de admissão utilizar.

3.2.2.3 Minimizar o tempo de resposta

O trabalho realizado em (BROWN *et al.*, 1994) tentou otimizar o desempenho relacionando dois parâmetros: o nível de multiprogramação e a quantidade de memória usada por transações. O algoritmo de ajuste proposto baseia-se na classificação das operações por objetivos esperados no tempo de resposta. O trabalho desenvolvido em (SCHROEDER, 2006) ajusta o nível de multiprogramação através de um controlador de agendamento externo com base em períodos de observações e reações. O controlador

utiliza um simples controle de *feedback* levando em consideração tanto o tempo de resposta como o throughput.

Em pesquisa similar, (LIU *et al.*, 2003) procurou otimizar o tempo de resposta do servidor web Apache. Eles se concentraram em ajustar o parâmetro *MaxClients* que controla o nível de multiprogramação do servidor. Diferentes algoritmos de controle foram utilizados, incluindo *Hill Climbing* e algoritmos baseados em heurística. Em seu trabalho, este ajuste ajudou a reduzir o tempo de resposta por um fator de 10 ou mais. Um dos itens de seus trabalhos futuros foi implementar essas técnicas em aplicações mais complicadas como um servidor de banco de dados ou um servidor de aplicação.

3.2.2.4 Comparação das abordagens

O trabalho realizado em (SCHROEDER, 2006) estudou exaustivamente todas as três abordagens e concluiu que para abordagens de controle do MPL baseado em *feedback*, apenas as medidas de desempenho baseadas no throughput e tempo de resposta são suficientes. O trabalho realizado em (ABOUZOUR *et al.*, 2010) propôs uma técnica híbrida para adaptar MPL baseado em *feedback*, utilizando as duas medidas dando um ganho de desempenho melhor comparado ao desempenho individual de cada uma.

Porém para o cenário apresentado estas duas medidas não são tão adequadas devido à taxa de aborto inerente aos bancos de dados replicados.

O throughput leva em consideração todas as transações finalizadas em um determinado intervalo de tempo, sem distinguir se houve aborto ou não. Quando a taxa de aborto começa a se tornar alta o throughput deixa de ser uma medida adequada para o problema apresentado.

O controle baseado em tempo de resposta também se mostra inadequado, devido às transações abortarem, este tempo tende a diminuir por deixar o sistema mais ocioso quando a ocorrência de abortos prematuros.

Portanto, estas duas medidas não refletem o que está acontecendo de fato com o sistema, outra medida de desempenho deve ser utilizada. A abordagem proposta utilizará a taxa de transações efetivadas. Isto é, o número de transações que são efetivadas em um determinado intervalo de tempo.

3.2.3 Balanceamento de carga para tarefas com restrições temporais

O trabalho realizado em (ORLEANS, 2007) busca um controle do banco de dados e introdução de restrições temporais de *QoS*. O trabalho procurou unir um balanceador desenhado com o objetivo de *QoS* juntamente com um controle de admissão adaptativo baseado no tempo de resposta das transações.

Apesar de bons resultados obtidos, evitando situações de pico e sobrecarga do sistema. A abordagem utilizada e o foco do trabalho não são aplicados para o cenário e arquitetura de banco de dados replicado considerado neste trabalho. O primeiro ponto de divergência reside no fato de que o trabalho foi aplicado para estratégias Primary Copy de replicação unidas a um sincronismo *lazy* entre as réplicas. O segundo ponto de divergência é o cenário no qual está sendo considerado: sobrecarga de trabalho e altas taxas de conflito. O trabalho focou na utilização de cargas moderadas, até mesmo, devido ao fato da arquitetura escolhida não suportar cargas elevadas como mostrado por (WIESMANN & SCHIPER, 2005).

4. Balanceamento de carga adaptativo com restrições de conflito

Este capítulo apresenta a proposta de balanceamento deste trabalho que pretende resolver os problemas, apresentados anteriormente, em banco de dados replicados: manter o sistema sobre controle em um cenário de sobrecarga e taxa de conflito elevada. A seção 4.1 faz uma descrição geral da proposta, a seção 4.2 aborda a questão do controle de admissão e a seção 4.3 o balanceamento baseado em conflito. A seção 4.4 apresenta o algoritmo proposto enquanto que a seção 4.5 apresenta as variáveis do algoritmo. Por fim a seção 4.6 apresenta as contribuições desta proposta.

4.1 Abordagem de balanceamento em dois níveis

Nesta seção será apresentada uma proposta de *middleware* que provê uma abordagem em dois níveis para balanceamento adaptativo em banco de dados replicados. A abordagem baseada em *middleware* foi escolhida por ser uma abordagem não intrusiva, permitindo a separação de responsabilidades e adequada para um ambiente heterogêneo de hardware e software. Além disso, é uma tendência tanto na indústria quanto na academia (CECCHET *et al.*, 2008).

No nível local, o foco está em maximizar o desempenho de cada réplica individual, ajustando o MPL em decorrência da carga. Em um nível global, o balanceador tenta maximizar o desempenho do sistema como um todo, decidindo como distribuir a carga entre diferentes réplicas. O desafio de realizar estes tipos de adaptações em um *middleware* se dá devido à reduzida informação que se tem disponível sobre as mudanças no comportamento e interior do banco de dados, tornando difícil detectar gargalos.

No nível local, cada instância do *middleware* possui um *pool* de conexões ativas para a réplica de banco de dados local. Isso determina o MPL já que cada transação requer sua própria conexão. Se existem mais requisições de transações do que conexões

ativas, estas são enfileiradas no *middleware*. Para se adaptar a mudanças nas características da carga de trabalho e da intensidade da mesma, o número de conexões ativas deve ser ajustado dinamicamente. Uma abordagem adaptada de (HEISS & WAGNER, 1991) foi utilizada. Nesta abordagem só é preciso monitorar a carga do banco de dados e a taxa de efetivação com a finalidade de ajustar o MPL apropriadamente. Outras abordagens (SCHROEDER, 2006) necessitam do conhecimento de componentes internos do banco de dados e, portanto, não são apropriadas. Utilizando esta abordagem, cada réplica adapta seu MPL individualmente de modo que possa executar sua carga da maneira mais eficiente, sem nenhum conhecimento interno da carga de trabalho, de detalhes internos do banco de dados, nem das outras réplicas que fazem parte do sistema.

No nível global, um balanceador de carga entre as réplicas será utilizado. O objetivo deste balanceador é distribuir a carga de trabalho entre as réplicas de modo que cada réplica não fique nem tanto sobrecarregada nem tanto ociosa, isto é, uma distribuição justa entre recursos. Isto é feito, redistribuindo a carga de forma que a capacidade ociosa de servidores distintos e heterogêneos fique tão uniforme quanto possível entre as réplicas. Portanto, este algoritmo de balanceamento de carga requer um meio para estimar a carga de dados atual em cada réplica.

Porém, como visto anteriormente, um dos grandes problemas em protocolos de replicação em banco de dados é a execução de transações conflitantes em réplicas distintas. Para cargas leves o problema não é tão relevante, porém para cargas pesadas, o algoritmo baseado em certificação acaba sofrendo altas taxas de aborto. Além disso, todos os algoritmos acabam tendo seu tempo de resposta aumentado e o throughput reduzido devido ao fato de que os conflitos, só serão resolvidos bem depois, na última fase (votação ou certificação). Portanto o balanceador de carga deve buscar tanto uma distribuição justa de carga sobre os recursos, como selecionar servidores preferenciais para execução de transações concorrentes de modo a evitar conflitos somente na última etapa de execução.

As próximas seções apresentam respectivamente a estratégia de controle de admissão baseado em mecanismos de *feedback*. Posteriormente a estratégia de balanceamento de carga baseado em conflito. Em seguida um novo algoritmo de balanceamento de carga desenhado para o cenário abordado é apresentado. Este

algoritmo, baseado nestas duas estratégias, é um exemplo de que o resultado da união das estratégias é superior que a soma destas isoladamente. Os benefícios de tal sinergia poderão ser analisados nos experimentos realizados.

4.2 Controle Adaptativo da carga de trabalho

Originalmente esta abordagem foi analisada em (ABOUZOUR *et al.*, 2010) e utiliza como medida de *feedback* o throughput do servidor. Porém, para o cenário estudado e a aplicação em banco de dados replicado esta medida não se mostra interessante. A combinação destes dois fatores gera como conseqüência uma alta taxa de aborto. O throughput não é uma medida adequada, pois, apesar de transações serem concluídas, uma grande parcela corresponde a transações abortadas. Uma medida muito mais adequada e natural neste caso seria o uso da taxa de efetivação. O restante deste algoritmo não sofre modificações em relação ao original, somente esta medida de *feedback*. O restante desta seção ilustra o funcionamento do algoritmo de controle de admissão adaptativo baseado em aproximação por parábola.

Este algoritmo tenta modelar a curva de efetivação por uma parábola. A parábola foi escolhida como um modelo por várias razões. Primeiro, uma parábola pode ter uma concavidade para baixo similar a função de efetivação encontrada em muitos servidores. A Figura 8 ilustra vários formatos da curva de efetivação devido a diferentes cargas de trabalho, dependência entre requisições, contenção de hardware e recursos de software. Segundo, a parábola é fácil de usar e fácil de calcular. Em terceiro lugar, não será necessário modelar a parte final onde o servidor pode ter níveis de degradação consideráveis no desempenho.

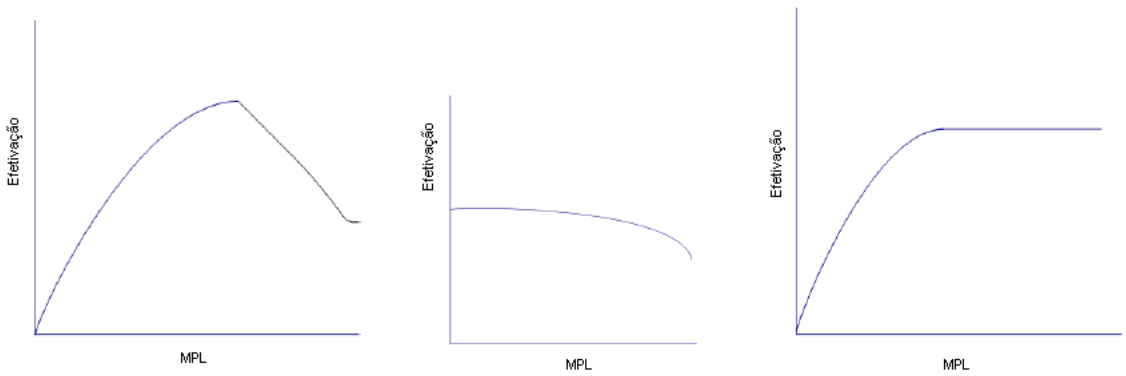

Figura 8 – Formas da curva de efetivação (ABOUZOUR *et al.*, 2010)

A fim de aproximar a função por uma parábola, três pontos de dados precisam ser coletados. Dois dos dados pontos será baseada em duas observações da taxa de efetivação para um determinado nível de multiprogramação. Para o terceiro ponto, a origem será utilizada. A equação da curva de efetivação como uma função parabólica é dada por:

$$E(t_i) = a * (mpl(t_i))^2 + b * (mpl(t_i)) + c$$

Os dois pontos que serão utilizados são ($mpl(t_i)$, $E(t_i)$) e ($mpl(t_{i-1})$, $E(t_{i-1})$). As seguintes equações serão utilizadas para resolver o problema *a*, *b* e *c*:

$$a = \frac{E(t_i) * mpl(t_{i-1}) - E(t_{i-1}) * mpl(t_i)}{mpl(t_i) * mpl(t_{i-1}) * (mpl(t_i) - mpl(t_{i-1}))}$$

$$b = \frac{E(t_{i-1}) - a * (mpl(t_{i-1}))^2}{mpl(t_{i-1})}$$

$$c = 0$$

Uma abordagem mais precisa seria a aproximação com o uso do método dos mínimos quadrados a partir de várias observações, porém por simplificação as equações acima foram utilizadas e os resultados obtidos satisfatórios.

Uma vez que os valores de *a, b*, *c* foram determinados o algoritmo tenta mover para um valor de MPL que maximize a taxa de efetivação. Este é o ponto onde a derivada é zero.

$$mpl(t_{i+1}) = \frac{-b}{2 * a}$$

Devido a possíveis ruídos nas medições, Figura 9, é possível que o coeficiente *a* seja positivo. Um coeficiente positivo modela a parábola com a concavidade para cima e, portanto, não é um modelo de curva de efetivação realista. Se a função parabólica calculada possuir concavidade para cima, então os dados coletados serão ignorados e o algoritmo apenas moverá para frente (ou para trás) por um número aleatório de passos entre 0 e Δ. Este procedimento é realizado cada vez que esta condição acontecer até que

funções parabólicas apropriadas com coeficiente *a* negativo apareçam. Alternar para frente e para trás, é necessário, pois os dados que foram coletados não dão uma direção correta para em qual direção ir. Alternar em ambas as direções fornece certa dose de aleatoriedade.

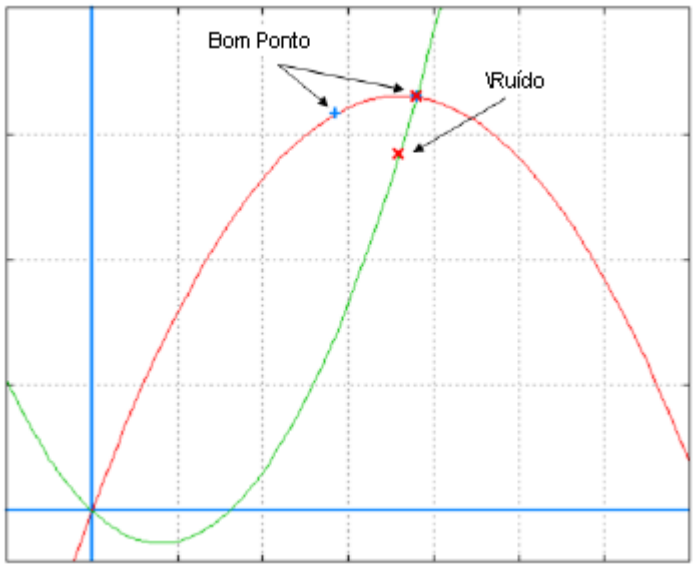

Figura 9 – Ruído na seleção de pontos para modelar a parábola (ABOUZOUR *et al.*, 2010)

Por outro lado, o ruído nos dados de medição, também, pode levar a resultados onde novo valor é muito longe do valor atual. Neste caso, o parâmetro α será utilizado para limitar o tamanho do passo. O valor de α é basicamente uma percentagem do número atual do MPL. Por exemplo, um α com valor de 0,3 limita o tamanho de passos para 30% do número atual do MPL. Portanto, as ações do algoritmo podem ser resumidas da seguinte maneira (ABOUZOUR *et al.*, 2010):

$$mpl(t_{i+1}) = \begin{cases} \min(-b/2*a, (\alpha+1)*mpl(t_i)) & se\ a<0\ e\ -b/2*a \neq mpl(t_i) \\ mpl(t_i) \pm rand(0,\Delta) & caso\ contrario \end{cases}$$

As vantagens da aproximação por parábola são:

- Rápida convergência.
- O valor inicial da variável de controle do MPL não afeta as decisões do algoritmo (métodos baseados em *Hill Climbing* possuem este problema).
- O algoritmo pode lidar com mudanças de carga de trabalho e não tem problemas com mudanças nas condições de carga de trabalho.

4.3 Balanceamento da carga de trabalho

A idéia central da proposta de balanceamento de carga consiste em diminuir o conflito entre transações concorrentes em servidores distintos com o objetivo de diminuir a taxa de aborto das transações, assinalado como um dos principais problemas dos algoritmos de replicação (ZUIKEVICIUTE & PEDONE, 2008).

O balanceador de carga deve se basear nas duas observações apresentadas anteriormente: (a) Se transações conflitantes são submetidas para o mesmo servidor, o controle de concorrência local será responsável por serializar operações conflitantes adequadamente, diminuindo abortos. (b) Na falta de conflitos, porém, o desempenho é melhorado se as transações executam concorrentemente em diferentes réplicas.

Uma restrição que este balanceador de carga deve respeitar, diz respeito ao número máximo de transações ativas que cada réplica pode suportar. Este valor não apenas servirá como um limite superior para a carga que cada servidor é capaz de suportar, mas também será utilizado como ferramenta útil de distribuição da carga de trabalho entre as réplicas. Antes de apresentar o algoritmo alguns conceitos devem ser analisados.

4.3.1 Cálculo do peso da transação

Existem duas abordagens para calcular o peso de uma transação. A primeira consiste em investigar e observar as características internas das transações. Características tais como número de operações, quantidade de atualizações, quantidade

de *locks* necessários e a partir destes dados, extrair um valor que corresponda ao peso da transação. Outra abordagem consiste em observar externamente estas características. O indicador usualmente utilizado para tal finalidade é o tempo de resposta da transação (MILAN-FRANCO *et al.*, 2004).

