Uma Experiência com o Modelo de Programação Orientada a Grupos

Jeferson Botelho do Amaral[♠] Raul Ceretta Nunes[♦]

Departamento de Eletrônica e Computação
CT – UFSM – Campus Camobi – Santa Maria – RS – CEP 97105-900
Tel. (055) 220 8418 – Fax: (055) 220 8030 – e-mail: *delc@ct.ufsm.br*

Resumo

A coordenação segura e consistente da cooperação entre processos num sistema distribuído sob falhas exige muita habilidade do programador. Primitivas para comunicação confiável entre grupo de processos criam uma estrutura básica poderosa para o desenvolvimento de aplicações distribuídas.

Este artigo descreve uma experiência de programação usando o sistema *x*AMp (*Extended Atomic Multicast Protocol*), um serviço de comunicação de grupo altamente versátil. Ele consiste de um pacote integrado, projetado para ser usado sobre redes locais. O objetivo é o desenvolvimento de uma aplicação simples, neste caso replicação de dados, usando o paradigma de grupo de processos.

Palavras-chave: comunicação de grupo, *x*AMp, replicação de dados

Abstract

In distributed systems, to reach secure and consistent processes coordenation demand expert programmers. Reliable group communication primitives create a powerful framework for the development of distributed applications.

This paper describes a programming experience using the *x*AMp system (*Extended Atomic Multicast Protocol*), a highly versatile group communication service. It consists of an integrated package, designed to be used over local-area networks. The goal is the development of a simple application, in this case data replication, using the process group paradigm.

Keywords: group communication, *x*AMp, data replication

1 Introdução

Desenvolver aplicações distribuídas robustas exige do programador um grande esforço para driblar os problemas oriundos de defeitos típicos na comunicação, principalmente quando é desejado que as aplicações possuam tolerância a falhas. No desenvolvimento destas aplicações, as tarefas e a cooperação entre os processos apontam para uma modelagem baseada no conceito de grupo de processos [BIR96]. A existência de primitivas para comunicação confiável entre os processos pertencentes a um grupo são vitais na determinação da confiabilidade e conseqüente possibilidade de suporte a falhas da aplicação. Sistemas com tolerância a falhas têm usado primitivas de comunicação de grupo para propagar atualizações em réplicas e para fazer, por exemplo, decisões distribuídas ao colecionar quorum num sistema de votação.

[♠] Bacharel em Informática pela UFSM – e-mail: *amaral@inf.ufsm.br*
[♦] Prof Msc. do DELC/CT – UFSM – e-mail: *ceretta@inf.ufsm.br*

Padronizações para o modelo de comunicação de grupo estão em estudo no X/Open e no IEEE, e um padrão de comunicação paralela orientada a grupo *Message-Passing Interface* [MPI97] foi introduzido em 1993. Alguns sistemas operacionais distribuídos têm fornecido mecanismos para a comunicação de grupo, adicionalmente aos métodos mais convencionais como o RPC (*Remote Procedure Call*) e o mecanismo de *stream* [BIR96]. Entretanto, com poucas exceções como o Amoeba [TAN91], fornecem muitas limitações no suporte a características como confiabilidade e consistência, que são essenciais em aplicações que executam durante um tempo longo ou em aplicações críticas. A desvantagem do Amoeba está na pouca portabilidade para outros sistemas proprietários.

Há uma variedade de ferramentas fornecendo suporte à programação de aplicações que desejam usar comunicação de grupo confiável. **Isis** [BIR87] foi o primeiro sistema a introduzir primitivas de comunicação de grupo; foi desenvolvido inicialmente como um projeto acadêmico na Universidade de Cornell e mais tarde tornou-se um produto comercial. Desde o Isis, muitas outras ferramentas têm surgido: **Transis** [AMI92], **Phoenix** [MAL95], **Relacs** [BAB95], ***x*AMp** [ROD92], **Horus** [REN95], etc. Estas ferramentas diferem nas primitivas de *multicast* que oferecem e, também, na forma de interfaceamento com as subcamadas do sistema distribuído.

