Um Simulador Distribuído Baseado no Paradigma de Eventos Condicionais

Luis Carlos A. P. Quintela[1]

Valmir C. Barbosa[2]

Resumo

Este artigo descreve o projeto, a implementação e uma avaliação experimental de um simulador distribuído de eventos discretos baseado no paradigma de eventos condicionais de Chandy e Sherman. O ponto central do projeto do simulador foi a total separação das funções pertinentes ao simulador daquelas pertinentes à aplicação sendo simulada, permitindo assim o tratamento de uma vasta classe de problemas. A implementação foi realizada em Occam2 sobre um hipercubo de *transputers* com oito processadores. Resultados de uma avaliação experimental sobre a simulação de uma rede fechada de filas em configurações variadas indicam excelentes valores de aceleração em diversas situações, e em geral sugerem uma forte dependência sobre a capacidade de *lookahead* da simulação.

Abstract

In this paper we describe the design, the implementation, and an experimental evaluation of a discrete-event distributed simulator based on Chandy and Sherman's conditional-event paradigm. The key point in the design of the simulator has been the complete separation between simulator-related functions and those pertaining to the particular application being simulated, thereby allowing the treatment of a vast class of problems. The simulator was implemented in Occam2 on an eight-node transputer hypercube. Results on an experimental evaluation of the simulator's performance on closed queueing networks of various configurations indicate excellent speedups in several situations, and in general suggest a strong dependency upon the lookahead capability of the simulation.

[1]Tecnologia Bancária S.A., Av. das Nações Unidas, 13797, bloco 2, 04794-000 São Paulo - SP. Participou deste trabalho quando aluno de mestrado da COPPE/UFRJ.

[2]Centro Científico Rio, IBM, Caixa Postal 4624, 20001-970 Rio de Janeiro - RJ.

1 Introdução

Este artigo apresenta o projeto e a implementação de um simulador distribuído de eventos discretos e medidas do seu desempenho numa máquina paralela de memória distribuída e topologia hipercúbica. O simulador descrito possibilita a criação de aplicações para a simulação do comportamento de sistemas reais que possam ser modelados por conjuntos de processos físicos independentes, conforme a definição de sistemas físicos encontrada em [M86]. Cada processo físico representa um componente do sistema real e a interação entre eles é realizada através da troca de mensagens.

A garantia da não ocorrência de erros causais e da progressão da simulação livre de *deadlocks* é conseguida pelo simulador pela aplicação do modelo conservador por abordagem de eventos condicionais, proposto por Chandy e Sherman em [CS89]. Este paradigma baseia-se na transformação de eventos condicionais — que podem não ocorrer, dependendo do escalonamento de eventos em outros processos físicos — em definitivos — seguros para processamento, pois ocorrerão independentemente da computação alheia ao processo que os escalonou. Por ser conservador, o paradigma em questão só processa eventos quando da certeza de que este processamento não levará a simulação a um estado no qual se configure um erro causal, em oposição aos modelos otimistas, que deixam a simulação progredir livremente para, posteriormente, estornar os eventuais erros de causalidade.

Em [CM79] pode ser encontrada a descrição do modelo clássico conservador de simulação distribuída, apoiado no envio de mensagens nulas — que não representam eventos, mas dão expectativas dos tempos associados a mensagens que chegarão no futuro — para a prevenção de *deadlock*. O grande trunfo do modelo de eventos condicionais está em evitar o envio de tais mensagens, provocando diminuição no tráfego da rede de processadores e, conseqüentemente, melhorando o desempenho da simulação.

O estudo aqui discutido do paradigma conservador é baseado em um projeto que promove algumas modificações no algoritmo original, possíveis pelo isolamento de detalhes do paradigma da aplicação. Esta se atém tão somente à simulação do sistema físico, deixando ao simulador as tarefas de prevenção de *deadlock*, progressão da simulação e garantia da não ocorrência de erros causais, controles não triviais em ambientes distribuídos de processamento.

Uma rede de filas parametrizada foi utilizada com o intuito de avaliar o desempenho do simulador. Os resultados obtidos apontam para a alta dependência do paradigma de eventos condicionais da capacidade da aplicação de simulação em explorar o *lookahead* intrínseco ao sistema físico simulado.

O artigo contém quatro seções adicionais. A seção 2 traz uma descrição do paradigma de simulação distribuída utilizado, enquanto na seção 3 são descritos o projeto e a implementação realizados. Resultados e conclusões são apresentados, respectivamente, nas seções 4 e 5.

2 O modelo de eventos condicionais

Considere um conjunto de procesos lógicos (PLs) representando os processos físicos de um sistema físico. Espera-se de um PL que ele produza uma seqüência $(t_0, m_0), \ldots, (t_{n-1}, m_{n-1})$,

onde $m_0, \ldots, m_{n-1}$ são as mensagens enviadas pelo processo físico correspondente nos tempos $t_0, \ldots, t_{n-1}$, com $t_0 \leq \cdots \leq t_{n-1}$ [M86]. As mensagens representam eventos a serem processados nos processos lógicos de destino destas, nos tempos associados. Cada PL deve possuir ainda um grupo de variáveis para armazenar o seu estado na simulação. A este conjunto de processos lógicos dá-se o nome de sistema lógico. Dada esta definição, passa-se à descrição do algoritmo de simulação por abordagem de eventos condicionais.

