**OGS/R: um Serviço para
grupos de objetos com
consistência relaxada**

MARCELO BRITO DOS SANTOS

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
Obtenção do grau
de
Mestre em Ciência da Computação

Área de Concentração: **Sistemas Distribuídos**

Orientador: **Prof. Dr. Markus Endler**

São Paulo - agosto - 2003

OGS/R: Um Serviço para Grupos de Objetos com Consistência Relaxada

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Marcelo Brito dos Santos e aprovada pela comissão julgadora.

São Paulo, agosto de 2003

Banca examinadora:

Prof. Dr. Markus Endler (orientador)	IME-USP
Prof. Dr. Fabio Kon	IME-USP
Prof. Dr. Edmundo Madeira	UNICAMP

Resumo

Replicação de objetos em sistemas distribuídos é comumente usada para garantir uma maior disponibilidade de serviços e de dados, bem como para implementar funções resilientes à falhas. O uso de serviços de grupo para a manutenção das réplicas de objetos torna possível uma separação entre as questões específicas da aplicação e as questões intrínsecas do gerenciamento de réplicas. Com a consolidação da computação móvel em nosso cotidiano, inúmeros trabalhos na área de replicação de objetos vêm pesquisando soluções para esta nova realidade. No entanto, há poucos serviços de grupo que dão apoio a sistemas nos quais haja a possibilidade de desconexões frequentes entre os nós da rede. Neste cenário, faz-se necessária a implementação de um estilo de replicação com consistência relaxada em um serviço de grupo de objetos.

Este trabalho discorre sobre as principais abordagens para manutenção de consistência entre réplicas, apresenta alguns trabalhos relacionados e especifica um serviço de grupo com consistência relaxada entre réplicas. Como resultado concreto, foi implementada uma extensão de OGS denominada OGS/R, que é um serviço CORBA de grupos de objetos que dá suporte à replicação com consistência relaxada.

Palavras-chave: Replicação de Objetos, Serviço de grupos, Consistência relaxada

Abstract

The replication of objects on distributed systems is a strategy to assure a higher availability of services and data, as well as to implement fail tolerant functions. It is possible to distinguish specific application issues from the ones related to replication management by using service groups for maintaining object replicas. As mobile computation settles down, much research has been made to find solutions for the new reality. However, there are few group services able to support systems in which frequent disconnections among the network nodes happen. In this scenario, the implementation of a replication style with relaxed consistency in a group service of objects is required.

This thesis presents the main approaches for maintenance of consistency among replications. It also presents some related works and specifies a group service with relaxed consistency among replications. From a more practical point of view, it was implemented in an extension of OGS defined as OGS/R, which is a CORBA service of object groups that support replication with relaxed consistency.

Keywords: Replication, Group Service, Relaxed Consistency.

Sumário

Resumo	ii
Abstract	iii
1 Introdução	1
2 Abordagens para Replicação de Dados e Trabalhos Relacionados	5
2.1 Abordagens para Replicação de dados	5
2.1.1 Consistência Forte	8
2.1.2 Consistência Fraca	11
2.1.3 Consistência Relaxada	13
2.2 Trabalhos Relacionados	18
2.2.1 CODA	18
2.2.2 Bayou	20
2.2.3 OGS+/Juggler	21
2.2.4 JGroup/ARM	22
2.2.5 Comparação e Avaliação	23
3 OGS e Juggler	25
3.1 OGS	25
3.2 Juggler	28
3.3 OGS+	29

4	O OGS/R	33
4.1	Operações estáveis	35
4.2	Operações instáveis	36
4.3	Integração & Visão	36
4.4	Especificação do Serviço	39
4.5	Implementação	42
4.5.1	Operações estáveis	42
4.5.2	Operações instáveis	42
4.5.3	Integração & Visão	43
4.6	Dificuldades Encontradas	45
5	Resultados Experimentais	48
5.1	Cenário 1	49
5.2	Cenário 2	50
6	Conclusão	55
	Referências Bibliográficas	57

Lista de Figuras

2.1	Um exemplo de Virtual-Primary-Copy com 3 réplicas.	17
2.2	Ambiente OGS/Juggler	21
2.3	Ambiente JGroup/ARM	23
3.1	Serviços do OGS.	26
3.2	Módulo mGroupAdmin do OGS+.	30
3.3	Módulo mGroupAccess do OGS+.	31
3.4	Módulo mGroupProtocol do OGS+.	32
4.1	Extensão do módulo mGroupAdmin.	39
4.2	Extensão do módulo mGroupAccess.	40
4.3	Extensão do módulo mGroupProtocol.	41
4.4	Diagrama de operação estável.	43
4.5	Diagrama de operação instável.	44
4.6	Diagrama da operação de Integração&Visão.	45
5.1	Comportamento dos estilos com 4 réplicas (1 primária)	50
5.2	Comportamento dos estilos com 6 réplicas (2 primárias)	51
5.3	Tempo de processamento para diferentes percentuais de operações instáveis, com 20% de reintegração	52
5.4	Tempo de processamento para diferentes percentuais de operações instáveis, com 50% de reintegração	52
5.5	Comportamento dos estilos com relação as operações instáveis, com 70% de reintegração	53

Capítulo 1

Introdução

Serviços de informação em redes de computadores devem idealmente apresentar um alto grau de disponibilidade e confiabilidade. Além disso, um usuário do serviço espera ter acesso às informações disponíveis independente de sua localização ou horário de acesso.

Percebendo os desafios atuais e inspirado em uma aplicação de sistema de controle de tráfego, o projeto SIDAM (Sistemas de Informação Distribuídos para Agentes Móveis)[16] voltou-se para o estudo de problemas relacionados ao projeto e implementação de sistemas de informação descentralizados para clientes fixos e móveis, onde os dados são eventualmente replicados em vários servidores. Neste cenário, encontramos aplicações como sistemas para monitoramento de tráfego urbano, sistemas de disseminação de notícias e sistemas para a coordenação de grupos de usuários/veículos móveis, tais como frotas de táxi, patrulhas de polícia, equipes de resgate, etc. Estas aplicações possuem uma característica em comum que é a existência de um grande número potencial de clientes fazendo consultas ao serviço de informações, como por exemplo para saber qual o fluxo médio de veículos em uma avenida, ou a posição geográfica de um taxi. Em muitos casos, a maioria dos clientes que efetuam consultas não requerem um alto grau de precisão das respostas, sendo suficiente que recebam respostas aproximadas dos dados. No entanto, estas mesmas aplicações eventualmente possuem um outro grupo de usuários “privilegiados”, como por exemplo os funcionários da Companhia de Engenharia de Tráfego, o coordenador de uma ação de resgate, etc, que precisam ter acesso aos dados consistentes e precisos.

Um dos principais desafios no desenvolvimento de sistemas de informação distribuídos

consiste justamente em conciliar os requisitos conflitantes de prover alta disponibilidade de dados e um acesso a dados consistentes e atualizados.

A busca por alta disponibilidade e acesso eficiente a dados em sistemas de informação tem sido um objetivo permanente da pesquisa e desenvolvimento em sistemas distribuídos. Dentro deste contexto, recentemente, o foco está voltado mais especificamente para sistemas de objetos distribuídos. Dentre as várias opções, a replicação de objetos tem se mostrado a abordagem mais promissora para garantir tanto uma alta disponibilidade, como uma maior resiliência à falhas. Porém, à medida que o número de réplicas utilizadas aumenta, mais custoso se torna o processo de mantê-las perfeitamente sincronizadas e/ou mutuamente consistentes. Portanto, a escolha da abordagem de manutenção de consistência de réplicas em um sistema de grande escala é uma questão complexa que deve levar em conta os requisitos específicos da aplicação com relação à disponibilidade e o grau de consistência de dados.

Há várias opções para a manutenção de consistência entre réplicas de um objeto, desde a manutenção de consistência forte até a manutenção de consistência fraca ou inexistente. Replicação de objetos com consistência forte é a abordagem que garante que o estado de todas as réplicas estarão sempre perfeitamente sincronizadas, ou seja, estarão consistentes. A replicação de objetos com consistência fraca tolera inconsistências entre os estados das réplicas em benefício de uma maior disponibilidade e eficiência de acesso à réplica.

Além destas, há ainda técnicas para manutenção de consistência relaxada. Estas técnicas propõem uma solução híbrida, que tenta combinar as vantagens da consistência forte e fraca. Para isso, estas técnicas toleram inconsistências entre as réplicas até um certo grau (ou durante um intervalo de tempo) e provêem mecanismos para a conciliação (ou reintegração) das réplicas divergentes segundo certos critérios de consistência, garantindo assim uma convergência gradual dos (estados) dos membros do grupo. Mais recentemente, replicação com consistência relaxada vem sendo bastante pesquisada e apontada como a melhor abordagem para serviços em redes móveis, onde replicação e *caching* são usadas para tornar mobilidade e desconexões temporárias transparentes para usuários móveis.

De qualquer forma, a escolha da melhor técnica de manutenção de consistência deve levar em conta não só particularidades da rede (por exemplo, mobilidade, grau de conectividade), como

também os requisitos da aplicação e o custo operacional da sua execução.

Outro fator relacionado à replicação de objetos é a forma de comunicação entre as réplicas. A maioria dos trabalhos estudados [20, 15] implementa comunicação de grupo. O paradigma de grupos de objetos [23] introduz uma abstração sobre o conjunto de réplicas, simplificando a comunicação entre a aplicação e as réplicas de um objeto. A maioria das implementações desse paradigma é baseada na replicação de processos e implementa primitivas de comunicação de grupo confiáveis, com ordenação causal (ou total) de eventos distribuídos, garantindo assim o comportamento previsível das aplicações distribuídas. Na utilização de serviços de grupo é importante tentar minimizar a sobrecarga na troca de mensagens entre os membros do grupo. Além disto, a escolha da forma de manutenção de consistência tem influência sobre o desempenho da aplicação que utiliza o grupo, que em grande parte é determinada pela complexidade (na troca de mensagens) do serviço de grupo utilizado.

O serviço de grupo para CORBA (*Common Object Request Broker Architecture*) [22] que serviu como base de pesquisa no projeto SIDAM foi o OGS (*Object Group Service*)[15], que implementa uma infra-estrutura básica para a criação de grupos de objetos. Sua versão inicial implementa consistência forte entre os membros do grupo e apresenta apenas uma interface de baixo nível para a administração de réplicas de seus grupos. O Juggler e o OGS+ [17], implementam respectivamente uma interface de alto nível para a administração de grupos de réplicas e uma extensão ao OGS com estilos adicionais de replicação de objetos. A implementação da comunicação de grupo no OGS, assim como no OGS+, tem um alto custo pois se baseia em um protocolo para consenso distribuído para a manutenção de consistência entre as réplicas, o que torna o serviço pouco escalável. Uma forma de reduzir o custo de comunicação em grupo e tornar o serviço de grupo mais escalável, seria a de incorporar o estilo de replicação com consistência relaxada no OGS.

O principal objetivo desta dissertação foi a especificação e a implementação de um serviço de grupo que incorporasse um estilo de replicação com manutenção de consistência relaxada, que foi chamado de OGS/R. O novo estilo de replicação implementado oferece três tipos de operações sobre um grupo de réplicas: operações estáveis, operações instáveis e operação de integração&visão. As operações estáveis são operações que mantêm as réplicas (ou um subconjunto

delas) com um estado consistente; as operações instáveis são operações realizadas somente em uma das réplicas, o que não garante a consistência com as demais réplicas; e a operação de integração & visão é responsável pela convergência entre os estados de réplicas divergentes. Através destas operações, uma aplicação cliente pode optar entre um acesso possivelmente mais lento, mas com garantia de consistência entre as réplicas, ou entre um acesso rápido (e não consistente) a uma réplica e reintegração explícita (por demanda) desta réplica com as demais.

Esta dissertação está organizada da seguinte forma: No Capítulo 2, discutiremos sobre os principais protocolos de manutenção de consistência entre réplicas e avaliaremos alguns trabalhos relacionados a cada uma delas. No Capítulo 3, apresentamos o Serviço de Grupos de Objetos (OGS), o serviço de gerenciamento de grupos Juggler e as extensões implementadas no OGS+. No Capítulo 4 apresentamos o OGS/R, o serviço de grupo de réplicas que implementa consistência relaxada, descrevendo o algoritmo de reintegração, interfaces e questões de implementação. O Capítulo 5 apresenta os experimentos realizados com o OGS/R e seus resultados. Por fim, no capítulo 6 são apresentadas as conclusões deste trabalho.

Capítulo 2

Abordagens para Replicação de Dados e Trabalhos Relacionados

Nas seções que seguem, discutiremos as diversas abordagens para a manutenção de consistência entre réplicas para depois comentarmos e compararmos ferramentas e sistemas que implementam algumas destas abordagens.

2.1 Abordagens para Replicação de dados

A replicação de dados pode prover tolerância a falhas em um sistema distribuído, mascarando falhas de réplicas e possibilitando o aumento da disponibilidade de um serviço. A replicação de dados possibilita ainda distribuir melhor o volume de requisições de acesso ao dado entre as réplicas, possibilitando assim um melhor balanceamento de carga e consequentemente aumentando o desempenho do sistema. Protocolos de replicação são utilizados para manter a consistência entre as réplicas que representam um mesmo dado. No entanto, o protocolo de replicação pode se tornar um fator limitante do desempenho do sistema, pois a sincronização e a propagação do estado entre as réplicas geralmente tem um alto custo, que aumenta com o número de réplicas do grupo.

