Relatório Técnico Interno
004/98

LaSiD

Uma Proposta de Controle de Fluxo para o Protocolo BCGcausal

George Marconi de A. Lima
Raimundo J. de A. Macêdo

Maio, 1998

Uma Proposta de Controle de Fluxo para o Protocolo BCGcausal*

George Marconi de A. Lima+
gmlima@lasid.ufba.br
http://www.lasid.ufba.br/~gmlima

Raimundo José de A. Macêdo++
macedo@lasid.ufba.br
http://www.lasid.ufba.br/~macedo

Universidade Federal da Bahia
Centro de Processamentos de Dados - CPD
Laboratório de Sistemas Distribuídos - LaSiD

Resumo

Comunicação em grupo é um paradigma de comunicação usado para a construção de aplicações distribuídas confiáveis. Em tal paradigma, devido a possibilidades de ocorrência de falhas durante o envio de uma mensagem (*multicast*), as mensagens devem ser mantidas nos *buffers* dos processos até que sejam conhecidas entregues por todos os elementos do grupo. Para tanto, deve existir serviços que previnam ou tolerem estouro de buffers de forma transparente e eficiente. Tais serviços são implementados pelos mecanismos de controle de fluxo. Neste trabalho apresentamos um mecanismo de controle de fluxo que previne estouro de *buffers*. Essa solução foi projetada para o protocolo BCGcausal, um dos protocolos implementados na BCG, com o objetivo de reduzir a contenção da comunicação sem apresentar custo extra em relação ao tamanho da mensagem e maximizar a utilização dos *buffers* dos processos do grupo.

Palavras Chaves: comunicação em grupo, ordenação causal, tolerância a falhas, controle de fluxo.

1. Introdução

Num sistema distribuído existem vários processos que executam tarefas distintas e comunicam-se a fim de cooperarem. A forma tradicional de comunicação é conhecido como comunicação *ponto-a-ponto*, ou seja, os processos se comunicam aos pares. Os sistemas distribuídos modernos, no entanto, adotam um paradigma de comunicação mais genérico, conhecido como *comunicação em grupo*. Neste paradigma toda comunicação se faz propagando mensagens no grupo, facilitando a construção de aplicações distribuídas tolerante a falhas [VRV93].

Entretanto, não se pode simplesmente garantir a propagação de mensagens no grupo. Para evitar que os processos do grupo observem uma ordenação inconsistente dos eventos, é necessário que existam protocolos de ordenação de mensagens. Os protocolos de *ordenação causal* garantem que um processo do grupo não observe o efeito antes da causa. Já os protocolos de *ordenação total* garantem que, além da ordem causal, todos os processos do grupo observem a mesma ordem dos eventos [Mac94]. Observe a figura 1 que ilustra a entrega em *ordem causal* para um grupo de 3 processos. As linhas horizontais indicam o tempo físico e as linhas diagonais representam o fluxo de mensagens entre processos. Observe que m2 foi recebida antes que m1 no processo p_3. Contudo, sua entrega foi atrasada até que m1 chegasse a p_3. A mensagem m3 não possui relação causal nem com m1 nem com m2 (m3 é *concorrente* a m1 e m2), então ela pode ser entregue logo que é recebida.

* Este trabalho recebeu apoio financeiro do CNPq
+ Pesquisador bolsista DTI-F/CNPq/ProTeM-CC fase III (Projeto Locus), número 380096/97-8.
++ Pesquisador II do CNPq, bolsa PQ número 300013/87-6 (projeto BCG)

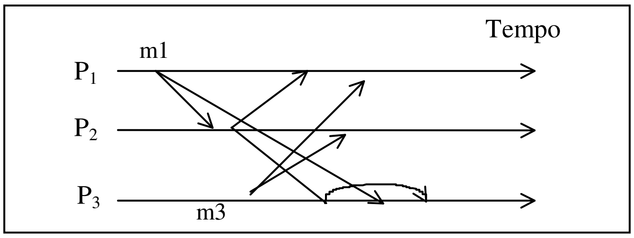

Figura 0: Entrega em ordem causal em comunicação em grupo.