Este trabalho descreve uma experiência prática utilizando o modelo de programação orientada a grupos. A aplicação escolhida e modelada foi **replicação de arquivos**, usando a técnica de cópia primária [JAL94]. O ambiente de programação utilizado foi o sistema operacional Linux, com primitivas de comunicação de grupo confiável fornecidas pela ferramenta *x*AMp. Como o objetivo foi apenas experimental, não se levou em conta todos os níveis de complexidades inerentes a este tipo de aplicação.

2 Comunicação de Grupo

A propriedade fundamental de grupo de processos é que quando uma mensagem é enviada para o grupo, todos os membros devem recebê-la, uma forma de comunicação **um-para-muitos** (***multicast***). Este tipo de comunicação contrasta com a comunicação **ponto-a-ponto** (***unicast***), onde um processo só é capaz de enviar uma mensagem para um outro processo.

Na utilização da comunicação de grupo, notam-se alguns aspectos semelhantes ao modelo comum de troca de mensagens [TAN92], por exemplo: **bufferização** e **não-bufferização**; **bloqueio** e **não-bloqueio**. Entretanto, há muitos detalhes novos, pois o envio de mensagens para um grupo de processos difere do envio a um único processo, e a estrutura interna dos grupos pode ser organizada de diversas formas.

Em **grupos fechados**, somente os membros podem enviar mensagens para o grupo; processos que não pertençam ao grupo não podem enviar mensagens para o grupo como um todo, embora possam fazê-lo para membros individuais do grupo. Por outro lado, na estruturação através de **grupos abertos** não há esta propriedade, pois qualquer processo do sistema é capaz de enviar mensagens para qualquer grupo.

Em alguns grupos há uma igualdade entre os processos, nenhum é superior, e todas as decisões são tomadas coletivamente (**grupos não-hierárquicos**). Em outros grupos, os processos são organizados hierarquicamente (**grupos hierárquicos**), podendo, neste caso, existir um processo coordenador com missão de gerenciar as tarefas dos demais processos no grupo.

Em **grupos estáticos**, os membros não podem deixar o grupo e nem novos processos podem juntar-se a ele. Já em **grupos dinâmicos**, novos grupos podem ser criados ao passo que grupos antigos podem ser destruídos, e um processo pode juntar-se ao grupo ou sair do grupo. Adicionalmente, um processo pode ser membro de vários grupos ao mesmo tempo, característica esta denominada **sobreposição**.

3 *x*AMp

O *x*AMp (*Extended Atomic Multicast Protocol*) [ROD92], desenvolvido no Instituto de Engenharia de Sistemas de Computadores da Universidade de Lisboa, é uma ferramenta de suporte ao desenvolvimento de aplicações distribuídas com exigências de performance, funcionalidade e dependabilidade. Fornece serviços para o gerenciamento de membros do grupo, permitindo a criação dinâmica e a reconfiguração de grupos de processos. Também permite que durante o tempo de vida de um grupo, processos possam juntar-se a um grupo (*join*) ou deixá-lo (*leave*). A falha de um membro do grupo é detectada e uma indicação do evento é enviada aos membros remanescentes do grupo.

A ferramenta *x*AMp fornece um suporte eficiente e versátil para a troca de informações entre os membros do grupo, um serviço de comunicação *multicast*. O serviço aceita uma lista de endereços, chamado **endereço seletivo**, como um endereço de destino válido para uma mensagem *multicast*, assim pode enviar, transparentemente, uma mensagem para os destinatários. Um **endereço lógico**, adicionalmente, pode ser associado com um grupo, permitindo a todos os membros do grupo serem endereçados através de um **nome lógico**. Isto libera o programador de ter que, explicitamente, gerenciar listas de endereços seletivos.