Chamamos de dado representado o dado/objeto lógico representado por um conjunto de réplicas (instâncias do dado). Considera-se réplica consistente aquela que apresenta coesão

com o dado representado. Portanto, toda réplica de objeto que esta no mesmo estado do dado representado é tida como consistente. Segundo a *teoria de serialização*[1], qualquer operação sobre um dado consistente deve levá-lo a um estado consistente. A teoria da serialização afirma ainda que a execução de operações concorrentes sobre um mesmo dado deve ter o mesmo efeito de uma execução seqüencial das mesmas sobre o dado.

Uma réplica é dita inconsistente, ou divergente, se não está no mesmo estado que o objeto representado. Em operações não coordenadas, ou seja, não serializadas a manutenção de réplicas consistentes é bem mais complexa. Chama-se, então, de *convergência*, a garantia de que todas as operações submetida às réplicas serão realizadas e que, em algum momento, todas as réplicas serão levadas a um estado consistente.

Um protocolo de replicação com *consistência forte* garante que todas as réplicas (de um mesmo objeto ou dado) estão sempre sincronizadas entre si[1]. Esta categoria de protocolos garante que as réplicas permaneçam consistentes com o dado representado, com isso incorporam o custo da sincronização das réplicas. Portanto, a consistência forte entre as réplicas pode comprometer o desempenho do sistema.

Protocolos de *consistência fraca* não garantem que as réplicas estejam sempre sincronizadas, deixando que as mesmas divirjam do dado representado. Devido à inexistência de um protocolo para a sincronização das réplicas, obtém-se uma maior disponibilidade de acesso ao dado, e aumenta-se o desempenho do sistema. Em computação móvel a manutenção de réplicas com consistência fraca é uma técnica praticamente inevitável, devido à possibilidade de haver nós temporariamente “desconectados”.

A *Consistência relaxada* permite um certo grau de inconsistência entre as réplicas. Para garantir a co-existência harmônica de réplicas consistentes e réplicas inconsistentes, que a qualquer momento podem ser levadas ao estado das réplicas consistente, os protocolos exigem uma reintegração entre as réplicas inconsistentes e as consistentes do grupo, que é realizada segundo *critérios de consistência* que determinam o grau/tipo de inconsistência máxima tolerada.

A disponibilidade de um dado recurso em um sistema é medida pela probabilidade de acesso instantâneo ao mesmo. A replicação de dados permite um aumento da disponibilidade, que atinge seu grau máximo nos protocolos de consistência fraca. O bloqueio de acesso às réplicas

durante as operações é uma característica dos protocolos de consistência forte, que reduz a disponibilidade das mesmas, mas garante a consistência das réplicas. Baseando-se nos requisitos de disponibilidade e de consistência da aplicação deve-se escolher a melhor forma de manutenção de consistência entre réplicas.

Protocolos de controle de concorrência são utilizados para restringir a execução concorrente de operações que podem levar o dado representado a um estado inconsistente. Os protocolos de controle de concorrência são divididos em duas categorias: pessimistas e otimistas. Os protocolos pessimistas impedem a execução de operações possivelmente não serializáveis, prevenindo inconsistências. Nestes casos a operação é anulada e mantém-se o dado no estado consistente anterior. Um exemplo de protocolo pessimista é o bifásico (Two-Phase Commit), amplamente usado em transações de banco de dados. Os protocolos otimistas, por outro lado, permitem a execução de operações não serializáveis, mas empregam mecanismos de detecção de anomalias. Operações não consolidadas são então abortadas evitando que as alterações afetem o estado do dado. Um exemplo de protocolo otimista é o protocolo de certificação[10], com validação antes da confirmação da operação.

A utilização de protocolos de consistência forte garante que as réplicas são idênticas ao dado representado. Dessa forma temos a idéia de *visão global*, onde as réplicas são uma representação lógica de um objeto/dado, que permanecem consistentes após cada operação. Em contraste, a utilização de protocolos de consistência fraca possibilita a existência de réplicas inconsistentes, introduzindo o conceito de *visão local* do dado. Na consistência fraca, qualquer réplica é uma *representação local* do objeto, podendo executar operações localmente e assim diferir de outra visão local do mesmo objeto. Este conjunto de visões locais geralmente são mutuamente inconsistentes.

Nas seções seguintes apresentamos uma visão geral das técnicas para manutenção de consistência de réplicas e suas principais aplicações. Na literatura, estas técnicas [1, 2] são geralmente classificadas de acordo com a forma de manutenção de consistência em técnicas de consistência forte, fraca e relaxada.

2.1.1 Consistência Forte

Uma regra é primordial na manutenção de consistência forte entre réplicas: cada execução concorrente deve equivaler a uma serial. A idéia de *one-copy serialiability* (1-SR) faz com que todas as réplicas participem da execução serializada para um objeto, sendo o resultado propagado para todas elas. Assim, é garantida a consistência de todas as réplicas. Toda operação de leitura e escrita em um dado consistente é atômica e leva a um estado consistente.

As técnicas de consistência forte, também denominadas consistência estrita, podem ser classificadas em três categorias: cópia corrente, votação e híbrida[1]. Em cópia corrente atualizações são aplicadas sincronamente em todas as réplicas acessíveis, enquanto que a leitura é feita em apenas uma réplica. Nas técnicas de votação, cada réplica possui o poder de voto. Cada operação recolhe os votos e realiza-se um consenso dos votos. Existem também técnicas que são híbridos das técnicas de cópia corrente e votação.

Muitas dessas técnicas não são escaláveis para um grande número de réplicas pois elas se baseiam em protocolos que envolvem muitas trocas de mensagens, sendo portanto inviáveis para um número grande de réplicas. Os protocolos de cópia corrente requerem participação de todas as réplicas e fazem um bloqueio temporário de todas as réplicas nas operações de escrita. Os protocolos de votação requerem muita comunicação entre as réplicas a fim de obter um consenso entre todas as réplicas para cada operação.

Cópia Corrente

As técnicas classificadas como cópia corrente são também chamadas de *Read-One-Write-All*(ROWA). A ROWA é a mais genérica das abordagens. As demais abordagens baseadas em cópia corrente são variações deste modelo. Este algoritmo converte uma operação de leitura no dado em uma única operação de leitura em qualquer uma das réplicas. Sendo n o número de réplicas, uma operação de escrita é convertida em n escritas, uma em cada réplica, e durante todo este processo todas as réplicas ficam bloqueadas. No caso de uma réplica não estar disponível, por exemplo devido à uma falha no respectivo host, todas ficam bloqueadas até a recuperação da falha. Para suprir esta deficiência, surgiu uma variação da técnica, denominada ROWA-A.

Read One/Write All Available (ROWA-A)

Nesta abordagem, uma operação de escrita já não requer uma resposta de todas as réplicas, mas somente das réplicas acessíveis. Um host que falhou, quando volta, tem sua réplica excluída até que se copie o estado atual de alguma réplica participante para a nova instância. Neste momento, a réplica que falhou se torna novamente disponível. Nesta técnica, uma das réplicas faz o papel de coordenador do grupo de réplicas, que decide quais operações serão efetivadas nas demais réplicas. As escritas em réplicas que não respondem ao coordenador são chamadas de *Missing Writes*(MW) e são armazenadas pelo coordenador. Antes de confirmar uma operação, o coordenador inicia um protocolo de validação em duas etapas:

1. *Missing Writes Validation*, determina se todas as réplicas que causaram MW estão inacessíveis. Para isto, o coordenador envia um UNAVAIL para todos os hosts que tem um MW. Se algum responder, a operação é abortada por inconsistência de uma réplica. Caso contrário é realizado o próximo passo.
2. *Access Validation*, determina se todas as réplicas que sofreram operação de leitura ou escrita estão de fato acessíveis. Para isto, o coordenador envia um AVAIL para todos os hosts que contenham réplicas acessíveis. Caso todos respondam, a operação é confirmada.

Esta abordagem tem algumas desvantagens: Por exemplo, a de que as repetidas tentativas de escrita em réplicas inacessíveis gera um grande volume de requisições armazenadas no coordenador. Como o tempo de indisponibilidade de uma réplica é indeterminado, pode acontecer que a lista de requisições armazenadas cresça indefinidamente.

Primary-Copy

Primary-copy é a mais utilizada das abordagens ROWA. O modelo básico garante que toda leitura e escrita seja feita em uma única réplica, chamada *cópia corrente* ou *primária*. As demais réplicas são chamadas de *backups*. Ou seja, as operações sofrem serialização na cópia corrente. Este protocolo se utiliza de um coordenador para operações de escrita que é encarregado de propagar imediatamente o resultado da operação para as demais réplicas. Só quando esta

propagação for confirmada em cada uma das réplicas backups confirma-se o sucesso da operação. Quando a cópia corrente falha, uma réplica backup assume o seu papel. Esta decisão é feita através de uma estratégia pré-determinada ou por votação e requer um monitoramento mútuo entre as réplicas. O sistema de nomes NIS(Naming Information Service)[21] basea-se na abordagem primary-copy.

A técnica de *primary-copy* tem como vantagem a alta disponibilidade em transações de leitura. Como desvantagem não tolera falhas esporádicas na comunicação ou partições de rede, necessitando portanto de redes confiáveis. Como alternativa para suprir esta deficiência temos as técnicas de votação.

Votação

A técnica de votação, também conhecida como *Quorum Consensus* (QC), garante a escrita em um subconjunto (quorum de escrita) das réplicas acessíveis e a leitura em um outro subconjunto de réplicas (quorum de leitura). Além disto, estes subconjuntos precisam ter uma intersecção (réplicas comuns) não vazia. Esta intersecção garante que a operação de leitura terá como resultado o valor da última escrita. Assumindo que as réplicas estão divididas em quorums, elas entram em consenso para efetuar o retorno das operações realizadas sobre o quorum ao qual pertencem.

Uma vantagem dos protocolos de votação é a tolerância a falhas de comunicação e de hosts, pois não precisam de intervenções para tratar do particionamento da rede ou junção de partições. No entanto, para manter esta tolerância é necessária muita comunicação entre as réplicas, o que pode vir a ser uma desvantagem. Buscando-se as vantagens de protocolos de votação e cópia primária temos protocolos híbridos como o ROWA/QC.

Híbrido ROWA/QC

Para redes confiáveis mostrou-se que o quorum consensus é caro, pois suas trocas de mensagens são desnecessárias neste cenário. Por outro lado, mostrou-se que a abordagem de cópia corrente é eficiente, mas pressupõe redes sem falhas, o que não é garantido na maioria das aplicações. Assim, a técnica híbrida de cópia corrente e votação propõe um método para

reduzir este custo, adotando ROWA durante períodos de comunicação confiável e chaveando para QC durante períodos nos quais eventualmente haja falhas de hosts ou comunicação. Ou seja, as réplicas executam em ROWA-A no *modo normal* e, quando percebem que uma réplica não responde, chaveiam para o *modo de falha*, quando passam a utilizar a técnica de votação.

A presença de um MW (ou seja, de uma operação pendente para uma réplica) indica que alguma réplica não está consistente com as demais. Para armazenar os MWs é associada a cada réplica uma lista de MWs com versões, chamada MWL. As MWLs ficam não nulas em todas as réplicas participantes quando alguma réplica falhou, a partir daí todas as réplicas ativas guardam os MWs e o coordenador faz o chaveamento para o modo de falha (modo QC). Quando a réplica tiver sido recuperada, ocorre uma transação de cópia em que qualquer uma das réplicas ativas repassa para a nova réplica as operações que esta não sofreu. Após esta transação todas as MWLs voltam a ficar vazias. Quando a MWL do coordenador está novamente vazia este faz o chaveamento para o modo normal e volta-se à técnica de cópia corrente. Quando o algoritmo está operando em modo de falha, armazena-se as MWs em todas as réplicas a fim de aumentar a tolerância à falha, o que não ocorre quando mantém-se a informação apenas no coordenador. Em caso de falha no coordenador, ocorre uma eleição para determinar o novo coordenador, e como este também possui as MWs necessárias, ele pode assumir imediatamente a sua função.

2.1.2 Consistência Fraca

A replicação é utilizada em sistemas distribuídos para aumentar a disponibilidade de dados e o desempenho no acesso. Tradicionalmente, os requisitos de consistência em bases de dados distribuídos são rígidos. Estes requisitos têm o objetivo de garantir a transparência da distribuição e a reprodução do paradigma de transações ACID para o ambiente distribuído. Mais recentemente, muitas abordagens vem propondo o relaxamento destes requisitos de consistência forte. O termo consistência fraca pode ser usado para qualquer abordagem que relaxa as restrições de consistência forte[2]. Neste trabalho, chamaremos de consistência fraca os protocolos que não especificam uma restrição de consistência entre réplicas ou os protocolos que fazem propagação epidêmica[3] e assim garantem que em algum momento haverá consistência entre as réplicas. Nesta linha, vamos comentar o protocolo de *Atualizações Independentes* e o *Timestamped Anti-Entropy*.

Atualizações Independentes

Atualizações Independentes[14] aumentam a autonomia de cada réplica e possibilitam um melhor desempenho em operações de escrita. Por outro lado, por não haver a propagação da operação de escrita imediatamente, o protocolo para manter a consistência entre as réplicas é bem mais complexos. Há o armazenamento das operações sofridas em cada réplica, prevendo o momento de propagação e sincronização das réplicas.

A aplicabilidade deste protocolo fica restrita a casos onde o grau de inconsistência da réplica local não afeta a solução. A aplicação que utiliza esta abordagem não deve precisar alcançar uma consistência global, pois inconsistências globais geradas não são resolvidas por rotinas de resolução de conflitos locais, como é o caso desta técnica. Protocolos de resoluções de conflitos globais requerem um grande número de trocas de mensagens, o que afeta a escalabilidade dos sistemas.