As mensagens recebidas são armazenadas temporariamente nos *buffers* dos processos e apenas são entregues se forem satisfeitas as condições estabelecidas pelo protocolo de ordenação. Mas, mensagens não podem ser descartadas logo após a entrega, pois se houver uma falha durante a transmissão de uma mensagem para o grupo, pode ser que alguns processos recebam algumas mensagens e outros não. Isso obriga que as mensagens recebidas sejam armazenadas localmente e apenas sejam descartadas no momento que se saiba que todos os processos do grupo já as entregou. Mecanismos responsáveis por tais funcionalidades são chamados de mecanismos de *controle de fluxo* e são essenciais para a construção de protocolos de comunicação em grupo tolerante a falhas.

Os mecanismos de controle de fluxo para comunicação *ponto-a-ponto*, tal como o usado pelo TCP/IP, não podem ser utilizados em comunicação em grupo, pois não contemplam o paradigma de grupo e tolerância a falhas. Geralmente, estão baseados na manutenção de uma janela deslizante sobre o *buffer* de transmissão para garantir que não haverá estouro no *buffer* do processo receptor. Recentemente, o controle de fluxo para comunicação em grupo recebeu maior atenção, no entanto as soluções até então apresentadas foram projetadas para protocolos de ordem total, onde podemos citar [Mac94, MES96, ACM95, BSS91, HR95, MW97, RBM96] entre outros.

A ordenação total é mais restritiva que a ordenação causal. Por exemplo, a ordem de entrega das mensagens ilustrada na figura 1 é (m1,m2, m3) para p_1 e p_2 e; (m3, m1,m2) para p_3. Para o protocolo de ordenação total, a entrega das mensagens teria que ser atrasada até que fosse possível entregar as mensagens na mesma ordem para todos os processos. Portanto, um algoritmo de controle de fluxo para protocolos de ordem total pode também ser válido para ordenação causal, como em [Mac94,MES96]. O preço pago por isso é uma contenção adicional na transmissão de mensagens.

É importante observar que o mecanismo de controle de fluxo exerce influência no desempenho do sistema como um todo, fato demonstrado em [Mac94, MES96, MW97, HR95]. Se o controle de fluxo for muito restritivo causará muita contenção na transmissão de mensagens, diminuindo o desempenho do protocolo. Por outro lado, se o controle de fluxo for ajustado à ordem de entrega de mensagens, pode-se otimizar o desempenho. Existem basicamente duas políticas de controle de fluxo para comunicação em grupo [MW97]: otimista e conservadora. Os algoritmos de controle de fluxo conservadores subestimam a velocidade de consumo de mensagens pelos processos do grupo e baseados numa *visão* do estado do grupo, processos bloqueiam a transmissão até que haja certeza que ao transmitir uma nova mensagem não haverá estouro no *buffer* de algum processo do grupo. A política otimista, por outro lado, superestima a velocidade de consumo de mensagens do grupo e, baseados numa visão do estado do grupo, os processos arriscam a transmissão de novas mensagens. Se, por acaso, não houver espaço no *buffer* de algum processo do grupo, o processo receptor descarta a mensagem que causaria o estouro. Tal política supõe a existência de procedimentos para recuperação de mensagens. Note que se houver muitas mensagens descartadas, serão necessárias muitas retransmissões, aumentando o tráfego na rede e diminuindo o desempenho do sistema.

