Relatório Técnico Interno 007/98

LaSiD

Aspectos de Confiabilidade em Sistemas de Tempo Real Críticos

George Marconi de A. Lima

outubro, 1998

Aspectos de Confiabilidade em Sistemas de Tempo Real Críticos

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

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

Resumo

Os sistemas de tempo real, especialmente os sistemas críticos não podem comportar-se fora do especificado, mesmo na presença de falhas. Tal requisito, sugere a utilização de sistemas distribuídos, que além do aspecto de tolerância a falhas possibilita o aumento da eficiência do sistema além de adaptar-se a distribuição intrínseca dos sistemas de tempo real. Entretanto, os sistemas distribuídos são bastante complexos, pois para prover tolerância a falhas é necessário que os diferentes componentes que o constituem cooperem entre si. Essa cooperação não é trivial de se obter devido à inexistência de tempo absoluto no sistema, o que pode violar a *previsibilidade*, requisito indispensável para os sistemas de tempo real críticos. Esse trabalho fornece uma visão geral do problema, resumindo diversos conceitos relativos a tolerância a falhas, confiabilidade, sistemas distribuídos, comunicação em grupo e sistemas de tempo real, através de uma extensa revisão bibliográfica.

1. Introdução e Motivação

Antigamente os computadores se limitavam ao processamento de dados. Eram usados como máquinas de calcular velozes ou para o processamento comercial. A rápida evolução tecnológica, no entanto, permitiu a inserção do computador nas mais variadas áreas proporcionando benefícios outrora inimagináveis.

Esse movimento proporcionou comodidade aos usuários, os quais são deslocados para um trabalho mais criativo, deixando os trabalhos repetitivos ou insalubres para a máquina. Consequentemente, há um aumento imediato da produtividade das empresas que usam sistemas informatizados e uma expressiva diminuição dos custos de produção. Entretanto, essa crescente automação e/ou informatização deixa-nos fortemente dependentes da máquina. Uma falha num computador de uma indústria pode causar prejuízos de dimensões assustadoras.

Como exemplos de sistemas críticos podemos citar os sistemas de tempo real. Eles se caracterizam por interagir com eventos do mundo real e possuir restrições temporais ao computar e responder a esses eventos. Por exemplo, um sistema de controle de bordo de um avião deve responder aos eventos num limite máximo de tempo sob pena de colocar vidas humanas em perigo. Tais sistemas idealmente devem ser, portanto, imunes a falhas.

A preocupação com falhas, no entanto, não está restrita aos sistemas de tempo real[1]. Qualquer sistema de computação deve prever a possibilidade de ocorrência de falhas e estabelecer mecanismos de prevenção ou recuperação dos possíveis defeitos que possam vir a ser causados por falhas. Deseja-se, portanto, que computadores sejam cada vez mais confiáveis, ou seja, que o funcionamento dos sistemas seja exatamente o que foi especificado, mesmo com a ocorrência de falhas.

É sabido que quanto maior o grau de redundância de um sistema maior é a tolerância deste à ocorrência de falhas. Por exemplo, se o sistema quer se prevenir contra falhas num disco magnético, manter dois discos, ambos com o mesmo conteúdo é uma boa estratégia. Assim, construir um sistema que implemente redundância parece ser a alternativa para manter sua confiabilidade. Neste sentido, sistemas distribuídos constituem uma caminho natural, pois são constituídos de vários componentes independentes, geograficamente distribuídos, portanto, se um dos componentes falhar os outros poderão continuar funcionando.

Nos últimos anos muitos resultados tem sido publicados na área de sistemas distribuídos. Mais especificamente, resultados importantes tem sido publicados no que se refere a tolerância a falhas[2], podemos citar [ACM95, ADKM92, BSS91, CM95, Mac94, RBM96, VA95, RBM96] entre outros. O objetivo deste texto é introduzir aspectos relativos ao uso de sistemas distribuídos no aumento da confiabilidade dos sistemas de tempo real críticos.