Cada PL_y deve manter um contador n_r para cada porta[3] r de entrada e um contador n_s para cada porta s de saída. Estes valores informam respectivamente o número de mensagens recebidas e enviadas pela porta correspondente. Além disso, um PL_y deve manter um valor $next_y$ que corresponde ao tempo do próximo evento condicional a ser enviado por PL_y. Em tempos arbitrários, um PL_y grava $next_y$ e os contadores n_r e n_s — com a restrição de que a gravação seja feita num tempo finito após a mudança dos valores — atomicamente, no sentido em que os valores não podem mudar durante a gravação. Também em tempos arbitrários, um PL_y irradia, para os outros processos lógicos do sistema, os valores gravados — com a restrição de que os valores sejam irradiados num tempo finito após a gravação. Um vetor $C_x[y]$ é utilizado por todo PL_x do sistema para o armazenamento dos valores $next_y$ recebidos. Em acréscimo a este vetor, um PL_y possui ainda dois outros, $D_y[r]$ e $D_y[s]$, que conterão, respectivamente, os valores n_r e n_s recebidos, para todas as portas r e s do sistema. É da responsabilidade de um PL_x garantir que $C_x[x] = next_x$, $D_x[r] = n_r$, para toda porta r de entrada de PL_x, e $D_x[s] = n_s$, para toda porta s de saída de PL_x, de forma que um processo lógico não envie mensagens com estes valores para si mesmo.

Dado um evento condicional com rótulo de tempo $next_x$, a ser enviado caso PL_x não receba nenhum evento por uma porta r de entrada com rótulo de tempo maior que o rótulo do último evento recebido por esta porta, este será convertido num evento definitivo caso valham

$$next_x = \min_y C_x[y] \tag{1}$$

e, para todas as portas r e s conectadas,

$$D_x[r] = D_x[s]. \tag{2}$$

Os processos lógicos progridem escalonando eventos em função das mensagens recebidas. Alguns eventos escalonados podem ser considerados definitivos através da consideração de características específicas do sistema físico simulado (como a certeza, por exemplo, no contexto de um servidor de *jobs* de diferentes prioridades, de que um *job* da mais alta prioridade possível não será jamais "preemptado" do servidor). Os eventos condicionais serão transformados em definitivos pela aplicação das equações 1 e 2. Somente os eventos definitivos serão enviados para os processos lógicos onde deverão ocorrer.

[3] Considerando que os processos lógicos se conectam por portas pelas quais trocam mensagens.

A prova de corretude do algoritmo está relacionada com a determinação de estados globais em sistemas distribuídos, conforme discutido em [CL85]. No artigo, Chandy e Lamport definem um estado global de um sistema distribuído como sendo um conjunto consistente de estados dos processos e canais que compõem o sistema. Considerando o estado de um canal como o conjunto de mensagens em trânsito por este canal, os autores provam que um conjunto de estados de processos e canais é consistente quando, entre a gravação do estado de um processo p e a gravação do estado de um canal c incidente a p e orientado para fora, não há o envio de nenhuma mensagem por p em c, para todo processo p do sistema e todo canal c de p orientado para fora.

No algoritmo assíncrono por abordagem de eventos condicionais, o estado de um PL_x é composto pelo seu valor $next_x$. Se considerarmos um canal como sendo a união entre uma porta s de saída e uma porta r de entrada, então o seu estado é um conjunto de mensagens que corresponde aos contadores n_r e n_s, mantidos, respectivamente, pelos processos de origem e destino do canal (deve ser observado, todavia, que este par de valores não é o estado do canal propriamente, definido como o conjunto de mensagens em trânsito).

O algoritmo de simulação, ao gravar para PL_y, de tempos em tempos, os valores de $next_y$, n_r e n_s para todas as portas r e s incidentes a PL_y, não está de fato executando nenhum algoritmo de gravação de estado global. Assim, apesar de a gravação desses valores ter sido realizada de maneira atômica, as cópias armazenadas por um PL_x em $C_x[y]$, $D_x[r]$ e $D_x[s]$ não são necessariamente consistentes, pois pode ocorrer que $D_x[r] > D_x[s]$ para algum canal, o que é evidentemente inconsistente (a atomicidade na gravação de $next_y$, n_r e n_s implica apenas em que os valores de $C_x[y]$ e $D_x[s]$ sejam consistentes entre si). Mesmo nos casos em que $D_x[r] \leq D_x[s]$ para todos os canais, as cópias guardadas por PL_x não constituem propriamente, como já vimos, um estado global, dado que as mensagens em trânsito não são conhecidas. Uma exceção é o caso particular em que $D_x[r] = D_x[s]$ para todos os canais, nos quais então não há mensagens em trânsito, daí a inclusão da equação 2, que garante então que a equação 1 está de fato sendo aplicada sobre um estado global armazenado em PL_x.

3 O simulador

O modelo por abordagem de eventos condicionais está encapsulado no simulador, o que permite que os processos lógicos se atenham exclusivamente à simulação do sistema físico.