Timestamped Anti-Entropy(TSAE)

O TSAE[3] é um dos protocolos de consistência fraca mais conhecidos, que provê garantias de que a consistência ocorre em algum momento, confiabilidade e garantias de ordem nas operações[4]. Isto significa que todas as réplicas de um dado executam todas as operações na mesma ordem, de forma não simultaneamente em um período de tempo finito, mas em um espaço de tempo indeterminado.

Um protocolo de consistência como o TSAE é *confiável* se garante que toda réplica recebe em algum momento uma cópia de toda mensagem enviada para o grupo[3]. O TSAE garante esta restrição armazenando as mensagens enviadas, baseando-se em *acknowledgments* de mensagem.

O protocolo mantém dois vetores de *timestamp*. Um guarda quais mensagens foram recebidas, enquanto que o outro registra os *acknowledgments* recebidos de outras réplicas. Os vetores contém um timestamp para cada operação nas réplicas. De tempos em tempos, uma réplica escolhe uma outra réplica e dá início a uma *sessão de anti-entropia*. Uma sessão começa com as réplicas alocando um timestamp, e trocando seus vetores de *acknowledgment* e mensagens. Cada réplica detecta se possui mensagens que a outra não recebeu, apenas checando se os seus timestamp de mensagens são maiores do que os correspondentes na outra réplica. As mensagens

ainda não recebidas por uma réplica são então enviadas pela outra réplica. Se algum destes passos falhar, a sessão é abortada e as mudanças são descartadas. A sessão termina com a troca de acknowledgments entre as réplicas. Ao término da sessão ambas as réplicas possuem o mesmo conjunto de mensagens. Isto é muito útil para difusão de mensagens em grupos de réplicas.

Os vetores de acknowledgment e mensagens determinam de forma precisa e eficiente o quão divergente está uma réplica da outra. Isso permite a transmissão apenas uma vez das mensagens não recebidas para cada réplica. Através do vetor de acknowledgments tem-se uma forma eficiente de verificar se todas as réplicas receberam a mensagem. De tempos em tempos é realizada a sessão de anti-entropia para fazer com que os estados das réplicas não divirjam do grupo de forma acentuada.

O protocolo TSAE foi usado no desenvolvimento do RefDBMS[3] que implementa um banco de dados bibliográfico distribuído. A mesma abordagem também foi utilizada no projeto Bayou[6] do Xerox PARC, que pesquisou a replicação em ambientes móveis. O Bayou é um sistema de armazenamento distribuído com consistência fraca para um ambiente de computação móvel segundo o modelo *peer-to-peer*, que inclui máquinas portáteis esporadicamente conectadas. Este sistema possibilita leituras e escritas em qualquer réplica acessível. Para obter consistência entre as réplicas, o Bayou possibilita desfazer operações e refazê-las para uma serialização global, baseando-se no timestamp das operações submetidas ao grupo.

2.1.3 Consistência Relaxada

Consistência relaxada é uma técnica de manutenção de consistência mais rígida do que as técnicas de consistência fraca, mas com menos restrições do que as exigidas para a consistência forte. Protocolos de consistência relaxada¹ são caracterizados pela especificação de momentos de reintegração, o relaxamento na serialização das operações ou o uso de consistência insular.

Os gatilhos para reintegrações podem ser divididos em duas categorias, de acordo com a especificação da dimensão temporal ou espacial de consistência entre réplicas[2]. A *dimensão espacial* descreve o grau de consistência (isto é, a divergência entre o dado representado e a

¹Alguns autores, como Lenz[2], não utilizam o termo consistência relaxada, denominando os protocolos de consistência relaxada também como protocolos de consistência fraca.

réplica), ou a qualidade do dado de uma certa réplica. A *dimensão temporal*, por outro lado, especifica quando a igualdade entre os estados das réplicas é necessária, ou por quanto tempo a inconsistência é tolerada. As restrições temporais podem ser eventos ou intervalos de tempo. Os eventos indicam um certo ponto no tempo no qual é necessário verificar a consistência de estados e atualizar as réplicas. Quanto às restrições espaciais, elas podem ser classificadas em consistentes, fracas e sem restrições. As consistentes garantem que a réplica é idêntica ao dado representado, enquanto que a fraca tolera uma “divergência máxima” entre os estados das réplicas. A divergência entre os estados pode ser especificada de diferentes formas, usando versionamento ou um valor de divergência.

As restrições necessárias para a consistência relaxada só são justificáveis para uma replicação de dados com requisitos de alta disponibilidade e baixo sincronismo, com operações não conflitantes. Um exemplo clássico de aplicação na qual a consistência relaxada é inviável é o débito e o crédito em conta corrente, processo este que necessita de serialização. Nos dias de hoje, protocolos com manutenção de consistência relaxada vem sendo aplicados em trabalhos na área de computação móvel[24], pois permitem mobilidade (e desconexões) às réplicas, possibilitando operações locais que mais tarde serão aplicadas ao grupo de réplicas consistentes.

A seguir discutiremos alguns protocolos de consistência relaxada, conforme descrito em [3].

Quasi-Copies

O protocolo Quasi-Copies[12] permite especificar limites de divergência entre as réplicas. A convergência das réplicas é garantida desde que as atualizações sejam feitas apenas na cópia corrente (primária). Através de um *predicado de convergência* há uma verificação de consistência e propagação contínua do estado da réplica primária para as réplicas divergentes. A especificação de intervalos de inconsistência entre réplicas é formulada em termos dos conceitos de réplica *master*(primária) e réplicas *slave*(secundária). Um usuário pode especificar que o valor da réplica slave pode divergir da réplica master não mais que uma determinada constante, ou que a réplica não deve ficar desatualizada por mais que um determinado período de tempo.

O protocolo Quase-Copies tem como vantagem o tratamento de inconsistências e como

desvantagem a serialização das operações na cópia corrente.

Epsilon Seriability

Epsilon Seriability(ESR)[13] é uma técnica de controle de concorrência, que não foi desenvolvida especificamente para sistemas distribuídos e sim como uma extensão da seriabilidade estrita que permite controlar inconsistências. Ao invés de reforçar requisitos da seriabilidade em todas as transações, como em um protocolo bifásico, a serialização é aplicada somente às operações de escrita.

Este protocolo de controle de réplicas utiliza a propagação assíncrona, ou seja, as réplicas não precisam executar as operações simultaneamente. Portanto, o protocolo pode ser classificado como de consistência relaxada por especificar operações a serem serializadas e operações não coordenadas. As escritas são realizadas na mesma ordem em todas as réplicas e cabe à aplicação limitar o quanto inconsistente poderá ficar uma réplica.

ESR é mais escalável do que a serialização estrita e pode definir mecanismos de serialização para a redução de conflito em operações concorrentes em sistemas de consistência fraca. As operações definidas como serializáveis utilizam-se de bloqueios para manter a consistência. A possibilidade de serialização parcial (em atualizações) melhora o grau de disponibilidade das réplicas, mas por ser um protocolo baseado em serialização das operações sobre o grupo de réplicas não é uma boa abordagem para serviços com grande número de réplicas[3].

O protocolo de Quasi-Copies oferece a possibilidade de especificar o grau de divergência, mas serializa qualquer operação de escrita sobre o grupo de réplicas. Por outro lado, a ESR possibilita a especificação de quais escritas necessitam ser serializadas e quais não são coordenadas.

Lazy-Replication

Lazy-Replication[11] garante consistência entre as réplicas em ordem causal ou total. As aplicações podem especificar exatamente quais as relações causais entre as operações nas réplicas que devem ser garantidas. Pode-se ainda especificar relações causais com eventos externos ao grupo de réplicas. Com isso o tratamento de escritas conflitantes é pontual, possibilitando uma maior disponibilidade das réplicas.

Em Lazy-Replication cada réplica mantém um vetor com as mensagens recebidas e troca mensagens (contendo este vetor) com as demais réplicas para a manutenção da consistência. Este protocolo envolve um número maior de mensagens, pois as informações de atualizações trocadas entre as réplicas são enviadas em um único sentido em mensagens *gossip*, que são semelhantes à sessão de anti-entropia do protocolo TSAE. Devido ao fato da troca de mensagens não ser bidirecional, esta operação é de desempenho inferior à sessão de anti-entropia, enviando informações redundantes já recebidas pela réplica destino.

Virtual-Primary-Copy

Nesta técnica, introduz-se a abstração de cópia primária virtual (*virtual primary copy*) [2], que é o termo utilizado para o conjunto de réplicas que estão consistentes com o dado representado pelas réplicas do grupo. A cópia primária virtual também é chamada de *ilha de consistência* e a técnica é conhecida também por *consistência insular*.

Réplicas podem entrar ou sair da ilha de consistência de acordo com os requisitos da aplicação. A réplica que entra na ilha recebe o estado das demais réplicas da ilha. Além disto, todos os outros membros da ilha são informados do novo membro, pois este, a partir deste momento, participará das sincronizações na ilha.

As operações realizadas na ilha de consistência são serializadas de acordo com 1-SR[1], mas especificamente seguindo o protocolo *ROWA-Available* (Seção 2.1.1), protocolo este que possibilita leituras locais em qualquer réplica e propagação de operações de escrita para todos os membros da ilha. Aplicações que acessam as réplicas fora da ilha também podem requisitar escritas. No entanto, atualizações em réplicas fora da ilha são chamadas *blind writes*. As blind writes são escritas nas quais o valor corrente da réplica é desconhecido. Isto difere do conceito de atualização, pois esta tipicamente depende do valor corrente do dado. A blind write é redirecionada para a ilha de consistência, onde a operação é realizada como se fosse requisitada diretamente a uma réplica da ilha. Operações bem sucedidas na ilha são propagadas para as demais réplicas de acordo com a especificação de consistência fraca para estas réplicas. Assim, a ilha de consistência funciona como uma cópia primária para as demais réplicas (Fig. 2.1).

Vale ressaltar que nesta abordagem não devem existir ilhas de consistência vazias. Uma

Figura 2.1: Um exemplo de Virtual-Primary-Copy com 3 réplicas.

estratégia simples para manter a ilha não nula é impedir que uma réplica deixe a ilha, caso esta seja a única integrante da ilha.

Virtual-primary-copy é parte de um projeto para especificação de consistência orientada a aplicação chamado ASPECT[2], que visa reduzir o número de réplicas síncronas e especificar os requisitos de consistência dinamicamente.

Outras Abordagens para Consistência Relaxada

Algumas pesquisas visam integrar consistência forte e fraca em um mesmo arcabouço[1]. Noha Adly[7] propõe em seu arcabouço operações *fast_read* e *fast_write* para consistência fraca, enquanto que as operações *slow_read* e *slow_write* são para consistência forte. Conflitos de consistência gerados por operações “fast” são resolvidas em “off-line” e as aplicações devem tolerar algumas inconsistências neste caso.

Sean Liu[9] estendeu o arcabouço proposto em [7], definindo quatro formas de consistência para as aplicações. Este arcabouço possibilita acesso transacional com consistência forte, consistência insular, consistência fraca ou sem manutenção de consistência. A consistência insular garante a convergência de réplicas, mas permite leituras em réplicas que não estão na ilha de consistência, ou seja, em réplicas com estados antigos do dado. Neste arcabouço as réplicas são

organizadas de forma hierárquica em uma estrutura de multiversionamento. Assim, uma atualização é realizada sobre a versão lida pelo cliente, mesmo que o dado já esteja com um novo estado (nova versão). As requisições são pré-processadas para uma definição de qual versão das réplicas deve ser alvo das operações. Este pré-processamento aliado a manutenção de múltiplas versões dá alta escalabilidade à abordagem.

2.2 Trabalhos Relacionados

Nesta seção apresentaremos alguns sistemas diretamente relacionados com o trabalho descrito nesta dissertação. Estes trabalhos implementam técnicas de consistência alternativas à consistência forte aumentando a disponibilidade do dado replicado. Primeiramente descreveremos o CODA (seção 2.2.1), um sistema de arquivos para ambientes distribuídos que implementa uma técnica de replicação baseada no protocolo de cópia primária e depois o Bayou (seção 2.2.2), um ambiente para suporte à replicação em ambientes móveis, que utiliza o TSAE para a manutenção de consistência fraca entre as réplicas. Em seguida, descreveremos o OGS/Juggler (seção 2.2.3), onde o OGS é um serviço de grupo que foi utilizado na implementação de nosso modelo de replicação e o Juggler que é um gerenciador de grupos de réplicas baseado no OGS. Por fim, discutiremos o JGroup/ARM (seção 2.2.4), uma abordagem em Jini para grupos de objetos junto com o ARM, uma ferramenta para gerenciamento de grupo de réplicas.

2.2.1 CODA

O CODA[8] é um sistema de arquivos para ambientes de computação distribuída compostos por estações de trabalho Unix e que se utiliza da abordagem de cópia primária para manter réplicas de volumes e dar suporte a desconexões dos mesmos. O serviço de replicação do CODA é composto de dois mecanismos:

- *Serviço de Replicação*, que armazena e sincroniza as réplicas em vários servidores; e
- *Operações Desconectadas*, que possibilita que uma réplica de um volume armazenado no cliente assuma o papel de servidora do volume durante o período de desconexão.

A unidade de replicação é o volume, que é uma subárvore de diretórios gerenciada pelo sistema de arquivos. O conjunto de servidores que replicam um *volume* é chamado de VSG (*Volume Storage Group*). O AVSG (*Accessible Volume Storage Group*) é um subconjunto do VSG acessível por um cliente em um determinado instante. As alterações realizadas em uma réplica do AVSG são propagadas para todos os servidores do AVSG e posteriormente para os demais servidores do VSG. Se compararmos o estilo de replicação² do CODA com a abordagem virtual-primary-copy, o AVSG funciona como uma ilha de consistência do grupo de servidores VSG. No entanto, cada cliente pode ter um AVSG diferente para um mesmo volume.