Este documento apresenta a seguir, com um pouco mais de detalhes, os principais conceitos envolvidos com o tema. O texto é organizado da seguinte forma: a seção 2 apresenta conceitos de sistemas de tempo real. A seção 3 resume alguns conceitos sobre falhas e a seção 4 descreve os sistemas distribuídos. Por fim, a seção 5 conclui esse documento.

2. Sistemas de Tempo Real

Sistemas de tempo real são aqueles que possuem requisitos de natureza temporal. Em tais sistemas não basta obter valores logicamente corretos, os valores devem ser gerados no momento certo (tempo real). Falhas de natureza temporal podem ser críticas em alguns casos.

Pode-se classificar os sistema de tempo real em críticos[3] e não-críticos[4]. Os primeiros possuem requisitos temporais rígidos, pois se uma tarefa não for executada até um dado limite de tempo (*deadline)* comprometerá a integridade do sistema. Por outro lado, os sistemas não-críticos, apesar de possuir requisitos temporais, se uma dada tarefa não for executada até seu *deadline* a integridade do sistema não é comprometida. Na prática os sistemas são híbridos [AB90], pois algumas de suas tarefas são críticas outras não.

---

[1] Vale ressaltar, contudo, que sistemas de tempo real críticos apresentam custos altíssimos se seus serviços fogem da especificação original.

[2] Usamos o termo tolerância a falhas como tradução do inglês *faut tolerance*. Sabe-se que uma *falha* pode provocar um *erro* que por sua vez chega ao usuário sob a forma de *defeito*. Tolerar falhas é, portanto, evitar os defeitos.

[3] Tradução do inglês *hard real time systems.*

[4] Tradução do inglês *soft real time systems.*

Visto as necessidades temporais e a preocupação com a ocorrência de falhas, conclui-se que é necessário conceber um modelo computacional adequado a ambos os aspectos. A busca de tal modelo tem sido tema de recentes publicações [VA95, CF97].

2.1 Escalonamento de tarefas

Algoritmos de escalonamento de tarefas são cruciais para a cumprimento dos requisitos temporais. Em sistemas tradicionais as execuções dos processos são intercaladas baseados em prioridades [Tan92]. Tal abordagem é inadequada para sistemas de tempo real, pois não leva em consideração as restrições temporais.

Um requisito importante para sistemas de tempo real críticos é a resposta a tarefas esporádicas dentro de um *deadline* específico. Por exemplo, um *air-bag* de um automóvel deve inflar em poucos milisegundos depois que o carro se choca. Pode-se notar com o exempo que a resposta do sistema é curta quando comparada com a taxa que o evento esporádico ocorre (no caso, apenas uma vez). Além das tarefas esporádicas, o sistema responde a eventos periódicos, os quais também possuem seus *deadlines*. Sendo assim, é essencial que o sistema possua completa previsibilidade, ou seja, todos os eventos devem ter sido anteriormente previstos e o sistema deve ter sido projetado para que todas as garantias de prazos sejam satisfeitas.

Pode-se categorizar uma abordagem para o planejamento de um sistema de tempo-real crítico da seguinte forma [Tindell93]: *configuração, escalonamento* e *análise.* A configuração é a atividade realizada *a priori* responsável por atribuir as características do sistema. O escalonamento é o procedimento que faz decisões em tempo de execução baseado em algum algoritmo. A análise usa o conhecimento a respeito do escalonamento e detalhes da configuração para determinar se os prazos especificados para as tarefas serão garantidos. Observe que o escalonamento de tarefas constitui um ponto crucial para a garantias dos *deadlines* das tarefas.

Os algoritmos de escalonamento podem ser divididos em estáticos e dinâmicos [ABTW95]. Os primeiros baseiam-se no conhecimento *a priori* das características dos processos e predeterminam assim o escalonamento (agenda de execução). Tal abordagem sofre limitações, pois não atende a requisitos de eventos esporádicos.