O simulador foi implementado na linguagem de programação Occam2 [B88] para uma máquina paralela distribuída de oito nós com topologia de hipercubo, desenvolvida pela COPPE/UFRJ [ACSC91]. Cada nó é composto por um processador T800 e um módulo de memória local, não acessível por outros nós. A comunicação entre os processadores é feita exclusivamente por canais físicos que os interconectam de acordo com a topologia citada.

O simulador tem uma estrutura padrão, composta de dois módulos — de comunicação e de gerência — que é replicada em todos os processadores da máquina distribuída. Os processos lógicos são distribuídos entre os processadores, sendo o balanceamento de carga estático e de responsabilidade única do programador do sistema lógico.

O módulo de comunicação faz a ligação entre os módulos de gerência da simulação, que executam em diferentes processadores. Todas as mensagens que fluem de um módulo de gerência (ou gerente, mais sucintamente) para outro — sejam estas mensagens eventos para serem processados em processos lógicos externos ou mensagens de controle da simulação que implementam o modelo por abordagem de eventos condicionais — passam pelos módulos de comunicação correspondentes aos gerentes envolvidos. Estes módulos cuidam para que as mensagens cheguem ao seu destino. Caso não haja um canal físico direto, conectando dois processadores cujos módulos de gerência trocam mensagens, os módulos de comunicação se encarregarão do roteamento destas mensagens por nós intermediários. O processo central da estrutura de um módulo de comunicação é o processador de comunicação, desenvolvido por Lúcia Drummond como tese de mestrado da COPPE em 1990 [D90].

A tarefa de um gerente da simulação é receber eventos escalonados pelos processos lógicos sob sua responsabilidade (internos) e encaminhá-los, no momento que considerar oportuno, para os processos lógicos onde estes eventos ocorrerão. Se o PL destino do evento é interno, a entrega do evento é feita pelo próprio gerente. Caso seja externo, o gerente envia o evento, através do seu módulo de comunicação, para o gerente externo responsável pelo processo lógico destino, o qual então se encarregará da entrega. Um evento segue, portanto dois caminhos distintos, caso os PLs de origem e destino executem no mesmo processador ou em processadores diferentes.

O algoritmo implementado pelo gerente baseia-se no algoritmo assíncrono do modelo de Chandy e Sherman. Introduz, porém, algumas modificações com a intenção de diminuir o trânsito de mensagens entre processadores e, conseqüentemente, melhorar o desempenho da simulação. As otimizações são possíveis em virtude da concentração das tarefas inerentes ao modelo nas mãos dos gerentes. Os valores $next_x$, por exemplo, para um conjunto de PLs que executam em um mesmo processador, são controlados pelo gerente deste processador. Este pode, portanto, saber qual dos processos lógicos possui o menor valor $next_x$ local ao processador, sem que seja necessária a troca de mensagens entre os PLs. O menor valor $next_x$ do sistema lógico pode ser descoberto pela troca de mensagens entre os gerentes, cujo número (igual ao número de processadores que, para a máquina utilizada na implementação, totaliza 8) é significativamente menor que o número de processos lógicos para uma aplicação padrão (no exemplo usado na avaliação do desempenho do simulador, este número chegou a 1.024 PLs). São as seguintes as modificações feitas no modelo original:

1. *Semântica de n_r e n_s.* Os contadores n_r e n_s são usados para o controle do fluxo de eventos entre gerentes e não mais entre processos lógicos. A idéia por trás desta modificação é que não é importante saber quais canais não estão vazios, mas sim se **algum** canal não está, já que o modelo exige que não haja mensagens em trânsito em **nenhum** canal no aproveitamento de um estado global gravado. Desta forma, as várias conexões entre os PLs que executam em dois processadores diferentes são vistas como dois grandes canais (nos dois sentidos) interligando os dois gerentes que controlam estes PLs. Os canais entre os processos lógicos sob a responsabilidade de um mesmo gerente são tratados como um único canal que parte deste gerente com destino ao próprio. Esta medida diminui substancialmente o número de contadores a serem mantidos.

2. *Atualização de n_r e n_s.* Um gerente possui uma estrutura de dados para armazenar eventos, condicionais ou definitivos, já recebidos de PLs internos ou externos, para serem entregues aos PLs de destino. Em linhas gerais, o algoritmo seguido por um gerente é formado por um laço englobando três operações básicas. Primeiramente, é verificada a existência de alguma mensagem externa, vinda de outro processador, que, se existir, será devidamente tratada. Em seguida, o gerente tenta transformar eventos condicionais em definitivos, seguindo as regras do modelo original. Na última fase, o evento definitivo com menor rótulo de tempo na estrutura de armazenamento de eventos é selecionado para ser entregue ao PL destino. Vários eventos (definitivos e condicionais) tendo como origem, por exemplo, um PL_x e como destino um determinado PL_y podem estar esperando nesta estrutura. Dado isto, será a seguinte a forma de atualização de n_r e n_s. Quando o gerente que controla PL_y entregar definitivamente a este um evento escalonado por PL_x, o contador n_r correspondente ao canal que conecta os gerentes responsáveis por PL_x e PL_y será incrementado. A estrutura que armazena os eventos em um gerente está organizada pelos rótulos de tempo dos eventos. Um eventos definitivo nesta estrutura, escalonado por PL_x para ocorrer em PL_y no tempo t, é considerado em trânsito se não há nenhum evento condicional de PL_x para PL_y na estrutura com rótulo de tempo menor que t. O contador n_s, relativo ao canal que une os gerentes responsáveis por PL_x e PL_y, deve ser acrescido de uma unidade para cada evento em trânsito de PL_x para PL_y na estrutura. Deste modo, se um canal parte de um gerente G_1 para outro gerente G_2, então G_1 atualiza o contador n_s do canal para cada novo evento em trânsito escalonado por um PL sob sua tutela para ocorrer em um PL sob a responsabilidade de G_2. Da mesma forma, G_2 atualiza o contador n_r do canal entre G_1 e G_2 para cada evento entregue a um PL sob sua responsabilidade, escalonado por um PL controlado por G_1.