Neste trabalho apresentamos um novo algoritmo de controle de fluxo baseiado na política conservadora e foi projetada para o protocolo BCGcausal que faz parte da BCG (Base confiável de Comunicação em Grupo), em fase de implementação no LaSiD. A BCG é uma plataforma distribuída tolerante a falhas, implementada em C++, que provê vários serviços de comunicação em grupo: ordenação de mensagens, causal (BCGcausal) e total; controle de fluxo e *membership.* A solução apresentada, quando comparada com a solução atualmente implementada na BCG, reduz a contenção da comunicação sem apresentar custo extra em relação ao tamanho da mensagem (para o protocolo BCGcausal) e maximiza a utilização dos *buffers* dos processos do grupo.

Antes de descrever o mecanismo de controle de fluxo, apresentaremos alguns conceitos. Primeiramente, na seção 2 descreve o protocolo BCGcausal. A seção 3 apresenta nossa proposta a seção 4 analisa-a, comparando-a com outros trabalhos. Por fim, a seção 5 conclui esse trabalho.

2. O Protocolo BCGcausal

Assumimos que os processos estão distribuídos, possivelmente em processadores distintos e não compartilham memória. Todas as mensagens chegam a seu destino num tempo finito, porém indeterminado, livres de falhas de omissão e não são corrompidas. A transmissão de mensagens entre quaisquer pares de processos é FIFO[1]. As falhas consideradas são do tipo *crash*.

Em relação à hipótese de falhas, apesar de considerar falhas tipo *crash* não há preocupação neste trabalho com a ocorrência das mesmas. Supõe-se que exista um mecanismo, tal como o apresentado em [Mac94,EMS95] que provê serviços de *membership*, o qual possibilita reconfiguração dinâmica do grupo. É importante ressaltar, contudo, que para que haja detecção de possívies falhas, cada processo fica obrigado de enviar mensagens periodicas ao grupo (*mecanismo de timesilence)*, mesmo se não há nada a transmitir, informando que está "vivo". Tais mensagens são chamadas de *mensagens nulas*. O algoritmo de controle de fluxo proposto neste trabalho assume a presença de tais mecanismos, e eles de fato estão presentes na BCG. As seções seguintes descrevem o protocolo BCGcausal.

2.1 Relógio Lógico

Considere o grupo de processos $g = \{p_1 , p_2 , ... , p_n\}$. Cada processo p_i mantém um relógio lógico BC_i (*Block Counter*), o qual é inicializado com zeros por todos os processos pertencentes a g, no momento de sua criação. Ao transmitir uma mensagem m para o grupo (evento $send_i(m)$), o processo p_i carimba-a com o valor corrente de BC_i, representando assim a causalidade entre as mensagens, semelhantemente a [Lam78]. Ao entregar uma mensagem à aplicação, evento $deliver_i(m)$, o processo p_i atualiza $BC_i = \max\{BC_i ,m.b\}$.

Baseado nas regras acima, pode-se deduzir que as mensagens com mesmo número de bloco são concorrentes e que se $send(m) \rightarrow send(m') \Rightarrow m.b < m'.b$. As propriedades e os detalhes podem ser vistos em [Mac94, Mac95]. Na seção seguinte descreveremos como as mensagens são representadas na BCG e as implicações desta representação.

2.2 Blocos Causais e Matriz de Blocos

Considere uma matriz bidimensional, BM, denominada *matriz de blocos*, mantida por todo processo p_i pertencente ao grupo g, onde cada linha B, denominada bloco causal BM[B], representa

[1]Essas funcionalidades estão presentes em protocolos de comunicação tal como TCP/IP.

as mensagens transmitidas ou recebidas por p_i , cujo número de bloco é igual a B. O número de colunas da matriz é igual a |g| (tamanho do grupo g).