Para uma proposta de balanceamento de carga, a segunda abordagem tem se mostrado adequada (ORLEANS & ZIMBRÃO, 2009) por possuir um cálculo mais simples, direto e rápido. Porém este cálculo não deve ser estático, as características do sistema mudam com o tempo, registros são inseridos ou removidos, alterações estruturais ou mudanças de hardware podem ocorrer. Outra alteração de natureza externa que pode ocorrer diz respeito à carga na qual o sistema está submetido. A carga de trabalho submetida pode alterar tanto de intensidade, com redução ou aumento, ou em termos de conflito ocasionando uma menor ou maior taxa de conflito.

Portanto o peso de uma transação não pode ser pensado isoladamente, este valor sempre deve ser possuir o referencial do sistema, referencial este que sofre também suas influências e se modifica ao longo do tempo. A estimativa do tempo de resposta utilizado neste trabalho se baseou no trabalho desenvolvido em (ORLEANS *et al.*, 2008). O tempo de resposta atual de uma determinada transação é estimado baseado nas últimas quatro observações, utilizando a seguinte fórmula:

$$RE(t_i) = 0.4 * R(t_{i-1}) + 0.3 * R(t_{i-2}) + 0.2 * R(t_{i-3}) + 0.1 * R(t_{i-4})$$

Onde $RE(t_i)$ corresponde ao tempo de resposta estimado da transação que requer o serviço, e as cláusulas R as durações reais das últimas transações do mesmo tipo. Nesta fórmula, as quatro últimas medidas reais são utilizadas para calcular a estimativa, tentando obter um tempo médio que reflete o comportamento recente do servidor de banco de dados.

4.3.2 Definindo grau de similaridade entre servidores

A carga do servidor é dada pelo somatório do peso das transações designadas a ele (peso agregado) em um dado momento.

$$w(S_k, t) = \sum_{T_i \in S_k} RE(T_i, t)$$

Para comparar a carga de dois servidores será utilizado um fator *f*, $0 < f \leq 1$. Os servidores S_i e S_j terão carga similar em um tempo t se as seguintes condições forem satisfeitas (ZUIKEVICIUTE & PEDONE, 2008):

$$f \leq w(S_i, t) / w(S_j, t) \leq 1 \; ou \; f \leq w(S_j, t) / w(S_i, t) \leq 1$$

Onde *w* representa o peso agregado de um determinado servidor em um dado momento. Por exemplo, com $f = 0.5$ permite que a diferença de carga entre duas réplicas seja até 50%. A escolha de *f* depende principalmente das características da carga de trabalho: o número de transações conflitantes, sua complexidade e a carga sobre o sistema.

O algoritmo tenta inicialmente designar cada transação T_i para a réplica contendo transações conflitantes com T_i. Se não há conflitos, o algoritmo tenta balancear a carga entre as réplicas. O fator *f* é importante, pois será utilizado na comparação de servidores. Este fator será utilizado para definir a similaridade de sobrecarga dos servidores, uma informação que será muito útil no algoritmo de balanceamento.

4.3.3 Detectando possíveis conflitos

Devido ao fato de ser custoso calcular e identificar, principalmente em nível de *middleware*, os conflitos entre transações concorrentes, nesta proposta o importante será identificar as réplicas onde se tem maior probabilidade de que tais conflitos ocorram. Portanto a questão fundamental é: dado duas transações T_i e T_j, deve-se descobrir (i) se tais transações possuem operações que possam conflitar e (ii) caso tal condição ocorra, qual a chance que isto se torne realidade. A técnica apresentada a seguir corresponde a uma adaptação do trabalho apresentado em (MUÑOZ-ESCOÍ *et al.*, 2006) e (ZUIKEVICIUTE & PEDONE, 2008).

Para a primeira questão, neste trabalho, uma simplificação será adotada. Duas transações terão possibilidade de conflitar se houver uma intersecção entre as tabelas

acessadas por ambas as transações. Da mesma forma, uma determinada transação terá a possibilidade de conflitar com um determinado servidor, se houver uma interseção entre todas as tabelas acessadas por suas respectivas transações locais e a transação em questão.

Para a segunda questão o peso das transações será utilizado. Esta abordagem traz consigo uma série de simplificações e facilidade de cálculo, tornando adequada para um *middleware* de balanceamento de carga onde pouca informação sobre o sistema é conhecido. Para medir a chance de que um conflito ocorra será utilizada a taxa de sobreposição entre uma transação T_i e um servidor S_i dado por $TS(T_i, S_k)$.

Considere $C_T(S_k, t)$ o subconjunto de S_k contendo transações conflitantes com T_i, e é dado por:

$$C_{T_i}(S_k, t) = \{T_j \mid T_j \in S_k \wedge T_i \sim T_j\}$$

Agora considere $w(C_{T_i}(S_k, t))$ o peso agregado de todas as transações que conflitam com T_i em S_k, dado por:

$$w(C_{T_i}(S_k, t)) = \sum_{T_j \in C_{T_i}} RE(T_j, t)$$

A taxa $TS(T_i, S_k)$ é definida como a razão entre o peso da transação T_i e o peso agregado de todas as transações conflitantes com T_i para um determinado servido S_k, portanto pode ser expresso pela seguinte fórmula:

$$TS(T_i, S_k) = \frac{RE(T_i, t)}{w(C_{T_i}(S_k, t))}$$

4.3.4 Controlando a variância entre servidores

Um dos grandes problemas das abordagens anteriores de balanceamento de carga baseado em conflito, é que na ocorrência do cenário estudado de sobrecarga e taxa de conflito elevada, alguns servidores se tornam preferenciais devido ao balanceamento baseado em conflito. Deixando algumas réplicas ociosas e degenerando os algoritmos

update every-where em um algoritmo de cópia primária. Este problema não toma proporções maiores devido ao limite superior imposto pelo nível de multiprogramação, isto é, algumas réplicas ficarão sobrecarregadas até o limite imposto pelo MPL e não indefinidamente como acontecia em trabalhos anteriores. Porém, até que este limite seja alcançado, este servidor concentrará boa parte da carga, e quando o limite for alcançado, outro servidor será escolhido e passará a ser o novo servidor preferencial e assim por diante. Para contornar este problema um limite de diferença de carga entre as réplicas será utilizado. O objetivo é que nenhum servidor fique mais sobrecarregado que outro em um determinado momento, este valor será denotado por v e assumirá valores percentuais em relação às outras réplicas. Por exemplo, assumindo um valor de 30%, a diferença entre o servidor mais ocioso e o servidor mais sobrecarregado não poderá passar de 30%. Porém falta definir como calcular a sobrecarga ou ociosidade de uma determinada réplica. Como os servidores possuem características distintas de hardware, o uso do tempo de resposta não seria uma boa escolha, além disso, esse valor reflete o tempo independente do sistema que foi atribuído. Uma escolha natural seria basear esta estimativa no MPL e no número de transações locais concorrentes no servidor e será dado pela seguinte fórmula:

$$CD(S_i, t) = mpl(S_i, t) - CA(S_i, t)$$

Onde *CD* corresponde ao número de conexões disponíveis e *CA* o número de conexões ativas. Esta fórmula além de expressar o quanto uma réplica está sobrecarregada, ou seja, quanto menor o valor de *CD*, mais sobrecarregado o servidor estará. Também, implicitamente, considera a capacidade de hardware destes. Pois quanto maior a capacidade, maior poder de processar transações este servidor terá e, portanto, maior será a taxa de efetivação e o mpl. Como as características de um servidor para outro pode variar, servidores mais robustos que possuem um valor de mpl muito elevado serão escolhidos em detrimento de servidores menos robustos. Ou seja, o valor de v pode privilegiar alguns servidores em detrimento de outros e deve ser cuidadosamente escolhido baseado na heterogeneidade do sistema. Para um determinado momento o seguinte objetivo deve ser buscado:

$$(1 - v) \leq CD(S_i, t) / CD(S_j, t) \leq 1 \; ou \; (1 - v) \leq CD(S_j, t) / CD(S_i, t) \leq 1$$

4.4 Algoritmo

Considere as réplicas $S_1, S_2, ..., S_n$. Para designar alguma transação T_i a algum servidor em um determinado tempo t executar os passos 1 ao 7.

1. Considerando $O(t) = \{S_i \mid CD(S_i, t) > 0\}$ o conjunto de réplicas que ainda não atingiram seu valor máximo de mpl.
2. Se $O(t) = 0$ rejeitar a transação e enviá-la para uma fila de espera.
3. Considere $V(t) = \{S_i \mid S_i \in O(t)\ e\ ((CD(S_i, t) - 1) * (1 + v)) \geq \max_{l \in O(t)} CD(S_l, t)\}$ o conjunto de servidores que podem receber a transação de modo a manter as restrições *v* de variância.
4. Se T_i for uma transação de consulta, designar T_i para qualquer servidor em *V(t)*, caso contrário executar o passo 5.
5. Considerando $W(t) = \{S_k \mid S_k \in V(t)\ e\ w(S_k, t) \geq \max_{l \in V(t)} (w(S_l, t)) * f\}$ o conjunto de réplicas com o maior peso agregado $w(S_k, t)$ no tempo t.
6. Se $|W(t)| = 1$, então designar T_i para a réplica em $W(t)$.
7. Se $|W(t)| > 1$, então considere $C_W(T_i, t)$ o conjunto de réplicas contendo transações conflitantes com T_i em $W(t)$:
    a. Se $|C_W(T_i, t)| = 0$, designar T_i para o servidor S_k em $W(t)$ que possuir o menor peso agregado $w(S_k, t)$.
    b. Se $|C_W(T_i, t)| = 1$, designar T_i para a réplica em $C_W(T_i, t)$.
    c. Se $|C_W(T_i, t)| > 1$, designar T_i para a réplica S_k em $C_W(T_i, t)$ com maior peso agregado de transações com possíveis conflitos com T_i; Se várias réplicas em $C_W(T_i, t)$ satisfazem esta condição, designar T_i para alguma destas réplicas.

O passo 1 do algoritmo consiste em justamente identificar aquelas réplicas que possuem conexões disponíveis para serem utilizadas. Isto acontece quando o número de conexões em uso for menor que o mpl. Caso nenhuma conexão esteja disponível, no passo 2, a transação será enviada para uma fila de espera onde uma política FCFS será utilizada para servir transações enfileiradas.

Dado que os servidores com conexões disponíveis sejam identificados, o passo 3, procura escolher aqueles servidores no qual a utilização de uma conexão disponível não causará um desvio ou uma diferença muito grande em relação às outras réplicas. Ou seja, este passo busca alocar as transações de maneira que a diferença entre o servidor mais sobrecarregado e o servidor mais ocioso não passe do fator *v* de variância previamente definido. Esta restrição é importante, pois evita que através do balanceamento de carga baseado em conflito, onde transações conflitantes tendem a serem alocadas na mesma réplica, servidores preferenciais fiquem sobrecarregados enquanto outros ociosos. Evitando, assim, que os algoritmos de replicação se degenerem em uma cópia primária. Outro benefício desta restrição é evitar, que em um ambiente de réplicas bastante heterogêneas, servidores menos robustos sofram com cargas comparativamente maiores enquanto servidores mais robustos fiquem ociosos. Com isto, busca-se nivelar os servidores baseado na relação carga e capacidade de processamento, deixando todo o sistema em um nível homogêneo de utilização.

O passo 4 do algoritmo diferencia as transações de consulta das transações de atualização. A primeira corresponde àquelas com apenas operações de leitura, enquanto a segunda mescla tanto operações de leitura quanto de escrita. Operações de leitura não passam por um balanceamento de carga baseado em conflito, isto se deve ao fato de que só serão processadas no servidor de delegação, devido à ausência de conjunto de escrita. Por outro lado as transações de atualização quando executadas concorrentemente com transações conflitantes em servidores distintos levam às taxas elevadas de abortos como já apresentado. Sendo necessário um tratamento especial que será dado pelo balanceador nos próximos passos.

O passo 5 do algoritmo procura aquelas réplicas com maior peso agregado, a partir do conjunto de servidores previamente selecionados. Dado que a carga dos servidores aumentará dentro de uma variância predeterminada, o algoritmo busca por aquela réplica mais sobrecarregada, isto se deve ao fato de que será esta réplica com mais chances de se encontrar uma quantidade grande de transações conflitantes com a transação sendo submetida. Para comparar os servidores o fator *f* de similaridade será utilizado de modo que as transações podem ser consideradas iguais em termos de carga dentro de um determinado intervalo predefinido. Importante ressaltar a importância da escolha do fator *v* de variância, este valor deve ser adequadamente escolhido de modo que se um valor muito alto for escolhido, o este passo do algoritmo sobrecarregará

algumas réplicas em detrimento de outras, se for muito baixo, o diferencial do balanceamento baseado em conflito não será alcançado: transações conflitantes não serão alocadas no mesmo servidor, isto é, serão alocadas em vários servidores distintos.

No passo 6, caso apenas um servidor seja encontrado, designar a transação para ele próprio, caso contrário executar o passo 7.

No passo 7 os servidores em $W(t)$ que possuam possibilidade de conflitar com a transação submetida são selecionados. Três condições distintas podem ocorrer: (a) não existe nenhuma réplica conflitante, neste caso alocar a transação em algum servidor em $W(t)$ com menor peso agregado; (b) caso apenas um servidor com possível conflito seja encontrado, designar a transação para este servidor; (c) Caso mais de um servidor conflitante seja encontrado, designar a transação para o servidor que possuir maior peso agregado de transações com possíveis conflitando com T_i, através da taxa $TS(T_i,S_k)$. Neste ponto o algoritmo atua de forma que quanto mais transações conflitantes e quanto maior o somatório do peso agregado destas transações, mais chances este servidor de ser escolhido para receber a transação. Ou seja, o balanceador tenta alocar transações com possíveis conflitos na mesma réplica, evitando conflitos de transações concorrentes em servidores distintos.

4.5 Variáveis do algoritmo

Na proposta apresentada, quatro variáveis são utilizadas para ajustar o comportamento do algoritmo às necessidades particulares. A Tabela 2 lista estas variáveis e seus respectivos valores a partir da pesquisa em diversos trabalhos na literatura. Estes valores levam em consideração o cenário sendo estudado: sobrecarga do sistema e altas taxas de conflito.

O trabalho realizado em (ABOUZOUR *et al.*, 2010) explorou o relacionamento entre α e Δ, através de vários experimentos chegou-se a conclusão de que valores ideais seriam 30% e 5 respectivamente. Resultados similares podem ser encontrados em (GUPTA & HARCHOL-BALTER, 2009).

O trabalho realizado em (ZUIKEVICIUTE & PEDONE, 2008), buscou um balanceamento híbrido entre maximização de conflitos e maximização de paralelismo. Um valor pequeno para f tende a maximizar os conflitos, isto é, um balanceamento atribuindo transações com possíveis conflitos sempre ao mesmo servidor, deixando vários outros ociosos. Por outro lado um valor alto para f tende a maximizar o paralelismo nos diversos servidores, não atribuindo transações conflitantes à mesma réplica. O trabalho chegou a um resultado equilibrado com o valor $f = 0.5$.

Tabela 2 - Variáveis do balanceamento de carga adaptativo com restrições de conflito

Variável	Descrição	Valor
α	Limitador da mudança do mpl, caso o novo valor estimado fique muito distante do valor atual.	0.3
Δ	Tamanho do passo aleatório, caso a parábola do controle adaptativo do mpl tenha concavidade para baixo.	5
f	Percentual que define a similaridade de peso das diferentes réplicas	0.5
v	Percentual que define a variância máxima de conexões disponíveis para as diferentes réplicas.	0.4

O último parâmetro está relacionado ao desbalanceamento máximo entre servidores. O trabalho desenvolvido em (COSTA & FURTADO, 2008) chegou à conclusão de que este valor deve estar idealmente entre 35% e 40%.

4.6 Contribuições

O diferencial deste trabalho está em tentar correlacionar duas áreas ainda não muito exploradas (CECCHET *et al.*, 2008) em banco de dados replicados: (i) controle adaptativo da multiprogramação e (ii) balanceamento de carga baseada em conflitos para algoritmos *eager replication update-everywhere*. Espera-se que os benefícios de ambas as estratégias criem condições ideais para controlar o sistema replicado em

situações de sobrecarga com transações conflitantes. Isto é, diminuir a taxa de aborto e o tempo de resposta de transações de atualização, apontado por vários trabalhos (WIESMANN & SCHIPER, 2005), (JUÁREZ *et al.*, 2008) e (KEMME & ALONSO, 2010) como um dos grandes problemas em banco de dados replicados e já discutido no capítulo 3.

Os trabalhos anteriores falham justamente ao não perceberem que somente atuando na causa do problema: execução de transações concorrentes conflitantes em servidores distintos, é que um resultado significativo pode ser alcançado. Além disso, quando a carga de trabalho atinge níveis intoleráveis, o balanceador não tem alternativa senão impedir que novas cargas de trabalho entrem no sistema.

Apesar das idéias de balanceamento baseado em conflito e controle adaptativo da carga existir em vários trabalhos como citado anteriormente, o algoritmo apresentado anteriormente traz contribuições originais e de fundamental importância para o cenário de sobrecarga e conflito estudado. Foi na união destas idéias que se pôde chegar ao algoritmo adaptado de trabalhos anteriores e, agora, desenhado para o cenário estudado. Especificamente, os passos 1, 2, 3 são todos de autoria deste trabalho e influenciam todos os passos subseqüentes. Além de adaptações nos passos 5 e 7, como a forma de calcular o peso das transações, taxa de sobreposição e controle da variância. Desta forma cria-se um encadeamento novo no fluxo e restrições no balanceamento da carga.