3. *Envio de n_r e n_s.* Os contadores não serão irradiados pelo sistema. Um contador n_r dando conta do número de eventos recebidos de um determinado gerente será enviado somente para este mesmo gerente, de forma que o mesmo possa compará-lo com o contador n_s correspondente. Analogamente, um contador n_s, mantido por um gerente, só será enviado para o gerente receptor dos eventos transmitidos pelo primeiro. Além de restringir o envio dos contadores, o novo algoritmo determina momentos precisos nos quais estes devem ser enviados. Um contador n_r será transmitido para o gerente de direito toda vez que incrementado. Quanto a um contador n_s, este somente será transmitido quando os eventos em trânsito já computados em seu valor forem efetivamente enviados ao gerente destino. (Um contador só é enviado para que seja comparado com o contador correspondente da outra ponta do canal. De nada adiantaria enviar um n_s se o contador n_r correspondente não pudesse ainda levar em conta os eventos em trânsito ainda não entregues, porém já computados em n_s.)

4. *Gravação do estado local.* Com a nova forma de envio dos contadores n_r e n_s, um gerente tem condições de determinar se os canais a ele incidentes, que o conectam com outros gerentes, estão vazios em algum estado global. Quando isto ocorrer, o gerente grava o seu estado local como sendo o mínimo dos valores $next_x$ dos processos lógicos sob sua responsabilidade. Feita a gravação, o gerente irradia o estado local para todos os outros gerentes do sistema. Desta forma, cada gerente possuirá uma cópia dos estados locais de todos os gerentes, gravados quando estes perceberam que não havia eventos em trânsito nos canais a eles incidentes. Este

conjunto de estados locais constitui um estado global no qual o estado de cada um dos canais do sistema lógico é uma seqüência vazia e, conseqüentemente, para o qual vale a aplicação da equação 1.

As modificações tentam tirar vantagem do controle da simulação centralizado nos módulos de gerência, de maneira a diminuir as mensagens em trânsito no sistema. No entanto, o princípio básico é o mesmo do algoritmo assíncrono do modelo original por abordagem de eventos condicionais: gravar um estado global do sistema no qual seja possível utilizar a equação 1 na transformação de eventos condicionais em definitivos.

O conjunto de estados locais determinado pelo algoritmo modificado constitui um estado global de acordo com a definição encontrada em [CL85], pois embora aparentemente o estado dos canais e o estado local de um gerente não sejam mais gravados atomicamente, o menor dos valores $next_x$ para um gerente é consistente com os valores de n_s sob sua responsabilidade. Segundo vimos anteriormente, este é o único aspecto da atomicidade que realmente importa.

4 Resultados

O modelo de simulação distribuído por abordagem de eventos condicionais — com as modificações descritas e implementado na máquina hipercúbica — teve o seu desempenho avaliado através de um sistema lógico que simula o comportamento de uma rede de filas computacionais. Esta rede possui uma topologia parametrizada por quatro fatores, os quais foram variados para que o desempenho do simulador pudesse ser avaliado em diversas situações.

A rede é fechada, com uma população de *jobs* fixa. Esta população (P) — o primeiro parâmetro a ser variado — é dividida pelo número de filas da rede para que se saiba o número de *jobs* inicial de cada fila. A rede é caracterizada também pelo número de *switches* (S) que a compõem. Um *job*, ao chegar a um *switch* vindo de alguma fila, deve escolher uma de suas saídas, de acordo com probabilidades associadas a elas, para prosseguir a uma outra fila (nos testes, foram consideradas probabilidades iguais para todas as saídas de um *switch*). Feita a escolha, o *job* entra numa seqüência de filas que o levará a um outro *switch*, no qual uma nova saída será sorteada. A esta seqüência dá-se o nome de aresta. O número de arestas (A) entre dois *switches* e o número de filas (F) por aresta são os outros parâmetros de caracterização.

Pode-se abstrair dos *switches* da rede de filas e considerar-se que as filas se conectam diretamente umas às outras. Desta forma, a última fila de cada aresta se conecta com cada uma das filas iniciais das arestas posteriores a um *switch*. Dito isto, pode-se tomar a rede como um conjunto de filas interconectadas por canais, de forma que a primeira fila de uma aresta tenha um número A de canais de entrada e um canal de saída, uma fila do meio da aresta tenha um canal de entrada e um de saída, e a última fila da aresta um canal de entrada e A canais de saída.