Com as características do serviço de replicação CODA, manter as réplicas consistentes entre si tende a aumentar o tráfego de rede. Para diminuir este custo, implementou-se no CODA um RPC específico para multicast entre os volumes replicados, com suporte a paralelismo, denominado de MultiRPC, que realiza um multicast atômico (confiável e com ordenação total) para todos os AVSGs.

O CODA possui um módulo no cliente (*venus*) e outro no servidor (*vice*), através dos quais as réplicas de volumes armazenadas no cliente se sincronizam com o volume no AVSG. Em momentos de desconexão, o módulo cliente se comporta como um proxy do servidor. No momento da reconexão do host móvel (cliente) com a rede ocorre a reintegração com as demais réplicas do AVSG. A reintegração com o AVSG pode causar cancelamentos de operações realizadas no cliente desconectado. A detecção de inconsistências entre os membros de um AVSG é responsabilidade da aplicação cliente, que solicita ao grupo a reintegração das réplicas.

O CODA oferece quatro operações sobre as réplicas:

Update Atualiza o estado de uma réplica, a partir do estado de outro réplica;

Force faz com que uma réplica A alcance o mesmo estado de uma réplica B, que possui todas as operações realizadas em A em seu conjunto de operações.

Repair Conduz uma réplica inconsistente ao estado atual do grupo (AVSG);

Migrate salva réplicas não reintegradas após uma reconexão.

²Chamaremos de estilo de replicação a implementação de um protocolo de replicação.

Estas operações são feitas pelo CODA tendo como base o conjunto de vetores de timestamp das operações realizadas nas réplicas e os identificadores dos volumes replicados.

2.2.2 Bayou

O Bayou[6] é uma plataforma para aplicações colaborativas com bases de dados móveis, que implementa um serviço de replicação com uma política de consistência fraca. Como discutimos na seção 2.1.1, algoritmos de replicação de consistência forte não apresentam bom desempenho em casos de partições de rede, desconexões ou falhas de sites. Em aplicações com mobilidade, a utilização de técnicas de *callback* tradicional para invalidar caches apresentam problemas de consistência e não é possível dar uma garantia de quando uma atualização será propagada entre as réplicas. Assim, o Bayou é baseado em consistência fraca e comunicação ponto-a-ponto (*peer-to-peer*), implementando o protocolo TSAE (seção 2.1.2). Devido ao tipo de comunicação, há defasagens temporais nas interações peer-to-peer entre as réplicas, logo não há tempo pré-determinado para que uma alteração feita em uma réplica seja consolidada por todo o grupo. Consequentemente, o Bayou garante apenas que o grupo de réplicas converge para o mesmo estado, caso não haja novas mudanças de estado no conjunto de réplicas.

O Bayou possibilita leituras e escritas em qualquer réplica e oferece ainda mecanismos para a consulta do estado das operações. Uma escrita realizada em uma réplica não é propagada imediatamente para as demais, mas é classificada como uma “tentativa” de escrita. Uma escrita é definida como “confirmada” quando é efetivada de forma estável no grupo de réplicas. Com estes rótulos sobre as operações é possível saber quais operações foram efetivadas no grupo de réplicas e quais ainda estão em processo de propagação.

Na abordagem de consistência fraca implementada no Bayou, após um cliente realizar atualizações em uma determinada réplica não há garantias de que este terá novamente acesso a mesma réplica do grupo. Para garantir ao cliente um acesso consistente, o Bayou implementa a abstração de “garantias de sessão”, que simula uma sessão de aplicação com seqüenciamento das transações realizadas por um cliente. No intuito de diminuir o impacto do acesso a réplicas inconsistentes, a garantia de sessão permite ao usuário e à aplicação ter uma visão do dado consistente com relação as suas próprias ações, mantendo as vantagens da leitura e escrita em

Figura 2.2: Ambiente OGS/Juggler

qualquer réplica. A garantia de uma visão consistente de um dado impacta na disponibilidade da réplica.

2.2.3 OGS+/Juggler

O *Object Group Service* (OGS)[15] é um serviço que permite a criação de grupos de objetos em CORBA. Ele foi o primeiro sistema a implementar a abordagem de serviço para a incorporação de mecanismos de comunicação de grupo em CORBA. O serviço OGS+[17] é uma extensão do OGS e é composto pelos serviços de pertinência a grupo, multicast, consenso e de monitoração. O Juggler[17] é uma ferramenta que fornece mecanismos para o gerenciamento flexível e automático de aplicações CORBA baseadas em objetos replicados. O Juggler é baseado no OGS+ e implementa a possibilidade de chavear dinamicamente o estilo de replicação de um grupo de objetos dentre os diferentes estilos do serviço (por exemplo, replicação ativa ou primary-backup).

O ambiente OGS+/Juggler(Fig. 2.2) tem um *GroupAccessor*(GAcc) como elemento de

comunicação entre o cliente e o grupo de objetos. O *GAcc* transmite a invocação do método do objeto para o *Group Administrator* (GAdm) associado às réplicas do grupo. O Juggler disponibiliza um *JugglerClient* (JC), aplicação cliente que faz requisições ao *Replication Manager* (RM), parte do *JugglerService* responsável por gerenciar os grupos com inclusão e remoção de membros, além de possibilitar a mudança dinâmica no estilo de replicação. O ambiente OGS+/Juggler é descrito com mais detalhes no capítulo 3.

2.2.4 JGroup/ARM

JGroup é uma plataforma para objetos distribuídos, integrante da tecnologia Jini, baseada em grupos de objetos que estende o Java RMI com o paradigma de comunicação de grupo e dá suporte ao particionamento de ambientes distribuídos. O ARM (*Autonomous Replication Management*) é uma ferramenta construída sobre o JGroup que provê um esquema de replicação distribuída extensível. JGroup/ARM[20] provêm juntos uma plataforma flexível de gerenciamento de aplicação distribuídas baseadas em replicação.

Muitos sistemas de grupos de objetos, inclusive o JGroup, não incluem mecanismos de controle de réplicas por hosts ou para recuperação de falhas. Em conjunto com o JGroup, o ARM provê uma interface na qual um cliente de gerenciamento pode instalar e remover grupos de réplicas, além de coordenar recuperação de réplicas de acordo com a política escolhida. O JGroup oferece suporte a particionamento de rede, mantendo visões consistentes do grupo para cada partição. Para isto, a aplicação cliente define o comportamento das réplicas do grupo em cada partição e como será a reintegração dos membros posteriormente.

O ambiente JGroup/ARM (Fig. 2.3) tem uma arquitetura similar ao OGS+/Juggler, o elemento de comunicação entre o cliente e o grupo de réplicas é o *group proxy* (GP), que transmite a invocação do método sobre o objeto replicado para um *Group Manager* (GM) associado a cada uma das réplicas do grupo. Através de um *Management Client* (MC) com interface gráfica é possível manipular os grupos gerenciados. O ARM se utiliza do JGroup para criar/gerenciar grupos, permitindo a inclusão e remoção de membros ao grupo e definição de estilo de replicação. Diferente do OGS+/Juggler, os estilos de replicação previstos no JGroup/ARM são apenas dois: replicação ativa ou passiva, onde a replicação passiva se assemelha ao estilo *COLD-PASSIVE* do

Figura 2.3: Ambiente JGroup/ARM

OGS+/Juggler. Após a escolha, um grupo não pode alterar seu estilo de replicação.

Como o Juggler, o ARM também possui um *Replication Manager*(RM) responsável pela criação e interação com as réplicas. Cada réplica é executada em uma *Máquina Virtual Java*(JVM) diferente, garantindo que problemas na execução de um objeto replicado não interfira na execução dos demais objetos. O próprio RM é replicado para garantir tolerância a falhas. O tipo de replicação adotado no ARM é a *primary-backup*, todas as operações são executadas sobre uma única réplica primária e as secundárias são mantidas sincronizadas com a primária.

2.2.5 Comparação e Avaliação

A maioria dos trabalhos estudados apresentam uma implementação de um mecanismo de replicação e têm como base uma única política de replicação. Dentre estes, apenas o OGS/Juggler implementa mais de um estilo de replicação (*ative*, *cold passive* e *virtual primary copy*). O CODA implementa *primary-copy* para manutenção de consistência entre réplicas. O Bayou implementa o TSAE e o Jgroup/ARM implementa *primary-copy*, mesmo tendo definido interface também para

replicação passiva.

Tratando-se de comunicação de grupo, todos dão suporte a comunicação assíncrona. Tanto o OGS quanto o JGroup implementam o conceito de grupo de réplicas de objetos e foram concebidos visando a possibilidade de se implementar vários estilos de replicação. O JGroup tem em sua implementação apenas o estilo de replicação primary-copy, já o OGS+ tem implementado outros modelos de replicação mais flexíveis quanto a manutenção de consistência. No entanto, uma das características do JGroup é o funcionamento em partições de rede e a idéia de subgrupo de réplicas, que não são previstos no OGS.

O Bayou é o único dos sistemas descritos que tem reintegração de membros do grupo continuamente realizada pelo protocolo. Os demais sistemas têm protocolos específicos para momentos de reintegração. No CODA, a inconsistência é detectada pela aplicação cliente e portanto réplicas podem ficar inconsistentes enquanto não houver acesso ao volume replicado. O JGroup/ARM e o OGS+/Juggler propagam o estado da réplica primária para a réplica que está se reintegrando.

Tabela de Comparação

Na tabela abaixo, segue um comparativo das funcionalidades implementadas em cada sistema com relação a replicação.

Funcionalidade	CODA	Bayou	OGS/Juggler	JGroup/ARM
Tipo de Consistência Implementados	Forte	Fraca	Forte/ Relaxada	Forte
Unidade de Replicação	Arquivo/ Diretório	Arquivo	Objeto	Objeto
Troca dinâmica de Estilo de Replicação	Não	Não	Sim	Não
Estilo de Replicação Parametrizável	Não	Não	Sim	Sim
Trata Particionamento/ Desconexão	Sim	Sim	Não	Sim

Capítulo 3

OGS e Juggler

Este capítulo descreve a arquitetura do OGS[15] e as modificações/extensões que caracterizam o OGS+. Além disto, apresentamos a ferramenta de gerenciamento de réplicas baseada no OGS+, chamada Juggler[17]. Estas duas ferramentas serviram de base para o nosso trabalho.

3.1 OGS

O *Object Group Service*(OGS) [15] é um serviço que permite a criação de grupos de objetos em CORBA. Ele foi o primeiro sistema a implementar a abordagem de serviço para a incorporação de mecanismos de comunicação de grupo em CORBA. Uma característica muito importante desse serviço é o fato dele não modificar o ORB a fim de fornecer mecanismos de comunicação de grupo. Assim, o OGS pode ser utilizado de forma interoperável em diferentes ORBs compatíveis com a especificação CORBA. A arquitetura do OGS, ilustrada na Figura 3.1, é modular e flexível, permitindo a extensão e a reutilização de seus componentes individualmente. O OGS especifica uma arquitetura de serviços e um conjunto de interfaces especificadas em OMG IDL. Cada um dos serviços do OGS é especificado independentemente, garantindo modularidade e transparência, em se mantendo as mesmas interfaces IDL.

O ambiente OGS é composto pelos seguintes serviços:

- *Serviço de Pertinência a Grupo*: Permite que os membros de um grupo de objetos mantenham uma visão consistente da composição do grupo;

- *Serviço de Multicast*: Provê a entrega de mensagens a todos os membros do grupo com diferentes semânticas e graus de confiabilidade;
- *Serviço de Consenso*: Implementa um protocolo para a obtenção de consenso distribuído, que é utilizado por outros componentes do OGS, como por exemplo pelo serviço de multicast para garantir a entrega de mensagens em ordem total;
- *Serviço de Monitoração*: Provê mecanismos para a detecção de falhas do tipo *fail-stop* em membros de um grupo.

Figura 3.1: Serviços do OGS.

Vamos descrever em detalhes apenas os dois serviços que tratam diretamente de grupos no OGS: o Serviço de Pertinência a Grupo e o Serviço de Multicast.

Pertinência a grupo O serviço de pertinência a grupo gerencia o ciclo de vida do grupo de objetos, isto é, a criação de um grupo, a inclusão e a remoção de um membro, etc. Para permitir grupos dinâmicos, o OGS implementa dois tipos de protocolos: o *Protocolo de mudança de visão*, pelo qual todos os membros são notificados da nova configuração do grupo; e o *Protocolo de transferência de estado*, que é uma operação atômica no qual um membro do grupo é atualizado com o estado dos demais membros sincronizados do grupo.

O Serviço de Pertinência a Grupo é composto por dois tipos de objetos: *GroupAdministrator*, que administra a entrada e saída de membros no grupo, e *Groupable*, que implementa protocolos de transferência de estado e mudança de visão entre os membros. Dentre as interfaces IDL do serviço de grupos temos a estrutura *GroupView* responsável por armazenar a visão de um grupo em um determinado momento. A visão é representada por um número de versão, que é incrementado toda vez que a configuração do grupo muda e é usada para detectar mudanças na formação do grupo de réplicas. Como em sua versão original o OGS implementa somente replicação onde todos os membros têm a mesma visão do grupo, a manutenção de consistência no serviço de pertinência a grupo é mantida por um protocolo de consistência forte.

Multicast para grupos O serviço de multicast provê primitivas para o envio de requisições para os membros de um grupo de objetos. As primitivas diferem quanto ao grau de confiabilidade e a ordem de execução (das operações nas réplicas). Além disto, o serviço possui modelos de comunicação tipada e não-tipada.

As primitivas de multicast podem ser confiáveis ou não-confiáveis. A entrega confiável de uma mensagem por si só não garante a consistência do grupo, pois são necessárias também garantias de ordenação na seqüência de entrega de operações às réplicas. Quanto à ordenação, o OGS provê entrega de mensagens sem ordenação, em FIFO ou com ordenação total. As primitivas de multicast não-confiável são utilizadas de forma restrita, como em operações somente de leitura.

