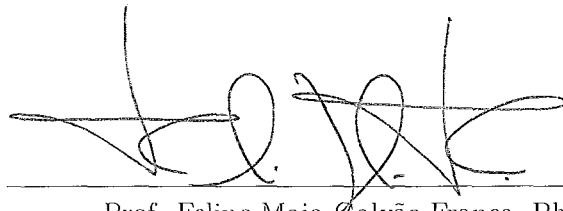
ESCALONAMENTO DISTRIBUÍDO POR REVERSÃO DE ARESTAS COM RECONFIGURAÇÃO DINÂMICA DE CARGA

Diego Moreira de Araujo Carvalho

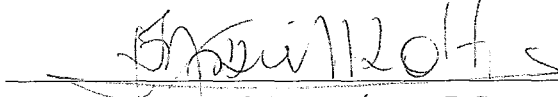
TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

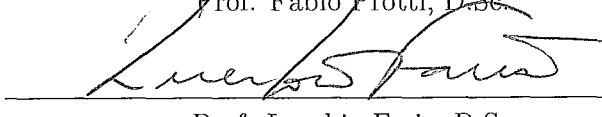


Prof. Felipe Maia Galvão França, Ph.D.

Prof. Valmir Carneiro Barbosa, Ph.D.



Prof. Fábio Protti, D.Sc.



Prof. Luerbio Faria, D.Sc

Prof. Josefino Cabral Melo Lima, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2004

CARVALHO, DIEGO MOREIRA DE ARAUJO

Escalonamento distribuído por reversão de arestas com reconfiguração dinâmica de carga [Rio de Janeiro] 2004

XI, 057 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2004)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 - exclusão mútua

2 - escalonamento, algoritmo distribuído, alta carga

I. COPPE/UFRJ II. Título (série)

Aos meus queridos Pais....

Agradecimentos

Várias pessoas contribuíram direta e indiretamente para essa tese. Em primeiro lugar, gostaria de agradecer ao meu grande amigo Pedro Henrique por ter me convencido a entrar no programa de pós-graduação da COPPE e depois pelos incontáveis momentos de incentivo, trabalho e discussões sobre a tese. Em segundo lugar, Luiz Felipe Canto, Joel Jones Jr e Curt Roloff que no trabalho, fizeram de tudo para que eu tivesse o tempo necessário para o desenvolvimento dessa tese. Além deles, gostaria de lembrar o nome de Philippe Gavillet, um grande mentor e amigo que me ensinou a nunca desistir para se obter uma meta. C'est ça.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc)

ESCALONAMENTO DISTRIBUÍDO POR REVERSÃO DE ARESTAS COM RECONFIGURAÇÃO DINÂMICA DE CARGA

Diego Moreira de Araujo Carvalho

Março/2004

Orientador: Felipe Maia Galvão França

Programa: Engenharia de Sistemas e Computação

Este trabalho introduz um novo algoritmo distribuído para o escalonamento de recursos atômicos compartilhados entre processos em situações onde pode ocorrer reconfiguração dinâmica de carga, i.e., quando a necessidade dos processos participantes pelo uso de tais recursos varia no tempo. O novo algoritmo, chamado de SER^{VT} (*Scheduling by Edge Reversal with Varying Topology*), apresenta um custo de comunicação, medido em número de mensagens, reduzido em aproximadamente 25% nas proximidades da situação de alta carga, quando comparado com a solução tradicional proposta por Chandy e Misra.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SCHEDULING BY EDGE REVERSAL WITH VARYING LOAD AND TOPOLOGY

Diego Moreira de Araujo Carvalho

March/2004

Advisor: Felipe Maia Galvão França

Department: Computing and Systems Engineering

This work introduces a novel distributed algorithm for the scheduling of shared atomic resources, in the context of dynamic load reconfiguration, i.e., when the necessity of the participating processes for such resources varies in time. The new algorithm, SER^{VT} (*Scheduling by Edge Reversal with Varying Topology*), has communication cost, accounted as number of exchanged messages, reduced by 25% in the proximities of the heavy load situation, when compared to the traditional solution proposed by Chandy and Misra.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Na literatura, encontramos uma plêiade de algoritmos para a solução do problema de exclusão mútua em sistemas distribuídos, dentre os quais, a solução de Chandy e Misra para o problema dos filósofos jantando (*Dinning Philosophers Problem – DPP*) [1] é uma dais mais famosas. Neste problema, os filósofos se encontram de três maneiras diferentes: comendo, pensando ou com fome. Além disso, na especificação do problema, temos que dois filósofos não podem comer ao mesmo tempo, quando eles compartilham um garfo entre eles. Neste trabalho, apresentamos o problema dos ursos de Pangéia, onde ursos hibernam temporariamente em cavernas, deixando sempre alguns ursos em vigília para verificar a existência de comida e quando ocorre a passagem de um cardume de salmão, os ursos, acordados pelos que se encontravam em vigília, entram num ritmo frenético de alimentação, usando pontos de pesca em que somente um urso pode se alimentar por vez. No final do cardume, os ursos, alimentados e contentes, voltam para o estado de hibernação, enquanto outros restam no estado de vigília.

A diferença entre os problemas acima é que o primeiro apresenta uma modelagem genérica para os problemas de exclusão mútua, enquanto o segundo tem como sua principal característica as ondas de hibernação. Além disso, sabemos que se o sistema operasse constantemente em ritmo frenético, poderíamos economizar nas mensagens que são trocadas entre os processadores [2].

O problema dos ursos de Pangéia encontra seus correspondentes na vida real em diversas áreas, como no projeto de micro-arquiteturas para micro-processadores (objetivando economia de energia), sistemas industriais de produção em linha com filosofia *Just in Time* (objetivando a eqüidade de comportamento na flutuação de demanda), circuitos digitais para comutação de alto desempenho (visando a redundância), sistemas de aquisição de dados em detectores de partículas (visando adequação à carga), sistemas de roteamento (redundância), etc.

Nesse trabalho apresentamos o algoritmo de escalonamento por reversão de arestas com variação de carga ou topologia (ou, em inglês, *Scheduling by Edge Reversal with Varying Topology* – SER^{VT}), que vem propor uma nova solução para os problemas da classe dos ursos de Pangéia. Além de sua prova formal, fizemos simulações para observar o seu comportamento em relação aos diferentes níveis de carga *vs* a quantidade de mensagens trocadas. Numa outra simulação, mostramos a solução do *problema do cruzamento*, onde utilizamos o SER^{VT}como algoritmo de controle de sinais de trânsito em cruzamento de várias vias e pedestres. Em todas as simulações são apresentadas comparações com o algoritmo proposto por Chandy e Misra para solução do problema dos filósofos jantando, para que possa ser verificado a aplicabilidade do novo algoritmo.

A apresentação desse trabalho se segue na seguinte forma: no Capítulo 2, apresentamos os algoritmos, Chandy e Misra para a solução do problema dos filósofos e o SER para sistemas operando em alta carga. Esses dois algoritmos vão estabelecer os padrões de comparação. No Capítulo 3 apresentamos o SER^{VT}e sua prova formal. No Capítulo 4, mostramos o resultado de simulações que mostram os mecanismos internos do SER^{VT}, junto com comparações com o algoritmo de Chandy e Misra. No Capítulo 5, finalizamos com as conclusões e sugestões de pesquisas futuras.

Capítulo 2

Definições e trabalhos correlatos

Neste capítulo apresentaremos as noções preliminares, notações e outros algoritmos distribuídos necessários para o entendimento do trabalho desenvolvido. Os algoritmos distribuídos apresentados são utilizados para prover garantia de *exclusão mútua* a recursos necessários para a computação de algum algoritmo substrato. Em alguns casos, os processadores que executam esse algoritmo tem a necessidade de acesso constante aos recursos, por períodos finitos de tempo, nesse caso dizemos que o sistema opera em *alta carga*. Este capítulo se desenvolve com a definição de nós, arestas e grafos nas Seções 2.1, 2.2, 2.8, a apresentação do algoritmo SER na Seção 2.3 e da solução do problema dos filósofos por Chandy e Misra nas Seções 2.6 e 2.7. Na Seção 2.9 é apresentada uma revisão dos trabalhos correlatos.

2.1 Nós e recursos

Em todo o trabalho, cada processador é representado por um nó e o conjunto dos nós é denotado por $N = \{n_1, ..., n_n\}$. Para simplificar a notação, são utilizadas as letras i, j e k para indicar respectivamente os nós n_i,n_j e n_k e por n o número total de nós no sistema. Para cada nó, existe um conjunto de recursos utilizado pelo algoritmo substrato que é chamado de R_i ($i \in N$). O conjunto de todos os recursos do sistema será indicado por R.

▷ **Variáveis:**
P_i^j para todo $j \in N_{viz}^i$;

▷ **Entrada:** $\mathbb{PP}$:= *msg* vinda de (i, j)
recebe($\mathbb{PP} \leftarrow j$);
P_i^j := **verdadeiro**;
Se $\forall_{j \in N_{viz}^i} P_i^j$ = **verdadeiro**
Acessa os recursos compartilhados;
para todo $j \in N_{viz}^i$
P_i^j := **falso**;
envia($\mathbb{PP} \rightarrow j$);

Figura 2.1: Pseudo-código de SER.