Um processo lógico foi alocado para simular cada fila da rede e se comporta como segue. Ao chegar um *job*, o PL toma o máximo entre seu relógio de simulação e o tempo de chegada do *job* e soma a este valor um tempo de serviço calculado (o modo de cálculo do tempo de serviço será explicado posteriormente). Feito isso, o PL sorteia um canal de saída — que será único caso o

PL simule uma fila do início ou do meio de uma aresta — e envia um evento, com o tempo de ocorrência calculado da forma descrita, para o PL com ele conectado através do canal escolhido[4]. Pelo comportamento descrito, pode-se verificar que um PL envia eventos na ordem crescente de rótulos de tempo. Sendo assim, por cada canal de entrada de um PL chegarão eventos também na ordem crescente de tempo.

Os *jobs* possuem todos a mesma prioridade, e por isto um PL que tenha somente um canal de entrada pode escalonar todos os seus eventos como definitivos, já que nenhum *job* será "preemptado", pois foi garantida a ordem crescente de tempos dos *jobs* que chegam. No entanto, um PL que possua A canais de entrada deve saber determinar quando um evento escalonado é definitivo ou condicional. Para isso, um relógio é mantido para cada canal de entrada de um PL, indicando o tempo de chegada do último evento por este canal. Se um evento e chega por um canal tal que seu tempo de ocorrência é menor que o relógio de todos os outros canais de entrada, então nenhum evento chegará cujo rótulo de tempo seja menor. Desta forma, o PL pode processar o evento e marcar o evento resultante do processamento como definitivo.

Eventos que não tenham rótulo de tempo menor que os relógios dos demais canais de entrada geram eventos condicionais quando processados. Estes eventos condicionais deverão ser cancelados caso seja constatado um erro na ordem de processamento dos eventos[5]. Com o intuito de minorar este cancelamento, os PLs com mais de um canal de entrada foram programados de maneira a acumular alguns eventos e depois processá-los a todos de uma vez. Com isso, eventos que chegam na ordem incorreta na fase de acúmulo poderão ser processados na ordem correta dos rótulo de tempo. Um PL acumula eventos escalonando eventos especiais para si mesmo. Se a lista de eventos retidos está vazia, o próximo evento recebido será posto na lista e causará o escalonamento de um evento especial para um tempo no futuro, enviado ao gerente. Eventos posteriores vão sendo introduzidos na lista. A lista será processada na ordem dos rótulos de tempo assim que o evento especial for recebido pelo PL.

O tempo de serviço de um *job* foi modelado como uma variável aleatória contínua, exponencialmente distribuída. Segundo Trivedi em [T82], o tempo entre duas chegadas sucessivas de *jobs*, o tempo de serviço num servidor de uma rede de filas e o tempo de falha de um componente de um sistema são exemplos de variáveis aleatórias exponenciais. As razões para o uso da distribuição exponencial incluem a sua relação com a distribuição de Poisson e o fato de possuir a propriedade de Markov. Esta reza que o comportamento da variável aleatória independe estatisticamente do seu comportamento passado.

Para a avaliação do desempenho do simulador, foram medidos os tempos de execução das simulações das redes de filas — obtidas com a variação dos parâmetros que as definem — utilizando-se 2, 4 e 8 processadores. No escopo definido para estas variações, limitou-se o número de filas inferiormente em 128 e superiormente em 1.024. Com relação à população de *jobs* no sistema, optou-se por simular cada rede de filas com 16K, 32K e 64K *jobs*[6].

Os parâmetros A, S e F foram variados como segue. Inicialmente, fixou-se $S = 2$ e $F = 32$

[4] Note que o envio de eventos é sempre feito através dos gerentes.

[5] Este cancelamento não se propaga por nenhum outro processo lógico, uma vez que o evento condicional sendo cancelado ainda está em poder do gerente, e não seria transformado em definitivo antes de ser cancelado.

[6] 1K *jobs* correspondem a 1.024 *jobs*.

e executou-se a simulação para $A = 2$, $A = 4$ e $A = 8$. Em seguida, S e F foram fixados em 2 e 16, respectivamente, e A tomou os valores 4 e 8. No que concerne o parâmetro S, os parâmetros A e F foram mantidos com os valores 2 e 32, respectivamente, e a simulação foi executada para $S = 2$, $S = 4$ e $S = 8$. Novamente fixou-se A e F, porém com $A = 2$ e $F = 16$, e atribuiu-se a S os valores 4 e 8. Finalmente, A e S foram mantidos com os valores 2 e 4, respectivamente, para que se pudesse avaliar a variação do parâmetro F. A simulação foi executada com $F = 16$, $F = 32$, $F = 64$ e $F = 128$. Cada u na das combinações de valores para A, S e F define uma rede de filas diferente. As combinações listadas acima foram simuladas com cada uma das três populações de *jobs* escolhidas. Uma configuração A, S, F e P foi simulada quatro vezes para que se pudesse extrair uma média dos tempos de execução, dado o não-determinismo da simulação.