A conclusão deste trabalho traz como contribuição científica para a comunidade acadêmica, um novo balanceador de carga adaptativo para banco de dados replicados em cenários de sobrecarga e taxa de conflito elevada capaz de manter o sistema harmônico, balanceado, com baixas taxas de aborto e estável sem picos de saturação, isto é, com um nível mínimo satisfatório de desempenho. Necessidades básicas para ambições maiores de níveis de serviço. Os conceitos aqui apresentados podem ser utilizados para pesquisas na área de banco de dados, como para sistemas de tempo real.

Como, também, fruto deste trabalho um simulador completo para banco de dados replicados foi desenvolvido. Onde tanto a parte lógica como a física são modeladas. Novas estratégias poderão ser analisadas e avaliadas de maneira mais rápida e controlada através deste simulador minimizando o tempo de pesquisa e experimentação, já que validações e avaliações pré-eliminares podem ser realizadas antes da implementação de estratégias de fato.

O capítulo seguinte apresenta o simulador que será utilizado e os experimentos que foram realizados para validar a abordagem proposta com o objetivo de comparar e mostrar como o balanceamento proposto resolve a questão principal de desempenho levantada.

5. Experimentos

Neste capítulo a seção 4.1 apresenta a estrutura do simulador, ele será utilizado para executar os experimentos com objetivo de validar a proposta de balanceamento. Na seção 4.2 uma descrição dos experimentos é fornecida, enquanto na seção 4.3 os resultados obtidos são apresentados e analisados.

5.1 Simulador

O simulador está baseado em um modelo completo de sistema de gerenciamento de banco de dados replicado. Portanto, contempla seus principais elementos: usuários (fontes das transações), sistemas de comunicação em grupo, estratégias de replicação e recursos físicos para armazenamento e processamento dos dados (disco e CPU). Além, das características da carga de trabalho e do banco de dados local.

A simulação foi escolhida em detrimento de uma implementação de fato por dois motivos principais: (i) experimentação da proposta em ambiente controlado e, principalmente, (ii) análise independente das características específicas de diferentes implementações dos algoritmos de replicação. Em relação ao primeiro item, um ambiente simulado e controlado é de fundamental importância para testar todas as premissas e cenários desejados nos experimentos, além de um rápido retorno acerca das estratégias que estão sobre análise. Além disso, as diversas implementações das estratégias de replicação adicionam características próprias em relação aos algoritmos clássicos de replicação. A abordagem adotada nesta dissertação e influenciada pelo trabalho desenvolvido em (WIESMANN & SCHIPER, 2005) buscou explorar as características destes algoritmos sem tal interferência. A simulação mostrou-se um ambiente isento destas questões.

O custo do controle de concorrência, controle de admissão, entre outros serão ignorados. O objetivo é estudar como o desempenho dos algoritmos de replicação é

afetado e influenciado pelo balanceador de carga proposto. Portanto, somente os elementos que interferem significativamente e são relevantes neste contexto serão analisados. Assim como, não serão abordadas questões relativas à degradação do sistema operacional.

Portanto o simulador é dividido em cinco partes principais. Esta divisão é baseada nos trabalhos desenvolvidos em (AGRAWAL *et al.*, 1985), (HOLLIDAY *et al.*, 1999), (WIESMANN & SCHIPER, 2005) e (JUÁREZ *et al.*, 2008):

- Módulo do Cliente
- Módulo lógico de banco de Dados local
- Módulo Físico
- Módulo de comunicação
- Módulo de replicação

5.1.1 Módulo do Cliente

Clientes são simplesmente as fontes das transações. Os clientes submetem uma transação, aguardam até que a transação seja processada, em seguida esperam um determinado tempo e iniciam o ciclo novamente. Um cliente só pode submeter uma transação por vez – múltiplas fontes são modeladas por múltiplos clientes.

O parâmetro mais importante dos clientes é o tempo entre transações. Se um cliente inicia uma transação no tempo t_1, uma vez a transação finalizada o cliente espera um tempo $t_1 + d$ antes de enviar uma nova transação. A variável d é uma variável aleatória com distribuição exponencial. Desta forma é possível criar vários cenários de carga variando o tempo médio da variável aleatória.

Portanto um cenário de sobrecarga pode ser modelado por duas maneiras distintas: aumentar o número de clientes e/ou diminuir o valor da variável d. A abordagem que será utilizada para simular um cenário de alta taxa de conflito será explicado mais adiante.

Tipicamente, cada servidor tem vários clientes associados. Na simulação os clientes não consomem banda de rede. Da mesma forma não consomem CPU, este consumo é considerado parte do *parsing* e otimização da transação e não é relevante para esta simulação. Os clientes coletam e computam todas as estatísticas, controlam a execução da simulação e a configuração dos parâmetros e finalizam a simulação quando os resultados obtidos estão dentro do intervalo de confiança desejado.

5.1.2 Módulo lógico de Banco de Dados local

O módulo lógico de banco de dados simula um sistema de banco de dados em uma determinada réplica. O módulo lógico de banco de dados local é dividido em três partes principais:

- **Sub-módulo de sistema de banco de dados:** contempla as características relevantes de software do sistema, as características do banco de dados (tamanho e granularidade), mecanismo de controle de número de transações ativas (Controle de Admissão) e o algoritmo de controle de concorrência.

- **Sub-módulo de E/S:** controla todas as operações de leitura e escrita de itens ao banco de dados, além de simular um sistema de *cache*.

- **Sub-módulo de transação:** contempla o comportamento e os requisitos de processamento das transações. Uma transação pode ser descrita por duas características: lógica e física. Na lógica, existem requisições de leitura e escrita para o controle de concorrência. Na física, requisições de acesso para itens físicos no disco, associado com processamento de CPU para cada item acessado. Estas características são descritas de maneira probabilística.

Este modelo pode ser visto na Figura 10. Existe um número fixo de clientes (terminais) de onde as transações se originam. Além disso, existe um número máximo de transações ativas, em um determinado momento, no sistema: *mpl – multiprogramming level*.

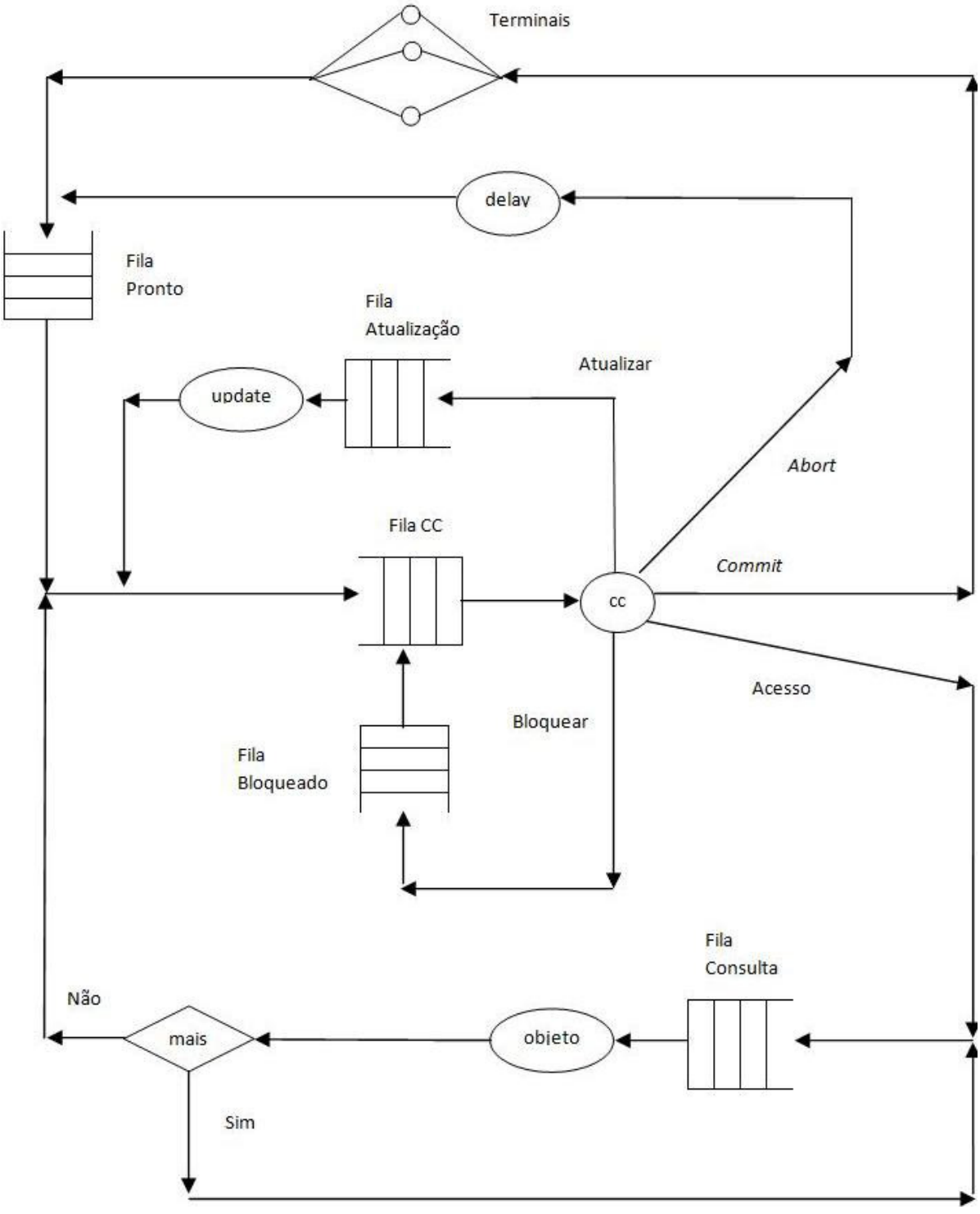

Figura 10 - Modelo Lógico de um Banco de Dados (AGRAWAL *et al.*, 1985)

Uma transação é considerada ativa se esta está recebendo um serviço ou esperando por um dentro do sistema. Quando uma nova transação chega, se o sistema já está com o número máximo de transações ativas (*mpl*), ela entra na fila de transações

prontas onde espera por uma transação ativa terminar (transações na fila de prontas não são consideradas ativas). A transação, então, entra na fila de controle de concorrência e faz o primeiro pedido de acesso. Se a requisição feita ao controle de concorrência é aceita, a transação é enviada para a fila de consulta a objetos e acessa o primeiro objeto. Se mais de um objeto tiver que ser acessado, a transação irá passar por esta fila o número de vezes necessário. Quando a próxima requisição ao controle de concorrência é necessária, a transação entra novamente na fila do controle de concorrência. Na modelagem foi assumido, por conveniência, que a transação executa todas as operações de leitura antes das operações de escrita.

Se o resultado da requisição ao controle de concorrência é que a transação deva esperar por um recurso, então a transação entra na fila de bloqueado até que possa voltar ao processamento. Se o resultado for para reiniciar a transação, ela volta para a fila de pronto após um *delay* determinado para o reinicio. Neste caso, a transação é reiniciada desde o inicio, isto é, todas as operações já executadas serão refeitas. Após a execução de todas as operações, a transação estará completa e o mecanismo de controle de concorrência decide efetivar a transação. Se a transação é somente de leitura, ela é finalizada.

Se um ou mais objetos foram escritos durante a execução, ela primeiro entra na fila de atualização e escreve todas as atualizações postergadas no banco de dados. Atualizações postergadas são utilizadas neste modelo, pois é necessário para suportar qualquer algoritmo de controle de concorrência – todos os algoritmos trabalham corretamente com atualizações postergadas, mas nem todos os algoritmos trabalham com esquemas de recuperação que permitam atualizações on-line ao invés de serem executadas no ponto de efetivação.

Uma transação é modelada de acordo com o número de objetos que são lidos e escritos. O número de objetos lidos se dá por uma distribuição uniforme. Esses objetos são aleatoriamente escolhidos dentre todos os objetos do banco de dados. Existe associado uma probabilidade de que um objeto lido irá também ser escrito. O tamanho do banco de dados é dado por um valor fixo.

Para modelar um cenário de taxas de conflito elevada, duas abordagens também existem. A primeira consiste em diminuir a quantidade de itens no banco de dados, desta forma, com uma sobrecarga de requisições, vários itens serão acessados e escritos

concorrentemente. A segunda abordagem consiste em diferenciar os itens de dados que serão acessados, atribuindo uma probabilidade maior de utilização de acesso e escrita de alguns determinados objetos do banco.

Neste trabalho o custo do controle de concorrência não será considerado. Será assumido que este custo é ínfimo comparado com o custo de acessar um objeto. O trabalho desenvolvido em (AGRAWAL *et al.*, 1985) mostra que o custo de algoritmos baseados em *locking* é comparável ao de algoritmos otimistas, a grande diferença está em quando estes custos acontecem e que o custo de detecção de *deadlock* não causa um *overhead* muito grande. Outros custos que não serão considerados é a degradação do sistema operacional e do banco de dados quando submetido a uma carga grande de transações concorrentes.

5.1.2.1 Algoritmos de Controle de Concorrência

Dois algoritmos de concorrência serão abordados neste trabalho: Bloqueio de duas fases (*2PL*) e multiversão (*MVCC*).

Bloqueio de duas fases (2PL): é uma técnica de controle de concorrência com o uso de *locks*. Um *lock* é um objeto de sistema associado com um recurso compartilhado como um item de dado, uma linha, ou uma página. Em banco de dados, um *lock* a um objeto do banco de dados precisa ser adquirido antes do acesso ao mesmo. O correto uso de *locks* previne indesejáveis, incorretas ou inconsistentes operações sobre os recursos compartilhados por outras transações concorrentes. No protocolo *2PL*, as transações controlam os *locks* em duas fases distintas durante a execução da transação:

- Fase de Expansão: O número de *locks* só pode crescer. Os locks são adquiridos e nenhum *lock* é liberado.
- Fase de encolhimento: Os *locks* são liberados e nenhum *lock* é adquirido.

A propriedade de seriabilidade é garantida para um histórico de transações que obedeçam este protocolo (BERNSTEIN *et al.*, 1987).

Multiversão (MVCC): é uma técnica de controle de concorrência onde as atualizações são realizadas sem sobrescrever um determinado item de dados, mas marcando o valor antigo como obsoleto e adicionando a nova versão. Portanto existem várias versões

armazenadas, mas apenas uma é a última. Transações de leitura tipicamente usam *timestamp* ou o identificador da transação para determinar qual estado do banco de dados será lido, evitando o uso de *locks* para leitura. Escritas afetam a versão futura, porém os dados estarão consistentes, pois as escritas ocorrem no último identificador da transação.

Resumindo, MVCC provê a cada usuário conectado um *snapshot* do banco de dados. Qualquer mudança feita não será vista por outros usuários até a transação ser efetivada.

No simulador o controle de concorrência é modelado utilizando as duas estratégias previamente apresentadas. Para o 2PL, cada requisição ao controle de concorrência corresponde a um pedido de *lock* para um objeto, e estas requisições se alternam com acessos aos objetos. Todos os *locks* são liberados juntos ao fim da transação (depois que as atualizações postergadas são realizadas). Filas de espera por *lock* e grafos de espera são mantidos no simulador. A cada novo bloqueio de transação este grafo é atualizado. Caso um *deadlock* seja detectado, alguma política de reinício deverá ser executada, geralmente a transação mais recente no sistema. Para o controle de concorrência otimista, o primeiro pedido ao controle de concorrência é aceito imediatamente; todos os acessos a objetos são executados sem nenhuma intervenção do controle de concorrência. Somente depois do último acesso, a transação volta para a fila do controle de concorrência, neste momento, uma validação é executada (seguido, caso haja sucesso, pelas atualizações postergadas).

5.1.3 Módulo Físico

O módulo físico representa o hardware de um único servidor. Existe uma instância deste módulo em cada servidor do sistema. Cada servidor é simulado usando dois recursos básicos: CPU e discos. Estes dois recursos são utilizados por serviços lógicos de mais alto nível como: controle de concorrência, acesso aos objetos e atualizações. O recurso CPU modela unidades de processamento. Os recursos de disco são utilizados pelo módulo lógico de banco de dados. Operações básicas, de entrada e saída, utilizam ambos os recursos.