Em se tratando do modelo de comunicação, o OGS implementa dois tipos: a tipada e a não-tipada. As *Interfaces não-tipadas* (ou requisições explícitas) possibilitam aos clientes enviar qualquer tipo de mensagem ao grupo, podendo ser realizadas em quaisquer grupos de objetos, independente das interfaces implementadas pelos seus membros. As *Interfaces tipadas* (ou requisições transparentes) permitem que um cliente invoque diretamente as operações definidas na interface dos membros de um grupo. Portanto, este tipo de invocação só pode ser realizada em grupos de objetos que implementam a mesma interface IDL.

Utilizando a versão tipada pode-se especificar a semântica para cada tipo de operação. Com

isto, pode-se por exemplo definir operações comutativas sobre o grupo, operações estas cujo resultado é independente da ordem na qual são executadas.

O serviço de multicast é implementado através de dois tipos de objetos: o *GroupAccessor* que define como os objetos clientes podem invocar multicast através do OGS, e o *Invocable*, que é usado pelo OGS para entregar multicasts para objetos da aplicação.

O OGS implementa o conceito de *encapsulamento de pluralidade* através do uso de proxy para o grupo, denominado de *group reference*, assim o grupo de réplicas é referenciado e se comporta como um único objeto, para a aplicação cliente. Com isso, uma *group reference* oferece transparência de replicação para o cliente no acesso ao grupo. O OGS, implementa dois níveis de transparência:

- *Transparência para o cliente do grupo*: o cliente envia uma requisição ao grupo e recebe apenas uma confirmação. Com isto, um objeto pode ser substituído por um grupo sem mudanças na aplicação cliente.
- *Transparência para os membros do grupo*: Os membros são apenas avisados de que pertencem a um grupo, mas recebem requisições para a execução de métodos como se fossem um único objeto.

A transparência oferecida pelo OGS torna possível a troca de política de replicação ou a configuração do grupo sem que o objeto cliente da aplicação tenha percepção desta mudança. A política de replicação inicialmente implementada no OGS é de replicação ativa, onde todos os membros do grupo são mantidos sincronizados.

3.2 Juggler

O Juggler [17] é um serviço que fornece mecanismos para o gerenciamento flexível e automático de aplicações CORBA baseadas na redundância de objetos que as compõem (grupos de objetos). Com o Juggler é possível definir o estilo de replicação, a localização e a quantidade mínima de réplicas que compõem o grupo de objetos. Além disto, o Juggler possibilita a modificação dinâmica do estilo de replicação e de propriedades do grupo de objetos.

O Juggler tem a implementação de sua arquitetura para comunicação de grupo baseada no Serviço OGS(seção 3.1), além de utilizar o serviço de monitoramento de objetos, fornecido pelo OGS, para obter notificações sobre eventuais falhas nos objetos.

Os grupos de objetos criados através do Juggler são gerenciados por uma única entidade lógica, composta pelos componentes *GroupManager*, *GroupObserver* e *GroupActor*. O componente *GroupManager* lida basicamente com a criação, destruição e modificação de propriedades de grupos de objetos. Para isto, o *GroupManager* invoca operações no *GroupActor*, o qual é responsável pela criação e destruição de réplicas de objetos. O componente *GroupObserver*, por sua vez, é responsável pela monitoração dos membros dos grupos de objetos.

O Juggler é especificado por uma coleção de interfaces IDL, dentre estas, a interface *GroupManagementOperations* é a principal interface herdada pelo *GroupManager*, fornecendo operações essenciais para um cliente do Juggler, como a criação de grupo (*create_group*), inclusão de membro (*add_members*) e mudança de estilo de replicação (*set_replication_style*).

A fim de garantir tolerância a falhas no serviço Juggler, os objetos que implementam as interfaces *GroupManager*, *GroupObserver* e *GroupActor* são grupos de objetos replicados no OGS, que adotam o estilo de replicação *ACTIVE*¹.

3.3 OGS+

Como parte do trabalho de desenvolvimento do Juggler, foi realizada também uma extensão do OGS para a disponibilização de diversos estilos de replicação, denominada *OGS+*. O *OGS+* implementa algumas modificações nos módulos IDL *mGroupAccess*, *mGroupAdmin* e *mGroupProtocol*. Estas modificações serão descritas ao longo desta seção.

No módulo *mGroupAdmin* foi especificada a estrutura *ReplicationStyle*, que permite a escolha entre os estilos de replicação *STATELESS*, *COLD-PASSIVE*, *WARM-PASSIVE*, *ACTIVE* e *VIRTUAL-PRIMARY-COPY*, como ilustrado na Figura 3.2. O estilo de replicação *ACTIVE* já estava implementado no OGS original, os demais estilos de replicação foram implementados no *OGS+* e são:

¹nome utilizado no Juggler para definir replicação ativa.


```

module mGroupAdmin {

    typedef unsigned short ReplicationStyleValue;
    const ReplicationStyleValue STATELESS = 0;
    const ReplicationStyleValue COLD_PASSIVE = 1;
    const ReplicationStyleValue WARM_PASSIVE = 2;
    const ReplicationStyleValue ACTIVE = 3;
    const ReplicationStyleValue ACTIVE_WITH_VOTING = 4;
    const ReplicationStyleValue VIRTUAL_PRIMARY_COPY = 5;

    struct ReplicationProperties {
        PrimaryReplicasValue primaries_;
        CheckpointIntervalValue checkpoint_interval_;
        UpdatesBetweenCheckpointValue updates_between_checkpoint_;
    };

    struct ReplicationStyle {
        ReplicationStyleValue style_;
        ReplicationProperties properties_;
    };

    interface ExtendedGroupAdministrator : GroupAdministrator {
        void set_replication_style(in ReplicationStyle replication_style)
            raises(BadReplicationStyle);
        ReplicationStyle get_replication_style();
    };

    interface GroupAdministratorFactory {
        ExtendedGroupAdministrator create(in string group_name,
                                           in ReplicationStyle repl_style)
            raises(mGroupAccess::GroupError,
                  mGroupAccess::InvalidGroupName);
        void release(in ExtendedGroupAdministrator adm)
            raises(mGroupAccess::GroupError);
    };
};

```

Figura 3.2: Módulo mGroupAdmin do OGS+.

STATELESS Neste estilo de replicação o OGS+ não executa o Protocolo de transferência de estado (vide seção 3.1), com isto os membros do grupo não possuem qualquer manutenção de consistências entre si. Este estilo de replicação é comumente empregado na disponibilização de serviços somente de leitura.

WARM_PASSIVE Estilo de replicação onde há uma única réplica primária que recebe e responde às requisições dos clientes, e que se encarrega de difundir a mensagem para as réplicas secundárias do grupo (*state update*).

COLD_PASSIVE Semelhante ao estilo de replicação *WARM_PASSIVE*, com a diferença de propagar o estado da réplica primária periodicamente para as réplicas secundárias.

VIRTUAL_PRIMARY_COPY No estilo de replicação *VIRTUAL_PRIMARY_COPY* [2], uma ou mais réplicas são designadas primárias e formam uma *ilha de consistência*. Dentro da ilha é utilizado o estilo de replicação *ACTIVE*, assim o conjunto das réplicas primárias é visto externamente como uma única réplica primária virtual. As réplicas que não estão na ilha fazem o papel de réplicas secundárias da réplica primária virtual. Já a interação entre a ilha e as réplicas secundárias é idêntica a utilizada no estilo de replicação *WARM_PASSIVE*.

Ainda foram incorporadas no *mGroupAdmin* duas novas operações: *set_replication_style*, que permite a modificação dinâmica do estilo de replicação de um grupo e *get_replication_style*, que retorna o estilo de replicação associado a um grupo de objetos. Além disto a interface *GroupAdministratorFactory* foi alterada para que o método *create* receba o estilo de replicação a ser utilizado pelo grupo.

```
module mGroupAccess {  
  
    enum Ordering {  
        UNRELIABLE,  
        PESSIMISTIC_RELIABLE,  
        RELIABLE,  
        ATOMIC,  
        NON_CRITICAL,  
        REPLICATION_STYLE_CHANGE  
    };  
  
    struct GroupView {  
        sequence<Invocable> composition_  
        unsigned long version_  
        unsigned short replication_style_  
        sequence<Invocable> primaries_  
    };  
  
};
```

Figura 3.3: Módulo *mGroupAccess* do OGS+.

No módulo *mGroupAccess* (Figura 3.3) foi criado o atributo *primaries_* que é um subconjunto do atributo *composition_* definido originalmente no OGS, pois este armazena as IORs para todos os membros de um grupo, sejam elas réplicas primárias ou não. Assim, com as informações contidas na estrutura *GroupView* é possível acessar uma determinada réplica primária. Este atributo não é utilizado em estilos de replicação como *ACTIVE* e *STATELESS*, mas somente nos estilos *COLD_PASSIVE*, *WARM_PASSIVE* e *VIRTUAL_PRIMARY_COPY* onde é usado para o envio de mensagens apenas as réplicas primárias do grupo de réplicas do objeto.

O módulo *mGroupProtocol* (Figura 3.4) especifica estruturas de comunicação entre os componentes *GroupAdministrator*, componentes estes que armazenam a visão do grupo no qual a réplica esta inserida. Neste módulo foi modificada a enumeração *Ordering*, que foi estendida para incorporar três novos tipos de mensagens internas ao serviço: *REPLICATION_STYLE_CHANGE*, *CHECKPOINT* e *STATE_UPDATE*, operações estas responsáveis por informar a mudança dinâmica de estilo de replicação, informar um estado comum as réplicas do grupo e por atualizar o estado de uma réplica, respectivamente.

```

module mGroupProtocol {

    enum Ordering {
        UNRELIABLE,
        PESSIMISTIC_RELIABLE,
        RELIABLE,
        ATOMIC,
        VIEW_CHANGE, // Private protocol message
        REPLICATION_STYLE_CHANGE, // Private protocol message
        CHECKPOINT, // Private protocol message
        STATE_UPDATE // Private protocol message
    };

    struct ServiceView {
        GroupManagerSeq gms_;
        mConsensus::ConsensusManagerSeq cms_;
        unsigned long version_;
        mGroupAdmin::ReplicationStyleValue replication_style_;
        GroupManagerSeq primaries_gms_;
        mConsensus::ConsensusManagerSeq primaries_cms_;
    };

};

```

Figura 3.4: Módulo *mGroupProtocol* do OGS+.

Uma requisição do tipo *REPLICATION_STYLE_CHANGE* é um tipo de mensagem que sempre é processada por todos os membros do grupo e de forma atômica, independente do estilo de replicação associado ao grupo de objetos. As mensagens *CHECKPOINT* e *STATE_UPDATE* são enviadas pela réplica primária para as secundárias de um grupo com estilo de replicação passiva. Em uma mensagem do tipo *CHECKPOINT* o estado completo da(s) réplica(s) primária(s) é enviado para as secundárias, enquanto que numa mensagem *STATE_UPDATE* apenas a última atualização de estado da réplica primária é transmitida.

Capítulo 4

O OGS/R

Neste capítulo vamos descrever o serviço de grupo OGS/R. O OGS/R é um serviço de grupos de objetos que implementa manutenção de consistência relaxada (seção 2.1.3), oferece recursos para criação e gerenciamento de grupos de objetos CORBA para aplicações que demandam alta disponibilidade de réplicas e permite um grau de consistência seletivo entre as réplicas.

O OGS/R é uma extensão do serviço OGS+ (vide seção 3.3), que implementa todos os estilos de replicação do OGS+, além de implementar um novo estilo replicação com consistência relaxada. Este serviço de grupos possui interfaces internas e externas, onde as interfaces externas permitem ao usuário do serviço a escolha do estilo de manutenção de consistência entre as réplicas e as interfaces internas definem a comunicação entre os membros do grupo.

A principal motivação para acrescentar consistência relaxada ao serviço OGS foi experimentar e avaliar os ganhos no desempenho de uma aplicação com um estilo de replicação que concilia as vantagens de alta disponibilidade das abordagens de consistência fraca (seção 2.1.2) com as garantias de consistência entre as réplicas das abordagens de consistência forte (seção 2.1.1).

A definição de grupo de réplicas encontrada em [23] e [15] descreve o grupo como: “uma coleção de entidades distribuídas (cópias do mesmo dado), onde cada membro do grupo se comunica com os outros através de multicast para todo o grupo, sendo este multicast confiável e de ordenação total”. Esta definição de grupo prioriza características de consistência e confiabilidade de transações no grupo em detrimento de eficiência de acesso e escalabilidade. Aplicações que utilizam grupos com estas características e com grande número de membros no grupo tendem

a apresentar baixo desempenho e uma alta sobrecarga de mensagens em cada comunicação de grupo. O OGS/R oferece uma solução híbrida, provendo um serviço de grupo com consistência relaxada, que flexibiliza o conceito de grupo de réplicas permitindo que aplicações possam realizar operações de consistência confiáveis ou não através da escolha da forma de difusão da mensagem no grupo. Além disto, permite tornar novamente consistente com as demais réplicas de um grupo as réplicas que realizaram operações inconsistentes.

A abordagem de consistência relaxada implementada no OGS/R deriva do proposto em *virtual-primary-copy*[2](ver seção 2.1.3), que serviu como base por implementar o conceito de (sub)grupos de réplicas consistentes e inconsistentes e por possuir uma implementação no OGS+. O estilo de replicação que implementamos, chamado de *RELAXED*, difere do *virtual-primary-copy* no que diz respeito ao tratamento de réplicas secundárias.