Ao receber ou transmitir uma mensagem m, cada processo p_i de g marca na sua matriz BM_i [m.b][m.s] = '+', onde m.s é o identificador do processo que enviou m, a fim de representar o recebimento ou a transmissão da mensagem. A figura 2 abaixo ilustra uma BM para um grupo de 6 processos. Suponha que o processo que mantém essa matriz é p_1, então a matriz indica que a última mensagem transmitida possui número de bloco 5 e as últimas mensagens recebidas dos processos p_2, p_3 , p_4 , p_5 e p_6 possuem, respectivamente, número de blocos iguais a 5, 3, 4, 5 e 2. Pode-se notar desse exemplo que as mensagens contidas num mesmo bloco causal são concorrentes e se $m \rightarrow m'$ então m.B < m'.B. Mais detalhes ver [Mac94, EMS95].

	p_1	p_2	p_3	p_4	p_5	p_6
1	+			+		
2		+			+	+
3			+			
4	+			+	+	
5	+	+			+	

Figura 1: Ilustração de uma matriz de blocos para um grupo de 6 processos.

Observando o exemplo da figura 2, notamos que nenhum processo do grupo transmitirá mensagem com número de bloco menor que 3. Desta forma, pode-se entregar todas as mensagens contidas nos blocos de números 1 e 2 (tais blocos são ditos *completos*). O protocolo de ordem total [Mac94, EMS95], assim como os algoritmos de controle de fluxo propostos em [MES96], baseiam-se em blocos completos

2.3 O Algoritmo de Ordenação Causal

Essa seção descreve resumidamente algoritmo do protocolo BCGcausal. Os detalhes de implementação podem ser encontrados em [LM97]. Considere que qualquer mensagem transmitida m, além de carregar seu número de bloco m.B, leva consigo os números de blocos das últimas mensagens entregues à aplicação (*delivered*). Quando m é recebida por um processo, ele pode entregar m imediatamente após entregar todas as mensagens que precedem causalmente m. Para tanto, cada processo $p_i \in g = \{ p_1 ,p_2 ,...,p_n \}$ mantém um vetor de números de blocos, chamado de LDV (*Last Delivered Vector*), cujo tamanho é |g|. O conteúdo de LDV, juntamente com o número de bloco da mensagem constituirão os *timestamps* de todas as mensagens, m, transmitidas (m.ldv e m.b, respectivamente).

Com relação a LDV_i, p_i executará o seguinte. Antes de transmitir uma mensagem m o valor corrente de LDV_i é atribuído a m.ldv. Depois de transmitir m o valor corrente de BC_i é colocado em LDV_i[i]. Depois de entregar m (evento $deliver_i(m)$), o processo p_i faz LDV[m.s] = m.b (m.s representa o identificador do processo que enviou m).

Considere os vetores LDV mantidos pelos processos p_i e p_j, LDV_i e LDV_j , respectivamente. Define-se a seguinte relação ≤ da seguinte forma: $LDV_i \leq LDV_j \Leftrightarrow \forall k: LDV_i [k] \leq LDV_j [k]$. Uma mensagem recebida por p_i, é entregue por p_i imediatamente após a seguinte condição tornar-se verdadeira:

Condição de Entrega: $m.ldv \leq LDV_i$

O vetor LDV tem propriedades similares à *vector clocks* [BSS91]. A diferença é que o segundo mecanismo representa o número de eventos que ocorreram no grupo, enquanto que o primeiro representa as últimas mensagens entregues ao grupo, o que constitui uma representação de causalidade que permite manipular grupos sobrepostos mais facilmente. As propriedades e vantagens podem ser vistas em [Mac94, Mac95].

3. Controle de Fluxo Proposto

O protocolo BCGcausal entrega as mensagens considerando informações sobre entrega das mensagens (LDV). Desta forma, pode haver num mesmo bloco, mensagens entregues e outras ainda não entregues. A fim de diminuir a contenção na transmissão das mensagens, o controle de fluxo deve agir sobre as mensagens entregues à aplicação, desconsiderando, portanto, blocos de causais. A seguir introduziremos algumas definições na seção 3.1 e apresentaremos na seção 3.2 o controle de fluxo proposto.