2.2 O grafo G

De maneira geral, o sistema é descrito por um grafo não direcionado G, exceto quando especificado o contrário. Nesse grafo, os nós são mapeados diretamente e existe uma aresta ligando os nós i e j ($(i, j) \in E$) se existe algum recurso que está sendo compartilhado pelos nós i e j, como mostrado em (2.1). Além disso, para um dado nó i, o seu conjunto de vizinhos é denotado por N_i^{viz}.

$$G = (N, E) \tag{2.1}$$
$$\forall_{i,j \in N} \quad (i, j) \in E \longleftrightarrow R_i \cap R_j \neq \emptyset$$

2.3 O escalonamento por reversão de arestas

O escalonamento por reversão de arestas (*Scheduling by Edge Reversal* – SER), apresentado em [2] e [3], é o algoritmo utilizado quando o sistema opera em alta carga. Em [4] é apresentado o código assíncrono do SER que é reproduzido na Figura 2.1.

O SER é modelado por um grafo G, como descrito em (2.1), mas o conjunto de arestas (E) é orientado. A orientação é uma função de E com a seguinte forma

$$\omega : E \rightarrow N,$$

tal que se $i, j \in N$ e $(i, j) \in E$, então $\omega((i, j))$ é igual a i ou a j dependendo se a orientação da aresta é respectivamente para i ou para j. Além disso, se a orientação ω não induz ciclos no grafo G, ela é chamada de acíclica. Todas as orientações acíclicas de G são denotadas pelo conjunto Ω e os nós que têm todas as suas arestas incidentes são chamados de *sumidouros* e os que tem todas as suas arestas orientadas para seus vizinhos são chamados de *fontes.*

Com o conceito de orientações acíclicas, podemos definir a *decomposição em sumidouros* (DS), como apresentado em [5] e [6]. A DS de uma determinada orientação ω é uma partição de N em λ subconjuntos $(S_0, S_1, ..., S_{\lambda-1})$ como mostrado abaixo

$$S_0 = \{i \in N \text{ e } i \text{ é um sumidouro em } G\},$$

e para $0 \leq q \leq \lambda - 1$:

$$S_q = \{i | i \in Q_{q-1} \text{ e } i \text{ é um sumidouro no grafo induzido por } Q_{q-1}\},$$

onde $Q_{q-1} = N - S_0 - ... - S_{q-1}$. Devemos observar que S_0 é o conjunto de sumidouros do grafo G e que se retiramos esses sumidouros $(N - S_0)$, os nós da partição seguinte se tornam sumidouros como mostrado na Figura 2.2. A DS dos nós de um grafo G apresenta as seguintes propriedades:

(i) S_q é um subconjunto independente de N;

(ii) Todos os nós de $S_{\lambda-1}$ são fontes, mas podem existir fontes que não pertençam a $S_{\lambda-1}$;

(iii) Se $i \in S_q$ e $j \in S_l$ e $0 \leq l < q \leq \lambda - 1$, então se existe uma aresta entre i e j, ela é tal que $\omega((i, j)) = j$;

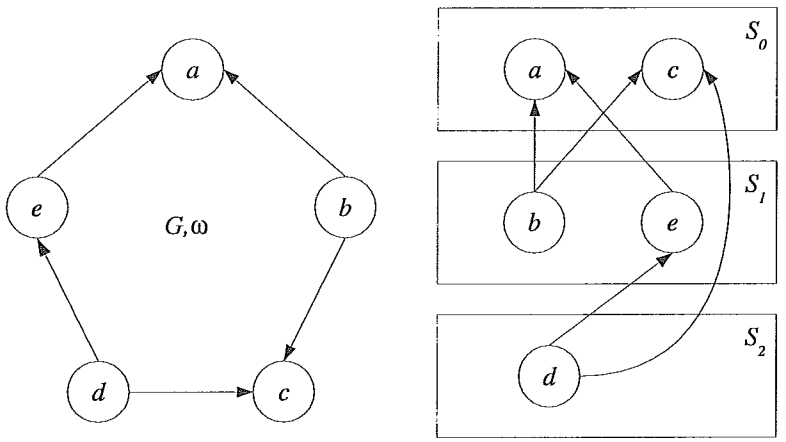


Figura 2.2: Decomposição de sumidouros.

(iv) Para $0 < q \leq \lambda - 1$, se $i \in S_q$ então existe um nó $j \in S_{q-1}$ onde $(i,j) \in E$ e $\omega((i,j)) = j$.

De uma maneira geral, utilizaremos a notação $DS(G)$ para nos referirmos a uma decomposição de sumidouros de um determinado grafo G com uma data orientação acíclica ω.

Com a definição da orientação do grafo G e da decomposição de sumidouros podemos facilmente entender o funcionamento do código do SER mostrado na Figura 2.1. Todo nó que é um sumidouro pode operar*. Depois da operação, ele deve reverter todas as suas arestas, se tornando uma fonte em G. Esse esquema de operação funciona da seguinte forma: todo o nó de S_0, depois de operar, troca as orientações de todas as suas arestas, gerando uma nova orientação $\omega(G)$, portanto uma nova $DS(G)$. Na nova decomposição de sumidouros (chamada aqui de $DS'(G)$), os nós que pertenciam a S_1 passam para S'_0, se tornando sumidouros e agora tendo o direito de operar, como mostrado na Figura 2.3. A aciclicidade de G é sempre mantida por

*por um tempo finito.

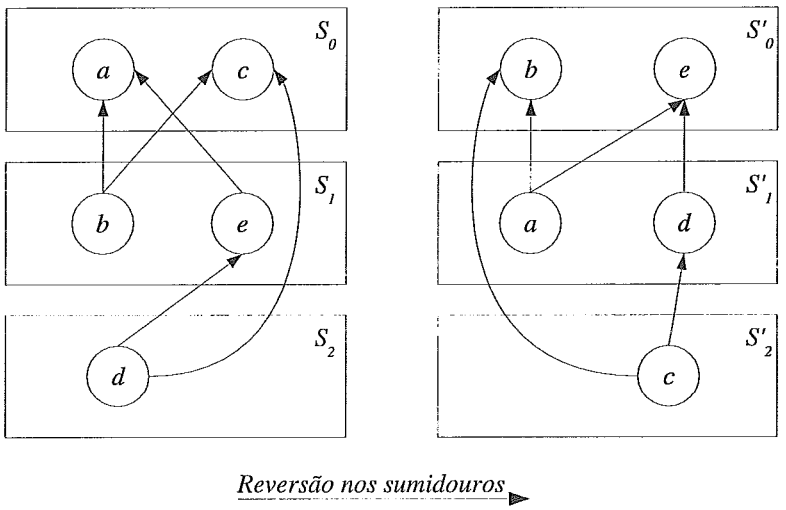


Figura 2.3: Reversão no SER.

esse esquema de operação, pois as únicas arestas modificadas são as arestas de cada sumidouro e como eles viram fontes em G, não podem ser gerados ciclos que passem por esses nós, garantindo a aciclicidade de $\omega(G)$. Além disso, podemos verificar que depois de operar, um nó espera um tempo finito para sua próxima operação, porque depois de virar uma fonte em G, ele vai deslocando nas DS's seguintes até virar novamente um sumidouro. A seqüência de orientações acíclicas do grafo G geradas pela operação do SER será chamada de $\sigma = \omega_1, \omega_2, ..., \omega_w$.

2.4 Definição de eventos

Durante as provas da corretude do algoritmo apresentado nessa Tese, vamos utilizar o conceito de evento, seu passado e seu futuro, como apresentado em [4] e [7]. Esses conceitos serão necessários na ordenação dos acontecimentos que ocorrem em um algoritmo distribuído assíncrono, i.e., onde não se faz uso de um relógio global. Passemos às definições.

Um evento $\xi \in \Xi$ da computação distribuída é a 6-upla, definida de seguinte maneira:

$$\xi = \langle i, t, \varphi, \sigma, \sigma', \Phi \rangle \tag{2.2}$$

onde:

- i é o nó em que o evento acontece;
- t é o tempo local quando o evento acontece;
- φ é a mensagem, se houver, que gerou o evento em i;
- σ é o estado local em que i se encontra antes do acontecimento do evento;
- σ' é o estado local em que i se encontra depois do acontecimento do evento;
- Φ é o conjunto de mensagens, se houver, enviadas pelo processador i em conseqüência do evento.