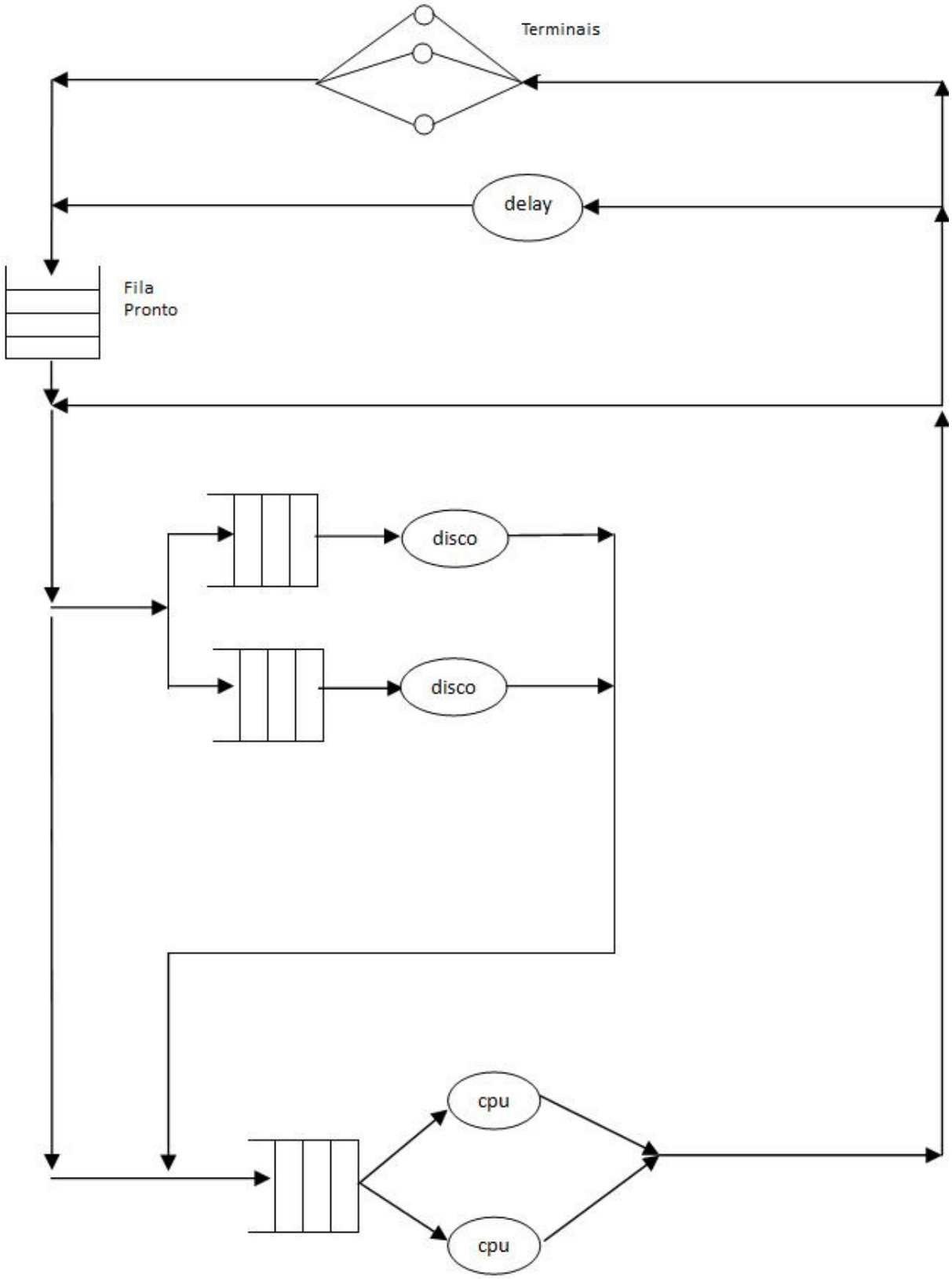

Figura 11 - Modelo Físico (AGRAWAL *et al.*, 1985)

A quantidade de tempo de uso de cada recurso (CPU e disco), por serviço, é especificada como parâmetros. O modelo físico é mostrado na Figura 11. Este modelo consiste em uma coleção de terminais, vários servidores de CPU e discos. A quantidade de cada recurso também é especificada via parâmetros.

Quando a transação precisa dos serviços da CPU, ela é atribuída a uma CPU livre, caso contrário a transação espera até que alguma fique livre. Portanto, as CPUs podem ser vistas como um conjunto de servidores, servindo uma única fila global de requisição a CPU. Requisições a CPU são servidas segundo a política FCFS, exceto as requisições do controle de concorrência que possuem prioridade em relação às outras. O modelo de E/S é probabilístico em relação a todos os discos. Existe uma fila associada a cada disco. Quando uma transação precisa de um serviço, aleatoriamente um disco é escolhido e a transação é enviada para a respectiva fila. Requisições ao disco também são servidas segundo uma política FCFS.

Ler um objeto leva um tempo igual a um acesso ao disco seguido por um processamento na CPU. Escrever um objeto leva um tempo igual a um processamento na CPU no momento do pedido de escrita e acesso ao disco no momento de atualizações postergadas. Assume-se que as transações mantêm listas de atualizações postergadas em *buffers* na memória principal.

5.1.4 Módulo de Comunicação

O módulo de comunicação modela toda a iteração com a rede. Existe uma instância desta para cada servidor. Primitivas de mais baixo nível como ponto-a-ponto e *multicast* são implementadas neste módulo, o uso de primitivas de mais alto nível como *total order broadcast* (TOB) são resultados da execução destas primitivas de mais baixo nível. Os algoritmos são simulados assumindo que não existam falhas na comunicação de mensagens.

Este módulo faz uso de dois recursos físicos: CPU e rede. Enquanto a CPU é um recurso de cada servidor, a rede é um recurso compartilhado. Para a rede existe uma fila de espera baseada em uma política FIFO.

O envio de mensagens é modelado em três passos: primeiro a mensagem a ser enviada é processada no servidor de envio, então a mensagens é enviada pela rede e por último a mensagem é processada no servidor que a recebe (WIESMANN & SCHIPER, 2005). Isso significa que três recursos estão envolvidos na simulação de envio de uma única mensagem: CPU no servidor que está enviando a mensagem, o recurso de rede e finalmente a CPU no servidor que está recebendo a mensagem. Desta forma é possível modelar a contenção de CPU e rede entre as mensagens, mas também a contenção entre o sistema de comunicação e o banco de dados.

5.1.5 Módulo de Replicação

Este módulo representa a estratégia de replicação. Existe uma instância deste módulo por servidor. Dependendo da estratégia de replicação, uma implementação diferente para este módulo é usada. Cada estratégia de replicação é representada por uma subclasse da classe de replicação. Os algoritmos de replicação baseados em certificação e votação foram implementados no simulador.

5.2 Configuração geral dos Parâmetros de simulação

Vários cenários serão utilizados para comparar o desempenho e validar a proposta de balanceamento de carga sobre as duas técnicas de replicação estudadas. Todas as técnicas irão ser analisadas nas mesmas condições de simulação, isto é, serão analisadas nas mesmas condições operacionais e de infra-estrutura. No caso de experimentos que necessitarem de customização, os valores serão especificados na descrição do respectivo experimento.

As simulações são executadas baseadas nos parâmetros operacionais da Tabela 3, com um ou mais parâmetros sendo variáveis ao longo do experimento. As configurações dos parâmetros do banco de dados são baseadas na literatura (AGRAWAL *et al.*, 1985), (HOLLIDAY *et al.*, 1999), (WIESMANN & SCHIPER, 2005) e (JUÁREZ *et al.*, 2008). Importante destacar que os parâmetros foram

escolhidos de tal forma para que se fosse possível ter uma base de comparação e alinhamento com estes trabalhos anteriores.

O conjunto de dados contém 10000 itens. O tamanho da transação é uniformemente distribuído entre 10 e 20 operações. Cada transação é tanto uma transação de atualização (50%) quanto consulta (50%). Para a escolha dos valores dos parâmetros, tomou-se cuidado para que o tamanho do banco de dados e o tamanho da transação juntos produzissem uma região de operação e conflito no qual fosse possível observar interessantes efeitos de desempenho.

Tabela 3 - Parâmetros do simulador

Parâmetro	Valor	Parâmetro	Valor
Itens no Banco de Dados	10000	Taxa de acerto no Buffer	20%
Número de Servidores	9	Tempo de Escrita	4 a 12ms
Número de Clientes por Servidor	4	Tempo de Leitura	4 a 12ms
Discos por Servidor	2	Tempo de CPU usado para operação E/S	0.4ms
CPUs por Servidor	2	Tempo de uma mensagem na rede	0.07ms
Tamanho da Transação	10 a 20	Tempo de CPU para enviar/receber uma mensagem	0.07ms
Operação é escrita	50%	Tempo para broadcast na rede	0.07ms
Operação é Leitura	50%	Tempo para enviar/receber um broadcast	0.07ms

Em (AGRAWAL *et al.*, 1985) os servidores são compostos por duas CPUs e duas unidades de disco. Tanto leitura quanto escrita utiliza entre 4 e 12ms (distribuição uniforme) de tempo do disco. Operações de leitura têm a chance em 20% de encontrar o item de dado no cache e, portanto, não utilizar o disco. Ambas as operações consomem um tempo de 0.4 ms da CPU.

Em (WIESMANN & SCHIPER, 2005) as configurações de rede foram baseadas em valores observados em um cluster de PCs. Neste estudo cada máquina era equipada com um processador de 733MHz e uma interface de rede de 100Mbit/s. As máquinas eram conectadas através de um Hub. Enviar uma mensagem pequena ponto-a-ponto ou *multicast* de 256 bytes consumia 0.07 ms de CPU no servidor de envio, 0.07 ms de rede e 0.07 ms de CPU no servidor de recebimento. Outros experimentos foram feitos usando adaptadores de rede de 10Mbit/s. Neste caso o custo de envio, recebimento e de rede subiram para 0.5 ms. Para este trabalho foi considerado uma interface de 100 Mbit/s.

A configuração de carga consiste em 36 clientes conectados a nove servidores, resultando em quatro clientes por servidor. O tempo entre transações varia de 200ms a 1800ms. O resultado disto consiste em uma variação entre 20 e 180 transações por segundo. A tabela 4 mostra estes valores. Para aumentar a taxa dos possíveis conflitos, alguns itens de dados deverão ser marcados, ao gerar o banco de dados, como itens de dados mais acessados. O segundo parâmetro da tabela informa o percentual destes itens em relação ao tamanho do banco de dados, 45% dos itens serão marcados como freqüentemente acessados. Uma vez identificado estes itens, a chance que um destes determinados itens serem acessados é de 85% como mostrado na Tabela 4.

Tabela 4 - Parâmetros de carga

Parâmetro	Valor
Tempo entre transações	200ms a 1800ms
Percentual de itens freqüentemente acessados	45%
Probabilidade de acesso	85%

A simulação é executada até estar dentro de no mínimo 95% de intervalo de confiança. Para evitar medidas viciadas devido a fatores de inicialização (JÁ, 1991), os resultados das primeiras 500 transações foram descartados.

5.3 Medidas de interesse

A seguir as medidas de interesse estudadas serão apresentadas, estas medidas podem ser de importância interna, isto é, são utilizadas para o funcionamento interno do simulador, ou externas e serão utilizadas nos experimentos e em uma posterior análise.

- **Tempo de resposta:** medido em segundos, essa medida corresponde à diferença entre quando o terminal submete uma nova transação e quando a transação retorna para o terminal após o término com sucesso, incluindo todos os tempos de espera, inclusive na fila de pronto.

- **Taxa de aborto:** Quantidade de transações que devem ser reiniciadas ou canceladas devido a conflitos de concorrência nos dados em razão do número total de transações.
- **Taxa de efetivação:** Quantidade de transações que são efetivadas em razão do número total de transações.
- **Taxa de conflito:** é calculada marcando as transações que tiveram que esperar por algum *lock* possuído por outra transação durante a sua execução.
- **Throughput:** é a medida de desempenho do sistema. Medido em transações por segundo.

5.4 Descrição dos experimentos

Três estudos distintos foram realizados com o objetivo de validar e analisar a proposta deste trabalho. Cada estudo tem por objetivo analisar e focar cada parte do balanceamento proposto, a saber: (i) Controle de admissão; (ii) Controle de variância e (iii) Balanceamento baseado em conflito. Para o primeiro item, como conseqüência da sua introdução, se tem uma geração de fila de espera. O estudo desta fila tem por objetivo realizar uma análise do impacto de sua existência no sistema de banco de dados replicado. Para o segundo item, um controle da diferença de carga entre as réplicas é a principal conseqüência e, portanto, um estudo da distribuição desta carga entre as réplicas parece razoável. Por fim, para validar tanto o último item, como também os restantes como um todo, medidas clássicas de desempenho de um banco de dados serão analisadas.

Nesta seção será apresentada uma descrição dos experimentos que foram realizados em cada estudo, os parâmetros específicos de cada um deles e seus objetivos. A seção seguinte apresenta os resultados obtidos e a seção 4.6 uma análise final.

5.4.1 Estudo do desempenho

Motivação principal deste trabalho, este estudo busca mostrar como a taxa de aborto foi reduzida, o tempo de resposta diminuído e o throughput aumentado. Estes experimentos, também, visam mostrar como o sistema de banco de dados foi controlado, mantendo-o estável mesmo em situações de sobrecarga. Este experimento foi realizado tanto para o algoritmo baseado em certificação quanto para o baseado em votação.

Descrição: Neste estudo será analisada a evolução da taxa de aborto, tempo de resposta e throughput em relação à carga do sistema (transações por segundo). Em relação ao tempo de resposta, a medição foi realizada apenas após a transação ter sido aceita no sistema, neste experimento não é considerado o tempo de espera na fila, que será posteriormente analisado. Todas as configurações descritas anteriormente se aplicam nestes experimentos. Além, disso será considerado que a capacidade de hardware é igual em todas as réplicas. Foram realizadas 17 medições para o intervalo de 20 a 180 transações por segundo em incrementos de 10 transações. Três curvas diferentes são geradas: (i) uma curva sem balanceamento; (ii) Balanceamento sem controle de admissão; e, (iii) Balanceamento proposto no trabalho. Para a curva sem balanceamento, assume-se que a carga será distribuída aleatoriamente entre as réplicas, de modo que a distribuição seja uniforme. Nenhum controle de admissão e nenhum balanceamento de carga serão utilizados. Importante frisar que esta curva foi adicionada ao experimento para fins de comparação. Para a segunda curva os passos 1, 2, 3 e 4 não serão executados, isto é, nenhum controle de admissão e variância entre os servidores. A curva três representa o balanceador completo proposto neste trabalho.

5.4.2 Estudo da distribuição da carga

Este estudo busca mostrar como a distribuição da carga se dá entre as réplicas distintas. O objetivo é comprovar como o balanceador de carga proposto é capaz de fazer uma distribuição equilibrada e justa sem que nenhum servidor fique sobrecarregado nem ocioso, levando em consideração as características físicas de cada

servidor. Resumindo, este experimento mostra como o controle de variância de carga é essencial para controlar e prover um melhor desempenho para um banco de dados replicado.

Descrição: Como o objetivo é estudar a distribuição da carga, neste caso, o algoritmo de replicação utilizado não altera o resultado dos experimentos. Portanto, por conveniência, o algoritmo baseado em certificação foi escolhido. Para estudar como a carga é distribuída entre as réplicas, alguns experimentos foram realizados.

O primeiro experimento se propõe a mostrar a evolução do desvio padrão de conexões disponíveis em relação ao aumento da carga de trabalho. O objetivo é mostrar como o balanceador de carga proposto é capaz de manter o desvio entre os servidores baixo, impedindo que alguns servidores fiquem sobrecarregados enquanto outros ociosos. A medida coletada para o cálculo do desvio padrão foi o número de conexões disponíveis, normalizado. Uma normalização foi necessária devido ao fato de que os servidores podem ter configurações de hardware diferentes, causando valores de MPL muitos distintos. A normalização traz os valores para um intervalo entre 0 e 1. A amostragem foi feita a partir de intervalos fixos de tempo no qual os valores de conexões disponíveis de cada servidor foram coletados. Três curvas foram geradas para analisar este experimento: (i) balanceamento de carga sem controle de admissão (sem os passos 1 e 2 do algoritmo), (ii) balanceamento de carga sem controle de variância (sem o passo 3 do algoritmo) e (iii) balanceamento completo, isto é, com o controle de variância. A curva sem balanceamento de carga não foi necessária neste experimento, pois como a chegada de requisições de dá de maneira uniforme entre os servidores, a distribuição seria, também, uniforme entre as réplicas.

Um segundo experimento tem por objetivo mostrar a distribuição da carga de trabalho em um determinado momento, para uma determinada carga (90 transações por segundo) em todos os servidores. O objetivo é mostrar uma foto de como o sistema se comporta com e sem o controle de variância.

O terceiro experimento visa estudar como as medidas de desempenho: taxa de aborto, tempo de resposta e throughput se comportam com a mudança do fator de variância. A evolução de quatro curvas em relação à carga de trabalho será analisada e correspondem a fatores diferentes de variância: 20%, 40%, 60% e 80%. Todas estas análises são realizadas considerando o balanceamento completo, isto é, baseado em

conflito e com controle de admissão. O objetivo é mostrar como a variância influencia no desempenho do banco de dados como um todo.

Por fim o quarto e último experimento deste estudo tem por objetivo estudar a distribuição de carga em um ambiente heterogêneo de hardware frente à variação da variância entre os servidores. Como no experimento anterior, somente o balanceamento completo será analisado. O objetivo é mostrar como a variância influencia a distribuição da carga sobre servidores mais robustos e analisar esta mesma distribuição sobre servidores menos robustos. Para esta análise os mesmos fatores do experimento anterior serão utilizados: 20%, 40%, 60% e 80%. Uma simplificação de configuração será feita neste experimento, apenas 4 servidores compõem o ambiente replicado. Apenas um servidor terá configuração superior a dos demais, o restante possuirá as mesmas configurações descritas na seção 5.2. A Tabela 5 ilustra tais configurações. A carga do servidor será fixada em 20 transações por segundo (TPS). Esta carga foi escolhida por não deixar o sistema sobrecarregado em nem tão ocioso.