3.1 O Conceito de *Slot*

Um *slot*, denotado pelo par (B,i), identifica o elemento da matriz de blocos, linha B e coluna i. Na verdade, o bloco BM[B] passa a ser definido como o conjunto de todos os *slots* (B,i), onde $1 \leq i \leq |g|$. As seguintes relações sobre *slots* são definidas. Note que não há relação entre *slots* pertencentes a colunas distintas.

1. (B,i) = (B',j) se e somente se B = B' e i = j.
2. (B,i) < (B',j) se e somente se B < B' e i = j.
3. (B,i) ≤ (B',j) se e somente se (B,i) < (B',j) ou (B,i) = (B',j)

Definição de estabilidade. Um *slot* (B,p) é dito estável num grupo g se é conhecido entregue a todos os processos pertencentes a g. A mensagem contida num *slot* estável também é considerada estável.

Lema 1. Se uma mensagem é considerada estável por um processo de um grupo, então ela será, num tempo finito, considerada estável por todo o grupo.

Prova: Pela definição de estabilidade, se $p_i \in g$, identifica (B,j) como estável, com $p_j \in g$, então p_i sabe que m_j.B foi entregue por todos os membros do grupo, então $B \leq \min(m_k.ldv[j])$, sendo m_k as últimas mensagens recebidas de cada processo do grupo por p_i. Sem perda de generalidade, considere p outro processo de g. Sabe-se, por hipótese, que p já entregou a mensagem m_j.B. Assim, p reconhecerá a estabilidade de (B,j) após receber todas as mensagens que p_i recebeu quando considerou (B,j) estável (ver hipótese de falha – seção 2).

Se um *slot* (B,i) é considerado estável, então todos os *slots* (B',i) < (B,i) serão estáveis. É fácil de verificar que esta propriedade é verdadeira, pois as mensagens são sempre entregues em ordem FIFO entre pares de processos. A seguir apresentaremos o algoritmo de controle de fluxo proposto.

3.2 Algoritmo de Controle de Fluxo

A idéia básica do algoritmo proposto é fazer com que cada processo p_i do grupo observe o estado do *slot* (B,i) e apenas transmita uma mensagem se houver espaço em todos os processos do grupo. Para tanto, um processo deve obedecer a seguinte regra antes de transmitir uma mensagem:

Condicão de transmissão (CT). Um processo p_i envia uma mensagem com número de bloco B se (m.B-N,i) é considerado estável, onde N é o número máximo de blocos na BM.

Ao transmitir uma mensagem obedecendo a condição CT, o processo não provoca estouro de *buffer*, esperando o referido *slot* ficar estável. Para implementar tal procedimento, cada processo do grupo mantém o vetor LSDV (*Last Slot Delivered Vector*). Cada entrada $LSDV_i[j]$ representa o último *slot* (B,i) entregue por p_j e conhecido por p_i. Assim, para verificar a estabilidade do *slot* (B,i), p_i verifica a seguinte relação:

Relação 1: $B\text{-}N \leq \min(LSDV_i)$

O vetor LSDV é atualizado da seguinte forma. Depois de executar *$receive_i(m)$*, o processo p_i faz LSDV[m.s] = m.ldv[i] e depois de executar *$transmitter_i(m)$* o valor de m.b é atribuído a LSDV[i].

Duas mensagens consecutivas, transmitidas pelo mesmo processo não necessariamente possuirão consecutivos números de blocos (ver figura 2). Isto causa uma sub-utilização do espaço da BM, pois existirão *slots* vazios (buracos) que não são levados em consideração pela relação 1. Podemos otimizar o algoritmo de controle de fluxo se cada processo p_i do grupo controlar o número de buracos existentes. De fato, isso pode ser facilmente implementado da seguinte forma. Seja Sv_i o número de buracos acumulados por cada proceso p_i do grupo, (Br, i), tal que (Br,i) < (min(LSDV),i). Assim, antes de transmitir uma mensagem, o processo p_i verifica a relação abaixo:

Relação 2: $B\text{-}N - Sv_i \leq \min(LSDV_i)$

Esse esquema proporciona um algoritmo de controle de fluxo que previne o estouro de *buffer* sem gerar *overhead* adicional no tamanho da mensagem, pois o cálculo de Sv_i pode ser feito apenas por informações locais e o LDV já é usado nas mensagens transmitidas pelo protocolo BCGcausal. Tal abordagem proporciona também a máxima utilização de espaço e menor contenção na transmissão[2].