Os tempos obtidos com as simulações paralelas foram posteriormente comparados com os tempos medidos para as mesmas combinações de parâmetros num simulador seqüencial. A estrutura de dados utilizada para a lista de eventos é a mesma do simulador distribuído, a *splay tree* [ST85]. Porém, como neste um gerente subdivide a estrutura de *splay tree* e usa uma lista simplesmente encadeada ligando as cabeças das subestruturas, estudou-se a melhor forma de implementação da lista de eventos para o simulador seqüencial: se como uma *splay tree* única ou como um conjunto de *splay trees*, encadeadas por uma lista. Considerando uma única *splay tree*, o tamanho máximo da estrutura seria a população P de *jobs* do sistema. Uma operação em tal estrutura teria custo de tempo $O(\log P)$. Com várias *splay trees*, a lista de cabeças teria tamanho m igual ao número de filas na rede (isto é, igual ao produto AFS) e cada *splay tree* teria em média tamanho p, sendo $P = mp$. As operações de inserção e remoção teriam custo de tempo $O(m) + O(\log p)$, caso a operação causasse uma atualização na lista simplesmente encadeada, ou $O(\log p)$, caso essa atualização não fosse necessária. Comparando o custo de o operações, das quais l demandam uma atualização na lista de cabeças, tem-se que a utilização de várias *splay trees* é vantajosa, ou seja,

$$oO(\log P) > l(O(m) + O(\log p)) + (o - l)O(\log p),$$

quando $o/l > m/\log m$. Experimentalmente, tal situação verificou-se para um rede de 128 filas e uma população de 64K *jobs*. As outras configurações foram simuladas utilizando-se uma única *splay tree* para armazenar a lista de eventos.

Usualmente (ver, por exemplo, [S87]), define-se o *speedup* para N processadores como a razão entre o tempo para executar o programa seqüencial mais eficiente que realiza uma computação, e o tempo de execução de um programa paralelo que realiza a mesma computação nos N processadores. Pela definição, o *speedup* é limitado superiormente por N, pois se a razão entre os tempos for maior que N, então existe pelo menos um programa seqüencial mais eficiente, formado pela seqüencialização do programa paralelo.

Os tempos medidos para as simulações paralelas foram avaliados relativamente às simulações seqüenciais que se utilizavam do simulador seqüencial referido acima. Este provou não ser o mais eficiente possível, conforme requer a definição de *speedup*, uma vez que algumas razões superaram o limite imposto pelo número de processadores usados. Atribui-se esta distorção ao tamanho excessivo de uma estrutura de *splay tree* gerenciada pelo simulador seqüencial. Por isso, define-se um novo conceito — aceleração — dada pela razão entre o tempo medido para a execução da simulação seqüencial com uma determinada configuração de parâmetros de caracterização da rede de filas,

e o tempo medido para a execução da mesma simulação utilizando-se o simulador distribuído por abordagem de eventos condicionais. Os gráficos que ilustram o desempenho do modelo paralelo foram traçados com o número de processadores como uma dimensão e a aceleração como a segunda dimensão.

As figuras 1 a 5 mostram a influência das variações de A, S e F, respectivamente, no desempenho do simulador distribuído para duas populações diferentes de *jobs* (excluindo-se $P = 16K$, por falta de espaço). De uma maneira geral, é possível notar que quanto menor o número de arestas que partem de um *switch*, quanto menor o número de *switches*, quanto maior o número de filas numa aresta e quanto maior a população, melhor o desempenho da simulação. Todas as condições de melhora do desempenho explicam-se porque aumentam a quantidade de eventos definitivos no sistema, resultado este esperado dado que o simulador se baseia num modelo conservador de simulação distribuída. O modelo de teste não foi muito esclarecedor, no entanto, no que se refere ao efeito da proporção entre eventos condicionais e definitivos escalonados pelos processos lógicos no desempenho da simulação. Nos testes realizados, uma grande parte dos eventos condicionais escalonados teve de ser cancelada posteriormente por erro na ordem de processamento dos eventos. Com isso, o impacto maior causado pelos eventos condicionais deveu-se aos cancelamentos e não propriamente à espera por sua transformação em eventos definitivos.

5 Conclusões

Os testes realizados para a avaliação do desempenho do modelo de simulação distribuída por abordagem de eventos condicionais expuseram seus dois limites opostos de comportamento. O paradigma conservador pode ter desempenho excelente quando o sistema físico apresenta níveis suficientes de *lookahead* e o sistema lógico que o implementa é capaz de explorá-lo com eficiência. Desempenho este que pode ser qualificado como sofrível quando pouco se pode prever no que concerne o comportamento futuro do sistema em simulação.

O ponto forte do paradigma está no fato de evitar a sobrecarga do sistema distribuído com mensagens nulas, tal qual o faz o modelo conservador clássico para garantir a progressão da simulação sem a ocorrência de *deadlock*. Aliado a isto, o isolamento do controle da simulação, ditado pelo paradigma, em módulos distribuídos de gerência, possibilitou uma otimização do modelo com o objetivo de diminuir o tráfego de mensagens. Em [F90], Fujimoto chegara à conclusão de que métodos conservadores podem ter êxito em pacotes de simulação, nos quais o algoritmo de sincronização de eventos é otimizado e ao usuário cabe a configuração de determinados módulos para a adaptação a sistemas específicos.

