Relatório Técnico - MAC 5701

Tópicos em Ciência da Computação

Replicação em Banco de Dados utilizando Consistência Fraca

Marcelo Camacho de Souza
mcsouza@ime.usp.br

Orientador :
Prof. João Eduardo Ferreira

ÍNDICE

1. Motivação

Ultimamente o uso de computadores móveis está ganhando muita popularidade. O número de usuários com tais computadores está crescendo e a tendência é que continue no futuro, onde o número de clientes móveis deverá superar em muito o número de clientes tradicionais ou *"desktop"*. Aplicações executadas em clientes móveis requisitam informações conectando-se periodicamente aos repositórios de dados. Clientes móveis constituem um novo e diferente tipo de carga de trabalho e exibem padrões de acesso diferentes daqueles vistos nos tradicionais sistemas cliente servidor.

Os sistemas de arquivo tem sido modificados para tratar clientes que podem fazer o *"download"* de informações, desconectar, e depois reintegrar. Já os banco de dados não tem sido reprojetados para acomodar os clientes móveis. Assim, existe a necessidade de suportar tais clientes no contexto de banco de dados cliente servidor.

Assim como a computação móvel torna-se mais e mais popular, os bancos de dados móveis também têm aumentado sua popularidade. Uma característica chave desses sistemas de banco de dados é sua habilidade para tratar a desconexão. Esta refere-se a condição onde sistemas móveis estão inaptos a comunicarem-se com parte ou todo o sistema. Nesta situação a unidade móvel não tem acesso aos dados compartilhados. Para tratar com o problema da desconexão tem-se utilizado esquemas de replicação otimista, que permitem às unidades móveis realizarem atualizações locais nos dados replicados. O problema fundamental desse esquema é a reconciliação, isto é, a serialização das atualizações potencialmente conflitantes dos clientes desconectados nas cópias do banco de dados.

Esta monografia está organizada da seguinte forma. Na seção 2, são descritos os objetivos desse estudo. A seção 3 descreve os principais trabalhos relacionados com o tema do projeto de dissertação. E finalmente, na seção 4 são apresentadas algumas ferramentas comerciais que tratam o conceito de banco de dados móveis.

2. Objetivo

O principal objetivo desejado com a realização desta disciplina foi obter um melhor subsídio para o planejamento do projeto de dissertação. Este, tem como proposta inicial estudar os diferentes esquemas de replicação de dados otimista envolvidos nas arquiteturas de banco de dados móveis, e propor uma solução, possivelmente híbrida, que permita conciliar diferentes modelos de replicação otimista, para tratar a replicação de dados em clientes desconectados. Essa solução envolverá além dos métodos de replicação, a própria formatação e composição dos chamados "Objetos Réplica", que conterão os dados a serem replicados. Tal proposta é muito bem aceita dentro dos padrões da arquitetura de computação móvel, pois permite tratar a mobilidade e desconexão dos clientes móveis, juntamente com a replicação de dados otimista. Nesta, as réplicas podem ser atualizadas a todo momento, estando ou não o cliente conectado ao servidor central. A reconciliação dessas atualizações, no servidor de dados central, é feita no momento da reconexão do cliente móvel, garantindo-se assim a propagação das atualização dos dados, mas sempre se respeitando as regras de consistência e integridade exigidas pelo sistema gerenciador de banco de dados.

Metas alcançadas conforme o plano de estudos proposto inicialmente :

- Levantamento bibliográfico relacionado as recentes pesquisas voltadas aos esquemas de replicação otimista. Foram estudados diversos artigos que tratam diferentes metodologias de replicação de dados otimista. Na seção 3 é apresentada uma avaliação comparativa dessas metodologias e das arquiteturas propostas nos artigos.

- Análise de arquiteturas comerciais relacionadas a bases de dados móveis. Foram avaliadas duas ferramentas comerciais que propõem um modelo de replicação de dados baseados na desconexão dos clientes, permitindo a realização de atualizações de dados mesmo quando desconectados, e a posterior propagação das mesmas ao servidor central, quando na reconexão dos clientes ao sistema. O interesse no estudo de tais ferramentas é proporcionar um melhor conhecimento das tecnologias empregadas e seus limites. Na seção 4 é apresentada uma descrição das características de cada ferramenta, bem como a possível contribuição no projeto de dissertação.