3.2.1 Prova de Correção

Demostraremos a seguir a correção do protocolo proposto.

Lema 2: Não há estouro de *buffer*.

Prova: Suponha, por contradição, que há estouro do *buffer* na matriz de blocos do processo p_k ao tentar inserir a mensagem m_i, cujo número de bloco é B. Como p_i verificou que $B\text{-}N - Sv_i \leq \min(LSDV_i)$ é certo que (i) o *slot* (B-N,i) é estável, $B\text{-}N \leq \min(LSDV_i)$ (portanto p_k já entregou a mensagem contida em (B-N,i)); ou (ii) (B-N,i) não é estável, mas existirão $Sv_i > 0$ buracos. Em ambas as situações existirá espaço para m_i em p_k.

Lema 3: Não há ocorrência de *deadlock.*

Prova: Suponha, sem perda de generalidade, que p_i ao tentar enviar uma mensagem m_i com número de bloco B, fique bloqueado esperando a Relação 1 (Relação 2) ser verdade. Então há pelo menos um processo $j \neq i$ e $LSDV_i[j] < m_i.B - N$ ($LSDV_i[j] < m_i.B - N - Sv_i$). O processo p_i espera, portanto, receber m_j, cujo $m_j.ldv[i] > B$. Temos que provar portanto que p_j enviará tal mensagem.

[2] Observe que se $B\text{-}N - Sv_i \leq \min(LSDV_i) < B\text{-}N$ poderá ocorrer colisão na BM, o que pode ser contornado por métodos de resolução de colisões em tabelas *hashing*.

Duas situações são possíveis: (i) $LDV_j[i] \geq B - N$, e neste caso, m_j será recebida por p_i (mecanismos de mensagens nulas); (ii) $LDV_j[i] < B - N$ e portanto, $\exists$ m' e m'', tal que m'' $\rightarrow$ m' e m'.B > m.B – N e *deliver$_j$(m')* não ocorreu. Como não há falhas de omissão e as falhas tipo *crash* são tratadas pelo mecanismo de *membership*, podemos afirmar que todas as m'' serão recebidas por p_j e portanto, a Relação 1 (e consequentemente a Relação 2) será verdade. Portanto p_i não ficará bloqueado para sempre.

Observe que o mecanismo de mensagens nulas pode ditar o ritmo com que os processos do grupo vêem a estabilidade de mensagens. Tal mecanismo é, de toda forma, necessário para o protocolo de *membership* [Mac94] e deve ser ajustado para cada aplicação, baseado no nível de atividade do grupo a fim de se obter uma relação de compromisso entre o número de mensagens transmitidas e a velocidade com que falhas e mensagens estáveis são percebidas [Mac94].

5. Comparações com outros Trabalhos

Um estudo sobre a influência do controle de fluxo no desempenho de sistemas de comunicação em grupo foi apresentado em [HR95]. Os algoritmos descritos baseiam-se na extensão do sistema de janelas deslizantes adaptado a comunicação em grupo, onde existem duas abordagens: local e global. No algoritmo local o transmissor controla a janela, preocupando-se apenas com as mensagens por ele enviadas e ainda não reconhecidas pelos demais processos do grupo. Já no algoritmo global, os processos controlam a janela baseados em informações sobre todas as mensagens transmitidas pelo grupo. Os autores demonstram que a abordagem local é mais eficiente do que a global. Essa solução é implementada no Horus [RBM96] na camada FC que funciona da seguinte forma. Quando o número máximo de massagens instáveis chega a um certo limite, a camada FC atrasa a transmissão, acumulando as mensagens localmente. Assim que o número de mensagens instáveis decresce novamente, as mensagens retidas são empacotadas e transmitidas numa única operação.