2.5 Passado e futuro de um evento

Como definido em [4] e [7], será apresentada a noção de passado e futuro de um evento. A noção de evento permite uma ordenação parcial ($\prec$) entre os eventos de uma computação distribuída (Ξ), de acordo com as seguintes condições:

- quando dois eventos ocorrem em um mesmo nó (ξ_1 e ξ_2) e seus tempos locais (t_1 e t_2) são tais que $t_1 < t_2$ e não existe nenhum evento ocorrido no mesmo nó tal que $t_1 < t < t_2$;
- quando dois eventos ocorrem em nós vizinhos (ξ_1 e ξ_2) e existe uma mensagem φ gerada no evento ξ_1 e recebida em ξ_2;

Além da ordenação entre dois eventos, é apresentada em (2.3) a definição de passado e futuro de um evento, utilizando o fecho transitivo direto do operador de ordenação $\prec$, denotado por $\prec^+$.

$$Passado(\xi) = \{\xi' | \xi' \in \Xi \wedge \xi' \prec^+ \xi\} \tag{2.3}$$

$$Futuro(\xi) = \{\xi' | \xi' \in \Xi \wedge \xi \prec^+ \xi'\}$$

2.6 O problema dos filósofos jantando

O *problema dos filósofos jantando (Dinning Philosophers Problem – DPP)*, primeiramente apresentado em [1], consiste de cinco filósofos sentados em torno de uma mesa redonda. Na frente de cada filósofo, um prato de macarronada no qual são necessários dois garfos para se comer (talvez por causa da falta de habilidade do cozinheiro na preparação do molho). Entre cada filósofo, temos disposto um garfo de maneira que dois filósofos vizinhos não podem utilizar os dois garfos simultaneamente, ou seja, eles não podem comer no mesmo instante. Os filósofos podem se apresentar nos seguintes estados: pensando, comendo e com fome. Uma representação gráfica é mostrada na Figura 2.4. No DPP, temos um problema típico de exclusão mútua, mas não temos a obrigatoriedade do sistema operar em alta carga. Se o fosse, teríamos o equivalente aos filósofos assumindo somente dois estados: comendo e com fome, o que seria perfeitamente modelado e resolvido pelo o SER apresentado anteriormente. Contudo, vamos estudar a solução apresentada por Chandy e Misra em [8] que vem a resolver introdução do novo estado. Apesar de ter sido escolhida essa ordem de apresentação dos algoritmos, cronologicamente, Dijkstra definiu e apresentou a primeira solução do DPP, seguido pela generalização distribuída mostrada por Chandy e Misra. Primeiramente, Gafni e Bertsekas desenvolveram o SER, depois Barbosa e Gafni formalizaram a solução em alta carga, utilizando o SER. Voltemos para a apresentação do algoritmo de Chandy e Misra.

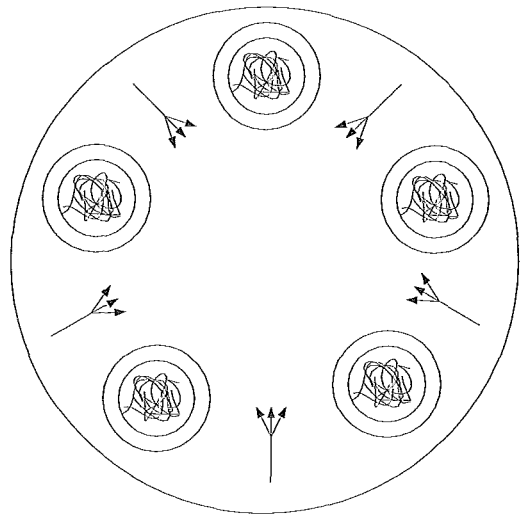


Figura 2.4: Problema dos Filósofos jantando.

2.7 A solução de Chandy e Misra

A solução do DPP, apresentada por Chandy e Misra, utiliza o grafo não orientado G para modelar o problema e tem o seu código apresentado nas Figuras 2.5 e 2.6. Contudo, quando dois filósofos vizinhos estão com fome, a simetria criada é quebrada com a utilização de um grafo de precedência H. Além disso, devemos notar que o algoritmo foi criado para qualquer configuração topológica de G, não existindo a obrigatoriedade do sistema ser um anel como mostrado no desenho dos filósofos. Discutiremos agora a utilização do grafo H para a quebra da simetria.

2.8 Grafo de precedência

O grafo de precedência H é construído com o mesmo conjunto de nós e arestas de G, mas apresenta uma orientação ω acíclica. Para cada aresta $e \in E$, existe uma ficha que hora esta com um nó, hora esta com o outro, nunca com os dois. No algoritmo, essa ficha simboliza a posse do garfo pelo filósofo, indicada no algoritmo pela variável local $P_i^j, i \in N \wedge j \in N_i^{viz}$ e sua troca é feita pela mensagem *garfo*. Lembre-se que

```
▷ Variáveis:
  comfome := falso;
  P_i^j para todo j ∈ N_viz^i; (Indica que o nó tem o garfo para essa aresta)
  M_i^j para todo j ∈ N_viz^i; (Indica que o nó tem a vez)
  T_i^j para todo j ∈ N_viz^i; (Indica que o nó deve o garfo ao seu vizinho)

▷ Entrada: nenhuma mensagem
  Quando deseja acessar os recursos compartilhados;
  comfome := verdadeiro;
  para todo j ∈ N_viz^i
    Se P_i^j = falso
      envia(pedido → j);

▷ Entrada: msg := pedido vinda de (i, j)
  recebe(pedido ← j)
  Se comfome = falso ou M_i^j = falso
    M_i^j := verdadeiro
    Se comfome = falso
      envia(garfo → j)
    Se não
      envia(garfo+pedido → j)
  Se não
    T_i^j := verdadeiro
```

Figura 2.5: Filósofos jantando por Chandy-Misra (parte 1/2).

essa aresta existe se e somente se $R_i \cap R_j \neq \emptyset$. Quando um nó é um sumidouro em H e deseja operar, ele envia uma mensagem para cada vizinho requisitando a ficha (se já não a possui) e seus vizinhos são obrigados a enviar a ficha como resposta em tempo finito. Quando ele tem todas as fichas, ele opera sobre os recursos e reverte suas arestas em H (variáveis locais $M_i^j, i \in N \wedge j \in N_i^{viz}$) para todos os vizinhos, indicado pelo envio das mensagens *vez* no texto do código. Como H era acíclico, ele continuará acíclico depois dessa transformação, pois nenhum ciclo pode passar pelo nó que executou a transformação. Esse sistema de transformações de H pode ser visualizado através da $DS(H)$. No caso dos nós intermediários i e j, quando i recebe um pedido do garfo vindo de j temos que i retorna o garfo imediatamente se

$$i \in S_q, j \in S_p, 0 < p < q \leq \lambda - 1$$

ou seja, quando $\omega((i,j)) = j$. Se o nó i estiver com fome, esse retorno é acompa-

```
▷ Entrada: msg := garfo+opt vinda de (i, j)
  recebe(garfo+opt ← j)
  P_i^j := verdadeiro
  Se opt = vez
    M_i^j := verdadeiro
  Se opt = pedido
    T_i^j := verdadeiro
    Se ∀_{j∈N_viz^i} P_i^j = verdadeiro
      Acessa os recursos compartilhados;
      comfome := falso;
      para todo j ∈ N_viz^i
        Se M_i^j
          M_i^j := falso;
          Se T_i^j
            T_i^j := falso;
            P_i^j := falso;
            envia(garfo+vez → j);
          Se não
            envia(vez → j);

▷ Entrada: msg := vez vinda de (i, j)
  recebe(vez ← j)
  M_i^j := verdadeiro
```

Figura 2.6: Filósofos jantando por Chandy-Misra (parte 2/2).

nhado pelo pedido do garfo através da mensagem *garfo+pedido*. No caso contrário, quando $\omega((i,j)) = i$, o nó i espera operar para enviar o garfo para j, satisfazendo o pedido.

2.9 Revisão bibliográfica e trabalhos correlatos

Existe uma plêiade de algoritmos para a exclusão mútua e escalonamento de processos variando topologia, mecanismos, objetivos, forma de alocação, *etc*, como apresentado na revisão feita por VELAZQUES em [9]. Uma taxonomia é apresentada por RAYNAL em [10] que divide estes algoritmos em três classes a saber:

- **Algoritmos baseados em permissões** – Utilizam a idéia de pedir a permissão para os vizinhos, quando um processo deseja utilizar os recursos compartilhados. As simetrias criadas são quebradas com prioridades temporais,

grafos acíclicos e *quorum* ou maiorias. Exemplos desses algoritmos podem ser encontrados em [11], [12], [13], [14], [15], [16], [17] e [18].

- **Algoritmos baseados em fichas** – Nesse caso, o processo só pode acessar os recursos compartilhados se for o detentor de um objeto especial que é único para o referido grupo de recursos. Exemplos desses algoritmos podem ser encontrados em [19], [20], [21], [22] e [23].

- **Algoritmos com coordenador central** – Nesse caso, os dois princípios anteriores são utilizados na solução. Os processos que precisam acessar os recursos, enviam mensagens para um coordenador central e no momento correto, eles recebem um objeto único, indicando o direito ao uso dos recursos.

Fora dessa taxonomia, exitem algoritmos distribuidos para exclusão mútua que focam a solução nos recusos e não nos processos que desejam o acesso (WELCH e LYNCH, [24]). No que concerne as aplicações, temos os algoritmos de exclusão mútua sendo utilizados em todos as áreas da computação, variando de aplicações em sistemas operacionais, micro-arquitetura, computação gráfica, redes, banco de dados, etc, além de usos específicos como aplicações de multimídia para vídeo conferência [25] e [26] e o seu uso em redes móveis [27].

Nosso trabalho, focou o seu estudo principalmente em algoritmos baseados em permissões, como os apresentados em [8], desenvolvido por Chandy e Misra, e [2], desenvolvido por Gafni e Bertsekas (primeiramente apresentado em [28]). Além disso, estudamos as variantes apresentadas em [29] e [30], que apresentam maneiras de modificar o grafo de precedência dinamicamente, normalmente utilizando um processo com conhecimento global. No Capítulo 3 passamos a apresentação formal do algoritmo desenvolvido nessa Tese.