Tabela 5 – Configuração de um servidor mais robusto

Parâmetro	Valor
Taxa de acerto no Buffer	30%
Tempo de Escrita	2 a 6ms
Tempo de Leitura	2 a 6ms
Discos	6
CPUs	6
Tempo de CPU usado para operação E/S	0.2ms

5.4.3 Estudo da fila de espera e tempo de resposta global

Este estudo busca mostrar como se comporta a fila de espera de transações que foram impedidas de entrar no sistema pelo controle de admissão devido à sobrecarga no sistema. O objetivo é mostrar como a introdução do controle de admissão, além de manter o sistema sobre controle, é capaz de prover um tempo de resposta melhor quando se leva em consideração o tempo de espera na fila, em relação a um balanceamento sem tal controle.

Descrição: Para este estudo, dois experimentos são importantes. O primeiro consiste em analisar o tamanho da fila enquanto o segundo o tempo de espera. Este último consiste em analisar comparativamente o tempo de resposta das transações sem o controle de admissão frente ao tempo de resposta da transação somado ao tempo de espera na fila, quando o controle de admissão passa a ser utilizado. Todos estes experimentos analisam a evolução destas medidas frente a um aumento da carga de trabalho. Como o objetivo é estudar a fila gerada devido à sobrecarga, neste caso, o algoritmo de replicação utilizado, apesar de influenciar o resultado, não traz nenhum benefício em termos de estudo. As características que os diferem já foram analisadas no primeiro experimento e, portanto, por conveniência, o algoritmo baseado em certificação foi escolhido para analisar os tempos de resposta.

5.5 Resultados

Esta seção apresenta os resultados obtidos dos experimentos especificados anteriormente, bem como uma analise dos mesmos.

5.5.1 Estudo do desempenho

O resultado do primeiro experimento pode ser observado na Figura 12. Este gráfico mostra a evolução da taxa de aborto do algoritmo baseado em certificação em relação ao aumento da carga de trabalho. Para esta medida quanto maior o valor, pior o resultado. Sem nenhum balanceamento o gráfico mostra claramente que a taxa de aborto cresce rapidamente mesmo com cargas pequenas de trabalho até atingir níveis onde quase a totalidade das transações será abortada. Esta curva, assim como para este experimento, será utilizada em todo restante deste estudo. O objetivo dela é de comparação em relação ao balanceador proposto.

Para um balanceamento sem controle de admissão, os abortos podem ser controlados para cargas menores, porém com o aumento da carga, este balanceamento por si só, não é suficiente. A terceira curva, a proposta deste trabalho, mostra como o

controle de admissão combinado com o balanceamento baseado em conflito pode controlar o sistema de banco de dados, impedindo uma alta taxa de aborto para cargas elevadas de trabalho.

A Figura 13 mostra o mesmo experimento, para o algoritmo baseado em votação. As características das curvas são similares ao gráfico anterior: crescimento da taxa de aborto em relação ao aumento da carga (apesar do algoritmo de votação por natureza já possuir taxa de aborto menor em relação ao de certificação como já observado na seção 3.1 do capítulo 3), curva sem controle de admissão com resultados melhores apenas para cargas menores. A tendência da curva é de aumento da taxa de aborto, enquanto o balanceador completo consegue controlar o sistema. Novamente, apenas, o balanceamento de carga com controle de admissão é capaz de conter a sobrecarga a manter o sistema em níveis aceitáveis de taxa de aborto.

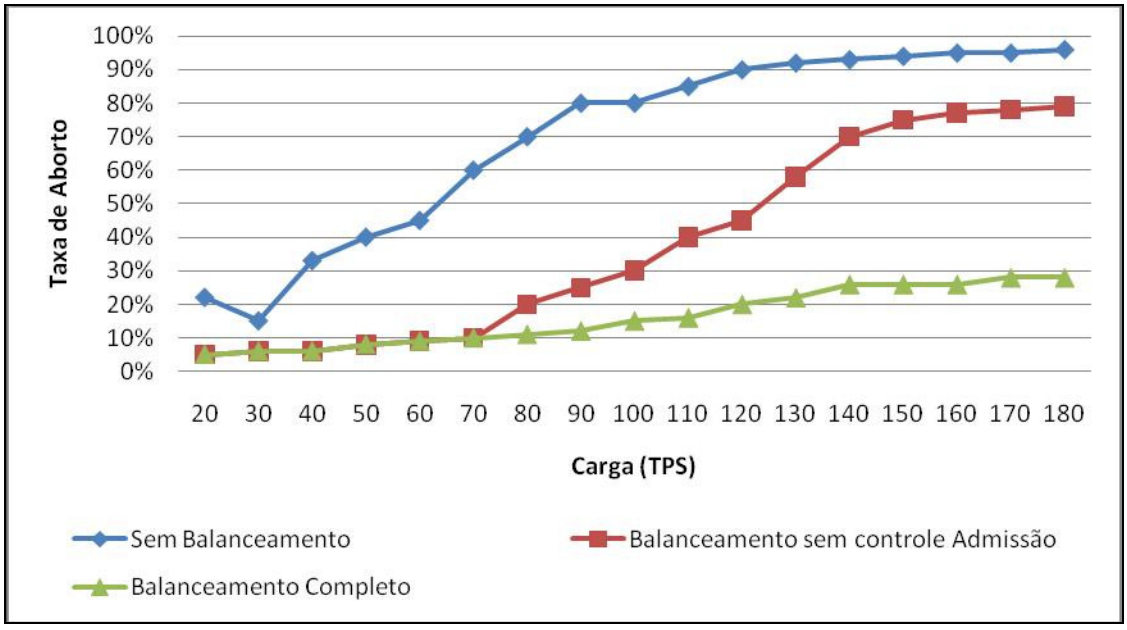

Figura 12 - Redução da taxa de aborto para o algoritmo baseado em certificação

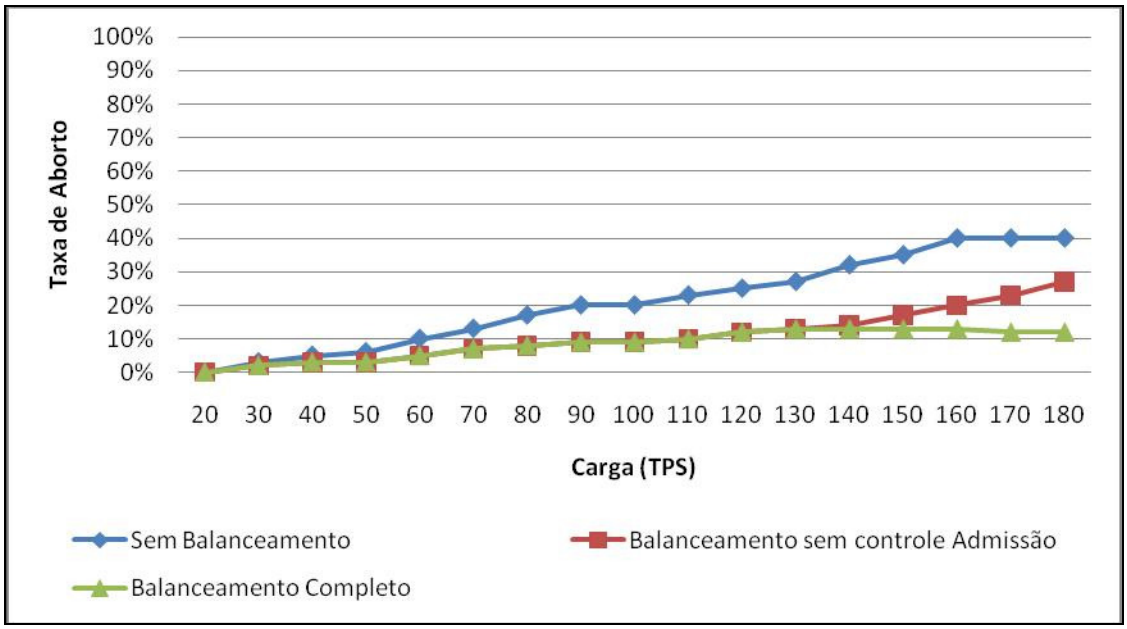

Figura 13 - Redução da taxa de aborto para o algoritmo baseado em votação

Os próximos dois gráficos ilustram o desempenho do balanceador de carga sobre a ótica do tempo de resposta. Neste experimento, como no anterior, quanto menor o valor da medida, melhor o resultado. Estes gráficos ilustram a evolução do tempo de resposta das transações submetidas ao sistema em relação ao aumento de carga.

A Figura 14 ilustra o experimento realizado para o algoritmo baseado em certificação. Para a curva sem balanceamento, claramente, no início o algoritmo comporta-se como esperado, isto é, o tempo de resposta aumenta com o aumento da carga. Isto se deve ao fato de que transações executando concorrentemente requerem os recursos de hardware ao mesmo tempo, levando a contenções, tempos de espera e execução maiores. Porém, após, uma determinada carga, este tempo cai rapidamente, este fato pode ser explicado com a ajuda do experimento anterior. Devido ao aumento da taxa de aborto, chega-se a uma carga onde quase a totalidade das transações será abortada, reduzindo drasticamente o tempo de resposta (aqui não está sendo considerado o tempo de reenvio e re-execução das transações abortadas, este estudo será feito mais adiante). A curva de balanceamento sem controle de admissão, apesar de apresentar resultados melhores que a curva anterior, sofre do mesmo problema original, não consegue controlar os abortos do sistema quando altas taxas são submetidas. A curva

com balanceamento completo tem os mesmos resultados para cargas menores, porém é capaz de manter o sistema sobre controle em situações de sobrecarga.

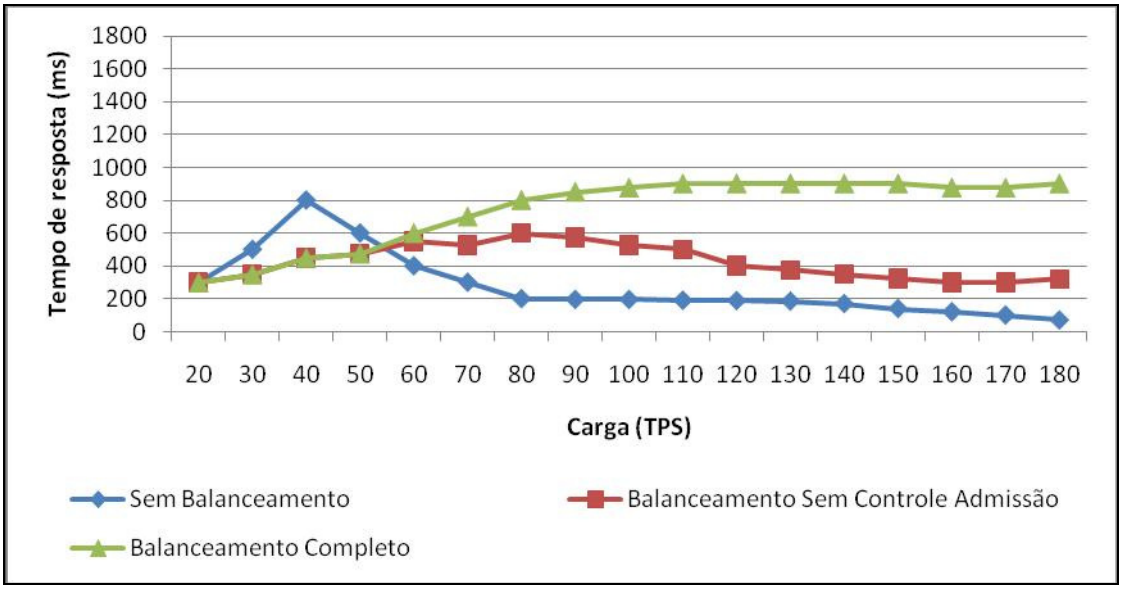

Figura 14 - Controle do Tempo de resposta para o algoritmo baseado em certificação

O tempo de resposta é justamente estabilizado porque o controle de admissão impede a entrada de novas transações e conseqüentemente evitando a degradação do sistema em termos tanto do tempo de reposta como da taxa de aborto (como visto anteriormente).

A Figura 15 apresenta o mesmo estudo anterior para o algoritmo de votação. Neste gráfico a curva sem balanceamento e balanceamento sem controle de admissão, apesar de resistirem um pouco mais ao aumento de carga (devido à própria natureza do algoritmo de possuir menores taxas de aborto), para uma determinada carga as taxas de aborto se tornam elevadas e o tempo de resposta passa a decrescer. Outro ponto importante a se notar é que os tempos de resposta deste algoritmo são maiores em relação ao algoritmo baseado em certificação. Isto se deve ao fato da necessidade de uma mensagem extra, no qual as transações remotas precisam esperar pela confirmação do servidor de delegação.

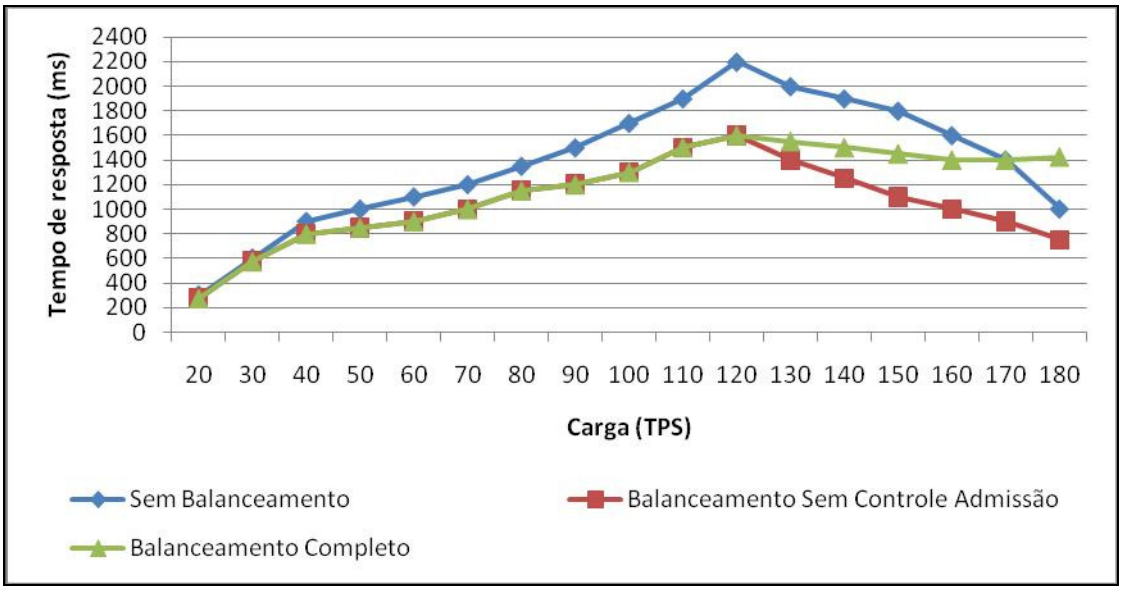

Figura 15 - Controle do Tempo de resposta para o algoritmo baseado em votação

O throughput será a última medida de desempenho que será analisada neste trabalho. Agora, quanto maior o valor desta medida, melhor o resultado. Os gráficos da Figura 16 e da Figura 17 mostram como esta medida evolui em relação ao aumento da carga de trabalho. Novamente em ambas as estratégias de replicação (certificação e votação) o balanceador de carga traz benefícios. Enquanto as outras duas curvas passam a ter um declínio com o aumento da carga, a curva do balanceador mantém o sistema estável. Outra questão a se notar é que as três curvas se comportam de maneira igual para cargas pequenas de trabalho, porém após uma determinada carga o balanceador sem controle de admissão passa ter o pior resultado, isto de deve ao fato de que servidores preferenciais foram escolhidos para receberem cargas conflitantes. Como não há controle de admissão nem tão pouco controle de variância entre as réplicas, alguns servidores ficam sobrecarregados com grande parte da carga de trabalho, reduzindo o throughput.

Também é fácil perceber que para cargas menores o algoritmo baseado em certificação possui desempenho superior, porém após uma determinada carga, esta relação é invertida, isto se deve ao fato da ausência de *log* de transações efetivadas no algoritmo de certificação. Para cargas menores este *log* é pequeno e como o algoritmo baseado em votação necessita de uma mensagem extra de confirmação, ocasiona para o algoritmo de certificação um throughput maior, porém o inverso ocorre depois da

sobrecarga, pois o *log* de transações efetivadas tende a crescer, causando contenção no sistema e conseqüentemente diminuindo o throughput.