Em [MW97] é apresentado um estudo onde são propostos dois algoritmos de controle de fluxo, um conservador e outro otimista, baseados num protocolo de ordem total que usa um *token* para ordenar as mensagens, o Pinwheel [CM95]. Os autores argumentam que na presença de falhas o protocolo otimista é menos eficiente que o conservador, aumentando o tempo de entrega de mensagens e o tempo de estabilidade. Essa conclusão pode ser estendida a outros algoritmos de controle de fluxo, tais como [MES96, ADKM92, BSS91, KT91].

Macedo, em [MES96], apresenta três algoritmos de controle de fluxo, um deles atualmente implementado na BCG, fornecendo suporte tanto ao protocolo BCGcausal quanto ao Total. Essas algoritmos de controle de fluxo estão baseados em blocos completos e portanto apresentam uma restrição adicional à entrega de mensagens quando trata-se do protocolo BCGcausal. Macêdo [Mac94, MES96] também apresenta dados relativos ao efeito do controle de fluxo no Newtop, onde vê-se que o controle de fluxo diminui o atraso médio na entrega das mensagens.

Nossa proposta utiliza informações sobre entrega de mensagens, já presentes no protocolo BCGcausal (LDV) para definir as relações de estabilidade de mensagens. Desta forma preveni-se o estouro de *buffers* de processos no protocolo BCGcausal, utilizando portanto a política conservadora. Na classificação apresentada por [HR95] podemos dizer que nossa proposta baseia-se em informação local para conter a transmissão de novas mensagens, pois cada processo observa as mensagens por ele transmitidas e ainda não entregues. A idéia apresentada neste trabalho assemelha-se ao esquema de janelas deslizantes adaptado à comunicação em grupo [HR95], contudo informações no nível do protocolo de ordenação.

5. Conclusões

Controlar o fluxo de mensagens num ambiente distribuído é um problema clássico e não possui solução trivial. Processos enviam e recebem mensagens usando diferentes taxas de transmissão e consumo, fazendo-se necessária a sincronização. Além disso, falhas devem ser consideradas e portanto, as mensagens devem estar disponíveis para possíveis retransmissões, ou seja, as mensagens são mantidas nos *buffers* dos processos e apenas são descartadas se reconhecidas entregues por todos os processos do grupo. O controle de fluxo deve, portanto, usar informações a respeito da estabilidade de mensagens, as quais estão diretamente relacionadas com o protocolo de ordenação.

Em comunicação ponto-a-ponto processos negociam o *buffer* que será usado na transmissão. O transmissor pode assim controlar o fluxo das massagens enviadas mantendo uma janela deslizante sobre o *buffer*. A janela avança a medida que mensagens de *acknowledge* são recebidas. Já em comunicação em grupo, a possibilidade de ocorrência de falhas durante a transmissão requer que os processos do grupo mantenham as mensagens recebidas nos *buffers* até que sejam reconhecidas entregues no grupo, pois retransmissões podem ser necessárias. Isto impede que o esquema de janelas deslizantes tal como concebido para comunicação ponto-a-ponto possa ser usado à comunicação em grupo.

O controle de fluxo exerce um grande efeito sobre os protocolos de comunicação em grupo. Se houver muitas retransmissões de mensagens, que podem ser causadas por falta de espaço nos *buffers* dos processos receptores, haverá além do trabalho perdido, congestionamento na rede. Se por outro lado o algoritmo de controle de fluxo for muito restritivo a fim de evitar estouro de *buffer*, pode haver uma contenção desnecessária na transmissão.