Os algoritmos dinâmicos, por sua vez, são capazes de produzir um escalonamento em tempo de execução sem conhecimento prévio dos processos. São baseados no conceito de prioridades associadas às tarefas (urgência na execução). A regra geral é atribuir altas prioridades às tarefas com *deadlines* curtos. Observa-se nesta abordagem que transferiu-se algum trabalho da atividade de *configuração* para o *escalonamento*. Sendo assim, o escalonamento dinâmico é mais poderoso que o estático. Entretanto, a atividade de análise fica mais difícil.

Se a decisão sobre o escalonamento é feita em tempo de execução diz-se que o algoritmo é *online* (prioridades dinâmicas, tais como [35 em Tindell93]), caso contrário são chamados de *offline* (prioridades fixas, tais como [Tindell93]).

Outra característica dos algoritmos de escalonamento é a possibilidade de haver preempção durante a execução de tarefas. Muitas vezes uma tarefa não pode ser simplesmente interrompida, pois pode estar executando um conjunto atômico de operações. Todas essas características exercem influências no tipo de serviço suportado pelo sistema: se periódico ou esporádico; se um determinado escalonamento é exeqüível ou não etc.

Considerações importantes devem ser feitas com relação ao escalonamento de tarefas em sistemas distribuídos. Mok e Dertouzos [MD88] mostraram que algoritmos ótimos em sistemas uniprocessados não são ótimos em sistemas multiprocessados. A complexidade de tais sistemas aumenta ainda mais devido a atrasos na comunicação, ocorrência de falhas das mensagens e o fato de que os canais de comunicação baseados na semântica FIFO não serem adequados aos sistemas de tempo real [AB90, ABTW95, TBW95].

3 Tolerância a Falhas

Para se projetar um sistema de computação confiável é necessário que estejam bem definidas quais tipos de falhas que podem ocorrer e com isso estabelecer estratégias a fim tolerá-las. Geralmente, o tipo de falha (ou *modelo de falhas*) suportado pelo sistema influirá diretamente em sua eficiência e complexidade. A figura 1[5] abaixo ilustra os vários tipos de falhas e a relação entre os mesmos.

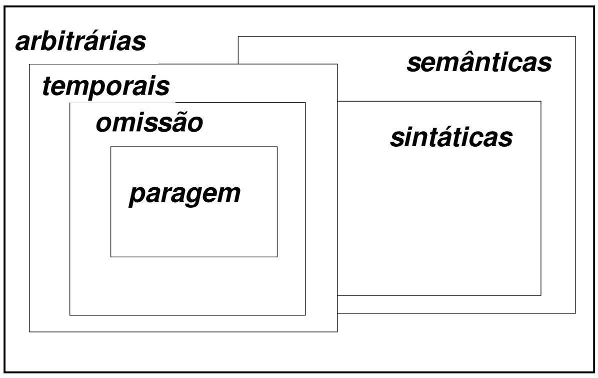

Figura 1. Tipos de falhas.

As falhas por paragem assumem que o componente falho não mais voltará a funcionar. Essa é uma hipótese, muitas vezes, considerada forte. Por exemplo, se alguma mensagem não chega a seu destino pode ser por alguma falha esporádica e o processo que a enviou poderá continuar a fazê-lo. Esse tipo de falha é fácil de ser detectada, o que simplifica a construção dos sistemas. A categoria de falha omissiva é um pouco mais genérica que a anterior (no exemplo dado, perdas de algumas mensagens na transmissão). As falhas temporais[6] são ainda mais genéricas que as omissivas. Nessa categoria estão as falhas em que a computação pode atrasar ou se adiantar no tempo e são ainda mais difíceis de serem detectadas. Observe que as

---

[5] Figura retirada do curso de sistemas distribuídos confiáveis ministrado pelo Prof. Dr. Paulo Veríssimo (Universidade de Lisboa) e disponibilizada pelo mesmo.