No estilo de replicação *RELAXED*, há a *ilha de consistência* composta por um subgrupo de réplicas mantidas com consistência forte e há as demais réplicas de fora da ilha que podem divergir, mas que eventualmente podem vir a ter o mesmo estado das réplicas da ilha. Entre as réplicas da ilha de consistência é utilizado o estilo de replicação *ACTIVE*, assim o conjunto das réplicas consistentes se comporta como uma única réplica primária virtual. As réplicas que não estão na ilha desempenham o papel de réplicas secundárias desta réplica primária virtual. No estilo de replicação *VIRTUAL-PRIMARY-COPY*(seção 3.3), a interação entre a ilha e as réplicas secundárias é a mesma utilizada no estilo de replicação *WARM-PASSIVE*, ou seja, todas as réplicas recebem a mensagem, mas apenas as réplicas da ilha executam a operação. No estilo de replicação *RELAXED*, as réplicas secundárias têm comportamento independente, semelhante ao estilo de replicação *STATELESS*, podendo ser acessadas e modificadas isoladamente por um objeto cliente e sem a necessidade de um multicast para todo o grupo.

Em nossa abordagem de consistência relaxada, a convergência de uma réplica secundária com o grupo ao qual pertence precisa ser requisitada explicitamente através da operação de reintegração. Esta operação executa um algoritmo de reintegração da réplica visando identificar um subconjunto de operações realizadas na réplica secundária e que pode ser efetuada na ilha a fim de produzir um novo estado consistente.

O OGS transporta as operações que serão executadas nos membros de um grupo sem

verificação de conteúdo, não distinguindo entre uma difusão de consulta ou atualização no grupo. Esta característica dá flexibilidade à comunicação, mas não atende algumas aplicações que precisam de um tratamento diferenciado de suas operações. O OGS/R permite à aplicação cliente categorizar suas diferentes operações sobre o grupo, oferecendo uma utilização personalizada do serviço de grupos de objetos.

O OGS/R oferece três categorias de operações sobre um grupo de réplicas, *operações estáveis*, *operações instáveis* e *integração & visão*.

1. *Operações estáveis* são operações que sempre são executadas na ilha, não são descartadas e produzem um novo estado consistente no grupo. Se uma réplica de fora da ilha recebe uma operação destas, redireciona para ser executada na ilha. Esse conjunto de operações abrange aquelas que só podem ser executadas nos moldes de transações ACID.
2. *Operações instáveis* são as operações que são executadas somente nas cópias locais, alterações em uma réplica não são propagadas imediatamente, passarão por um processo de reintegração com a ilha e poderão ser incorporadas ou não no conjunto de operações sobre o dado replicado. Uma operação sobre uma réplica na ilha implica no bloqueio de todas as réplicas da ilha, mas em uma réplica de fora da ilha apenas esta é bloqueada no processo.
3. *Integração & Visão*, operações deste tipo são realizadas em uma réplica secundária para que esta sincronize seu estado com o da ilha de consistência, e em caso da réplica secundária ter realizado operações inconsistentes é feita uma serialização das operações da réplica na ilha, descartando-se operações locais conflitantes.

A seguir, vamos descrever estas operações com mais detalhes.

4.1 Operações estáveis

Operações estáveis são operações que quando submetidas ao grupo de réplicas são executadas sobre a ilha de consistência. Uma operação estável atribui um novo estado consistente ao grupo de réplicas.

A operação estável está implementada no OGS/R utilizando o multicast atômico do OGS+. Ao contrário do multicast original, o multicast implementado no OGS/R é feito somente para os membros da ilha, reduzindo assim o custo da comunicação no grupo de réplicas.

Devido às características do protocolo de manutenção de consistência na ilha, as operações estáveis de uma aplicação deverão ser aquelas que demandam um alto grau de consistência e cujo resultado deve estar consolidado no objeto replicado.

4.2 Operações instáveis

Operações instáveis são operações locais executadas somente em uma réplica secundária, sendo que as atualizações feitas nesta réplica não são propagadas para a ilha. Dado que no mesmo instante as réplicas da ilha podem estar sofrendo uma outra operação (estável), o resultado da operação instável em uma réplica secundária pode eventualmente ser perdido no momento de uma reintegração desta réplica com a ilha. Como as operações instáveis são dirigidas a uma única réplica local não integrante da ilha, evita-se o alto custo do consenso entre as réplicas do grupo e garante-se um acesso imediato à réplica. Operações como consultas, *Blind writes*, ou atualizações intermediárias que deviam ser visíveis para outros “clientes da réplica” podem ser submetidas ao grupo como operações instáveis, desde que para a aplicação o resultado da execução na réplica secundária não necessite ser consolidado na ilha.

4.3 Integração & Visão

A operação de *Integração & Visão (I&V)* é uma requisição explícita de reintegração de todas as atualizações ocorridas em uma réplica secundária com o estado da ilha de consistência, produzindo um novo estado consistente entre a ilha e a réplica secundária.

Em nossa abordagem, usamos o conceito de *prefixos de operações* como sendo o conjunto de operações realizadas antes que o estado da réplica tenha divergido do estado da ilha. E um *prefixo comum maximal* identifica um último estado comum entre uma réplica e a ilha de consistência. Portanto, a operação de I&V, ao produzir um novo estado consistente entre a

réplica secundária e a ilha, determina um novo prefixo comum maximal.

O algoritmo responsável pela reintegração utiliza um número de sequência (identificador da operação) para comparar as operações na ilha e nas réplicas secundárias. Após cada reintegração, a ilha e a réplica assumem o identificador da última operação conjunta realizada.

Vamos explicar o algoritmo de reintegração a partir de um exemplo. Supondo que uma réplica secundária R e as réplicas primárias da ilha de consistência estejam em um estado comum S_1 (que corresponde à execução do prefixo comum maximal), onde 1 representa o identificador da última operação comum. Sejam as seguintes operações executadas independentemente em R e nas réplicas da ilha:

- Op_2 é executada na ilha de consistência, gerando o estado S_2 ;
- Op'_2 e Op'_3 são executadas em R , gerando o estado (divergente) S'_3 .

Agora, suponha que é solicitada uma operação de I&V entre R e a ilha de consistência. O algoritmo obtém inicialmente a sequência de todas as operações realizadas na ilha desde o último estado comum. No caso, somente a operação Op_2 .

Estas operações realizadas na ilha são então submetidas à réplica secundária para execução. Todas as operações com identificador maior do que o identificador da última operação do prefixo comum maximal são executadas na réplica secundária. Neste exemplo, a operação Op_2 da ilha seria executada na réplica secundária, devido ao prefixo comum maximal possuir identificador 1.

Seja n o identificador da última operação estável, as operações instáveis com identificadores maiores do que n são então submetidas para execução na ilha, uma a uma, como se fossem operações estáveis. Em nosso exemplo, a operação instável Op'_3 seria executada na ilha. Após esta operação, portanto, a réplica secundária estará sincronizada com a ilha de consistência e o novo prefixo comum maximal será 3. Quando a réplica não possui operações com identificador maior do que n é automaticamente sincronizada com o estado da ilha, não havendo a necessidade de submeter operações à ilha.

A reintegração baseada em prefixos nos garante que uma sequência de operações RU contribui para o novo estado da ilha, caso não tenham ocorrido atualizações na ilha desde a última reintegração da réplica secundária. Se tiverem ocorrido atualizações na ilha, somente uma

parcela “final” da seqüência RU contribuirá para o novo estado da ilha. Portanto, o conjunto de operações recebidas pela ilha na I&V corresponde às operações instáveis realizadas na réplica após algum estado consistente da ilha.

A reintegração de uma réplica inconsistente com a ilha ocorre como descrito a seguir. Para a descrição utilizou-se a notação Op_t para indicar a operação Op realizada com identificador t . Considerando uma réplica secundária solicitando reintegração, onde $(U_1, U_2, \dots, U_m)$ é a seqüência de operações locais instáveis, e considerando LU (índice da operação mais recente) o prefixo comum maximal da réplica secundária com a ilha, a reintegração ocorre de acordo com o seguinte algoritmo:

1. Gerar a seqüência ordenada de operações estáveis $(Y_i, \dots, Y_n)$, onde $i = LU + 1$ e n é o identificador da última operação estável realizada.
2. Se $m > n$, gerar seqüência ordenada de operações instáveis a serem executadas na ilha $(U_k, \dots, U_m)$, onde $k = n + 1$.
3. Executar as seqüências dos itens 1 e 2 na réplica secundária.
4. Definir o novo LU' na réplica secundária como sendo $\max(n, m)$.
5. Executar sequencialmente cada operação da seqüência $(U_k, \dots, U_m)$, como operação estável na ilha de consistência do grupo, fazendo com que o novo estado S' da ilha satisfaça $\{S\}(U_k, \dots, U_m)\{S'\}$.

Como se nota, a operação de *Integração&Visão* (I&V) requer comunicação adicional entre a réplica secundária e a ilha de consistência, mas seu custo adicional sobre o desempenho do serviço pode ser compensado se houver uma grande quantidade de operações instáveis (de baixo custo).

4.4 Especificação do Serviço

O OGS/R incorpora novas funcionalidades ao OGS+, implementando mudanças e criando novas interfaces nos três principais módulos do serviço: *mGroupProtocol*, *mGroupAdmin* e

mGroupAccess. A maioria das extensões para incorporar estas novas funcionalidades ao OGS/R são internas ao serviço. As interfaces IDL do OGS/R atendem às características de diferentes níveis de consistência no grupo e permitem um maior controle sobre a consistência dos membros do grupo além de flexibilizar a comunicação entre a aplicação e o serviço OGS/R.

```
module mGroupAdmin {

    typedef unsigned short ReplicationStyleValue;
    const ReplicationStyleValue STATELESS = 0;
    const ReplicationStyleValue COLD_PASSIVE = 1;
    const ReplicationStyleValue WARM_PASSIVE = 2;
    const ReplicationStyleValue ACTIVE = 3;
    const ReplicationStyleValue ACTIVE_WITH_VOTING = 4;
    const ReplicationStyleValue VIRTUAL_PRIMARY_COPY = 5;
    const ReplicationStyleValue RELAXED = 6;

    interface ExtendedGroupAdministrator : GroupAdministrator {
        void set_replication_style(in ReplicationStyle replication_style)
            raises(BadReplicationStyle);
        ReplicationStyle get_replication_style();
    };

    interface GroupAdministratorFactory {
        ExtendedGroupAdministrator create(in string group_name,
                                           in ReplicationStyle repl_style)
            raises(mGroupAccess::GroupError,
                  mGroupAccess::InvalidGroupName);
        void release(in ExtendedGroupAdministrator adm)
            raises(mGroupAccess::GroupError);
    };
};
```

Figura 4.1: Extensão do módulo *mGroupAdmin*.

No módulo *mGroupAdmin* as mudanças foram realizadas na estrutura *ReplicationStyleValue*. Esta estrutura é responsável por armazenar o estilo de replicação escolhido para o grupo e foi alterada para comportar também o novo estilo de replicação *RELAXED* (Figura 4.1).

No Módulo *mGroupAccess* foi especificada a interface *OGSRGroupAccessor* (Figura 4.2), que descreve os métodos de um *GroupAccessor* para um grupo com consistência relaxada. Um *OGSRGroupAccessor* implementa dois novos métodos: *tentative*, responsável por requisições de operações instáveis e o *reintegrate*, que solicita a reintegração da réplica secundária com a ilha de consistência do grupo. Seguindo os padrões de interface do OGS+, foi especificada uma *OGSRGroupAccessorFactory* que define métodos para criação e destruição de objetos do grupo.

No módulo *mGroupProtocol*, as modificações se concentraram na interface *GroupManager*. Nesta interface foram criados os métodos *get_primary*, *deliver_tentative*, *reintegrate* e *up-*

```

module mGroupAccess {

    interface OGSRGroupAccessor : GroupAccessor {
        AnySeq setup_multicast(in any msg,
                               in NumReplies replies,
                               in Ordering order)

        raises(GroupError);
        AnySeq tentative(in any msg)
        raises(GroupError);
        void reintegrate()
        raises(GroupError);
    };

    interface OGSRGroupAccessorFactory {
        OGSRGroupAccessor create(in string group_name)
        raises(GroupError,
               NoGroup,
               InvalidGroupName);
        void release(in OGSRGroupAccessor acc)
        raises(GroupError);
    };
};

```

Figura 4.2: Extensão do módulo mGroupAccess.

date_backup (Figura 4.3). O método *get_primary* é utilizado pelo *OGSRGroupAccessor* na identificação da ilha de consistência e réplicas secundárias do grupo. A entrega da operação instável para o *GroupAdministrator* da réplica secundária é feita pelo método *deliver_tentative*. A operação de *Integração & Visão* é composta pelos métodos *reintegrate* (sobre a réplica primária) e *update_backup* (sobre a réplica secundária). Para a transferência da sequência de operações entre as réplicas foi especificada uma nova estrutura, chamada *ReintegrateMsg*. Esta estrutura é usada para armazenar a sequência de operações e identificadores e é utilizada como parâmetro e objeto devolvido pelo protocolo de reintegração (métodos *reintegrate* e *update_backup*). A *ReintegrateMsg* desempenha o papel de levar operações estáveis de uma réplica primária para uma réplica secundária e entregar as operações instáveis da réplica secundária ao *OGSRGroupAccessor* que as executará nas réplicas da ilha de consistência.