Capítulo 3

O SER^{VT}

3.1 Os ursos de Pangéia

Para motivar a introdução do SER^{VT}, vamos apresentar o problema dos ursos de Pangéia. Por causa de modificações climáticas na Terra, ocasionadas pelo efeito estufa e outros agentes químicos, os ursos de uma determinada região da Pangéia* modificaram o seu hábito de hibernação por causa da mudança no regime de desova dos cardumes de salmão, que passaram a subir o rio em ondas durante todo o ano. Por causa dessa mudança, os ursos, organizados em grupos, ficam espalhados por uma grande área geográfica, dormindo dentro de cavernas. Na porta de cada caverna, um ou mais ursos são escolhidos para ficar em vigília, os quais tem como principal tarefa o monitoramento da passagem dos cardumes, contudo, eles executam outras atividades durante esse período. O rio apresenta alguns pontos para a pesca e logo que um cardume passa, cada urso em vigília acorda o seu grupo que estava hibernando, ocupando imediatamente os pontos de pesca. Contudo, esses pontos são de tal maneira que somente um urso pode pescar por vez. Eles ficam nessa pescaria frenética até que o cardume acabe e cada grupo volte para sua caverna. Esse problema apresenta duas características importantes: a primeira é que os nós, ou melhor, os ursos podem hibernar de tempos em tempos, mais quando eles estão operando, eles devem operar em alta carga para poder comer o maior número de

*na realidade não existiam ursos em Pangéia.

peixes possível, formando uma reserva de nutrientes para uma possível carência de alimentos.

No problema dos ursos, não poderíamos usar o SER porque os ursos tem que dormir quando estão dentro das cavernas, ou seja, existe um período em que os ursos não estão operando em alta carga. Por outro lado, o algoritmo de Chandy e Misra é uma solução para o problema, onde cada nó é um urso e os recursos compartilhados são os pontos de pesca. Contudo, gostaríamos de um algoritmo que fosse econômico na fase de alta carga, já que o caso do Chandy-Misra seria necessário o envio de uma mensagem *pedido* para cada vizinho depois de cada operação. Para solucionar o problema dos ursos, desenvolvemos o SER^{VT}, onde operamos como o SER entre os nós em alta carga e permitimos a hibernação dos outros.

3.2 O funcionamento do SER^{VT}

O escalonamento por reversão de arestas com variação de carga ou topologia (ou, em inglês, *Scheduling by Edge Reversal with Varying Topology* –SER^{VT}) é a solução apresentada para o problema dos ursos de Pangéia, combinando o comportamento do SER para execuções em alta carga, além de permitir que um nó se desligue do sistema até que um outro nó, que ainda esteja operando, o acorde para mais um período de operação em alta carga.

Para uma descrição informal do funcionamento do SER^{VT}, mostramos na Figura 3.1(a) um sistema com 12 nós. Os nós são interconectados seguindo as regras apresentadas em 2.1, onde a existência um canal de comunicação, ou aresta no grafo apresentado, indica que os nós compartilham pelo menos um recurso. Para cada canal de comunicação reservamos duas bolas: uma com a cor $\oplus$ e a outra com a cor $\ominus$. Na inicialização, distribuímos as bolas em pares de cores distintas de maneira que seja formado um grafo direcionado com uma orientação ω acíclica, como

observamos em 3.1(a).

Como no SER, os nós que são os sumidouros têm o direito de operar sobre os recursos compartilhados. No final da utilização dos recursos, os sumidouros revertem suas arestas como indicado pelas setas na Figura 3.1(b) e a operação se segue. A noção de sumidouro no SER^{VT} é diferente da existente no SER. Em nosso exemplo, ser um sumidouro significa ter todas as bolas de cor $\oplus$. Outro conceito diferente é a reversão total que existe no SER. No SER^{VT}, seguindo o nosso exemplo, a reversão de bolas de cor $\oplus$ só ocorre se o nó têm a correspondente bola de cor $\ominus$ para a aresta a qual esta ocorrendo a reversão. Essa diferença nos mecanismos existe para a introdução do estado de hibernação, não existente no SER.

Para compreender a diferença dos conceitos de sumidouro, reversão total e estado de hibernação, vamos acompanhar a evolução do exemplo na Figura 3.1(c). Nessa Figura, o nó f decide entrar em estado de hibernação. Nesse caso, ele só reverte as bolas de cor $\oplus$ para seus vizinhos, permanecendo com as bolas de cor $\ominus$ em seu poder. Podemos observar que os nós c, d, e, h, k e l são sumidouros de bolas $\oplus$, como citado acima. Nesse caso eles estão em condições de operar sobre os recursos compartilhados.

No nosso exemplo, os nós d, h e l operaram e reverteram suas arestas seguindo a convenção apresentada acima, como mostrado na Figura 3.1(d). Os outros sumidouros não fizeram a reversão das bolas pois ainda se encontram operando sobre os recursos. Nesse ponto vamos focar no nó g. Esse nó acabou de se tornar um sumidouro e quando terminar de operar sobre os seus recursos vai fazer sua reversão de arestas, indo para o estado de hibernação. Devemos observar que nesse caso, ele não reverte a bola de cor $\oplus$ para o seu vizinho f, porque g não possui a bola de cor $\ominus$ correspondente a essa aresta. Nesse momento, devemos lembrar que o nó f se encontra em estado de hibernação, ou seja, não apresenta interesse em usar

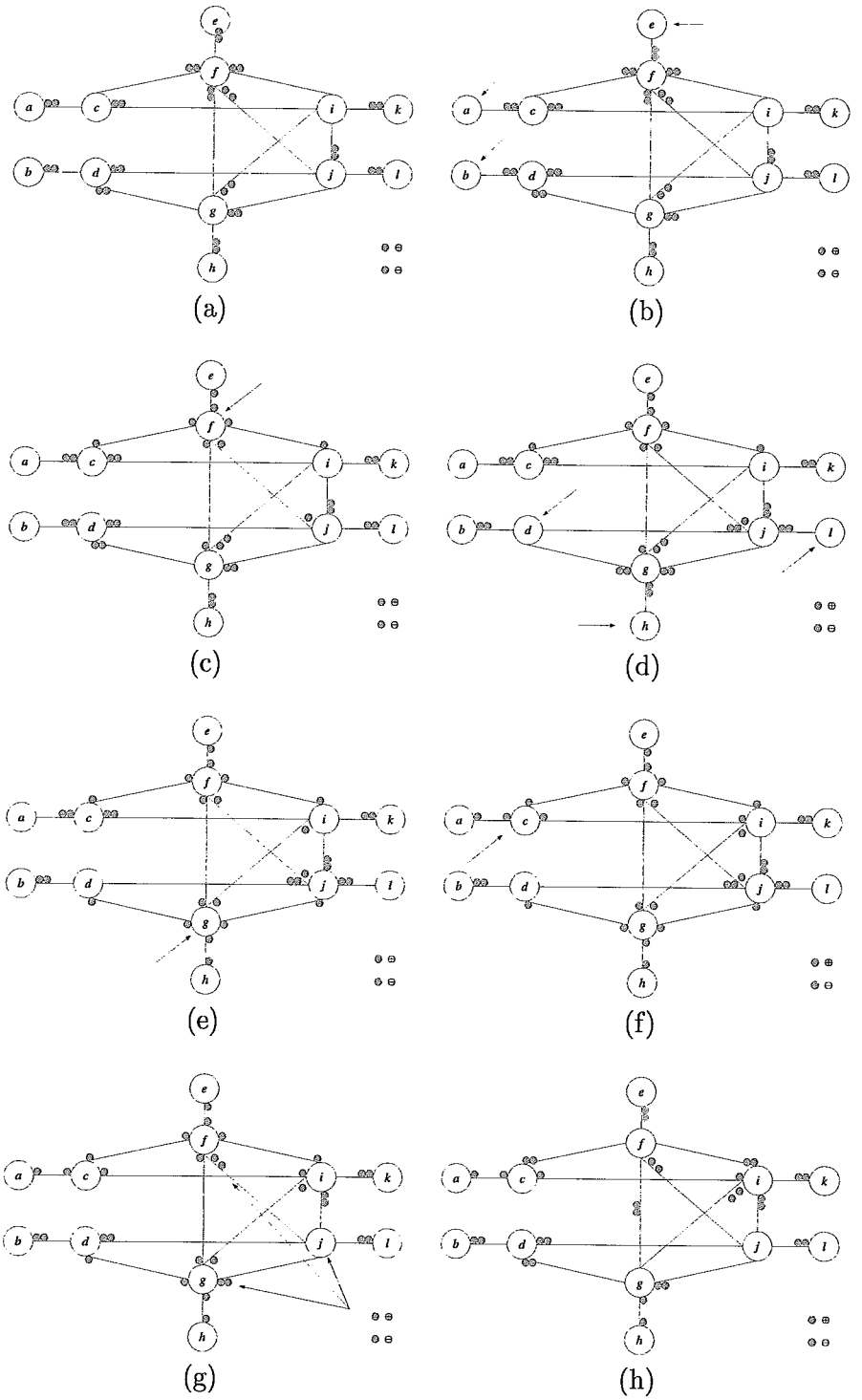


Figura 3.1: Exemplo de funcionamento.

os recursos que são compartilhados com seus vizinhos. O resultado da operação é mostrado na Figura 3.1(e).