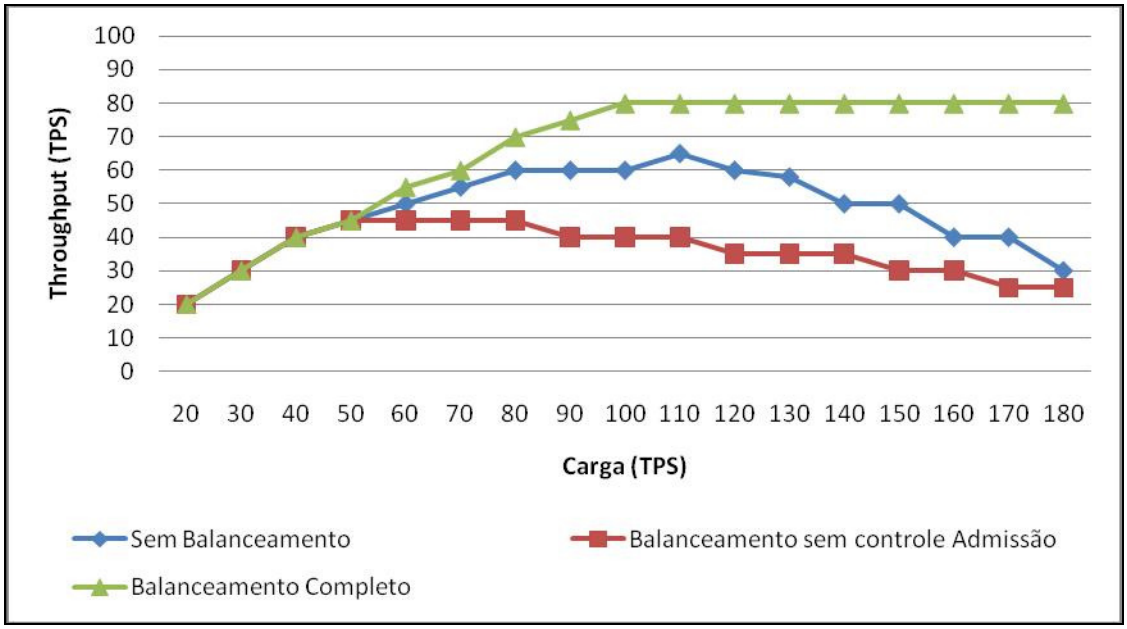

Figura 16 - Throughput para o algoritmo baseado em certificação

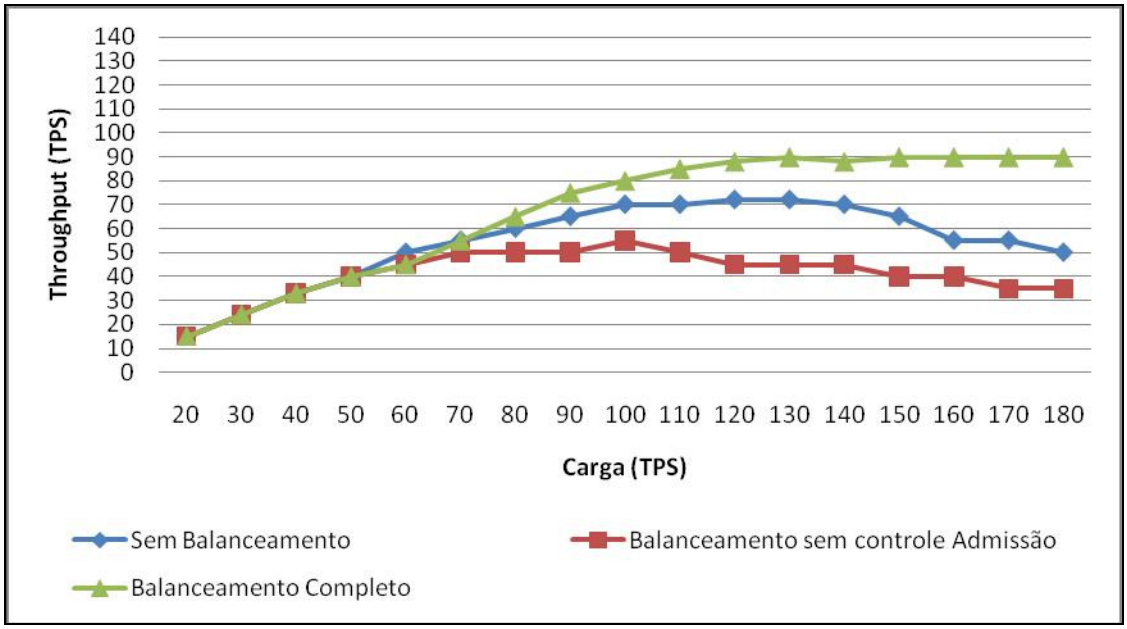

Figura 17 - Throughput para o algoritmo baseado em votação

5.5.2 Estudo da Distribuição da carga

O gráfico da Figura 18 apresenta os resultados do primeiro experimento. Este mostra a evolução do desvio padrão das conexões disponíveis normalizadas de todas as réplicas. Para as curvas sem o controle da variância, o desvio padrão entre os servidores é muito alto, isto se deve pelo fato de que o balanceamento baseado em conflito elege servidores preferenciais. Porém a partir de um ponto de saturação, o controle de admissão, da segunda curva (Sem controle de variância), faz com que todas as conexões livres sejam utilizadas, levando todos os servidores a ficarem ocupados. Na primeira curva isso não ocorre devido ao fato de que o MPL não é adaptado dinamicamente frente a uma carga que diminua seu desempenho (o valor do MPL foi configurado suficientemente grande para suportar a carga do experimento). A curva do balanceamento completo, isto é, com o controle da variância mostra como o desvio padrão entre os servidores fica pequeno, sem grandes variações bruscas com o incremento da carga submetida.

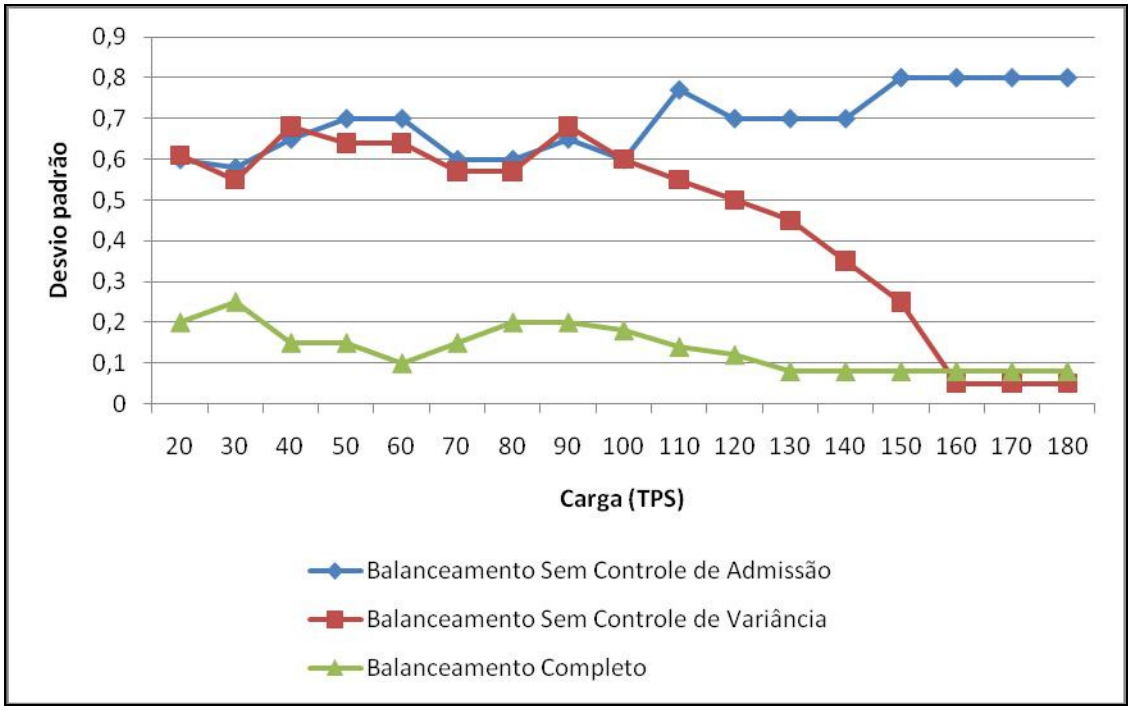

Figura 18 - Desvio padrão da distribuição da carga

O gráfico da Figura 19 representa uma foto de como as transações se distribuem quando não há controle de variância. Fica evidente que alguns servidores ficam sobrecarregados enquanto outros ociosos. O gráfico da Figura 20 mostra como o controle da variância promove uma distribuição equilibrada do trabalho entre as cargas.

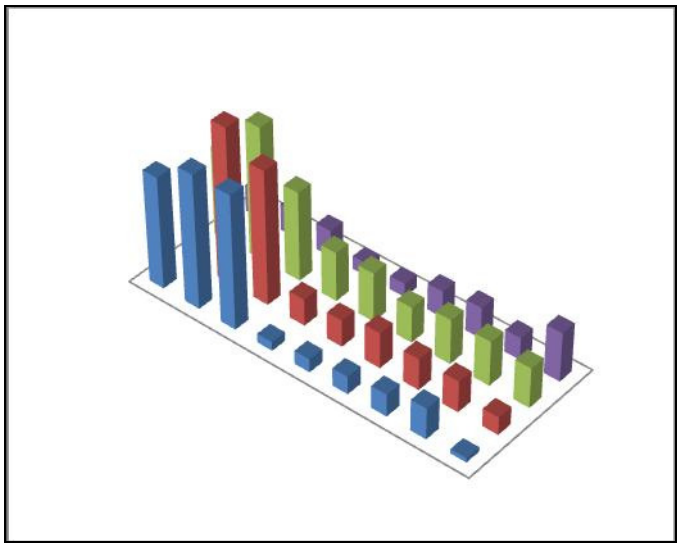

Figura 19 - Distribuição da carga sem controle de variância

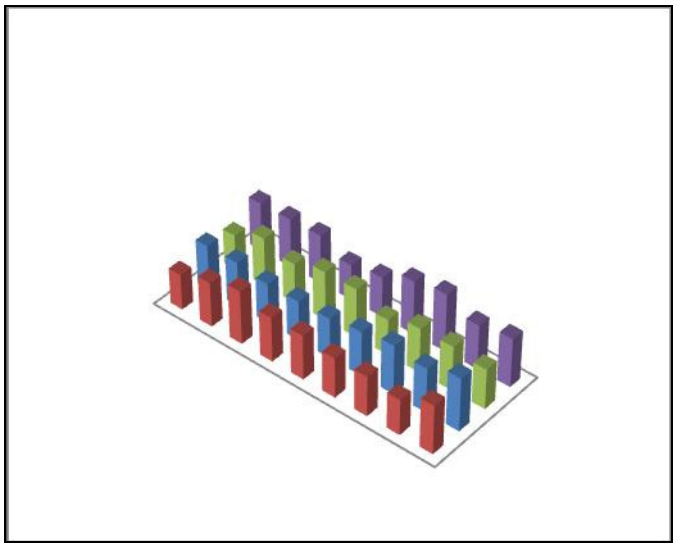

Figura 20 - Distribuição da carga com controle de variância

Os próximos três gráficos ilustram o resultado do terceiro experimento deste estudo: o comportamento das medidas de desempenho em relação à carga de trabalho a partir da observação de quatro curvas distintas, representando, cada uma, fatores de variância distintos. Quanto maior o fator de variância, maior o número de servidores que ficarão ociosos em detrimento a um conjunto pequeno de servidores que ficarão sobrecarregados, por outro lado quanto menor este fator, maior o paralelismo entre os servidores.

O gráfico da Figura 21 ilustra o comportamento da taxa de aborto. Quanto maior o fator da variância, menor a taxa de aborto, isto se deve ao fato de que transações com probabilidade de conflito serão enviadas para a mesma réplica, evitando conflito em servidores distintos, conseqüentemente reduzindo a taxa de aborto. Portanto, em relação à taxa de aborto quanto maior o fator de variância melhor o resultado.

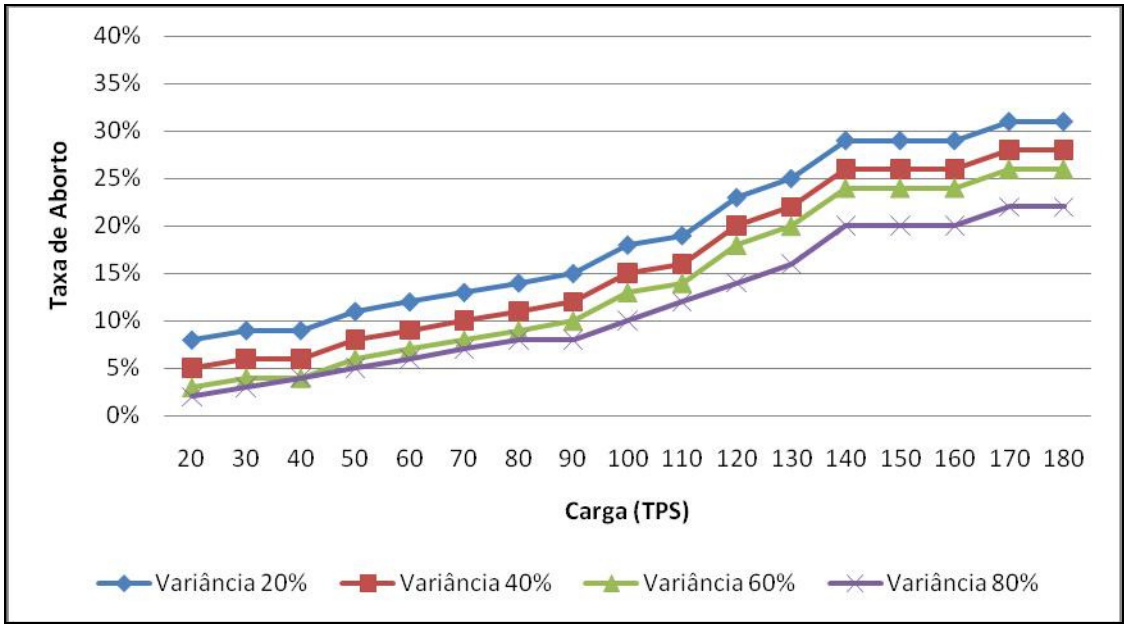

Figura 21 - Estudo da taxa de aborto em relação à variância

O gráfico da Figura 22 ilustra o comportamento do tempo de resposta. Quanto menor a variância, menor o tempo de resposta. Quanto menor o valor da variância, maior o paralelismo entre as réplicas, evitando com que as transações sejam designadas

para poucas réplicas e assim, diminuindo a contenção e utilizando todo hardware disponível em todas as replicas para a execução de transações.

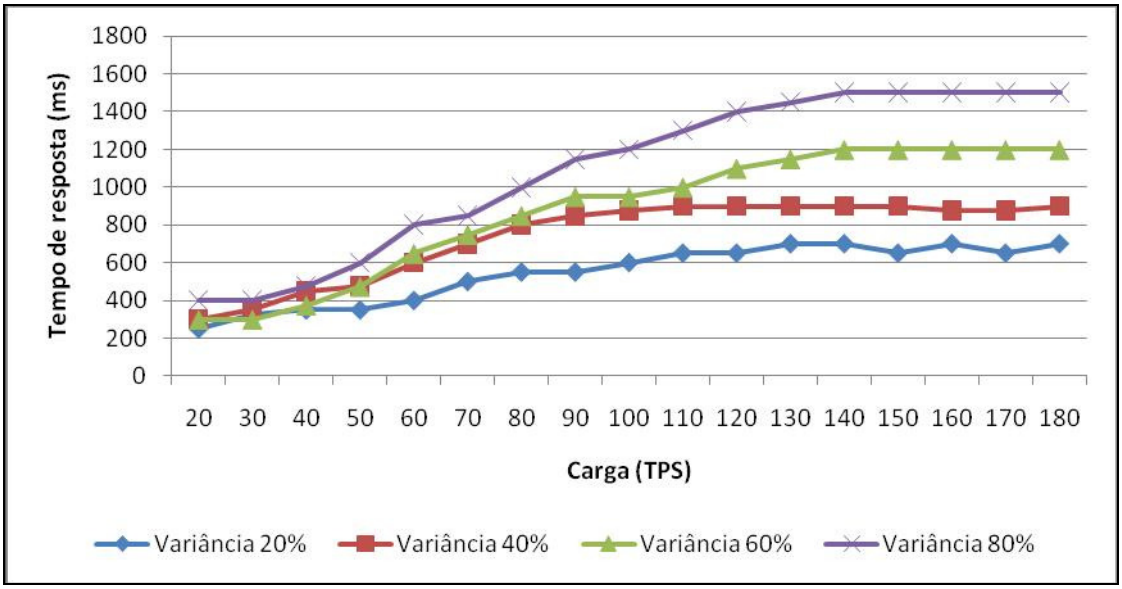

Figura 22 - Estudo do Tempo de resposta em relação à variância

O último gráfico deste experimento, Figura 23, ilustra a evolução do throughput. O mesmo princípio explicado para o tempo de resposta serve para esta medida. Portanto, quanto menor o fator de variância, melhor o resultado obtido.