Outra característica do *x*AMp, é fornecer um ambiente de execução com propriedades de **validade** e **sincronismo**, as quais são relevantes para a maioria dos sistemas de comunicação. Isto faz com que o usuário possa confiar no sistema, no sentido de que as mensagens não serão corrompidas, não serão perdidas arbitrariamente, nem mesmo geradas espontaneamente.

Propriedades de **acordo** surgem quando do envio de mensagens *multicast*; neste conjunto, a propriedade de **unanimidade** é a que fornece maiores garantias, pois se uma mensagem for enviada a um membro correto, será enviada a todos os membros corretos, mesmo na presença de falhas. Entretanto, este tipo de propriedade, pode não ser tão interessante para alguns tipos de aplicações, pois degrada o desempenho. Por exemplo, pedidos a servidores replicados precisam alcançar apenas uma das réplicas, ao invés de todas, uma vez presumido que todas as respostas são idênticas. Protocolos baseados em quorum são outro exemplo onde a propriedade de unanimidade não é exigida. Devido a fatos deste tipo, o *x*AMp possui diversas propriedades de acordo.

Finalmente, o *x*AMp possui propriedades que especificam o tipo de **ordenação** que o protocolo deve utilizar para a troca de mensagens entre seus membros. O serviço de comunicação de grupo do *x*AMp proporciona diversas primitivas, intencionando satisfazer a maioria das exigências das aplicações com respeito à comunicação de grupo. A **ordenação total** é a propriedade que fornece maiores garantias, pois as mensagens são enviadas na mesma ordem para os diferentes membros. **Ordenação causal** e **FIFO** provêm menores garantias, contudo há uma redução no custo, o que melhora o desempenho em aplicações que não exigem ordenação total.

4 Replicação de Arquivos e a Técnica de Cópia Primária

Replicação de arquivos se insere no contexto de várias tecnologias criadas para se conseguir maior dependabilidade nos sistemas de computação. A replicação pode ser conseguida através de técnicas com enfoque no *hardware* ou em mecanismos de *software* (utilizados neste trabalho).

As técnicas de replicação por *software* baseiam-se em sistemas com processadores do tipo *fail-stop* e enquadram-se em três classes: **técnica de cópia primária**, **técnica de cópias ativas** e **técnica de quorum**. O modelo de grupo de processos adequa-se bem tanto à técnica de cópias ativas quanto a de cópia primária. Em decorrência do caráter experimental deste trabalho, foi escolhida a técnica de cópia primária, pela maior simplicidade de implementação. Esta técnica garante que operações sobre os dados possam ser realizadas, mesmo se alguns nós que estejam atuando como servidores de arquivos, ou as comunicações, falhem [JAL94].

Na especificação original da técnica de cópia primária, para que seja possível suportar a falha de n nós, é necessária a existência de, no mínimo, $n+1$ nós. Um destes nós é denominado como *primário*, e a cópia de um volume existente nele é conhecida como *cópia primária*. O restante dos nós são denominados *backups*. Um servidor *backup* não interage diretamente com os clientes, mas somente com o servidor primário. Portanto, há um centralizador de pedidos na técnica, o que facilita a ordenação dos pedidos. O servidor primário ao receber um pedido de leitura interage somente com o leitor. Mas, ao receber um pedido de escrita, deverá difundir a atualização para os seus servidores *backup* e, somente após receber a confirmação, volta a interagir com o cliente escritor.

Em caso de falha, um fator importante para manter a consistência das réplicas são as propriedades de ordenação e atomicidade aplicadas sobre as mensagens usadas na atualização das réplicas. Por exemplo, quando o servidor primário falha após já ter enviado uma mensagem de operação para qualquer um dos *backups*, mas ainda não tenha enviado a mensagem para todos os *backups*, a falha poderia fazer com que um servidor *backup* se tornasse primário sem possuir o estado mais atual da cópia. Isto pode ser evitado com o uso de um protocolo de difusão de escritas atômico com comunicação confiável, similar ao fornecido pela ferramenta *x*AMp.