Quando algum nó, que se encontra como sumidouro de bolas $\oplus$ decide acordar um vizinho, ele envia a respectiva bola de cor $\oplus$ para o vizinho hibernante como mostrado na Figura 3.1(g), ação executada pelo nó j, que acorda os nós f e g. Esses nós, para sair do estado de hibernação, enviam todas as bolas de cor $\ominus$, para seus vizinhos, exceto para o nó que enviou o pedido para acordar, como mostrado na Figura 3.1(h). Os vizinhos, recebendo a bola de cor $\ominus$ devem reverter imediatamente a bola de cor $\oplus$, como será mostrado na apresentação formal do algoritmo. Além desse detalhe, existem outros que serão abordados nas seções seguintes. Um deles, é o que ocorre quando temos uma bola de cor $\oplus$ e outra de cor $\ominus$ transitando no mesmo canal em direções diferentes, como mostrado entre os nós f e g na Figura 3.1(h).

Passaremos agora para a apresentação formal do algoritmo que é ilustrado nas figuras 3.2 e 3.3.

3.3 O estado dos nós

$$s : N \rightarrow \{R, W, H\} \tag{3.1}$$

$$s(i) = \begin{cases} R, & \text{se o nó } i \text{ está executando;} \\ W, & \text{se o nó } i \text{ está esperado a reversão das arestas;} \\ H, & \text{se o nó } i \text{ está em hibernação.} \end{cases}$$

Numa primeira análise do código, podemos observar que os nós podem se apresentar em três estados distintos e excludentes: executando (R), em espera (W) e hibernando (H). Definimos a função $s(i)$ em (3.1). Essa função faz o mapeamento dos nós do sistema com os seus respectivos estados. Além disso, será utilizado a

▷ **Variáveis:**
P_i^j para todo $j \in N_{viz}^i$; *(Indica que o nó tem direito aos recursos de R_{ij})*
M_i^j para todo $j \in N_{viz}^i$; *(Indica a forma de reversão da aresta)*

▷ **Entrada:** msg := $\mathbb{PM}$ vinda de (i, j)
Se $s(i) = \mathrm{W}$
  P_i^j := **verdadeiro**
  M_i^j := **verdadeiro**
  Se $\forall_{k \in N_{viz}^i} P_i^k$ = **verdadeiro**
    $s(i) := \mathrm{R}$;
    Acessa os recursos compartilhados;
    Se *próximoestado* = H
      para todo $k \in N_{viz}^i$
        Se M_i^k = **verdadeiro**
          P_i^k := **falso**;
          envia($\mathbb{PP} \rightarrow k$);
    Se *próximoestado* = W
      para todo $k \in N_{viz}^i$
        Se M_i^k = **falso**
          Se desejoAcordar(j)
            P_i^k := **falso**;
            envia($\mathbb{PP} \rightarrow j$);
          Se não
            P_i^k := **falso**;
            M_i^k := **falso**;
            envia($\mathbb{PM} \rightarrow k$);
  $s(i)$:= *próximoestado*;

Figura 3.2: Código do SER^{VT}(parte 1/2).

```
▷ Entrada: msg := ℙℙ vinda de (i, j)
  Se s(i) = W
    P_i^j := verdadeiro
    Se ∀_{k∈N_viz^i} P_i^k = verdadeiro
      s(i) := R;
      Acessa os recursos compartilhados;
      Se próximoestado = H
        para todo k ∈ N_viz^i
          Se M_i^k = verdadeiro
            P_i^k := falso;
            envia(ℙℙ → k);
      Se próximoestado = W
        para todo k ∈ N_viz^i
          Se M_i^k = falso
            Se desejoAcordar(j)
              P_i^k := falso;
              envia(ℙℙ → j);
            Se não
              P_i^k := falso;
              M_i^k := falso;
              envia(ℙ𝕄 → k);
    s(i) := próximoestado;
  Se s(i) = H
    P_i^j := verdadeiro;
    para todo k ∈ N_viz^i ∧ k ≠ j
      Se P_i^k = falso
        envia(𝕄𝕄 → k);
    (i) := W
▷ Entrada: msg := 𝕄𝕄 vinda de (i, j)
  Se s(i) = W
      P_i^j := falso;
      envia(ℙ𝕄 → j);
  Se s(i) = H
    P_i^j := falso;
    M_i^j := verdadeiro;
    envia(ℙℙ → j);
```

Figura 3.3: Código do SER^{VT}(parte 2/2).

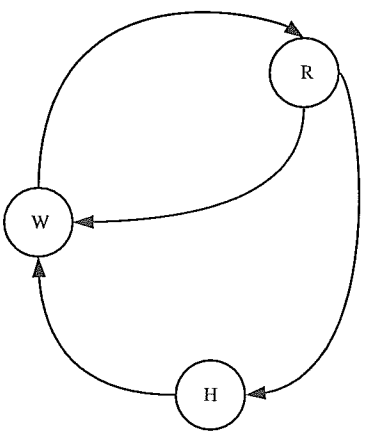


Figura 3.4: Diagrama de estados de um nó.

noção de nó *ativo* e *inativo*. Os nós ativos são os nós que se encontram em execução ou em espera (R ou W). Já os inativos são todos os nós em hibernação no sistema.

A Figura 3.4 mostra o diagrama de estados do algoritmo. Nela devemos observar que o nó só pode ir para o estado H vindo de um estado R e que quando o nó sai de um estado de hibernação ele passa obrigatoriamente pelo estado de espera. Além disso, o algoritmo pressupõe que sempre exista pelo menos um nó no estado R em todo o sistema. Essa obrigatoriedade existe porque um nó não pode sair do estado de hibernação expontaneamente, ou seja, ele tem que se tornar ativo através do pedido de um outro nó ativo que seja seu vizinho.

3.4 Mensagens e canais de comunicação

O SER^{VT} utiliza três tipos de mensagens, chamadas de $\mathbb{PM}$, $\mathbb{PP}$ e $\mathbb{MM}$. Como será mostrado adiante, elas indicam a reversão de arestas em dois grafos distintos (mensagem $\mathbb{PM}$), ou somente em um deles (mensagens $\mathbb{PP}$ e $\mathbb{MM}$). Essas mensagens podem ser mapeadas em canais de comunicação de dois *bits* de largura e eles existem quando ocorre uma aresta entre dois nós no grafo G, como definido em 2.2. Esses canais são construídos de forma a existir uma precedência entre as mensagens. Essa precedência é feita da seguinte maneira: num dado canal (i, j), quando

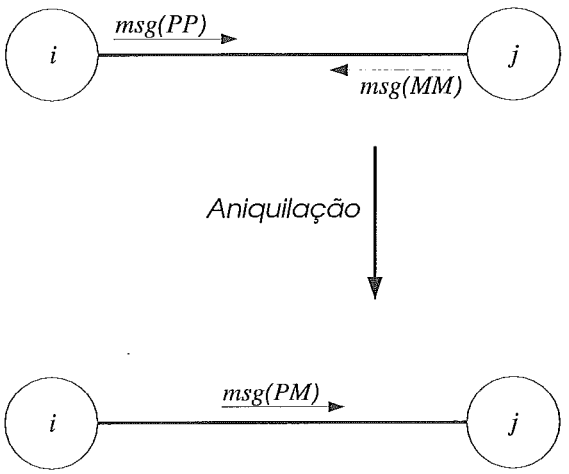


Figura 3.5: Precedência de mensagens.

existe simultaneamente uma mensagem $\mathbb{PP}$ trafegando de i para j e uma mensagem $\mathbb{MM}$ trafegando de j para i, as mensagens se aniquilam[†], resultando em uma única mensagem $\mathbb{PM}$ trafegando de i para j, como mostrado na Figura 3.5.

Esse caso é um dos problemas indicados na apresentação informal do algoritmo, quando entre os nós f e g existiam mensagens trafegando simultaneamente (Figura 3.1(h)). Essa configuração acontece normalmente quando temos vizinhos ativos acordando vizinhos que já estão em processo de ativação, ou quando dois vizinhos estão saindo do estado de hibernação simultaneamente. Nesse caso, é formada uma simetria (ex. entre f e g, quem deveria ficar com o par de bolas?) entre os nós que pode ocasionar uma espera circular, ou seja um *deadlock*. Para a quebra dessa simetria, vamos simplesmente utilizar o conceito de aniquilação de mensagens como descrito acima. Nas seções seguintes, vamos apresentar uma maneira para se implementar um protocolo que quebre a simetria criada nessas situações.

[†]como ocorre nos processos de colisão de elétrons e pósitrons em um acelerador de partículas.

3.5 Variáveis locais

Cada nó participante do SER^{VT}deve armazenar duas variáveis booleanas para cada canal de comunicação existente, *i.e.*, para cada vizinho no grafo G, como mostrado no topo da Figura 3.2. A primeira variável, P_i^j $(i \in N, j \in N_i^{viz})$, tem como função o mapeamento distribuído do grafo orientado de recursos $\mathcal{G}^+ = (N, \mathcal{E})$, como o existente no SER e quando $P_i^j = \textit{verdadeira}$ $(\omega((i,j)) = i)$, indica o direito do uso exclusivo dos recursos do conjunto $R_{i,j} = R_i \cap R_j$. Os nós indicam para os seus vizinhos as reversões no grafo de recursos através das mensagens $\mathbb{PP}$ e $\mathbb{PM}$.