A fim de ser coerente com as extensões incorporadas no OGS/R, o serviço de gerenciamento de réplicas *Juggler* sofreu uma adaptação para possibilitar o uso do novo estilo de replicação implementado no OGS/R. Acrescentou-se à interface *JugglerClient* a opção de utilizar o estilo *RELAXED* do OGS/R, integrando assim as funcionalidades de gerenciamento de grupos do *Juggler* com o implementado no serviço OGS/R. É através do *Juggler* que se controla o número de réplicas

```

module mGroupProtocol {

    struct ReintegrateMsg {
        any state_;
        long cid_;
        MessageIdSeq deliver_;
        StateUpdateSeq updates_;
    };

    interface OGSRGroupAccessManager :
        GroupAccessManager,
        mGroupAccess::OGSRGroupAccessor {
    };

    interface GroupManager :
        mGroupAdmin::ExtendedGroupAdministrator,
        GroupAccessManager,
        mMonitoring::Monitorable,
        mMonitoring::Notifiable,
        mConsensus::ConsensusParticipant {
        boolean get_primary();
        oneway
        void deliver_message(in Message msg);
        oneway
        void deliver_setup_message(in Message msg);
        void deliver_tentative(in Message msg);
        void reintegrate()
            raises (mGroupAccess::GroupError,
                mGroupAdmin::NotMember,
                mGroupAdmin::AlreadyMember);
        ReintegrateMsg reintegrate()
            raises (mGroupAccess::GroupError,
                mGroupAdmin::NotMember,
                mGroupAdmin::AlreadyMember);
        ReintegrateMsg update_backup(in ReintegrateMsg state)
            raises (mGroupAccess::GroupError,
                mGroupAdmin::NotMember,
                mGroupAdmin::AlreadyMember);
    };
};

```

Figura 4.3: Extensão do módulo mGroupProtocol.

que participam de uma ilha de consistência, ou que se define o número mínimo de membros da ilha.

4.5 Implementação

No desenvolvimento do OGS/R foram utilizados o ORB Visibroker 4.0 e a linguagem Java (SDK 1.2.2), as mesmas tecnologias utilizada no desenvolvimento do OGS+. Nas seções a seguir vamos descrever como interagem os objetos implementados no OGS/R durante a execução

de cada uma das três operações especificadas para o serviço.

4.5.1 Operações estáveis

A operação estável (Figura 4.4) é um multicast para as réplicas pertencentes à ilha de consistência e que pode ser invocada por qualquer objeto que implemente uma interface *GroupAccessor*, *OGSRGroupAccessor* ou *ExtendedGroupAccessor*. Caso a operação estável seja o primeiro multicast para o grupo é executado o método *setup_multicast* (Figura 4.2) que é um multicast que entrega a mensagem aos participantes do grupo e através de um consenso sincroniza o grupo de réplicas. Sua principal função é a de informar cada *GroupAdministrator* dos membros do grupo que participarão dos próximos consensos (os membros da ilha de consistência) sobre quais são as réplicas da ilha de consistência e secundárias do grupo. Caso a operação estável não seja a primeira operação sobre o grupo, a mesma é realizada através do método *multicast*.

Quando um cliente solicita uma operação estável, o *OGSRGroupAccessor* (OGSRAcc na Figura 4.4) encaminha a operação para cada *GroupAdministrator* da ilha de consistência do grupo, para que ela seja executada nas réplicas da ilha. A mensagem é difundida para os *GroupAdministrator* através do método *deliver_message*, sendo então executada sobre as réplicas do grupo. O resultado da operação sobre o objeto de aplicação é retornado como resultado de um consenso no grupo de objetos composto apenas pelos integrantes da ilha de consistência.

4.5.2 Operações instáveis

As operações instáveis, foram implementadas através de uma requisição direta do *OGSRGroupAccessor* instanciado na aplicação cliente para o *GroupAdministrator* associado a uma réplica secundária do grupo, ou seja, o cliente executa a operação em uma réplica secundária (fora da ilha de consistência)(Figura 4.5). As operações instáveis são solicitadas pelo método *tentative* do *OGSRGroupAccessor*. O *GroupAdministrator* da réplica secundária recebe a mensagem através do método *deliver_tentative* e repassa a mensagem para o objeto replicado associado a ele, sem iniciar qualquer consenso ou propagar o estado para os demais membros do grupo. Após a execução da operação, a mensagem é armazenada na estrutura *tent_list*. A *tent_list* é uma lista de operações instáveis utilizada no processo de reintegração da réplica com a ilha de consistência. O

Figura 4.4: Diagrama de operação estável.

resultado da execução da operação instável sobre a réplica é devolvido para o *OGSRGroupAccessor* como resultado da operação *tentative*.

A implementação do método *tentative* no OGS/R garante a atomicidade da operação, utilizando funcionalidades de bloqueio já implementadas no OGS. Portanto, mesmo que a réplica secundária esteja atendendo à requisições de vários *OGSRGroupAccessor*, não há a possibilidade de ocorrência de inconsistência no estado da réplica. No entanto, como já mencionado, ocorrerá uma divergência entre o estado da réplica secundária e a ilha de consistência.

4.5.3 Integração & Visão

Na *Integração & Visão* uma aplicação pode fazer com que a réplica secundária "local", de estado divergente, volte a assumir um estado consistente com o estado da ilha. O método do *OGSRGroupAccessor*, que implementa a reintegração da réplica secundária foi chamado de *reintegrate*.

Em um *reintegrate* (Figura 4.6), o *OGSRGroupAccessor* do cliente executa o papel fundamental de coordenar a reintegração, fazendo a ponte entre a réplica secundária e a ilha. Ao executar um *reintegrate*, o *OGSRGroupAccessor* invoca o método *reintegrate* do *GroupAdminis-*

Figura 4.5: Diagrama de operação instável.

trator de uma réplica primária. O resultado desta operação é o conjunto de operações estáveis realizadas na ilha desde o prefixo comum maximal da réplica secundária com a ilha e é entregue ao *OGSRGroupAccessor*. Este conjunto de operações é repassado para a réplica secundária, através do método *update_backup*. A réplica secundária implementa o algoritmo de reintegração discutido na seção 4.3. Inicialmente, todas as operações realizadas na ilha são executadas na réplica secundária. Em seguida, é retornado para o *OGSRGroupAccessor* o conjunto de operações instáveis com identificadores maiores que o identificador da última operação estável da lista. De posse das operações instáveis que devem ser consolidadas na ilha, o *OGSRGroupAccessor* executa cada uma das operações da lista retornada pelo método *update_backup* como uma operação estável (método *multicast*), atualizando assim a ilha com as operações mais recentes da réplica secundária.

4.6 Dificuldades Encontradas

Foram vários os problemas de implementação que tiveram de ser resolvidos ao longo deste trabalho. Desde questões relacionadas à implementação do OGS, como comunicação de grupo, até adaptações necessárias para o Juggler, responsável pelas *Factories* dos objetos replicados. Nesta seção vamos discutir um pouco das dificuldades encontradas para implementar as três operações

Figura 4.6: Diagrama da operação de Integração&Visão.

disponíveis no OGS/R.

A implementação da operação estável teve como maior desafio permitir o consenso do OGS mesmo com um número menor de réplicas (as réplicas da ilha), mantendo as funcionalidades do OGS original e todas as réplicas como membros do grupo. O OGS é um serviço de grupo de objetos que tem como elemento central a comunicação multi-ponto entre os membros do grupo. Em sua versão original, todos os membros participam dos consensos de multicast, mesmo os que não irão executar a mensagem. No OGS/R a configuração de um grupo pode conter inúmeras réplica secundárias e em caso extremo somente uma primária, sendo portanto eventualmente desnecessária a comunicação entre muitas réplicas. Para implementar a consistência relaxada do OGS/R e manter compatibilidade com o funcionamento do OGS+, dividimos os membros em subgrupos de réplicas primárias(ilha de consistência) e secundárias. Com isso, a lista de membros para a entrega de mensagens no *GroupAdministrator* foi reduzida para apenas a ilha de consistência.

Outra dificuldade foi a implementação da operação instável. No OGS, uma operação é executada em uma réplica e esta operação é repassadas para todos os membros de forma idêntica. No OGS/R, era necessário que a operação fosse recebida e processada por apenas uma réplica. A melhor forma para implementar isto no OGS+ seria um multicast com o retorno do resultado por apenas uma réplica do grupo. Isso teria duas conseqüências que não desejávamos: (a) todas

as réplicas receberiam a operação, apesar de apenas uma executá-la, causando uma sobrecarga desnecessária na rede; (b) se houvesse uma segunda solicitação de operação instável pelo mesmo cliente, então outra réplica poderia responder a solicitação e em outro estado e assim o cliente eventualmente não perceberia o estado recém-modificado do objeto. A solução para evitarmos (a) foi implementar o método *tentative* no *OGSGroupAccessor* especialmente para este tipo de operação. Buscando uma solução para (b) optamos por eleger uma única réplica secundária para cada cliente, durante todo o ciclo de vida de um *OGSGroupAccessor*. Portanto, a operação instável é entregue diretamente ao *GroupAdministrator* de uma réplica secundária e é executada sobre o objeto da mesma forma como ocorre no OGS original, o resultado da operação é devolvido diretamente ao *OGSGroupAccesor* solicitante. Com isso, a aplicação cliente tem uma visão consistente do grupo, simulada pelo OGS/R.

A implementação da operação de Integração&Visão foi a que causou maiores dificuldades. Foi especificado que a réplica secundária receberia a lista de operações estáveis do grupo, executaria-as, selecionaria todas as operações instáveis realizadas nela após o prefixo comum maximal e devolveria para a ilha a lista de operações instáveis a serem consolidadas na ilha. Na primeira tentativa de implementação, o *OGSGroupAccessor* solicitava um *reintegrate* à réplica secundária e esta tentava se sincronizar com a ilha de consistência, simulando o comportamento que ocorre numa mudança de estilo de replicação, onde a operação interna *STATE_UPDATE* leva a réplica a um estado consistente. No entanto, utilizando esta abordagem entrava-se em uma espera infinita, pois a réplica secundária não possuía identificador de consenso com estado semelhante ao dos membros da ilha, o que inviabilizava o uso do serviço de consenso. Uma segunda opção seria a de simular a entrada da réplica secundária na ilha, como acontece quando inserimos um novo membro no grupo e o definimos como membro da ilha de consistência. Este membro, ao ser inserido no subgrupo de membros da ilha de consistência iria adquirir o mesmo estado da ilha. Esta opção de implementação não pôde ser concretizada devido às restrições impostas pelo Juggler, pois com a simulação de entrada no grupo, a referência às réplicas utilizada pelo serviço era perdida. Dado que uma alteração do Juggler não era o foco do trabalho, optamos por uma terceira abordagem, que foi a implementada no OGS/R. Nesta abordagem, o *OGSGroupAccessor* através de seu método *reintegrate* coordena a interação entre dois *GroupAdministrators*,

um de uma réplica primária e o outro da secundária utilizada pelo *OGSRGroupAccessor*. A implementação desta abordagem esta descrita na seção 4.5.3.

Capítulo 5

Resultados Experimentais

Este capítulo apresenta os resultados de alguns testes realizados com o serviço de grupos de objetos OGS/R. Os testes tiveram o objetivo de medir e comparar o desempenho de grupos de réplicas que utilizam os estilos de replicação RELAXED, VIRTUAL_PRIMARY_COPY e ACTIVE, medindo o tempo de resposta nas operações estáveis, instáveis e reintegrações de réplicas para diferentes configurações de grupo.

Para a realização dos testes, criou-se uma aplicação simplificada que simula um sistema de informações sobre a velocidade média de veículos em trânsito em um determinado ponto da cidade. As informações de velocidade média são armazenadas em grupos de réplicas no OGS/R, grupos estes gerenciados pelo Juggler[17].

Para a avaliação dos serviços, fizemos baterias de testes para dois cenários, com os seguintes objetivos: (1) obtenção do tempo médio de execução de uma operação para cada um dos diferentes estilos de replicação, variando o número de clientes da aplicação; (2) obtenção do tempo médio de execução de uma sequência de operações distintas sobre os diferentes estilos de replicação, variando o percentual de operações instáveis e reintegrações no estilo RELAXED.

Para os experimentos foram utilizadas 6 máquinas com sistema operacional Linux, interligadas através de em uma rede local Ethernet de 10 Mbps. Os testes foram realizados no período noturno e nos fins de semana, de forma a minimizar a interferência decorrente da execução de outras aplicações no mesmo ambiente.

Para os testes foram criados três grupos no OGS/R: A, B e C, cada qual executando um

estilo de replicação distinto: ACTIVE, VIRTUAL_PRIMARY_COPY e RELAXED, respectivamente. Estes grupos eram compostos por quatro ou seis réplicas, dependendo do cenário a analisar. Em cada cenário, a bateria de testes consistiu da execução de lotes de 300 iterações das operações em cada um dos grupos. As requisições partiam de 1 ou mais clientes para avaliar o tempo de resposta no atendimento a várias requisições simultâneas.

5.1 Cenário 1

No primeiro conjunto de testes, verificamos o comportamento das operações do OGS/R em cada estilo de replicação.

Uma operação estável, implementada através de um multicast confiável, pode ser realizada em qualquer um dos diferentes estilos de replicação oferecidos pelo OGS/R. Assim, é possível comparar o desempenho desta operação em cada um dos três estilos de replicação. Neste cenário, coletamos o tempo de execução de operações estáveis sobre cada um dos grupos de réplicas, usando 1, 3, 5 e 7 clientes simultâneos. Para efeito de comparação, inserimos no gráfico também o desempenho da execução de *operações instáveis* no grupo de réplicas C, seguindo a mesma variação de número de clientes simultâneos.

Para a obtenção do tempo médio de uma operação estável foram realizados 300 operações estáveis por cliente sobre cada um dos três grupos, ou seja, 900 multicasts para 3 clientes, 1500 para 5 clientes, etc.