3. Trabalhos Relacionados

Recentemente muitas pesquisas tem sido direcionadas para esquemas de replicação otimistas em banco de dados móveis e reconciliação. O problema tem sido muito estudado no contexto de sistemas de arquivo. Gray et. al.[4] apresentam uma arquitetura de sistema e um modelo de replicação para bases de dados móveis. BAYOU[2,3] propõe um outro modelo para repositório de dados móveis. Neste as informações necessárias para o processo de reconciliação estão incluídas em cada atualização do repositório de dados. Dataman [5,6,7,8,9] apresenta uma arquitetura totalmente relacionado a fragmentação de objetos, não necessitando ou exigindo da aplicação qualquer semântica para realizar a fragmentação dos dados. O modelo proposto diferencia-se dos demais na flexibilidade do uso das políticas de reintegração das réplicas, escolhidas e gerenciadas pelo sistema de forma transparente a aplicação, e ao conceito de "Objeto Réplica", que representa a cópia do dado do servidor de banco de dados, replicado na aplicação cliente.

3.1. Bayou

3.1.1. Arquitetura

Em Bayou [2,3], as réplicas são cópias locais completas do repositório de dados. Cada servidor deve manter uma cópia. Essas cópias podem ser acessadas por outros *"hosts"* conectando em tais servidores. Todos os servidores estão interrompidamente conectados. Bayou define uma atualização como uma consistência tripla de um mecanismo de detecção de conflito, uma chamada de escrita, e um manipulador de resolução de conflito. Cada componente dessa tripla de atualização deve ser definida pela aplicação. Uma escrita ocorre somente se o mecanismo de detecção de conflito falhar ao detectar um conflito, de outro jeito o manipulador de resolução de conflito é invocado. Cada atualização feita desde a última vez que o banco de dados esteve globalmente consistente é registrada em um *"log"*. Sempre que dois servidores Bayou conectam-se, todas as transações no *"log"* são refeitas e os *"log's"* são combinados na ordem do *timestamp*. *"Log's"* combinados são usados para repetir as atualizações em ambos os servidores e tornar as bases de dados mutuamente consistentes. Bayou garante que na ausência de qualquer nova atualização, a base de dados convergirá para um estado consistente em todos os servidores, tão logo os servidores não fiquem desconectados para sempre. A responsabilidade da manipulação de conflitos entre as atualizações é das aplicações e o sistema não provê qualquer controle de concorrência global. Além disto, o próprio sistema não é um sistema de banco de dados totalmente maduro, e pode prover somente algumas garantias fracas para as aplicações.

3.1.2 Políticas de Replicação com Consistência Fraca

As políticas de replicação de dados, na arquitetura Bayou, variam segundo o grau de rigor exigido na consistência. Tal escolha afeta diretamente a latência das operações realizadas, a eficiência do canal de comunicações e a própria consistência dos dados. Assim, quanto maior for a consistência exigida, mais consistentes os dados tendem a ser, embora o tempo de latência das operações e o uso do canal de comunicação aumentem.

Um opção de política de replicação seria manter as réplicas totalmente independentes, onde as alterações nos dados são permitidas, sem a obrigatoriedade de que tais mudanças sejam comunicadas a outros servidores. Tal situação é muito útil em sistemas que não necessitam compartilhar as alterações das réplicas.

Outra opção é a política de consistência total, onde toda alteração deve ser propagada imediatamente a todo o sistema, impossibilitando que as demais réplicas dos dados sejam utilizadas por outras operações, até que todo o processo de propagação esteja concluído. Tal modelo apresenta o maior grau de consistência possível, já que garante que todas as réplicas estarão coerentes. Em geral, a latência das operações é muito alto, pois cada modificação representa a atualização de todos os servidores. Tal situação é desejada em sistemas nas quais os dados exijam alto grau de consistência.