5 Modelagem da Aplicação

Segundo Tanenbaum [TAN92], grupos fechados são adequados ao processamento paralelo e grupos abertos, por outro lado, são adequados para aplicações do tipo cliente-servidor. Entretanto, verificou-se que as ferramentas de comunicação de grupo existentes não costumam oferecer grupos abertos. Normalmente isto não é oferecido devido à complexidade de gerenciar e implementar, de forma confiável, tal tipo de grupo. No caso do *x*AMp, ferramenta utilizada, os grupos são fechados.

A modelagem da aplicação, como mostra a figura 1, fez uso de grupos fechados (único oferecido) e dinâmicos, dado que um grupo deve definir um novo servidor primário sempre que o corrente falhar. A interação cliente-servidor foi obtida fazendo uso de 3 grupos distintos: **grupo dos leitores** (formado por processos clientes que desejam ler um arquivo), **grupo dos escritores** (formado por processos clientes que desejam alterar o conteúdo de determinado arquivo) e **grupo dos servidores** (formado pelo processo com função de servidor primário e os processos com função de *backup*).

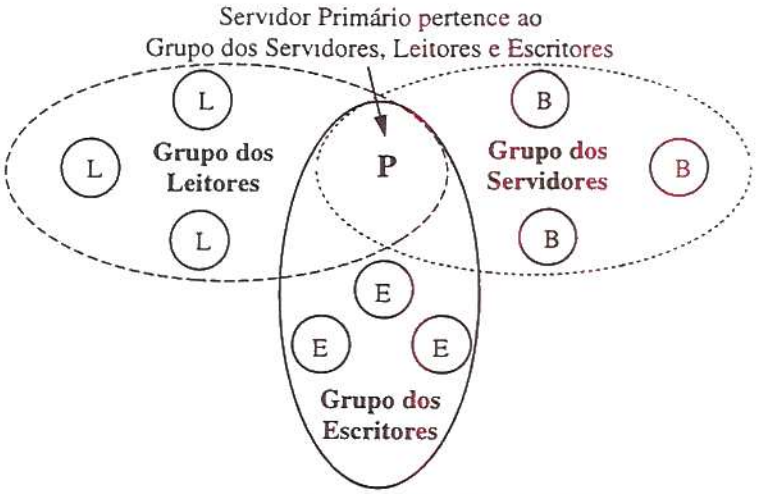

Figura 1: Modelagem utilizada na aplicação

Os processos do grupo dos servidores possuem o mesmo código, facilitando o mecanismo de eleição de um novo servidor primário que, como mostra a figura 1, apresenta característica de sobreposição. Desta forma, já que um cliente (leitor ou escritor), pertence ao mesmo grupo do servidor primário, torna-se possível a comunicação cliente-servidor. No grupo dos servidores há uma hierarquia entre o servidor primário e os servidores *backup*, pois o primário é quem os coordena. Por outro lado, pode-se dizer que o grupo também possui aspectos não-hierárquicos, pois os processos têm o mesmo código e, conseqüentemente, são capazes de desempenhar a mesma função, ou seja, um *backup* é capaz de tornar-se primário.

Para eleger um novo servidor primário, é utilizado o conceito de **visões** (em síntese, uma representação de quem é membro do grupo num determinado instante). Uma visão sofre modificação sempre que um novo membro se junta ao grupo, ou que um membro sai do grupo, de modo voluntário ou involuntário (uma falha). Cada grupo de processos possui uma lista contendo os processos membros do grupo. O conceito de visões pode ser aliado à identificação de um índice fixo na lista de membros como sendo o processo servidor primário [GUE97] [JAL94]. O *x*AMp mantém o controle da visão de cada grupo; caso um membro falhe, todos os demais são notificados da falha. Entretanto, é preciso construir um algoritmo que identifique se o membro que falhou foi o servidor primário e, em caso positivo, que inicialize um algoritmo de eleição.