[6] Observe a importância de detecção de falhas temporais em sistemas de tempo real. Tanto tarefas antecipadas quanto atrasadas podem comprometer a previsibilidade do sistema.

falhas por paragem são as omissivas onde o tempo de omissão é infinito e essas são as falhas temporais onde o tempo de atraso é infinito em algumas computações do sistema.

Por outro lado podemos classificar as falhas quanto ao valor produzido pela computação. Elas podem ser sintáticas ou semânticas. As primeiras são fáceis de serem detectadas. Por exemplo, o termômetro indicando que a temperatura de um forno ligado é de 100ºC negativos. Mas, se a temperatura correta fosse 103ºC e o termômetro indicasse 105ºC teríamos uma falha do tipo semântica. Todos os tipos citados pertencem a classe de falhas arbitrárias, como ilustrado na figura 1.

As falhas se manifestam através de erros. Assim, são necessárias estratégias para detectá-las, tratá-las e manter o sistema funcionando. Podemos citar as seguintes estratégias [Pra96,Tise]:

- Verificação da ocorrência de erros - *detecção* de erros. Procedimentos devem ser previstos para detectar se houve algum erro na computação. Por exemplo, se uma mensagem chegou corrompida ao seu destino ou se o valor de um dado não foi escrito corretamente no disco.
- Correção dinâmica dos erros – *mascaramento*. Alguns tipos de erros podem ser mascarados. Por exemplo, se for possível detectar quais os bits de uma mensagem que foram trocados durante a transmissão é possível recuperar a mensagem original. Com isso evita-se que o defeito seja exposto ao sistema.
- Prevenção da propagação de erros - *confinamento de erros.*
- Identificação do módulo em falha - *diagnóstico*
- Eliminação/substituição do componente falho – *reparo.*
- Correção do sistema a um estado aceitável – recuperação.

A palavra chave em tolerância a falhas é redundância. Há vários tipos de redundância [Pra95]: de *hardware,* onde vários componentes executam a mesma função, precavendo-se contra falhas de algum dos componentes; de *software*, onde código extra é adicionado ao sistema para a detecção de possíveis erros; de *informação*, que mantém informação extra para possibilitar a recuperação/detecção de erros; e de *tempo*, baseada na repetição da computação. Como esses tipos de redundância são naturalmente concebidos nos sistemas distribuídos, eles proporcionam um ambiente adequado à tolerância a falhas.

5. Sistemas Distribuídos

Sistemas distribuídos se caracterizam por serem constituídos de componentes, dispersos geograficamente e interconectados por uma rede de comunicação. A única forma de comunicação é através de troca de mensagens através da rede. Tal modelo é adequado para a construção de sistemas confiáveis, visto que seus componentes são independentes, suportando a implementação de redundância. Pode-se por exemplo manter várias cópias do banco de dados em cada um dos seus N componentes, tolerando assim N-1 falhas por paragem. Entretanto, manter as cópias consistentes exige um esforço do sistema para coordenar os componentes sem diminuir sua confiabilidade.

Prover essa cooperação não é tarefa trivial devido a inexistência de um sistema de tempo absoluto e das incertezas quanto à transmissão de mensagens. Esses fatores influenciam a concepção de sistemas de tempo real críticos, visto que esses não podem exibir um comportamento não previsto por causa de falhas. No entanto, o uso de sistemas distribuídos para construir sistemas de tempo real críticos constitui uma alternativa justificável, pois além do aspecto de tolerância a falhas, podemos citar o aumento do desempenho devido ao potencial paralelismo de execução e a disposição dos eventos/componentes do sistema, que geralmente, exigem o processamento de tarefas distribuídas [Tindell93].