Uma terceira opção é a utilização do protocolo "Anti-Entropia" para propagação das atualizações. Nessa política, cada servidor utiliza um *"log"* que representa a ocorrência das alterações nos dados. Quando uma atualização é realizada, um registro é feito no *"log"* contendo uma marca (*"timestamp"* ou número sequencial) gerada pelo servidor, juntamente com o identificador da máquina servidora. O propósito dessa marca é definir uma ordem parcial em todo o sistema. A propagação das atualizações consiste em fazer com que o *"log"* dos servidores contenha o mesmo conjunto de marcas de atualizações. A propagação se dá entre pares de servidores, sempre da forma unidirecional.

3.2. Anywhere Anytime Anyway

Gray [4] propõe o modelo de atualização "*anywhere-anytime-anyway"*. A base de dados é tratada como uma coleção de réplicas de objetos com cópias primárias controladas através de nós coordenadores conhecidos como objetos mestres. O modelo em questão distingue entre um nó móvel que fica desconectado a maioria do tempo, e nós base, que estão sempre conectados. Os objetos são replicados em um conjunto de nós usando um esquema de replicação de duas camadas. Uma camada abrange os nós móveis enquanto que a outra camada é contemplada pelos nós base. Cada nó móvel mantém uma cópia mestre do banco de dados, na qual é a cópia válida mais recente de um objeto mestre e uma cópia tentativa, na qual as atualizações foram realizadas. Cada transação *"read-write"* no nó móvel é considerada uma tentativa. Todas as transações tentativa são enviadas para o nó base na reconexão com um conjunto de critérios de aceitação. Se os resultados da transação tentativa na cópia mestre satisfizerem os critérios de aceitação, a transação base escreve na cópia mestre.

3.3. Arquitetura Dataman

Tal arquitetura é proposta pelo Laboratório de Computação Móvel DATAMAN, do Departamento de Ciências da Computação, da Universidade de Rutgers, Nova Jersey, Estados Unidos.

3.3.1. Modelo Básico da Arquitetura do Banco de Dados

O modelo consiste de uma base de dados centralizada com dois tipos de clientes : os tradicionais e os desconectados. Clientes desconectados são capazes fazer o *"download"* dos dados móveis, operar sobre eles (mesmo quando não há conexão com o servidor) e propagar atualizações a base de dados centralizada. Clientes tradicionais não podem operar de forma desconectada.

Os dados móveis, replicados nos clientes móveis, são fragmentos horizontais das relações. Assim o problema da escalabilidade está resolvido, pois não requer que relações inteiras sejam replicadas. O cliente pode desconectar do servidor e as aplicações podem continuar consultando e atualizando fragmentos no cliente. Quando conectados, clientes móveis comportam-se como clientes tradicionais e tem total acesso a base de dados.

Com a introdução de clientes móveis, servidores de bancos de dados podem esperar duas distintas cargas de trabalho. Uma gerada pelas requisições dos clientes móveis e outra resultante das consultas convencionais dos clientes tradicionais. A carga apresentada pelos clientes móveis é fracionada devido a natureza de seu modo de operação (*"download"* do dado móvel, seguido por períodos de pouca ou ausência de atividade, e por excessivas validações e escritas).

O servidor é o ponto de sincronização da arquitetura. Todas as atualizações devem finalmente ser propagadas ao servidor. Este também controla o privilégio de acesso para vários usuários. Relações inteiras são armazenadas no servidor (ao contrário dos clientes móveis que podem armazenar parte das relações). A organização física das relações no servidor está em termos dos fragmentos físicos, de forma a auxiliar uma rápida reintegração e replicação dos dados. As transações dos clientes tradicionais somente são executadas no servidor.

3.3.2. Organização do Dados no Servidor

É desejado que os clientes móveis demonstrem localidade de acesso. Para permitir isso as relações do servidor são fragmentadas horizontalmente, de forma a otimizar a eficiência da replicação e da reintegração. O cliente não precisa explicitamente especificar quais fragmentos fará o *"download"*.

Os fragmentos físicos são unidades de particionamento, reintegração, controle de concorrência e controle de acesso do modelo. Devem ser cuidadosamente projetados para capturar a localidade de acesso dos típicos clientes móveis. Cada cliente móvel deve informar ao servidor os fragmentos que ele pretende utilizar para replicar o dado. Cada cliente não precisa replicar o fragmento inteiro, entretanto do ponto de vista do servidor, o cliente pode replicar todos os fragmentos. É da responsabilidade do servidor resolver os conflitos devido a transição da réplica e prover algum meio de resolução de conflitos. O servidor deve também controlar o acesso a cada fragmento e manter um mapeamento entre clientes e os fragmentos que ele replicou.