A concentração de tarefas inerentes ao paradigma, se permite a diminuição do tráfego de mensagens, introduz no entanto o cancelamento de eventos condicionais que não mais ocorrerão. A eliminação excessiva de eventos condicionais pode impactar substancialmente o desempenho da simulação. Há uma perda associada à gerência da estrutura de dados de armazenamento de eventos, exacerbada com a inserção e remoção de eventos inúteis.

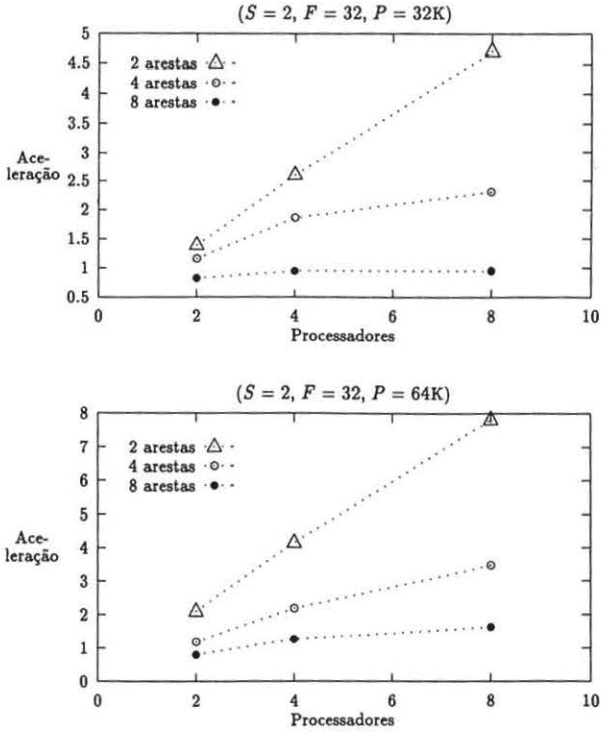

Figura 1: Variação do número de arestas com $S = 2$ e $F = 32$

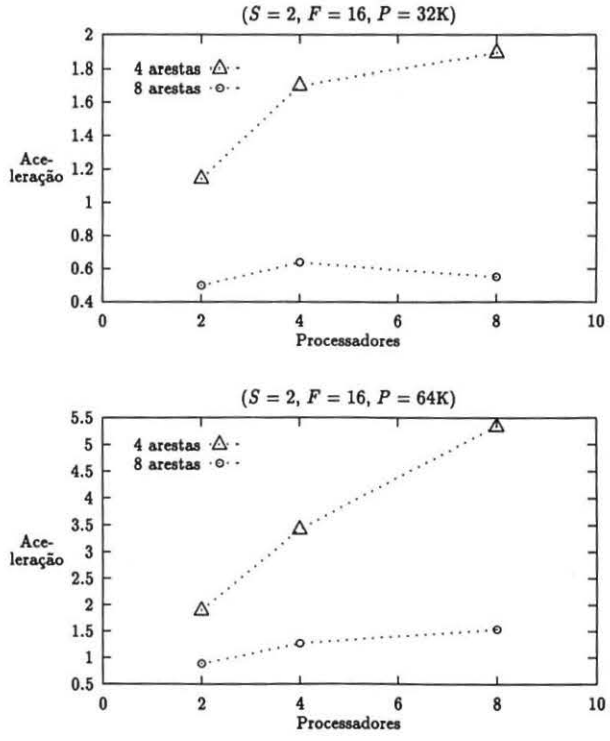

Figura 2: Variação do número de arestas com $S = 2$ e $F = 16$

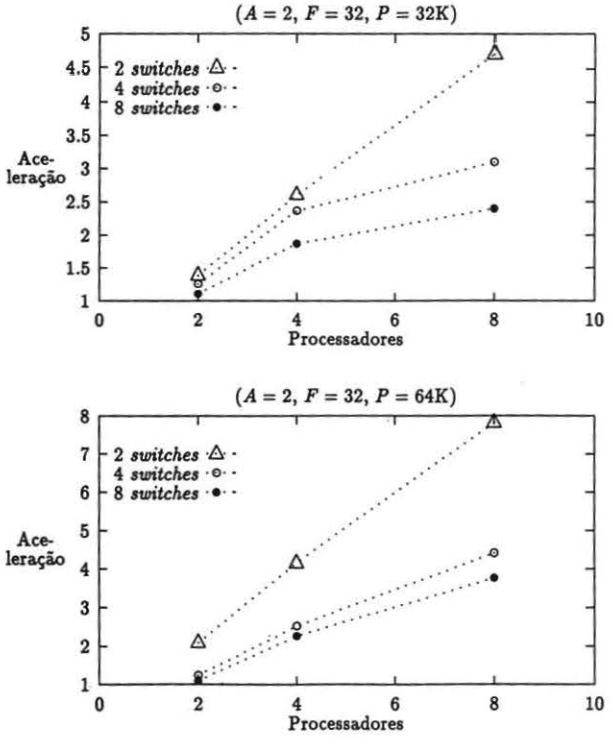

Figura 3: Variação do número de *switches* com $A = 2$ e $F = 32$

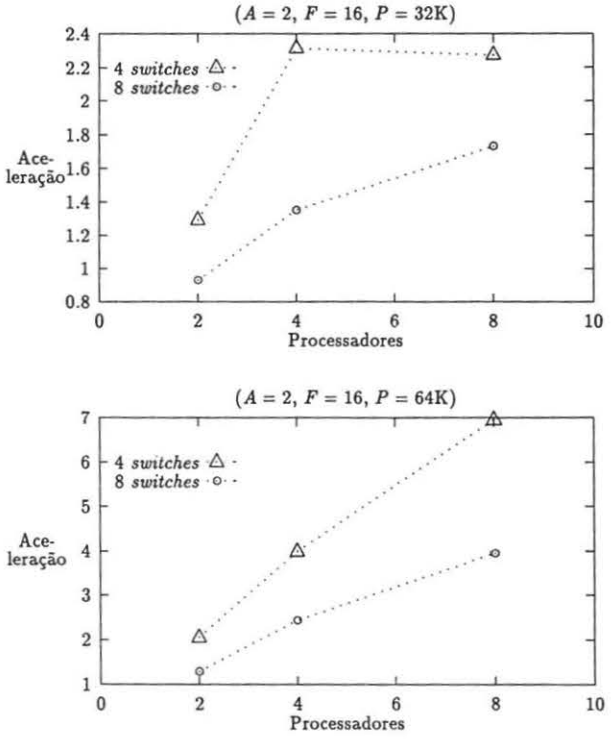

Figura 4: Variação do número de *switches* com $A = 2$ e $F = 16$

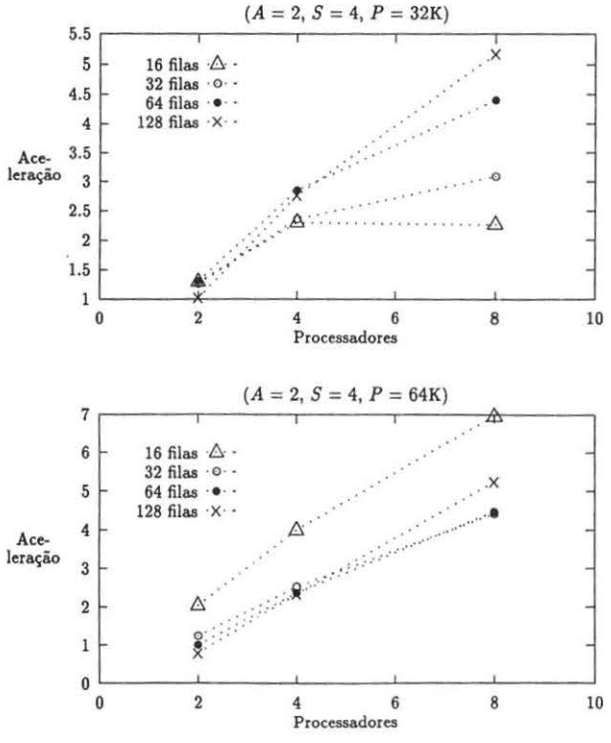

Figura 5: Variação do número de filas entre *switches* com $S = 4$ e $A = 2$

Referências

[ACSC91] Amorim, C. L.; Citro, R.; de Souza, A. F. & Chaves Filho, E. M., *The NCP-I Parallel Computer System.* COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, Relatório Técnico ES-241/91, 1991.

[B88] Burns, A., *Programming in Occam2.* Addison-Wesley, Wokingham, Inglaterra, 1988.