4.1 Tempo num Sistema Distribuído

Ordenamos os eventos no mundo real fundamentados num conceito de tempo absoluto. Podemos aplicar tal paradigma em sistemas computacionais centralizados e assim, conseguimos estabelecer uma ordem absoluta para os eventos, baseada num relógio único. No entanto, em sistemas distribuídos[7] qualquer troca de informação entre diferentes participantes (processos) é feita através de troca de mensagem, cujo tempo de trânsito entre um ponto e outro é indeterminado. Isso impossibilita a concepção de um relógio único e obriga-nos a pensar em vários *relógios* distribuídos, impedindo que a visão absoluta de tempo seja aplicada. Essa característica nos leva a observar os eventos sob uma visão *relativista de tempo*, onde cada processo possui uma visão particular da computação.

Neste sentido, um importante contribuição foi apresentada por Lamport [Lam78] que introduziu o conceito de relógio lógico, com o qual é possível estabelecer uma ordem de *causa e efeito* (relação "*happened before*":$\rightarrow$) dos eventos distribuídos. Tal conceito permite a construção de mecanismos para garantir que processos possam observar diferentes ordens da computação, preservando, contudo, a *causalidade* entre as mensagens.

4.1.1 Modelos de Sistemas Distribuídos

Sincronismo, uma propriedade fundamental em sistemas distribuídos, divide os sistemas em dois modelos extremos: síncrono e assíncrono. No modelo assíncrono nada é assumido a respeito do tempo no sistema, o tempo máximo de trânsito de mensagens é indeterminado e não há sincronismo entre os relógios de cada componente. Por outro lado, o

[7]Não estamos nos referindo aos sistemas *fortemente acoplados,* onde os diversos processadores residem na mesma máquina.

modelo síncrono assume várias hipóteses a respeito do tempo no sistema, tais como, desvio máximo entre os relógios dos componentes, tempo máximo de transmissão de uma mensagem pela rede etc. Para que este sistema funcione corretamente, nenhuma das hipóteses deve ser violada.

No modelo assíncrono, quando o sistema está sujeito a falhas, pode não haver consenso [FLP85]. Por outro lado, no modelo síncrono a construção de aplicações distribuídas é mais simples, pois as hipóteses fortes fornecem um comportamento previsível ao sistema. Entretanto, essas mesmas hipóteses reduzem a aplicabilidade do modelo, pois nem sempre são factíveis. Esse impasse entre a previsibilidade e a aplicabilidade fez surgir modelos intermediários que adicionam algumas hipóteses de tempo ao modelo assíncrono *timed-asyncronous* [CF97], ou relaxam as hipóteses do modelo síncrono, *quasi-syncronous* [VA95]. De fato, devido a previsibilidade ser um aspecto imprescindível para a construção de sistemas de tempo real críticos, os modelos síncrono ou quasi-syncronous se adeqüam a tal propósito.

4.2 Comunicação em Grupo

A fim de contornar as dificuldades de construção de aplicações distribuídas confiáveis devido às complexidades já mencionadas, surgiu o paradigma de comunicação em grupo. Em tal paradigma os processos são agrupados em conjuntos, chamados de grupos e toda a comunicação envolve todos os elementos do grupo (implementação de redundância). Essa abstração é fortemente recomendada por diversos autores [MR93, Mac94, EMS94, PBS89, BSS91, PRV93, CV95].

Um dos serviços providos pelos protocolos de comunicação em grupo tem a finalidade de *ordenar* os eventos percebidos pelo grupo. Todos os elementos do grupo têm que perceber uma ordem consistente dos eventos. Em [BJ87] é apresentado o conceito de *ordenação causal*, que contempla tanto a causalidade entre as mensagens quanto o conceito de grupos. Protocolos baseados em tal conceito fornecem suporte a diversos tipos de aplicações (CSCW, banco de dados replicados etc.). Se além de não violar a ordem causal, todos os processos observarem a mesma ordem dos eventos, diz-se que a ordenação de mensagens é *total* [Mac94].