3.3.3. Controle de Concorrência e Modelo de Consistência

Como aos clientes desconectados é permitida a atualização local dos dados, essa arquitetura utiliza a consistência fraca. Entretanto, diferentes clientes podem tentar atualizar o mesmo dado de diferentes formas. Assim o estado global do banco de dados torna-se inconsistente e o servidor tem que tratar as atualizações conflitantes.

Existem dois esquemas para tratar os problemas de consistência na presença de desconexão eminente. O primeiro esquema é o pessimista e requer explícita sincronização antes da desconexão para manter a consistência. Um exemplo é o modelo *"checkin"* / *"checkout"* onde o servidor simplesmente trava todos os acessos ao dado replicado pelo cliente, eliminando conflitos com o custo de requerer muitas informações avançadas, e reduzindo a disponibilidade do sistema. O segundo esquema é o otimista, e permite que atualizações conflitantes de todos os clientes móveis sejam resolvidas na reconexão. Esse esquema é difícil de implementar devido a conflitos na reconexão. Entretanto, é o mais atrativo já que provê alta disponibilidade do sistema, na qual é precisamente desejado nessa arquitetura.

3.4. Discussão

Em Bayou [2,3], as réplicas são armazenadas em repositórios de dados, e não em banco de dados. Além disso, não existe ordem total na execução das transações, assim a consistência dos dados não é totalmente garantida. Quaisquer eventuais conflitos que venham a surgir devem ser tratados pela aplicação, pois o sistema não dispõe de tal recurso. A utilização de tal arquitetura pode ser indicada para sistemas onde não existe um grande compromisso com a garantia da consistência dos dados. Tal proposta não condiz com as características desejadas no modelo de replicação a ser proposto no projeto de dissertação.

Gray [4] propõe uma arquitetura de banco de dados baseada no modelo Mestre/Escravo de controle de consistência. Nele todas transações somente são validadas se forem atualizadas na cópia Mestre, segundo critérios de aceitação estabelecidos pelo sistema. Esta é uma forma de consistência bastante utilizada comercialmente, mas não atende a forma como é desejado o controle de consistência na solução proposta. Possivelmente poderá ser utilizado como uma das opções do modelo de consistência na solução a ser projetada.

Em Dataman [5,6,7,8,9] é proposto um modelo de banco de dados relacional mais maduro e não requer que o cliente mantenha uma cópia de toda a relação ou base de dados. Além disto, a resolução de conflitos é transparente a aplicação. Esta é informada sobre a presença de conflitos para que possa resolvê-los. O sistema, que utiliza tal arquitetura, pode também ser especialmente projetado para prover automaticamente a resolução de conflitos para certas aplicações.

O modelo a ser proposto no projeto de dissertação se assemelha mais ao proposto em Dataman. Estamos criando o conceito de "Objeto Réplica", que representa o conjunto de dados a serem replicados nas unidades móveis, e contém a referência ao modo de reintegração da réplica no banco de dados. Será utilizado o esquema otimista para tratar os problemas de consistência, bem como diversos modelos de reintegração a serem escolhidos no momento da criação do "Objeto Réplica". Tal flexibilidade resulta em um modelo dinâmico de escolha de modelos de reintegração, permitindo de maneira fácil a introdução, a posteriori, de novos modelos ao sistema. Essa característica, não observada nos esquemas e modelos estudados acima, é a grande motivação dessa solução proposta.

4. Ferramentas Comerciais

4.1. Cloudscape

Cloudscape[10], desenvolvido pela Informix, é um sistema gerenciador de banco de dados relacional objeto (ORDBMS), escrito inteiramente em linguagem java, que permite ser encapsulado em qualquer aplicação java, não necessitando que esta se conecte a um banco de dados central, pois realiza chamadas locais ao ORDBMS. Para isso não necessita de quaisquer softwares adicionais para conexão em rede.