O gráfico da figura 5.1 apresenta os tempos de execução de operações sobre grupos compostos por quatro réplicas, sendo que nos grupos B e C uma destas é réplica primária. Pelo gráfico é possível perceber que, à medida que se aumenta o número de clientes, as curvas dos grupos A e B crescem bastante, o que acontece de forma bem mais lenta no grupo C (com replicação relaxada). O tempo de execução de uma operação instável no grupo C também está representado no gráfico e permanece quase constante, independente do número de clientes, devido ao fato de ser uma operação executada sem consenso no grupo.

O crescimento das curvas para os grupos A e B se deve ao fato de que todas as operações são difundidas através de multicast confiável para todos os membros do grupo. No grupo A, todos

Figura 5.1: Comportamento dos estilos com 4 réplicas (1 primária)

os membros do grupo participam do multicast e executam a operação recebida. No grupo B com estilo VIRTUAL_PRIMARY_COPY, as réplicas secundárias também participam do multicast, mas não executam as operações recebidas. O fato de termos no grupo C o multicast executado apenas na ilha garante uma menor degradação no tempo de resposta, mesmo com o aumento de acessos simultâneos ao grupo.

No gráfico da figura 5.2, fizemos o teste com grupos de 6 réplicas, sendo duas delas réplicas primárias. O desempenho de cada estilo de replicação ficou mais evidente neste gráfico. A curva de desempenho do grupo B se aproxima do desempenho do grupo A, pois com um total de 6 réplicas o multicast demanda mais tempo do que o multicast realizado com 4 réplicas. Estes dados mostram a escalabilidade do estilo de replicação RELAXED.

5.2 Cenário 2

Neste conjunto de testes avaliamos o desempenho do OGS/R para diferentes estilos de replicação, quando submetido a uma seqüência de operações de acesso ao objeto replicado, variando o percentual de operações instáveis e reintegrações no estilo RELAXED.

Figura 5.2: Comportamento dos estilos com 6 réplicas (2 primárias).

Para estes testes, os grupos A, B e C eram compostos por seis réplicas, onde a ilha de consistência nos grupos B e C era composta de duas réplicas. As seqüências de operações eram submetidas por um único cliente.

Nestes testes essencialmente avaliamos como o custo combinado de operações instáveis com eventuais reintegrações se compara ao custo de operações estáveis. Para tal, medimos o tempo total de execução de seqüências de operações geradas aleatoriamente em cada um dos estilos de replicação. No caso do grupo C (estilo RELAXED), definimos que para cada operação estável seria executada uma operação de reintegração da réplica secundária com a ilha. Isto equivaleria a uma operação estável (para leitura/escrita) de um valor consistente, nos demais estilos de replicação.

Para a obtenção do tempo médio de execução das seqüências foram variados: (a) o percentual de operações instáveis que poderiam assumir os valores de, no caso 20%, 35%, 50%, 60% e 70% do total de operações realizadas; e (b) o percentual de reintegrações realizadas para tornar consistente o estado de uma réplica secundária durante a execução da seqüência, que assumiu os valores 20%, 50% e 70% do total de operações realizadas. As seqüências eram executadas com operações instáveis e reintegrações apenas no estilo RELAXED, nos demais estilos as seqüências eram executadas através de multicast confiável.

Figura 5.3: Tempo de processamento para diferentes percentuais de operações instáveis, com 20% de reintegração

A figura 5.3 apresenta o gráfico do tempo médio usado por cada estilo de replicação para executar seqüências de operações com 20%, 35%, 50%, 60% e 70% de operações instáveis e percentual de 20% de operações de reintegração da réplica secundária.

Figura 5.4: Tempo de processamento para diferentes percentuais de operações instáveis, com 50% de reintegração

Para a obtenção do tempo médio de processamento da sequência nos estilos de replicação foram geradas 300 sequências, onde uma mesma sequência gerada era submetida aos três grupos de réplicas (cada qual com um estilo de replicação).

Ao analisarmos o gráfico que representa o comportamento dos estilos de replicação, quando o grupo C realiza reintegrações em somente 20% das operações estáveis, podemos concluir que o desempenho do estilo de replicação RELAXED se mostra melhor quando o percentual de operações instáveis na sequência é maior do que 30%.

A Figura 5.4 apresenta o gráfico do tempo médio usado por cada estilo de replicação para executar sequências com 20%, 35%, 50%, 60% e 70% de operações instáveis, com percentual de 50% de operações de reintegração nas réplicas secundárias.

Figura 5.5: Comportamento dos estilos com relação as operações instáveis, com 70% de reintegração

Neste gráfico percebe-se que com o percentual de operações instáveis entre 35% e 60% o estilo RELAXED tem desempenho semelhante ao VIRTUAL-PRIMARY-COPY, pois com um percentual médio de operações instáveis a serem reintegradas em 50% das operações estáveis, o custo relativo as várias reintegrações não compensa o ganho obtido com as operações instáveis.

Quando o percentual de operações instáveis é menor do que 35%, as operações estáveis se tornam muito freqüentes na sequência e desta forma reduzem a probabilidade de divergências

das réplicas secundárias, diminuindo assim o custo das reintegrações. A partir do percentual de operações instáveis maior que 60% já percebemos um melhor desempenho do estilo RELAXED quando comparado com os estilos VIRTUAL_PRIMARY_COPY e o ACTIVE, isto se deve a quantidade majoritária de operações instáveis, cujo o custo é baixo e praticamente constante.

A Figura 5.5, deixa claro que para seqüências de operações onde 70% de reintegrações são realizadas, não se justifica a utilização do estilo de replicação RELAXED, que passa a ser menos vantajoso do que o estilo VIRTUAL_PRIMARY_COPY.

No entanto, através dos testes foi possível perceber que o estilo de replicação RELAXED (e VIRTUAL_PRIMARY_COPY), mesmo considerando o custo da reintegração, apresenta um tempo de processamento (e resposta) muito abaixo do estilo ACTIVE, o que se deve ao custo menor da manutenção da consistência das réplicas da ilha em comparação com o custo de manter todas as réplicas do grupo consistentes.

Capítulo 6

Conclusão

Replicação de objetos em sistemas distribuídos é usada para garantir uma maior disponibilidade de serviços e de dados, bem como para implementar serviços/aplicações resilientes à falhas. O uso de serviços de grupo para a manutenção das réplicas de objetos de uma aplicação torna possível uma separação entre as questões específicas da aplicação e as questões intrínsecas do gerenciamento de réplicas.

Cada vez mais, serviços de grupo de objetos implementam um conjunto maior de funções para a criação e a manutenção de consistência em grupos de réplicas de objetos em aplicações distribuídas, fornecendo ao usuário do serviço ferramentas poderosas e flexíveis para o gerenciamento e a configuração de diferentes estilos de replicação. No entanto, há poucos serviços de grupo que dão apoio a sistemas nos quais haja a possibilidade de desconexões freqüentes entre os nós da rede. Neste cenário faz-se necessária a implementação de um estilo de replicação com consistência relaxada em um serviço de grupo. Neste serviço de grupo, a aplicação tem um controle maior sobre a consistência das réplicas no grupo, podendo inclusive determinar a forma como cada operação será propagada no grupo.

A principal contribuição desta dissertação é a especificação, implementação e avaliação do desempenho do OGS/R, um serviço de grupos de objetos CORBA que implementa um estilo de replicação com manutenção de consistência relaxada entre réplicas. Esta abordagem garante uma maior escalabilidade do grupo, pois devido ao relaxamento das regras de consistência no grupo, o custo da manutenção da mesma é consideravelmente reduzido e é menos dependente

do tamanho do grupo. O OGS/R e seu estilo de replicação RELAXED oferecem um mecanismo eficiente e flexível para a comunicação de grupo em sistemas com desconexões. O serviço é baseado na estruturação do grupo em uma ilha de consistência (com réplicas primárias) e réplicas secundárias, e provê *consistência seletiva* para as operações sobre estas réplicas. A aplicação tem a possibilidade de efetuar operações estáveis e instáveis sobre o grupo de réplicas, sendo que as operações instáveis afetam somente uma réplica secundária “local”. Além destas, há a operação de *Integração e Visão*, onde, a qualquer momento, a aplicação pode forçar que réplicas divergentes sejam sincronizadas com a ilha de consistência do grupo.

Os testes realizados com o OGS/R mostraram os benefícios do estilo de replicação RELAXED. Através destes, foi possível perceber a escalabilidade da abordagem, e comparar o desempenho do estilo de replicação RELAXED com outras abordagens, indicando as situações nas quais o uso do estilo RELAXED tem melhor desempenho.

A experiência adquirida no processo de inclusão do estilo de replicação no OGS é bastante significativa. As características deste estilo de replicação possibilitaram uma interação com toda a arquitetura do serviço de grupo, onde pudemos identificar e alterar os módulos envolvidos.

O estilo de replicação RELAXED tem uma maior aplicabilidade em sistemas com baixa taxa de reintegração de réplicas secundárias, como é o caso de algumas aplicações em redes móveis, nas quais usuários desconectados podem conviver com dados locais temporariamente imprecisos e inconsistentes com as demais réplicas. Exemplos são sistemas de informação nas quais os dados armazenados em uma réplica secundária contém dados preliminares, incompletos, ou interessantes apenas para usuários geograficamente próximos, enquanto réplicas primárias contém dados consolidados.

Como trabalhos futuros, podemos sugerir a implementação da funcionalidade de troca dinâmica do estilo de replicação RELAXED para os demais estilos implementados no OGS/R. Esta mudança pode ser vista como um processo de reintegrar todas as réplicas secundárias às réplicas primárias do grupo, para só então mudar o estilo de replicação do grupo.

Referências Bibliográficas

- [1] A. A. Helal, A. A. Heddaya and B. B. Bhargava. *Replication Techniques in Distributed Systems*, Kluwer Academic Publishers, 1996.
- [2] Lenz, Richard. *The “Virtual-Primary-Copy Approach” Compared to Other Approaches with Weak Consistent Data Replication*, University of Erlangen-Nuremberg, 1996.
- [3] R. A. Golding. *Weak-Consistency Group Communication and Membership*, Ph.D. Thesis, University of California, december 1992.
- [4] R. A. Golding, and D. D. E. Long. *Desing Choices for Weak-Consistency Group Communication*, UCSC-TR-92-45, University of California, october 1992.
- [5] K. P. Birman. *A Review of Experiences with Reliable Multicast*, Software Practice and Experience, Vol. 29, no.9, 1999.
- [6] A. Demers and K. Petersen et al, *The Bayou Architecture: Support for Data Sharing Among Mobile Users*, In Proc. IEEE Workshop on Mobile Computing Systems & Applications, California, december 1994.
- [7] N. Adly, *Performance Evaluation of HARP: A Hierarchical Asynchronous Replication Protocol for Large Scale Systems*, Technical Report TR-378, Computer Laboratory, University of Cambridge, August 1995.
- [8] M. Satyanarayanan, J. Kistler, P. Kumar, *CODA: A Highly Available File System for a Distributed Workstation Environment*, IEEE Transactions on Computers, vol.39, n° 4, april 1990.

- [9] X. Liu, *Data Communication and Replication Strategies in Large-Scale Distributed Databases*, Ph.D. Thesis, Purdue University, december 1995.
- [10] H. T. Kung and J. T. Robinson, *On Optimistic Methods for Concurrency Control*, ACM Transactions on Database Systems, Vol. 6, 1981.
- [11] R. Ladin, B. Liskov and L. Shira, *Lazy Replication: Exploiting the Semantics of Distributed Services*, Operating Systems Review, january 1991.
- [12] R. Alonso, H. Garcia-Molina and D. Barbara, *Quasi-Copies: Efficient Data Sharing for Information Retrival Systems*, EDBT'88: Extending Database Technology, march 1988.
- [13] C. Pu and A. Leff, *Epsilon-Seriability*, Technical Report CUCS-054-90, Columbia University, 1991.
- [14] S. Ceri, W. Keller, P. Samarati et al, *Independent Updates and Incremental Agreement in Replicated Databases*, Distributed and Parallel Databases Vol. 3, 1995.
- [15] P. Felber, *The CORBA Object Group Service: A Service Approach to Object Groups in CORBA*, Ph.D. thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 1998.
- [16] D. Silva, M. D. Gubitoso and M. Endler. *Sistemas de Informação Distribuídos para Agentes Móveis*, SEMISH'98, August 1998.
- [17] M. A. de Moura and M. Endler, *Juggler: A Configuration Management Service for CORBA Object Groups*, 18º Simpósio Brasileiro de Redes de Computadores (SBRC 2000), May 2000.
- [18] D. Ratner, P. Reiher and G. J. Popek, *Replication Requirements in Mobile Environments*, Department of Computer Science University of California, Technical Report n° 970021, Los Angeles, 1997.
- [19] M. A. de Moura, *Uma Infra-Estrutura para o Gerenciamento de Aplicações CORBA Tolerantes a Falhas*, Dissertação de Mestrado, IME-USP, fevereiro 2001.

- [20] O. Babaoglu, H. Meling, A. Montresor and B. Helvik, *Jgroup/ARM: A Distributed Object Group Platform with Autonomous Replication Management for Dependable Computing*, University of Bologna, Italy, February 2002.
- [21] D. Safford, D. K. Hess and U. Pooch, *A Unix Network Protocol Security Study: Network Information Service*, Technical report, Texas A & M University, 1992.
- [22] Michi Henning e Steve Vinoski, *Advanced CORBA Programming with C++*, Addison-Wesley, 1999.
- [23] G. Morgan and P. D. Ezilchelvan, *Policies for using replica groups and their effectiveness over the Internet*, Networked Group Communication, pág. 119-129, 2000.
- [24] J. Gray, P. Helland, P. O'Neil, and D. Shasha, *The dangers of replication and a solution*. In *Proceedings of the 1996 ACM, SIGMOD*, June 1996.