No paradigma de 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 em 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* [Mac94, MES96, ACM95, BSS91, HR95, MW97, RBM96, LM98] e são essenciais para a construção de protocolos de comunicação em grupo tolerante a falhas.

A parte do protocolo que implementa a tolerância a falhas é o *serviço de membership*. Esse serviço é um módulo extremamente importante de diversos sistemas distribuídos tolerantes a falhas: Newtop [EMS95], Relacs [BDGB94], Transis [ADKM92], Amoeba

[KT91], Isis [Bir93], Horus [RBC+93]. Em linhas gerais, o *membership* controla a entrada (*join*) e a saída (*leave*) de processos no grupo. Para decidir sobre a entrada ou saída de um processo do grupo, o *membership* precisa estabelecer um *consenso* entre os demais processos. Ou seja, todos os demais processos devem concordar que determinado processo deve ser retirado. Sabe-se, entretanto, que o problema do consenso não tem solução num sistema assíncrono, mesmo se um único processo falha e esta falha se dá por *paragem* [FLP85]. Isso ocorre devido à impossibilidade de se distinguir entre um processo lento e um processo falho.

Nos últimos tempos diversos protocolos de comunicação em grupo foram sugeridos. Alguns são próprios para sistemas síncronos [BSD88, Fis83, LSP82, SD83]. Outros são adequados para sistemas assíncronos [ADKM92, CM84, M-SMA90, PBS89, RV91]. Outros, mais recentes, fornecem suporte à sobreposição de grupos [BSS91, EMS95, G-M91, MR93].

6. Conclusão

Os sistemas de tempo real, especialmente os sistemas críticos, os quais possuem restrições temporais rígidas não podem comportar-se fora do especificado, mesmo na presença de falhas. Em outras palavras, se alguma falha ocorrer o sistema deve ser projetado para tolerá-las evitando que o defeito seja percebido pela aplicação. Tal requisito, sugere a utilização de sistemas distribuídos, que além do aspecto de tolerância a falhas possibilita o aumento da eficiência do sistema além de adaptar-se a distribuição intrínseca dos sistemas de tempo real.

Entretanto, os sistemas distribuídos são bastante complexos, pois para prover tolerância a falhas é necessário que os diferentes componentes que o constituem cooperem entre si. Essa cooperação não é trivial de se obter devido à inexistência de tempo absoluto no sistema, o que pode violar a *previsibilidade* (mesmo na presença de falhas), requisito indispensável para os sistemas de tempo real críticos.

Construir um sistema de tempo real crítico usando um modelo distribuído é, na verdade, solucionar o problema contemplando aspectos contraditórios: previsibilidade e incertezas; possibilidade de falhas e confiabilidade. Essas soluções possuem grande importância prática, devido a nossa crescente dependência do computador.

Esse texto forneceu uma visão geral do problema, resumindo diversos conceitos relativos a tolerância a falhas, confiabilidade, sistemas distribuídos, comunicação em grupo e sistemas de tempo real, através de uma extensa revisão bibliográfica. Acreditamos assim que esse trabalho possa servir para motivar aqueles que gostariam de investigar mais profundamente esse assunto, servindo como um ponto de partida.

Bibliografia

[ABDTW95] N.C. Audsley, A. Burns, R.I. Davis, K.W. Tindell and A.J. Wellings, Real-Time System Scheduling, in Predictably Dependable Computing Systems, eds B. Randell, J.-C. Laprie, H. Kopetz and B. Littlewood, ESPRIT Basic Research Series, Springer, pp41-52, 1995.

[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.

[BDGB94] Ö. Babaoglu, R. Davoli, L-A. Giachini, M. G. Baker. *RELACS: a communication infrastructure for constructing reliable applications in large-scale distributred systems.* BROADCAST project deliverable report, 1994. University of Newcastle upon Tyne, UK.