A mensagem $\mathbb{PM}$, além de indicar a reversão no grafo de recursos, faz a reversão de arestas em um segundo grafo $(\mathcal{G}^- = (N, \mathcal{E}))$, no qual a orientação da aresta indica, quando concordante com a aresta de G', o direito de reversão da aresta de G' e quando discordante, o direito de ordenar a reversão da aresta de G'. A orientação desse segundo grafo é mantido em cada nó pela variável M_i^j $(i \in N, j \in N_i^{viz})$ e também pode ser modificado pela mensagem $\mathbb{MM}$.

3.6 O grafo $\mathcal{G}$

Os grafos de recursos e de direito de reversão podem ser visualizados em um único multi-digrafo $\mathcal{G} = \{N, \mathcal{E}\}$. Esse grafo tem suas arestas $((i,j,c) \in \mathcal{E})$ definidas através de triplas ordenadas, onde o terceiro elemento é a cor $c \in \{\oplus, \ominus\}$. A cor $\oplus$ colore as que representam as arestas do grafo de recursos, indicado no algoritmo pela variável local $P_i^j, i \in N \wedge j \in N_i^{viz}$. Já a cor $\ominus$ colore as que representam as arestas do grafo de direito de reversão, indicado pela variável local $M_i^j, i \in N \wedge j \in N_i^{viz}$.

Usando a definição de G apresentada em (2.1) e dos valores das variáveis locais P e M faremos uma definição livre do grafo $\mathcal{G}$, apresentada em (3.2). Devemos lembrar que durante o trânsito de mensagens, essa definição não se aplica, mas podemos imaginar imagens do sistema, onde todos os nós concordam com o direcionamento

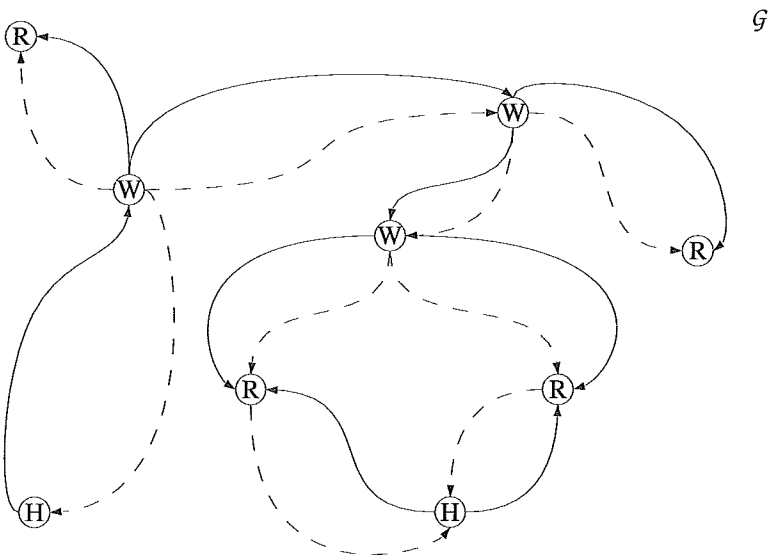


Figura 3.6: Grafo $\mathcal{G}$.

das arestas com os seus respectivos vizinhos. Além disso, usaremos o termo *par de arestas* para indicar as duas arestas de cores diferentes que ligam os mesmos vizinhos.

$$\begin{aligned} \mathcal{G} &= (N, \mathcal{E}) \qquad (3.2) \\ \forall_{(i,j)\in E} \quad & (i,j,\oplus) \in \mathcal{E} \wedge \omega((i,j,\oplus)) = i \longleftrightarrow P_j^i \\ \forall_{(i,j)\in E} \quad & (i,j,\ominus) \in \mathcal{E} \wedge \omega((i,j,\ominus)) = i \longleftrightarrow M_j^i \end{aligned}$$

3.7 Inicialização do SER^{VT}

O SER^{VT} deve ser inicializado seguindo quatro regras. Essas regras determinam as orientações das arestas no $\mathcal{G}$ e os possíveis estados que os nós podem se encontrar para a execução correta do algoritmo.

Regra 1 *Todos as arestas de $\mathcal{E}$ entre os nós ativos ($s(i) = R$ ou $s(i) = W$), têm seus pares de arestas concordantes, ou seja, $\omega((i,j,\oplus)) = \omega((i,j,\ominus))$, como apresentado em (3.3).* □

$$\forall_{i\in N \wedge j \in N_i^{viz}} \quad s(i) \neq H \wedge s(j) \neq H \quad \rightarrow \quad (i,j,\oplus),(i,j,\ominus) \in \mathcal{E} \wedge \omega((i,j,\oplus)) = \omega((i,j,\ominus)) \tag{3.3}$$

Regra 2 *Um par de arestas entre um nó ativo ($s(i) = R$ ou $s(i) = W$) e um nó inativo ($s(j) = H$) tem sempre a seguinte orientação: $\omega((i,j,\oplus)) = i$ e $\omega((i,j,\ominus)) = j$, como apresentado em (3.4).* □

$$\forall_{i\in N \wedge j \in N_i^{viz}} \quad s(i) \neq H \wedge s(j) = H \quad \rightarrow \quad (i,j,\oplus),(i,j,\ominus) \in \mathcal{E} \wedge \omega((i,j,\oplus)) = i \wedge \omega((i,j,\ominus)) = j \tag{3.4}$$

Vamos aproveitar a Regra 3.4 para definir um sub-grafo de $\mathcal{G}$ que chamaremos de fronteira. A fronteira ou $\mathcal{G}^0$ é formada pelos nós ativos que tem vizinhos hibernando e esses próprios vizinhos como mostrado em (3.5).

$$\begin{aligned} \mathcal{G}^0 &= (N^0, \mathcal{E}^0) \\ P(\mathcal{G}^0) &: \forall_{i,j\in N} \quad s(i) \neq H \wedge s(j) = H, i, j \in N \rightarrow i, j \in N^0 \wedge \\ &\quad (j,i,\oplus),(i,j,\ominus) \in \mathcal{E}^0 \end{aligned} \tag{3.5}$$

Regra 3 *Entre dois nós inativos, seus pares de arestas tem orientações contrárias ($\omega((i,j,\oplus)) \neq \omega((i,j,\ominus))$), como apresentado em (3.6).* □

$$\forall_{i,j\in E} \quad s(i) = H \wedge s(j) = H \quad \rightarrow \quad (i,j,\oplus),(j,i,\ominus) \in \mathcal{E} \wedge \omega((i,j,\oplus)) \neq \omega((i,j,\ominus)) \tag{3.6}$$

As regras 2 e 3 têm como conseqüência que todos os nós inativos apresentam suas arestas discordantes como apresentado em (3.7).

$$\forall_{i\in N} \quad s(i) = H \quad \rightarrow \quad (i,j,\oplus),(i,j,\ominus) \notin \mathcal{E} \quad tq. \quad \omega((i,j,\oplus)) = \omega((i,j,\ominus)) \tag{3.7}$$

Para a definição da quarta regra de inicialização, vamos apresentar alguns subgrafos de $\mathcal{G}$. O primeiro é o grafo orientado $\mathcal{G}^+$, definido em (3.8). Esse subgrafo é induzido quando retiramos todas as arestas de cor $\ominus$ de $\mathcal{G}$ e preservamos as orientações das arestas de cor $\oplus$. A partir de $\mathcal{G}^+$ podemos ter o grafo orientado $\mathcal{G}^{ativo}$ que só apresenta os nós ativos do sistema como mostrado em (3.9).

$$\begin{aligned} \mathcal{G}^+ &= (N, \mathcal{E}^+) \\ P(\mathcal{G}) &: \forall_{(i,j,\oplus),(i,j,\ominus)\in\mathcal{E}} \quad \omega((i,j,\oplus)) = \omega((i,j,\ominus)) \rightarrow (i,j) \in \mathcal{E}^+ \end{aligned} \tag{3.8}$$

$$\begin{aligned} \mathcal{G}^{ativo} &= (N^{ativo}, \mathcal{E}^{ativo}) \\ P(\mathcal{G}^+) &: \forall_{i,j\in N}(i,j) \in \mathcal{E}^+ \wedge s(i) \neq H \wedge s(j) \neq H \rightarrow (i,j) \in \mathcal{E}^{ativo} \end{aligned} \tag{3.9}$$

O grafo orientado $\mathcal{G}^-$ é formado de maneira similar ao $\mathcal{G}^+$, sendo que nesse caso retiramos as arestas de cor $\oplus$ como mostrado em (3.10). Contudo no grafo $\mathcal{G}^{inativo}$, fazemos a indução com a retirada dos nós ativos do sistema, como apresentado em (3.11). Passemos agora para a definição da quarta regra de inicialização.

$$\mathcal{G}^- = (N, \mathcal{E}^-) \tag{3.10}$$

$$P(\mathcal{G}) : \forall_{(i,j,\oplus),(i,j,\ominus)\in\mathcal{E}} \quad \omega((i,j,\oplus)) \neq \omega((i,j,\ominus)) \rightarrow (i,j) \in \mathcal{E}^+$$

$$\mathcal{G}^{inativo} = (N^{inativo}, \mathcal{E}^{inativo}) \tag{3.11}$$

$$P(\mathcal{G}^-) : \forall_{i,j\in N}(i,j) \in \mathcal{E}^- \wedge s(i) = H \wedge s(j) = H \rightarrow (i,j) \in \mathcal{E}^{inativo}$$

Regra 4 *As orientações de $\mathcal{G}^+$ e $\mathcal{G}^-$ são acíclicas.* □

Devemos observar que quando a Regra 4 é respeitada, temos os grafos $\mathcal{G}^{ativo}$ e $\mathcal{G}^{inativo}$ acíclicos, pois sua indução é feita somente com a retirada de arestas.