Sua utilização com outras aplicações java torna-se transparente ao usuário, pois não requer administração e executa na mesma máquina virtual. Permite também armazenar dados exatamente como a aplicação usa, isto é, objetos java, não necessitando de conversão dos modelos de negócio para outro tipo de dado.

A razão para usar um sistema gerenciador de banco de dados, ao invés de outra forma de armazenamento de informação, é permitir consultas mais complexas sobre um grande conjunto de dados, utilizar o processamento transacional, e ser um meio persistente de armazenagem de dados.

Cloudscape é dividido em três produtos : Cloudscape, que é o núcleo do produto; Cloudconnector, *"framework"* para construção de aplicações cliente/servidor, provendo conectividade JDBC e HTML para Cloudscape; Cloudsync, responsável pela sincronização de dados do sistema entre as várias aplicações de um sistema distribuído.
Cloudscape dispõe também de um *"framework"*, em seu núcleo, para a criação de aplicações cliente servidor, chamado RmiJdbc. Este é uma alternativa ao uso do Cloudconnector, que constitui um segundo produto sendo adquirido separadamente. Já a sincronização de dados somente pode ser realizada com a aquisição do produto Cloudsync.

A sincronização em Cloudscape é um processo na qual copia-se o dicionário de objetos, dados e objetos java originários de um banco de dados central para um ou mais banco de dados destino, e controla-se a consistência do dado. Como a sincronização em Cloudscape é baseada em mensagem, as bases de dados remotas não precisam estar continuamente conectadas por uma rede. Essas bases precisam somente se conectar intermitantemente, para através de envio e recepção de mensagems HTTP, atualizarem seus dados locais na base de dados central. Tal sincronização é realizada de forma transparente, sem necessidade de interação do usuário.

Ao utilizar Cloudscape deve-se seguir algumas orientações para que as atualizações nos dados sejam mantidas. Após o término do uso do banco de dados, deve-se finalizá-lo para que todas as alterações tenham a garantia de que foram escritas no *"log"* da base de dados. Caso não se proceda da forma descrita, na primeira execução do sistema o processo de recuperação será executado, restaurando o último estado consistente da base de dados, baseado nas informações armazenadas no *"log"*. Tal processo de restauração também é acionado após a interrupção da execução do sistema devido a ocorrência de alguma falha. O *"log"* do banco de dados não armazena apenas erros, mas também informações sobre a inicialização e finalização da execução da base de dados.

Clousdscape dispõe de uma aplicação gráfica chamada Cloudview que facilita a manipulação da base de dados, permitindo a criação de tabelas e chaves primárias e estrangeiras, a entrada de dados graficamente ou por comandos SQL, a abertura e execução de arquivos *"batch"* de SQL, etc. Dispõe também de ferramentas administrativas, tais como *Lock Diagnostics*, *Consistency Checker* e *Backup Utility*, para facilitar o gerenciamento do banco de dados.

4.2. JDataStore

JDataStore[11], desenvolvido pela Inprise, é um banco de dados relacional objeto, desenvolvido inteiramente na linguagem java, que permite ser anexado a outras aplicações java, e apresenta as seguintes características : tamanho pequeno, alta eficiência, portabilidade, custo zero de administração, replicação / sincronização com qualquer banco de dados que tenha suporte a *"drive"* JDBC, gerenciamento transacional e recuperação de queda. Também é um sistema de arquivo portável. Um simple arquivo JDataStore pode conter múltiplos streams. Um stream pode ser uma tabela, um índice secundário para uma tabela, um arquivo arbitrário ou mesmo um objeto java serializado.

Existem diversas formas para acessar os dados armazenados em um banco de dados JDataStore. Uma escolha é usar o padrão industrial, isto é, os *"drive's"* JDBC escritos em java, que tem suporte a SQL-92, ou utilizar algum componente padrão JavaBean.

O *"drive"* JDBC fornecido é do tipo 4, e permite acessos do tipo local e remoto utilizando o padrão industrial SQL call-level interface. Proporciona consultas otimizadas capazes de acessar as tabelas armazenadas no arquivo JDataStore, e acesso multi-usuário através da rede.