[Bir93] K. Birman. *The Process Group Approach to Reliable Distributed Computing*. Communications ot the ACM, Vol. 9, No. 12. pp. 36-53, dezembro de 1993.

[BSD88] O. Babaoglu, P. Stephenson e R. Drummond. *Reliable Broadcasts and Communication Models: Tradeoffs and Lower bounds*. Distributed Computing. pp. 177-189, Springer-Verlag. 1988.

[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.

[CF95a] Flaviu Cristian and Frank Schmuck. *Agreeing on Processor Group Membership in Asynchronous Distributed Systems.* UCSD TR CSE95-428, 1995.

[CF95b] Flaviu Cristian and Frank Schmuck. *On the Possibility of Consensus in Asynchronous Systems.* UCSD TR CSE95-428, 1995.

[CM84] J. Chang, N. F. Maxemchuk. *Reliable Broadcast Protocolos*. ACM Transactions on Computer Systems. Vol. 2, No. 3, pp. 251-273, agosto de 1984.

[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.

[CF97] Flaviu Cristian, Christof Fetzer. *The Timed Asynchronous System Model* UCSD Technical Report CSE97-519, Jan,1997.

[FC95] Christof Fetzer and Flaviu Cristian. Pacific Rim Int'l Symp. *On Fault-Tolerant Systems*. Dec, 1995.

[FLP85] M. Fischer N. Lynch, M. Peterson. *Impossibility of Distributed Consensus with One Faulty Process*. J.ACM, 32, pp. 374-382, abril 1985.

[G-M91] H. Garcia-Molina. *Ordered and Reliable Multicast Communication*. ACM Transactions on Computer Systems. Vol. 9, No. 3, pp. 242-271, agosto de 1991.

[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.

[LM98] G. M. de A. Lima and R. J. A. Macêdo. *Uma Proposta de Controle de Fluxo para o Protocolo BCGcausal*. Relatório Técnico Interno do LaSiD, RTI-004/98. 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.

[MR93] A. Mostefaoui, M. Raynal. *Causal Multicasts in Overlapping Groups: Towards a Low Cost Approach*. Broadcast deliverable report, Vol I, First Open Broadcast Workshop, Newcastle, outubro de 1993.

[M-SMA90] M. P. Melliar-Smith, L. E. Moser, V. Agarwala. *Membership Algorithms for Asynchronous Distributed Systems*. proc. of the 12th International Conference on Distributed Com. Systems. pp. 480-488, maio de 1991.

[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.

[Pra95] D. K. Pradhan. *Fault-Tolerant Computer System Design*. Prentice Hall PTR, 1996.

[PBS89] L. Peterson, N. Bucholz, R. Schlichting. *Preserving and Using Context Information in Interprocess Communication.* ACM Transactions on Computer Systems. Vol. 7, No. 3, pp. 217-246, agosto de 1989.

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

[RBC+93] R. Renesse, K. Birman, R. Cooper, B. Glade, P. Stephenson. The Horus System. In K. Birman e R. Renesse, editores, *Reliable Distributed Computing with the Isis Toolkit*, pp. 133-147. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[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.

[RV91] L. Rodrigues, P. Veríssimo. *xAMp: A Multi-Primitive Group Communication Service*. Proceedings of the 11th Symposium on Reliable Distributed Systems, Houston, Texas, 1991.

[Tan92] A. S. Tanenbaum. *Modern Operating Systems*. Pentrice-Hall Inc. 1992.

[TBW95] K.W. Tindell, A. Burns and A.J. Wellings, *Analysis of Hard Real-Time Communications*, Real-Time Systems, Vol 9, No 2, pp147-171, 1995

[Tin93] K. W. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems*. PhD tesis, University of York, 1993.

[VA95] P. Veríssimo and C. Almeida. *Quasi-Syncronism: A Step Away from Traditional Fault-Tolerant Real-Time System Models*. IEEE TCOS Bulletin, 04(7), December 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.