3.8 Corretude do SER^{VT}

Para apresentar a corretude do SER^{VT} devemos nos voltar primeiro para o diagrama de estados apresentado (Figura 3.4). Nesse diagrama temos as seguintes transições ξ de estado,

- ξ^{RW} – quando o nó sai do estado de execução para o estado de espera;
- ξ^{RH} – quando o nó determina que deve hibernar depois do acesso aos recursos;
- ξ^{WR} – quando o nó recebe o direito sobre todos os recursos;
- ξ^{HW} – quando o nó é despertado por algum vizinho.

Nessas transições, devemos observar que trechos específicos do código do SER^{VT} são executados e que mensagens são geradas, possivelmente ocasionando outras trocas de estado nos seus vizinhos. A partir desse momento, vamos mostrar através

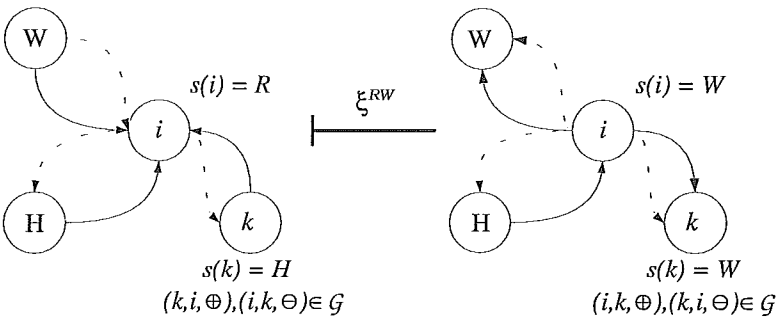


Figura 3.7: Transição ξ^{RW}.

de alguns lemas que o SER^{VT} respeita as direções das arestas e o estado dos nós determinados pelas regras 1, 2 e 3.

Lema 1 *Todas as reversões de arestas geradas por uma transição ξ^{RW} respeitam as regras 1, 2 e 3.*

Prova: Vamos primeiro reproduzir o procedimento executado em uma transição ξ^{RW} na Figura 3.8. Devemos observar que essa transição só ocorre em um nó ativo que venha receber uma mensagem $\mathbb{PM}$ ou $\mathbb{PP}$ e vamos supor que o sistema foi inicializado respeitando as quatro regras que foram definidas acima. Chamaremos o nó onde ocorre a transição de i. Antes da transição, por causa da inicialização, todas as arestas de cor $\oplus$ são orientadas para i, como mostrado na Figura 3.7, pois essa transição só ocorre depois que $\forall_{k \in N^i_{viz}} P^k_i = \textit{verdadeiro}$, como mostrado no código. Além disso, os pares de arestas no nó i só podem se apresentar em duas formas. A primeira forma em que os pares de arestas podem se encontrar antes da transição é

$$\omega((i,j,\oplus)) = i \wedge \omega((i,j,\ominus)) = i,$$

que ocorre quando um vizinho j está no estado W, por causa da Regra 1, determinando a orientação da aresta de cor $\ominus$. Nesse caso, o par resultante pela a execução do $SER^{VT}((ii)$ na Figura 3.8) tem a seguinte orientação:

$$\omega((i,j,\oplus)) = j \wedge \omega((i,j,\ominus)) = j,$$

que respeita a Regra 1, já que os dois nós estão ativos.

A segunda forma em que os pares de arestas podem se encontrar antes de ξ^{RW} é

$$\omega((i,j,\oplus)) = i \wedge \omega((i,j,\ominus)) = j,$$

que é o resultado da regra de fronteira (Regra 2), quando j, vizinho de i, se encontra no estado de hibernação. Nesse caso temos dois comportamentos. O primeiro é quando o nó i resolve não acordar o vizinho j, resultando na mesma orientação anterior do par que respeita a regra da fronteira. O segundo caso é quando i acorda o vizinho j através de uma mensagem $\mathbb{PP}$ (como mostrado em (i) na Figura 3.8). Nesse caso, a orientação do par resultante é:

$$\omega((i,j,\oplus)) = j \wedge \omega((i,j,\ominus)) = j,$$

contudo o nó vizinho j necessariamente sairá do estado H para W, respeitando a Regra 1. Se j já foi acordado por um outro nó, ele já enviou uma mensagem $\mathbb{MM}$, que será aniquilada pelo canal de comunicação, resultando num par de aresta orientado para j, o que respeita a Regra 1 □

Lema 2 *Todas as reversões de arestas ocasionadas por uma transição ξ^{RH} respeitam as regras 1, 2 e 3.*

Prova: Reproduzimos o procedimento responsável pela transição ξ^{RH} na Figura 3.10 e supomos que as regras de inicialização foram respeitadas. Esse procedimento

```
Se próximoestado = W
  para todo k ∈ N^i_viz
    Se M^k_i = falso
      Se desejoAcordar(j)
        P^k_i := falso;
(i)     envia(PP → j);
      Se não
        P^k_i := falso;
        M^k_i := falso;
(ii)    envia(PM → k);
```

Figura 3.8: Procedimento executado em uma transição ξ^{RW}.

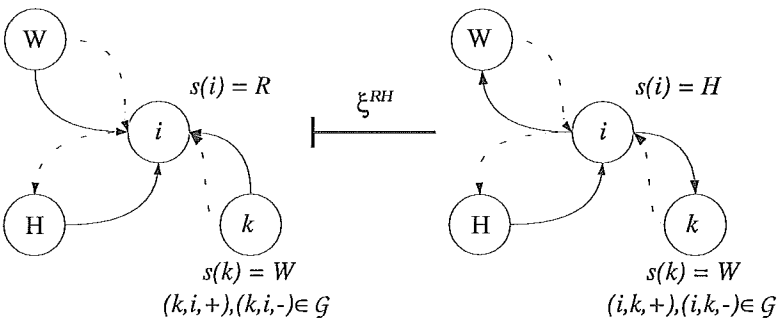


Figura 3.9: Transição ξ^{RH}.

só pode ser executado em um nó i que seja um sumidouro em $\mathcal{G}^+$ e antes da transição, os seus pares de arestas apresentam as seguintes orientações, como mostradas na Figura 3.9:

$$\omega((i,j,\oplus)) = i \wedge \omega((i,j,\ominus)) = i$$

e

$$\omega((i,j,\oplus)) = i \wedge \omega((i,j,\ominus)) = j,$$

por causa dos mesmos motivos apresentados no lema 1.

No final da transição, temos que $s(i) = H$, que nos obriga a verificar somente a validade das regras 2 e 3.

Se *próximoestado* = H
  para todo $k \in N^i_{viz}$
    Se M^k_i = **verdadeiro**
      P^k_i := **falso**;
(i)   **envia**($\mathbb{PP} \rightarrow k$);

Figura 3.10: Procedimento executado em uma transição ξ^{RH}.

Olhando o procedimento executado nessa transição (Figura 3.10), verificamos que só ocorre reversões nas arestas existentes entre i e os vizinhos que se encontram no estado W. Esses pares de arestas são revertidos parcialmente (somente as de cor $\oplus$) o que respeita a regra de fronteira. Já os pares que não foram revertidos respeitam a Regra 3. □

Lema 3 *Todas as reversões de arestas ocasionadas por uma transição ξ^{HW} respeitam as regras 1, 2 e 3.*

Prova: Supondo que as regras de inicialização foram respeitadas, reproduzimos o procedimento executado nessa transição abaixo (Figura 3.12). Como podemos visualizar na Figura 3.11, o nó i no qual ocorre uma transição ξ^{HW} só pode ter seus pares de arestas orientados da seguinte forma:

$$\omega((i,j,\oplus)) = i \wedge \omega((i,j,\ominus)) = j,$$

nas arestas inicializadas pela Regra 3 e

$$\omega((i,j,\oplus)) = j \wedge \omega((i,j,\ominus)) = i,$$

nas arestas inicializadas pela Regra 2 (regra da fronteira).

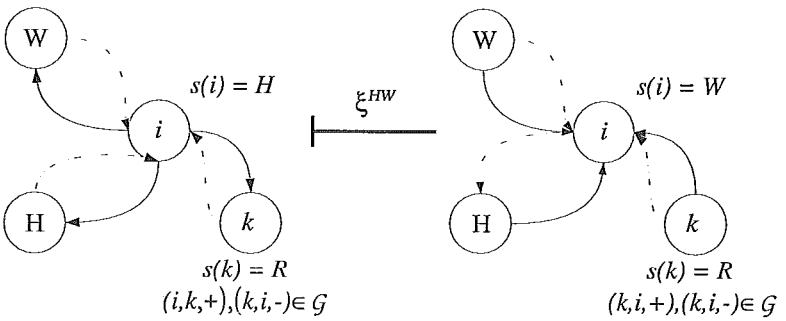


Figura 3.11: Transição ξ^{HW}.