Neste trabalho propusemos uma solução inovadora que utiliza a semântica de ordenação causal para aumentar a eficiência do algoritmo de controle de fluxo para o protocolo BCGcausal. Para tanto, apresentamos o conceito de *slots* na matriz de blocos, sobre o qual definimos estabilidade de mensagens. O algoritmo proposto possui regras simples e não apresenta nenhum custo adicional em relação ao tamanho das mensagens, pois utiliza as relações causais, presentes no cabeçalho das mensagens, que já são usadas pelo protocolo BCGcausal. O controle de fluxo proposto maximiza a utilização do *buffer* e em conseqüência minimiza a contenção na transmissão.

Referências Bibliográficas

[ACM95] G. A. Alvarez, F. Cristian, and S. Mishra.. On-demand Asynchronous Atomic Broadcast. In Proc. Of the Fifth International Conference on Dependable Computing for Critical Applications, Urbana-Champaign, IL, Sep 1995.

[ADKM92] Y. Amir, D. Dolev, S. Kramer and D. Malki. *Transis: A Communication Sub-system for High Availability.* In Proc. Of the 22nd International Sysmposium on Fualt-Tolerant Computing. Pp 76-84, Boston, MA, July 1992.

[BSS91] K. Birman, A. Schiper and P. Stephenson. *Lighweight Causal and Atomic Group Multicast*. ACM Transaction on Computing Systems, (9,3), pp. 272-314, Aug 1991.

[CM95] F. Cristian and S. Mishra. *The Pinwhel Asynchronous Atomic Broadcast Protocols*. In Proc. Of the Second International Sysposium on Autonomous Decentralized Systems, pp 215-221, Phoenix, AZ, Apr 1995.

[EMS95] P. D. Ezhilchelvan, R. A Macêdo and S. K. Shrivastava. *Newtop: A Fault-Tolerant Group Communication Protocol.* Proc. Of the International Conference on Distributed Computing Systems, IEEE Computer Society, pp. 296-306, Canadá , May 30 - Jun 2, 1995.

[HR95] T. M. Hickey and R. van Renesse. *Incorporating System Resource Information into Flow Control*. Technical Report TR95-1498. Dept. of Computer Science, Cornell University.

[Lam78] L. Lamport. *Time, Clocks and Ordering of Events in a Distributed System*. CACM, (21, 7), pp. 558-265, July 1978,.

[LM97] G. M. de A. Lima and R. J. A. Macêdo. *Avaliação de Desempenho do protocolo BCGcausal*. Relatório Técnico Interno do LaSiD, RTI-003/97. Nov 1997.

[KT91] M. F. Kaashoek and ª Tanenbaum. *Group Communicatio in the Amoeba Distributed Operating Systems.* pp 222-230, Arlington, TX, May 1991.

[Mac94] R. J. Macêdo. *Fault Tolerant Group Communication Protocols for Asynchronous Systems*. PhD Thesis, Aug de 1994.

[Mac95] R. A Macêdo. *Causal Order Protocols for Group Communication*. SBRC 95, pp. 265-283, May de 1996.

[MES96] R. A Macêdo, P. D. Ezhilchelvan and S. K. Shrivastava. *Buffer Overflow Avoidance Techniques for Groups Communication Protocols*. SBRC 96, pp. 633-652 1996.

[MW97] S. Mishra e L. Wu. *Flow Control in High Performance Atomic Multicast Services*. Proc. of the International Symposium on High Performance Computing. pp. 295-306, July 1997.

[RBM96] R. van Renesse, K. Birman, and S. Maffeis. *Horus: A Flexible Group Communication System.* CACM, 39(4), pp 76-83, April 1996.

[RHB95] R. van Renesse, T. M. Hickey and K. Birman. *Design and Performance of Horus: A Lightweigth Group Communications System.* Techinical Report TR95-1500. March, 1995.

[VRV93] P. Veríssimo, L. Rodrigues and Werner Vogels. *Group Orientation: a Paradigm for Modern Distributed Systems.* ESPRIT Basic Research Project First Year Report, volume 1, outubro 1993.