JDataStore aumenta a capacidade atual de armazenamento de dados nos clientes, pois é um banco de dados que pode manipular até 2 bilhões de registros por tabela. Permite também aos clientes móveis trabalhar com persistência, proporcionando computação desconectada com *"rollback"* de transações e recuperação de queda.

Transações e recuperação de queda podem ser habilitadas no banco de dados JDataStore apenas configurando-se algumas propriedades como diretório do arquivo de *"log"*, tamanho máximo do arquivo de *"log"*, tamanho dos blocos do arquivo de *"log"* e etc. Existe um valor *"default"* para todas essas propriedades, o que reforça ainda mais a característica de custo zero de administração da ferramenta. Em algumas situações críticas a aplicação decide qual decisão tomar. Por exemplo, após uma queda do sistema, na primeira vez que JDataStore for utilizado, automaticamente iniciará a recuperação dos processos para refazer o trabalho das transações que estavam em progresso, e refaz o trabalho também das transações ainda não propagadas. O sistema também controla a limpeza automática de antigos arquivos de *"log"*, na qual as transações já foram todas propagadas com sucesso.

Componentes específicos da arquitetura tratam a replicação dos dados e a sincronização das atualizações. Logo que os dados são retornados do servidor origem, tornam-se automaticamente persistentes na base de dados JDataStore. Agora a aplicação pode se desconectar e acessar os dados localmente. O dado pode ser lido e escrito usando-se *"drive's"* JDBC JDataStore. Tabelas dentro do JDataStore automaticamente registram todas as operações de inserção, atualização e remoção. Então quando a aplicação necessitar, se reconecta ao servidor de origem dos dados e componentes do JDataStore saberão quais operações devem ser propagadas. Esses componentes tem a capacidade de sincronização, onde através de mecanismos de resolução de conflitos, salvam as mudanças no servidor de origem, identificando os relacionamentos e ordenando operações para manter a integridade referencial do banco de dados. É claro que existem situações onde não se pode automaticamente gerenciar a reconciliação das modificações no servidor origem. Nesses casos, existem certas configurações que especificam como o sistema deve proceder.

4.3. Discussão

O intuito do estudo de tais ferramentas foi proporcionar um maior conhecimento das necessidades tecnologicas, de modo a facilitar a definição dos objetos e a solução híbrida de replicação de dados otimista, a ser proposta no projeto de dissertação.

Foram apontadas as seguintes necessidades de tecnologias emergentes :

- Computação fora do *"firewall"* : Ultimamente as organizações clássicas estão buscando aumentar a colaboração comercial de sua empresa, através da interação com os sistemas comerciais de seus parceiros. Lutam para conseguir ou aumentar a intimidade com seus clientes, por meio de ferramentas de informação de fácil uso. E finalmente, seus funcionários estão locomovendo-se cada vez mais, e precisam do sistema para permitir que mantenham sua produtividade sempre quando estiverem desconectados da empresa.

- Dilema do cliente leve ou pesado : As aplicações cliente/servidor provêem a rica funcionalidade e alta eficiência esperada para aplicações coorporativas, mas são muito complexas de serem desenvolvidas fora do *"firewall"*. Em contraposição, os *"browser's"* conectam-se a um servidor central web que provê alta conectividade, mas com limitada funcionalidade e eficiência. Para muitas aplicações que envolvem somente acesso de leitura, simples interação ou somente uso ocasional, a arquitetura de aplicação *"browser"* web é adequada. Mas para aplicações de negócio que são utilizadas por parceiros colaborativos, para aumento da intimidade com clientes, e para suporte a trabalhadores *"offline"*, o desafio é conciliar o melhor dos dois mundos.

- Sincronização das aplicações : Permite as aplicações receberem dados, lógica e esquemas atualizados de um computador central, e também o envio de eventos de negócio dos clientes remotos ao servidor central para processamento coorporativo. Tem como característica a alta disponibilidade e escabilidade, o aumento de segurança, a habilidade para integrar dados locais aos demais, e a possibilidade de trabalho *"offline"*.