[CL85] Chandy, K. M. & Lamport, L., "Distributed snapshots: determining global states of distributed systems", *ACM Transactions on Computer Systems* **3** (1), 63–75, 1985.

[CM79] Chandy, K. M. & Misra, J., "Distributed simulation: a case study in design and verification of distributed programs", *IEEE Transactions on Software Engineering* **SE-5** (5), 440–452, 1979.

[CS89] Chandy, K. M. & Sherman, R., "The conditional event approach to distributed simulation", *Proceedings of the SCS Multiconference on Distributed Simulation*, 93–99, 1989.

[D90] Drummond, L. M. de A., *Projeto e Implementação de um Processador Virtual de Comunicação.* COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, Tese de Mestrado, 1990.

[F90] Fujimoto, R. M., "Parallel discrete event simulation", *Communications of the ACM* **33** (10), 30–53, 1990.

[M86] Misra, J., "Distributed discrete-event simulation", *ACM Computing Surveys* **18** (1), 39–65, 1986.

[ST85] Sleator, D. D. & Tarjan, R. E., "Self-adjusting binary search trees", *Journal of the ACM* **32** (3), 652–686, 1985.

[S87] Stone, H. S., *High-Performance Computer Architecture.* Addison-Wesley, Reading, MA, 1987.

[T82] Trivedi, K. S., *Probability and Statistics with Reliability, Queueing and Computer Science Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1982.