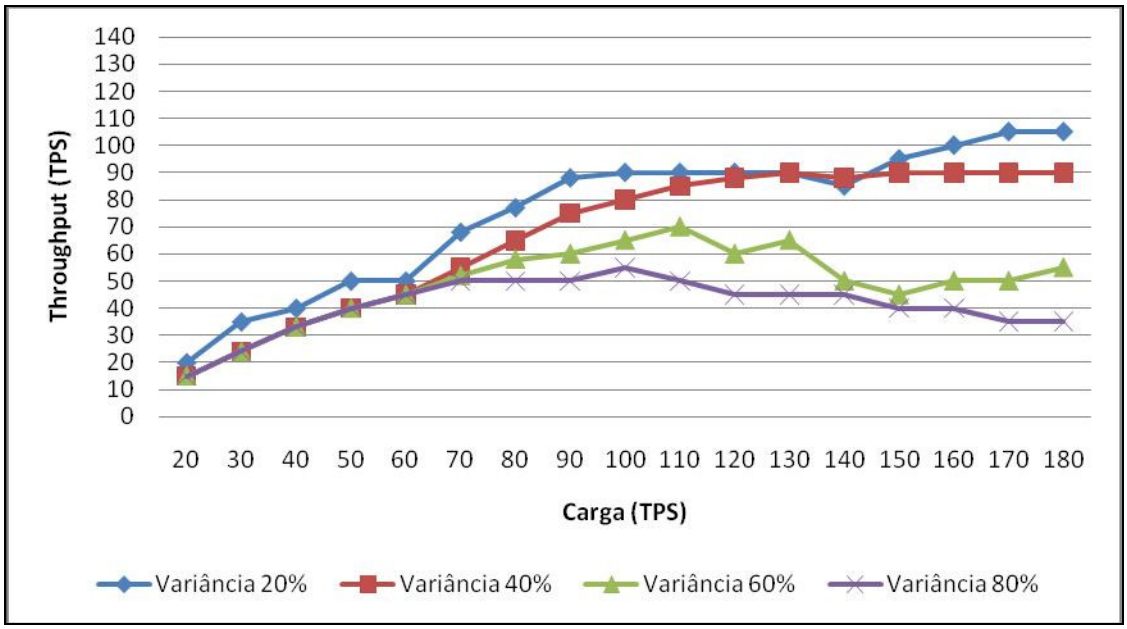

Figura 23 - Estudo do throughput em relação à variância

Para satisfazer as três medidas, um valor nem muito alto, nem muito pequeno deve ser utilizado, a fim de se obter um resultado satisfatório. O fator que foi utilizado neste experimento foi 40% e os gráficos apresentados neste experimento são de importância para ratificar a escolha feita anteriormente.

O gráfico da Figura 24 ilustra o último experimento deste estudo. Apenas quatro servidores foram utilizados. O servidor 1 corresponde ao servidor mais robusto em termos de hardware, os outros três servidores possuem a mesma configuração de hardware. Este experimento ilustra a distribuição da carga sobre os servidores, para os seguintes fatores de variância: 20%, 40%, 60% e 80%. Na introdução de um servidor mais robusto, o ideal seria que este aceitasse uma quantidade maior de carga que os demais, isto por dois motivos. O primeiro seria pelo fato de que por ser mais robusto pode aceitar mais transações concorrentes e mesmo assim, continuar a ter um desempenho melhor que os demais, até o ponto onde seu desempenho passa a ser equivalente ao dos servidores menos robustos. O segundo deve-se ao fato de como este servidor pode aceitar cargas maiores, mais transações conflitantes podem ser atribuídas a ele, diminuindo conflitos entre transações executadas em servidores distintos e conseqüentemente diminuindo as taxas de aborto.

Como pode ser observado no gráfico, um fator de variância muito alto, faz com que a carga seja distribuída de maneira uniforme entre as réplicas, e, portanto, servidores mais robustos acabam por ficar ociosos. Por outro lado, um valor muito baixo faz com que estes servidores fiquem muito sobrecarregados, isto se deve ao fato de que o número de conexões disponíveis é muito maior que nos demais servidores, devido a um valor alto de MPL (devido as suas características de hardware). Portanto, este servidor já inicia quebrando a restrição de balanceamento entre as réplicas, sendo sempre escolhido pelo balanceador a fim de respeitar a restrição.

Porém, como os valores correspondem apenas às médias do sistema, algo mais grave ocorre para valores altos de variância. Neste caso, em alguns momentos, servidores inferiores em termos de hardware, acabam por receber boa parte da carga deixando o servidor mais robusto totalmente ocioso. Portanto, novamente, a reafirmação da necessidade da escolha de um valor mediano de variância.

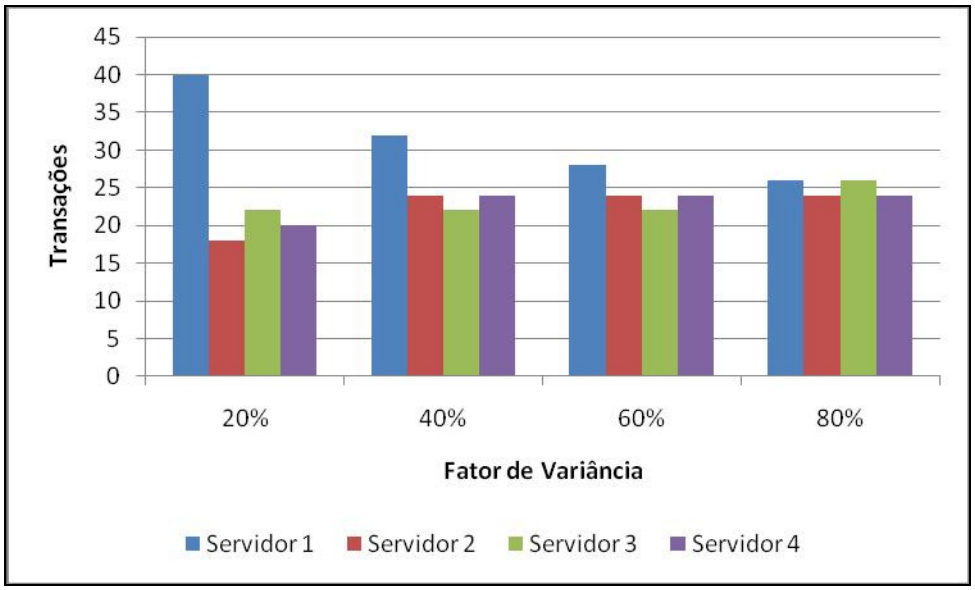

Figura 24 - Distribuição da carga sobre servidores heterogêneos

5.5.1 Estudo da fila de espera e tempo de resposta global

O gráfico da Figura 25 mostra como a o tamanho da fila cresce em relação à carga submetida ao sistema. Como era de se esperar, a fila para o algoritmo de votação possui sempre um tamanho maior para qualquer carga submetida.

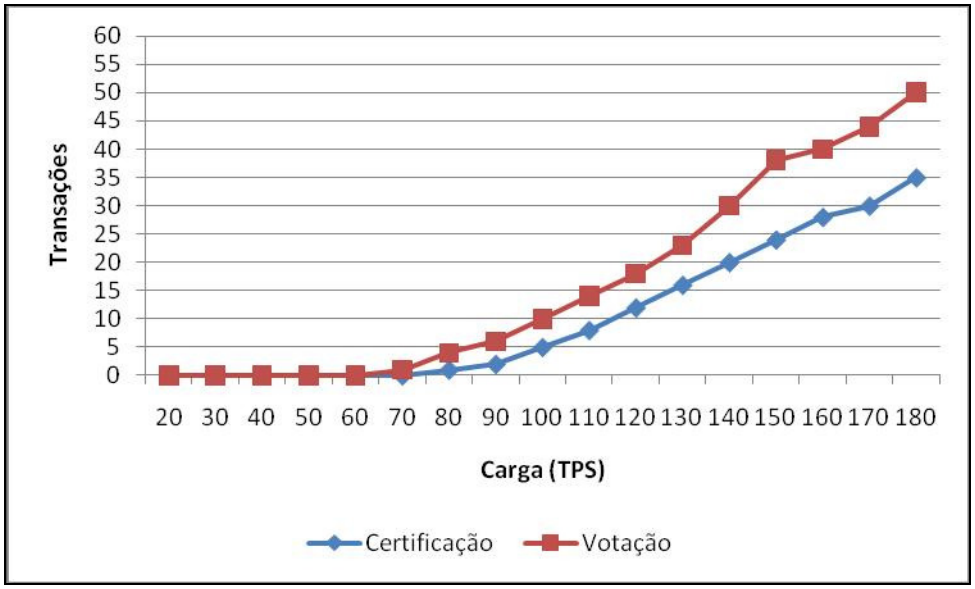

Figura 25 – Estudo do tamanho da fila de espera

Isto se deve pelo fato de que o tempo de resposta para este algoritmo supera o do algoritmo de certificação (como apresentado anteriormente), porém a tendência da curva é igual para ambos os algoritmos. Outra questão a se observar é o fato de que para cargas menores nenhuma ou quase nenhuma fila se forma, obviamente pelo fato de que o sistema ainda não entrou em um estado de sobrecarga, onde o controle de admissão impediria a entrada de novas transações.

O último experimento deste trabalho, Figura 26, analisa o tempo de resposta, considerando, agora, o tempo de espera na fila. Este experimento foi realizado para o algoritmo baseado em certificação. Todas as curvas deste gráfico consideram a existem do balanceado de carga baseado em conflito. As curvas sem controle de admissão e a curva com controle de admissão já foram estudadas e estão neste experimento para fins de comparação. Relembrando que a primeira e a segunda curva possuem valores baixos

de resposta, pois, o tempo de executar novamente uma transação abortada não foi computado. As duas curvas mais importantes aqui serão a curva que considera o tempo de resposta para o balanceamento completo, mas agora com o respectivo tempo de espera somado ao tempo de resposta de cada transação.

Para comparação uma quarta curva derivada da primeira foi utilizada, nesta curva é levado em consideração o tempo total de execução de uma transação, incluindo o tempo de re-execução de transações abortadas. Isto é, o tempo de execução de todas as vezes que a transação foi abortada, somado ao tempo de execução quando conseguiu ser efetivada. Em todas as curvas, quando a carga do sistema ainda é baixa, o sistema ainda não está sobrecarregado, todas possuem comportamento similar, porém após a sobrecarga pode-se notar, comparando a segunda curva (controle de admissão) com a terceira (controle admissão + tempo de espera), que o tempo de espera na fila, faz com que o tempo de reposta continue a crescer enquanto que na curva com controle de admissão uma estabilização ocorra. Este comportamento é esperado, enquanto o controle estiver impedindo a entrada de carga, o sistema mantém um nível estável de desempenho, por outro lado o tempo na fila de espera cresce com o aumento da carga. Comparando com a última curva é que se pode ver, sem dúvidas, o benefício do controle da carga. Os tempos de resposta desta curva mostram valores superiores de tempo de resposta comparado aos tempos do balanceamento com controle de admissão somado ao tempo de resposta.

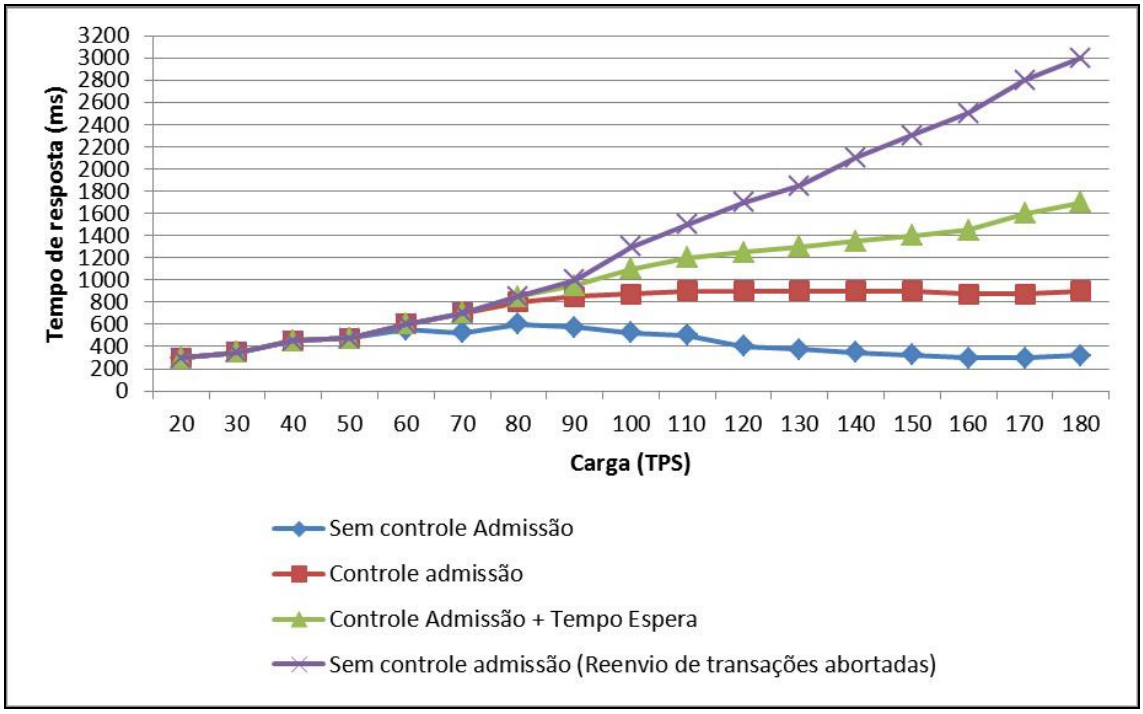

Figura 26 - Estudo do tempo de espera global

Portanto, o controle de admissão, ao manter níveis aceitáveis de desempenho no sistema, impedindo a sua degradação e como conseqüência a geração de uma fila de espera, acaba por possuir um desempenho global (quando se leva em consideração o tempo de espera na fila) superior em relação a uma abordagem sem controle de admissão. Isto mostra que um sistema já sobrecarregado, quando aceita novas transações sem nenhum controle, leva a uma situação de deterioração, onde novas requisições tendem a piorar cada vez mais o desempenho quando comparado a uma abordagem com o controle das requisições.

5.5.2 Análise final

O objetivo deste trabalho é alcançar um sistema que deva se comportar de maneira a atingir sempre um mínimo satisfatório ao longo do tempo, que se comportasse de forma homogênea e sem oscilações, independente da carga que está sendo submetido. Para tal finalidade, um balanceador adaptativo baseado em conflito foi proposto, esperava-se que tal balanceador fosse capaz de manter o sistema

previsível, balanceado, com baixas taxas de aborto e estável sem picos de saturação, isto é, com um nível mínimo satisfatório de desempenho. Necessidades básicas para ambições maiores de níveis de serviço.

Em resumo, para atingir o objetivo esperado, as seguintes estratégias foram propostas: (i) Distribuição justa de carga entre os recursos (réplicas); (ii) Respeitar o limite de carga de cada réplica (MPL); (iii) Diminuir número de transações conflitantes em réplicas diferentes. Com tais abordagens, os seguintes resultados eram esperados: (i) Maximizar o throughput; (ii) Minimizar taxa de aborto; (iii) Minimizar o tempo de resposta; (iv) Escalabilidade (principalmente em ambientes heterogêneos de hardware). Como tais resultados de maximização e minimização são muitas vezes conflitantes entre si, buscava-se que o balanceador proposto alcançasse resultados intermediários que trouxesse resultados satisfatórios para estas variáveis.

Para mostrar os resultados do balanceador, vários experimentos foram feitos e seus resultados apresentados. Cada estratégia do balanceador foi testada e os resultados mostraram seus benefícios. Tais resultados muitas vezes vieram para reafirmar as estratégias e parâmetros escolhidos. Os resultados mostraram: (i) como a taxa de aborto e tempo de resposta podem ser reduzidos e o throughput aumentado; (ii) como o balanceamento justo entre as cargas, levando em consideração a heterogeneidade de hardware dos servidores e como esta característica pode ser aproveitada para diminuir as taxas de aborto e melhorar o desempenho do sistema como um todo e (iii) o controle de admissão, evitando a sobrecarga do sistema, impede sua deterioração, produzindo resultados melhores quando se compara tal sistema deteriorado com o sistema sob controle, porém considerando o tempo gasto na fila de espera. Um, outro, fator importante a se destacar é que para cargas moderadas o resultado da introdução do balanceador não alterou o desempenho dos algoritmos, todas as curvas do estudo de desempenho se comportaram basicamente da mesma maneira independente da introdução do balanceador.

Porém, um segundo resultado mais sutil pode ser esperado destes experimentos. Não somente o simulador foi utilizado para analisar e validar os resultados do balanceador proposto, como a execução de tais experimentos serviu para validar e melhorar continuamente o simulador que é um dos frutos deste trabalho.

6. Conclusão

A meta principal deste trabalho consistia em prover um balanceador de carga que possibilitasse que o sistema se comporte de forma homogênea e sem oscilações, independente da carga que está sendo submetido. O sistema deve se comportar de maneira a atingir sempre um mínimo satisfatório ao longo do tempo, sem picos de saturação ou sobrecarga. Estas características são imprescindíveis quando se deseja objetivos mais ambiciosos como QoS (qualidade de serviço) e QoE (qualidade de experiência).

Um dos grandes problemas dos algoritmos de replicação em banco de dados está relacionado com a execução de transações conflitantes em réplicas distintas. Nestes algoritmos a execução de transações concorrentes, em servidores distintos, ocorre sem que o sistema tenha conhecimento dos potenciais conflitos existentes. Somente em uma fase posterior é que tais conflitos serão resolvidos. Para cargas leves o problema não é tão relevante, porém para cargas pesadas uma das conseqüências são as altas taxas de aborto. Uma carga muito grande de operações de escrita em combinação com uma taxa elevada de conflitos pode levar a uma degradação considerável do sistema, independente do algoritmo que está sendo utilizado.