Benefícios alcançados com tais tecnologias :

- Acesso : Operações na base de dados destino são mais rápidas porque seu tamanho é menor do que a base origem. Além disso, sua execução não compete com recursos no servidor de dados compartilhado. O DBMS está na mesma máquina da aplicação, ou, no pior caso, na mesma rede LAN, na qual tem velocidade muito maior do que uma rede WAN para acesso a base de dados central na origem.

- Disponibilidade : Aplicações podem executar mesmo quando a base de dados origem ou seu acesso estão indisponíveis. Essa flexibilidade resulta em maior disponibilidade da aplicação, porque uma falha em um simples ponto não consegue afetar todo o sistema.

- Segurança : Acesso restrito apenas a subconjuntos dos dados da base origem, segundo autorização permitida ao usuário. Restriçoes nas conexões com a base de dados origem. Proteção dos dados destino através de encriptação dos dados em disco.

- Balanceamento de Carga : Distribuição de operações do banco de dados central para outros computadores, mesmo com processamento *"offline"*, sem adicionar custo de manutenção ao sistema.

- Escalabilidade : Aplicações distribuídas podem tomar vantagem do fato da soma do poder dos computadores remotos exceder ao poder dos servidores centrais.

- Portabilidade : Fácil adaptação desses banco de dados as existentes aplicações desenvolvidas em java. Execução transparente a aplicação cliente (mesma *"JVM"*), não necessitando de nenhum recurso adicional.

- Mobilidade / Computação *"offline"* : Aplicações continuam disponíveis mesmo na ausência de conexão ao sistema coorporativo. A sincronização dos dados garante a propagação das alterações realizadas ao sistema.

- Persistência dos Dados : Armazenamento em disco dos dados no cliente.

A característica que mais motivou o estudos de tais ferramentas foi a replicação e a sincronização das atualizações. Por se tratarem de produtos comerciais, detalhes de tais implementações, como : algoritmos de sincronização e de resolução de conflitos, as estruturas de dados empregadas para armazenamentos dos dados, e as metodologias de controle de transação são omitidas, passando-se apenas uma visão geral das funcionalidades das ferramentas.

Foi realizado um estudo inicial da replicação e sincronização das atualizações em banco de dados. Deseja-se, como continuidade, traçar uma linha de testes para comparação com a solução híbrida de replicação de dados otimista, a ser proposta no projeto de dissertação.

5. Referências

[1] B. R. Badrinath and K. Ramamritham, Performance evaluation of semantics-based multilevel concurrency control protocols, ACM SIGMOD International Conference on Management of Data (37/206), pages 163-172, May 1990

[2] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer and B. Welch, The Bayou architecture : Support for data sharing among mobile users, Proc. of the IEEE Workshop on Mobile Computing and Applications, pages 2-7, Dec. 1994

[3] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, A. J. Demers. Flexible update propagation for weakly consistent replication. Proceedings of the 16th ACM Symposium on Operation Systems Principles, Saint Malo, França, páginas 288-301, Outubro 1997

[4] J. Gray, P. Helland, P. E. O'Neil and D. Shasha. The dangers of replication and a solution. Proceedings of ACM SIGMOD, pages 173-182, June 1996

[5] S. H. Phatak. and B. R. Badrinath, Database server organization for handling mobile clients, Department of Computer Science Technical report DCS-TR-324, Department of Computer Science, Rutgers University, New Jersey, 1997

[6] S. H. Phatak and B. R. Badrinath, An architecture for mobile databases, Department of Computer Science Technical report DCS-TR-351, Department of Computer Science, Rutgers University, New Jersey, 1998

[7] S. H. Phatak and B. R. Badrinath, Multiversion reconciliation for mobile databases, Proc. of International Conference on Data Engineering (ICDE), Sydney, Australia, pages 582-589, Mar.'99.

[8] S. H. Phatak and B. R. Badrinath, Data partitioning for disconnected client server databases, Proc. of International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'99), Seattle, Washington, Aug.'99.

[9] S. H. Phatak and B. R. Badrinath, Conflict resolution and reconciliation in disconnected databases, Proc. of Mobility in Databases and Distributed Systems (MDDS), Florence, Italy, Sep.'99.

[10] www.cloudscape.com, Cloudscape, Inc., Informix, Inc.

[11] www.jdatastore.com, Borland®, JdataStore™ versão 3.5, Inprise Corporation