6 Conclusões

Embora existam ferramentas fornecendo suporte à programação de aplicações utilizando o modelo de comunicação de grupos, o mercado ainda apresenta um número reduzido de aplicações fazendo uso desta técnica. Ao que parece, isto se deve ao pouco entendimento, por parte dos programadores, sobre como modelar diferentes classes de aplicações através de grupo de processos cooperantes.

Essa foi a principal observação deste experimento, pois a modelagem do exemplo tratado inicialmente parecia complicada, no que se refere a interação cliente-grupo (o *x*AMp usa um modelo de grupos fechados), contudo a característica de sobreposição de grupos aliada ao conceito de mudança dinâmica de visões, acabou facilitando bastante a programação Com base nesta constatação, verifica-se que a afirmação de Tanenbaum [TAN92], mencionada anteriormente na seção 5, pode ser mais uma questão de paradigma no uso de grupos do que propriamente uma adequação dos grupos abertos e fechados.

Atingiu-se o objetivo de verificação experimental do modelo de grupos de processos e a ferramenta *x*AMp mostrou-se bastante útil neste contexto, fornecendo ao programador a abstração de grupos de processos aliada a uma variedade de primitivas com características de atomicidade e ordenação. Observou-se que dispor deste tipo de ferramenta facilita bastante a programação.

Para trabalhos futuros, sugere-se implementar a mesma aplicação através de outra ferramenta que forneça a abstração de grupos, com a finalidade de realizar um comparativo levantando questões como flexibilidade, portabilidade e facilidade de programação.

7 Referências Bibliográficas

[AMI92] AMIR, Y.; DOLEV, D.; KRAMER, S.; MALKI, D. **Transis: A Communication Sub-System for High Availability**. 22nd International Symposium on Fault-Tolerant Computing (FTCS-22nd), Boston, July 1992.

[BAB95] BABAOGLU, O.; DAVOLI, R.; GIACHINI, L.A.; BAKER, M. **Relacs: A Communications Infraestruture for Constructing Reliable Apllications in Large-Scale Distributed Systems**, Technical Report UBLCS-94-15, June 1994.

[BIR87] BIRMAN, K.P.; JOSEPH, T.A. **Reliable Communication in the Presence of Failures**. ACM Trans. on Computer Systems, Vol. 5, No. 1, February 1987.

[BIR96] BIRMAN, K.P. **Building Secure and Reliable Network Applications**. Greenwich: Manning, 1996.

[GUE97] GUERRAOUI, R.; SCHIPER, A. **Software-Based Replication for Fault Tolerance**. IEEE Computer Magazine, Vol. 30, No. 4, pp. 68-74, Abril de 1997.

[JAL94] JALOTE, P. **Fault Tolerance in Distributed Systems**. New Jersey: Prentice-Hall, 1994.

[MAL95] MALLOTH, C.; FELBER, P.; SCHIPER, A.; WILHELM, U. **Phoenix: A Toolkit for Building Fault-Tolerant, Distributed Applications in Large Scale**, Départment d'Informatique, Ecole Polytechique Fédérale de Lausanne, Switzerland, July, 1995.

[MPI97] PACHECO, P.S. **Parallel Programming With Mpi**, San Francisco, Morgan Kaufmann, 1997. 418p.

[REN95] RENESSE, R.V.; BIRMAN, K. **Protocol composition in Horus**, Technical Report 95-1505, Cornell University, Dept. of Computer Science, March 1995.

[ROD92] RODRIGUES, L.; VERÍSSIMO, P. ***x*AMp: A Protocol Suite for Group Communication**, INESC, Technical University of Lisboa, January 1992.

[TAN91] TANENBAUM, A.S.; KAASHOEK, F.M.; RENESSE, R.V. **The Amoeba Distributed Operating System - A Status Report**. Department of Mathematics and Computer Science. Vrije Universiteit. Amsterdan, The Netherlands, 1991.

[TAN92] TANENBAUM, A.S. **Modern Operating Systems**. Englewwod Clifs, NJ.: Prentice-Hall, 1992. 728p.