Este trabalho apresentou um balanceador de carga adaptativo para banco de dados replicados em cenários de sobrecarga e alta taxa de conflito. Toda a estrutura do balanceador se baseou em três observações: (a) Se transações conflitantes são submetidas para o mesmo servidor, o controle de concorrência local será responsável por serializar operações conflitantes adequadamente, diminuindo abortos, (b) Na falta de conflitos, porém, o desempenho é melhorado se as transações executam concorrentemente em diferentes réplicas e (c) em situações de sobrecarga, é melhor impedir a entrada de novas requisições. Processamento de novas transações, apenas, degrada ainda mais o sistema, causando prejuízo do sistema como todo.

Logo em seguida, foi apresentado o simulador que foi desenvolvido neste trabalho. Este simulador, apesar da obvia simplificação, representa um modelo

razoavelmente completo de um banco de dados replicado, com todas suas características e componentes principais. A simulação é importantíssima, em projetos e pesquisas, onde ainda não se conhece com exatidão os efeitos das premissas estabelecidas e principalmente quando se tem dificuldade de realizar os testes e experimentações no mundo real. O simulador, correspondendo a uma simplificação do mundo real e por se tratar de um ambiente mais controlado torna tal tarefa mais simples e menos dispendiosa quando se quer validar e analisar as pesquisas. Portanto, o simulador, neste trabalho, traz duas contribuições. A primeira corresponde ao fato de que ele foi utilizado para realizar os experimentos com o objetivo de validar e analisar a proposta do trabalho e a segunda é que este simulador poderá ser utilizado em vários outros estudos e trabalhos com o objetivo de antecipar resultados de pesquisas para uma posterior fase em um ambiente real.

Por fim, vários experimentos e análises foram apresentados com o objetivo de mostrar como o balanceamento proposto é capaz de manter o sistema estável e controlar a sobrecarga. A partir dos gráficos de desempenho foi possível observar que sem este balanceador o sistema entraria em níveis de degradação onde tornava o uso do banco de dados inviável, principalmente no que se refere à taxa de aborto. Ao mesmo tempo, estes experimentos serviram para validar, refinar e melhorar continuamente o simulador desenvolvido.

Como contribuição, este trabalho trouxe: (i) um melhor entendimento sobre replicação de banco de dados em cenários de sobrecarga e taxa de conflito elevada; (ii) um balanceador de carga específico para tal cenário e (iii) um simulador que contempla as principais características de um banco de dados replicado. Tais contribuições abrem portas para pesquisas nas áreas de qualidade de serviço dentre outras.

Apesar dos avanços deste trabalho, o tema está longe de estar finalizado, abre várias frentes de pesquisa e alguns temas podem ser propostos:

- **Implementação:**
    - Desenvolvimento de uma API para o balanceador e implementá-lo em algum *middleware* de replicação do mercado;
    - Testar o balanceador com algum *benchmark* tipo o TPC-C (POESS & FLOYD, 2000);

Seguindo a tendência da computação em nuvem (LIM *et al.*, 2009):

- Automatizar o algoritmo para definir um número de servidores necessários para atender um critério específico de desempenho;
- Estudo e abordagem mais específica quando se trata da heterogeneidade de hardware das diferentes réplicas;

- **Seguindo a tendência da computação autonômica (KEPHART & CHESS, 2003):**
  - Tornar o balanceador automaticamente adaptável no que diz respeito aos seus parâmetros de funcionamento;
  - Tornar o balanceador sensível a variação da taxa de conflito. Atualmente o balanceador apenas considera que existe conflito, mas sem nenhuma sensibilidade a variação deste;
  - Outros estudos para torná-lo autonômico;
- **Nível de Serviço:**
  - Pesquisas e desenvolvimento relacionados a QoS e QoE;

7. Referências Bibliográficas

ABOUZOUR, M., SALEM, K., BUMBULIS P., 2010, "Automatic tuning of the multi programming level in Sybase SQL Anywhere". *In Workshop on Self-managing Database Systems (SMDB)*.

AGRAWAL, R., CARAY, M.J., LIVNY, M., 1987, "Concurency Control Performance Modeling: Alternatives and Implications," *In: ACM Trans. Database Systems*, vol. 12, no. 4, pp. 609-654.

AMIR, Y., TUTU, C., 2004, "From Total Order to Database Replication". *In Proc. of Int. Conf. on Distr. Comp. Systems (ICDCS)*.

AMZA, C., COX, A.L., ZWAENEPOEL, W., 2005, "A Comparative Evaluation of Transparent Scaling Techniques for Dynamic Content Servers". *In: 21st International Conference On Data Engineering*.

ATTIYA, H., WELCH, J., 2004, *Distributed Computing*, Wiley-Interscience.

BERNSTEIN, P., HADZILACOS V., GOODMAN, N., 1987, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

BROWN, K.P., MEHTA, M., CAREY, M.J., et al., 1994, "Towards automated performance tuning for complex workloads". *In Proceedings of the Twentieth International Conference on Very Large Databases*, pp. 72-84, Santiago, Chile.

CECCHET, E., CANDEA G., AILAMAKI A., 2008, "Middleware-based database replication: the gaps between theory and practice". *In SIGMOD*, pp. 739-752.

CHEN, S., GORTON, I., 2002, "A predictive performance model to evaluate the contention cost in application servers". *In APSEC '02: Proceedings of the Ninth Asia-Pacific Software Engineering Conference*, pp. 435, Washington, DC, USA, IEEE Computer Society.

COSTA, R.L., FURTADO, P., 2008, "A qos-oriented external scheduler". *In: SAC 2008: Proceedings of the 2008 ACM symposium on Applied computing*, pp. 1029–1033. ACM, New York.

COSTA, R., FURTADO, P., 2011, "Quality of experience in distributed databases". *In: Distributed and Parallel Databases*, v. 29, pp. 361-396, Springer Netherlands.

CORREIA, A., PEREIRA, J., RODRIGUES, L., et al, 2010, "Practical Database Replication". *In Replication: Theory and Practice*, Springer.

CORREIA, A. Jr., SOUSA, A., SOARES, L., et al., 2005, "Group-based replication of on-line transaction processing servers". *In Proc. IEEE/IFIP Latin-American Dependability Conf. (LADC'05)*.

FUERTES, M., 2011, *On the Consistency, Characterization, Adaptability an Integrity of Database Replication Systems*. Ph.D. dissertation, University of Valencia, Valencia, Spain.

GRAY, J., HELLAND, P., O'NEIL, P.E., et al., 1996, "The dangers of replication and a solution". *In: SIGMOD*, pp. 173-182.

GUPTA, V., HARCHOL-BALTER, M., 2009, "Self-adaptive admission control policies for resource-sharing systems," *In Proc. of ACM SIGMETRICS Conf.*, pp. 311–322.

HARIZOPOULOS, S., 2005, *Staged Database Systems*. PhD thesis, Carnegie Mellon University.

HOLLIDAY, J., AGRAWAL, D., ABBADI, A., 1999, "The performance of database replication with group multicast". *In Proceedings of the 29thIEEE International Symposium on Fault Tolerant Computing*, pp. 158–165. IEEE Computer Society Press, June.

HEISS, H., WAGNER, R., 1991, "Adaptive load control in transaction processing systems". *In VLDB '91: Proceedings of the 17th International Conference on Very Large Data Bases*, pp. 47-54, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc.

JA, R., 1991, *The art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley and Sons.

JUÁREZ, J.R., ARMENDÁRIZ-IÑIGO, J. E., MENDÍVIL, et al., 2007a, "A weak voting database replication protocol providing different isolation levels". *In NOTERE'07*.

JUÁREZ, J. R., ARMENDÁRIZ-IÑIGO, J. E., MUÑOZ-ESCOÍ, et al., 2007b, "A deterministic database replication protocol where multicast writesets never get aborted". *In OTM Workshops*, (1), LNCS 4805, pp. 1–2, Vilamoura, Portugal, November, Springer.

JUÁREZ, J. R., ARMENDÁRIZ-IÑIGO, J. E., MUÑOZ-ESCOÍ, F. D., et al., 2008, "A database replication protocol where multicast writesets are always committed". *In 3rd Intnl Conf on Availability, Reliability and Security (ARES)*, pp. 120–127, Barcelona, Spain, IEEE-CS Press.

JUNG, H., HAN, H., FEKETE, A., et al., 2011, "Serializable Snapshot Isolation for Replicated Databases in High-Update Scenarios" *In: VLDB*.

KEMME, B., 2000, *Database Replication for Clusters of Workstations*, PhD dissertation, ETH ZÄurich, Departement of Computer Science.

KEMME B., ALONSO, G., 2000, "Don't be lazy, be consistent: Postgres-R, a new way to implement database replication". *In VLDB*, pp. 134-143.

KEMME, B., ALONSO, G., 2010, “Database replication: a tale of research across communities.” *In: VLDB*.

KEPHART, J.O., CHESS, D.M., 2003, "The vision of autonomic computing", *Computer* , vol.36, no.1, pp. 41- 50

LIM, H., BABU, S., CHASE, J., et al., 2009, “Automated control in cloud computing: challenges and opportunities”. *In Proceedings of the 1st workshop on Automated control for datacenters and clouds (ACDC '09)*, pp. 13-18, ACM, New York, NY, USA.

LIU, X., SHA, L., DIAO, Y., et al., 2003, “Online Response Time Optimization of Apache Web Server”. *In Proceedings of the 11th International Workshop on Quality of Service*, pp. 461-478.

MILAN-FRANCO, J. M., JIMÉNEZ-PERIS, R., PATIÑO-MARTÍNEZ, M., et al., 2004. “Adaptive distributed middleware for database replication”. *In Proceedings of 5th ACM/IFIP/USENIX Middleware Conference*. Toronto, Canada, 175–194.

MISHIMA, T., NAKAMURA H., 2009, “Pangea: an eager database replication middleware guaranteeing snapshot isolation without modification of database servers”. *In: Proc. VLDB Endo,* pp. 1066-1077.

MONKEBERG, A., WEIKUM, G., 1992, “Performance evaluation of an adaptive and robust load control method for the avoidance of data-contention thrashing”. *In VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, pp. 432-443, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc.

MUÑOZ-ESCOÍ, F.D., PLA-CIVERA, J., RUIZ-FUERTES, M.I., et al., 2006, “Managing transaction conflicts in middleware-based database replication architectures”. *In: IEEE Int. Symp*.

MUÑOZ-ESCOÍ, F.D., DECKER, H., ARMENDÁRIZ, J.E., et al., 2007, “Database Replication Approaches”. Technical Report TR-ITI-ITE-07/19

NARVÁEZ, V., 2009, *On Non-Intrusive Workload-Aware Database Replication*. Ph.D. dissertation, University of Lugano, Lugano, Switzerland.

ORLEANS, L. F, 2007, ORBITA: a load-balancing strategy for tasks with temporal constraints, M.Sc Thesis, PPGI/UFRJ

ORLEANS, L. F., ZIMBRÃO, G., FURTADO, P., 2008, “Controlling The Behaviour of Database Servers With 2PAC And DiffServ”. *In: International Conference on Database and Expert Systems Applications*, Turin.

ORLEANS, L.F., ZIMBRÃO, G., 2009, “MIDAS: A middleware for information systems with QoS concerns”. *In: 11th International Conference on Enterprise Information Systems*, Milan – Italy

OZSU, M., VALDURIEZ, P., 2011, *Principles of Distributed Database Systems*, Prentice-Hall, Englewood Cliffs, N.J.

PATIÑO-MARTÍNEZ, M., JIMÉNEZ-PERIS, R., KEMME, B., et al., 2005, "Middle-R: Consistent database replication at the middleware level". *In: ACM Trans. Comput. Syst*, pp. 375-423.

PEDONE, F., WIESMANN M., SCHIPER, A., et al., 2000, "Understanding replication in databases and distributed systems". *In ICDCS*, pp. 464-474.

PLATTNER, C., ALONSO, G., OZSU, M. T. Ä., 2006, "Dbfarm: A scalable cluster for multiple databases". *In Middleware*, pp. 180-200.

POESS, M., FLOYD, C., 2000, "New TPC benchmarks for decision support and web commerce". *In SIGMOD Rec. 29*, pp. 64-7

SCHROEDER, B., HARCHOL-BALTER, M., IYENGAR, A., et al., 2006, "How to determine a good multi-programming level for external Scheduling". *In Proceedings of the 22nd International Conference on Data Engineering*, pp. 60, Washington, DC, USA, IEEE Computer Society.

THOMSON, A., ABADI, D. J., 2010," The case for determinism in database systems". *In VLDB*.

WEIKUM, G., VOSSEN, G., 2002, *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann.

WIESMANN, M., SHIPER, A., PEDONE, F., et al., 2000, "Database Replication Techniques: A Three Parameter Classification". *In Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pp. 206–215.

WIESMANN, M., SCHIPER, A., 2005, "Comparison of Database Replication Techniques Based on Total Order Broadcast". *In: IEEE Transactions on Knowledge and Data Engineering (TKDE)*, pp. 551–566, April.

ZUIKEVICIUTE, V., PEDONE, F., 2005, "Revisiting the Database State Machine Approach". *In Proceedings of the VLDB Workshop on Design, Implementation and Deployment of Database Replic*ation, Trondheim (Norway).

ZUIKEVICIUTE, V., PEDONE, F., 2008, "Conflict aware load balancing techniques for database replication". *In 23rd ACM Symposium on Applied Computing (ACM SAC 2008)*. ACM.

Anexo I

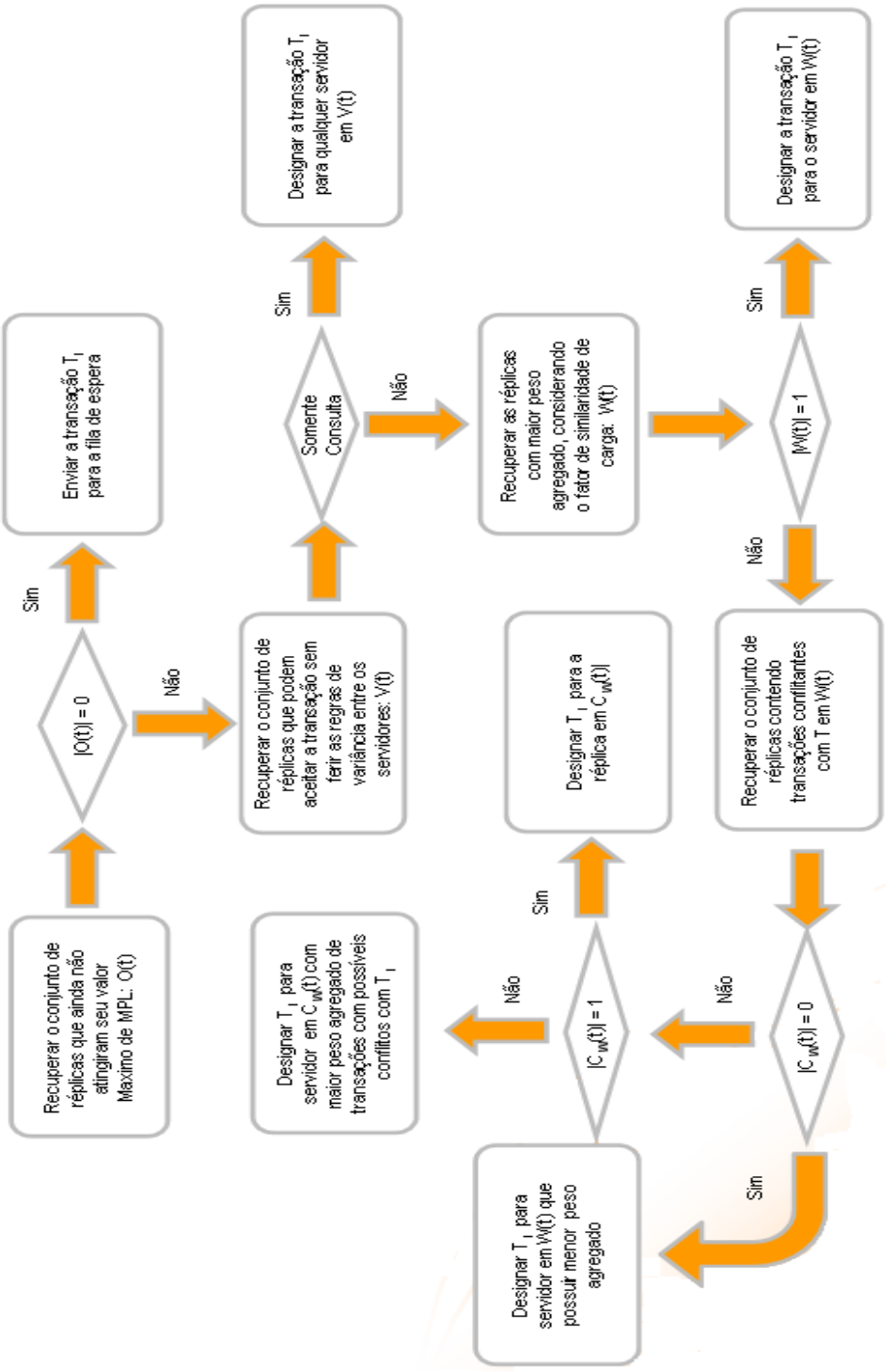

Figura 27 - Fluxograma de Balanceamento de carga adaptativo com restrições de conflito