Se $s(i) = \mathrm{H}$
(i) P_i^j **:= verdadeiro;**
para todo $k \in N_{viz}^i \wedge k \neq j$
Se $P_i^k =$ **falso**
(ii) **envia**($\mathbb{MM} \rightarrow k$);
$s(i) = \mathrm{W}$

Figura 3.12: Procedimento executado em uma transição ξ^{HW}.

No final da transição, temos que $s(i) = W$. Além disso, essa transição é disparada por uma mensagem $\mathbb{PP}$, vinda de um nó que passou por uma transição ξ^{RW}. Esse par de aresta respeita a Regra 1, pois ele está concordante no final da transição. Já os pares orientados na primeira forma (onde $\omega((i,j,\oplus)) = i$ e $\omega((i,j,\ominus)) = j$) não são revertidos, respeitando a regra da fronteira. No caso dos outros pares, temos três possibilidades: a primeira é quando o estado do nó vizinho é W, nesse caso o par resultante respeita a Regra 1, pela reversão mostrada no código da Figura 3.3. A segunda possibilidade é quando o nó vizinho se encontra no estado H. Nesse caso, o vizinho reverte a aresta de cor $\oplus$ para i como mostrado na Figura 3.3, respeitando a regra da fronteira. A terceira possibilidade ocorre quando

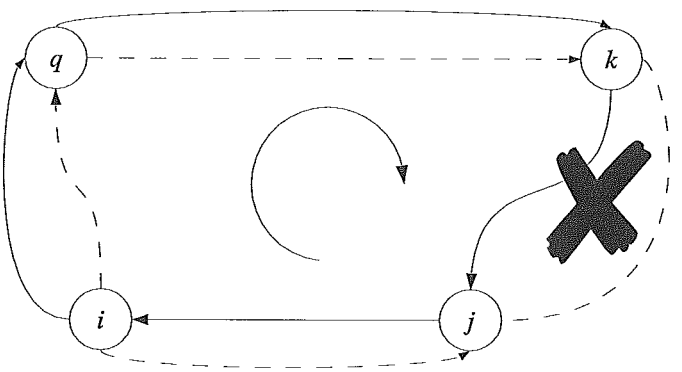


Figura 3.13: Ciclo hipotético em uma transição ξ^{RW}.

o vizinho se encontra no estado R e resolve acordar o nó i simultaneamente. Nesse caso a aniquilação de mensagens resulta em uma aresta que respeita a Regra 1. □

Lema 4 *As transições ξ^{WR} não geram reversões.*

Prova: Uma rápida inspeção no código das figuras 3.2 e 3.3, nos mostra que essa transição não gera mensagens de reversão. □

Os quatro lemas garantem que dada uma determinada orientação dos pares de arestas entre dois nós quaisquer, podemos inferir sobre o respectivo estado local de cada um e vice-versa. Usaremos esse fato na prova de aciclicidade dos grafos $\mathcal{G}^+$ e $\mathcal{G}^-$. Essa aciclicidade é importante por que garante a não existência de esperas circulares induzidas pela execução do algoritmo, o que ocasionaria *deadlocks*. Antes da prova do teorema de aciclicidade devemos provar os seguintes lemas.

Lema 5 *Uma transição de estado ξ^{RW} não gera ciclos nos grafos $\mathcal{G}^+$ e $\mathcal{G}^-$.*

Prova: Apresentamos na Figura 3.13 um nó i que acabou de passar por uma transição de estado ξ^{RW}. Pelo Lema 1, só existem duas configurações de pares de aresta com i em $\mathcal{E}$ depois da transição. A primeira configuração, revertida pelo procedimento da transição, tem a seguinte forma:

$$\omega((i,*,\oplus)) \neq i \wedge \omega((i,*,\ominus)) \neq i$$

e a segunda configuração, resultante dos pares que não foram revertidos pelo procedimento da transição, tem a seguinte forma:

$$\omega((i,*,\oplus)) = i \wedge \omega((i,*,\ominus)) \neq i,$$

Seja q o nó vizinho de i no par concordante e seja j o nó vizinho de i no par discordante. Se existe um ciclo em $\mathcal{G}^+$, esse ciclo deve passar obrigatoriamente pelas arestas

$$(i,q) \in \mathcal{E}^+ \wedge \omega((i,q)) = q$$

e

$$(i,j) \in \mathcal{E}^+ \wedge \omega((i,j)) = i$$

mostradas na Figura 3.13 como arestas de linha cheia. Além disso, sabemos que $s(q) = W$, por causa do Lema 1 e que $s(j) = H$ como demonstrado pelo Lema 2. Então, seja um ciclo no grafo $\mathcal{G}^+$ que passe pelo nó i formado pelo seguinte caminho:

$$i \rightarrow q \rightarrow ... \rightarrow k \rightarrow ... \rightarrow j \rightarrow i.$$

Sabemos que nesse caminho existe um nó k no qual $s(k) = H$ e $s(anterior(k)) = W$ (no limite podendo até ocorrer em $k = j$). Então, provamos por absurdo que o ciclo apresentado acima possa existir, porque pelo Lema

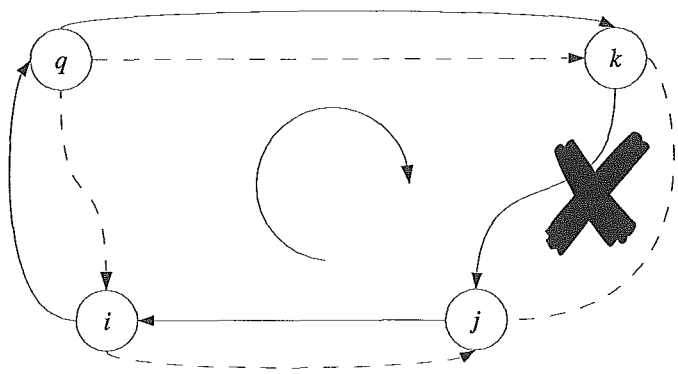


Figura 3.14: Ciclo hipotético em uma transição ξ^{RH}.

2, a aresta $(anterior(k), k, \oplus)$ não pode estar orientada para o nó k. A aciclicidade de $\mathcal{G}^-$ é trivial porque um nó i que passa por uma transição de estado ξ^{RW} se transforma numa fonte em $\mathcal{G}^-$. □

Lema 6 *Uma transição de estado ξ^{RH} não gera ciclos nos grafos $\mathcal{G}^+$ e $\mathcal{G}^-$.*

Prova: Na Figura 3.14, temos um nó i que acabou de passar por uma transição de estado ξ^{RH} e pelos lemas 2 e 3, só existem duas configurações em que os pares de aresta de i em $\mathcal{E}$ podem se encontrar:

$$\omega((i, *, \oplus)) = i \wedge \omega((i, *, \ominus)) \neq i,$$

e

$$\omega((i, *, \oplus)) \neq i \wedge \omega((i, *, \ominus)) = i$$

sendo que a primeira configuração resulta de um par onde não ocorreu reversão nessa transição. Seja q um nó, vizinho de i, no qual tenha ocorrido uma reversão

parcial do par de arestas e um vizinho j no qual não tenha ocorrido uma reversão ocasionada pela transição de estado. Se existe um ciclo em $\mathcal{G}^+$, esse ciclo deve passar obrigatoriamente pelas arestas

$$(i,q) \in \mathcal{E}^+ \wedge \omega((i,q)) = q$$

e

$$(i,j) \in \mathcal{E}^+ \wedge \omega((i,j)) = i$$

mostradas na Figura 3.14 como arestas de linha cheia. Além disso, sabemos que $s(q) = R$ ou W e $s(j) = H$, por causa do Lema 2. Então, seja um ciclo no grafo $\mathcal{G}^+$, que contenha o nó i, formado pela seguinte seqüencia de nós:

$$i \rightarrow q... \rightarrow k \rightarrow ... \rightarrow j \rightarrow i.$$

Sabemos que nesse caminho existe um nó k onde $s(k) = H$ e $s(anterior(k)) = W$, no limite $k = j$ e $anterior(k) = q$. Então, provamos por absurdo a existência do referido ciclo pois o Lema 2 garante que

$$\omega((k, anterior(k), \oplus)) = anterior(k),$$

determinando a orientação da aresta em $\mathcal{G}^+$ e que

$$\omega((k, anterior(k), \ominus)) = k,$$

determinando a direção da aresta em $\mathcal{G}^-$. □

Lema 7 *Uma transição de estado ξ^{HW} não gera ciclos nos grafos $\mathcal{G}^+$ e $\mathcal{G}^-$.*

Prova: Apresentamos na Figura 3.15 um nó i que acabou de executar uma transição de estado ξ^{HW} e supomos que $\mathcal{G}^+$ e $\mathcal{G}^-$ são acíclicos antes da transição. Temos considerar três pares de arestas distintos:

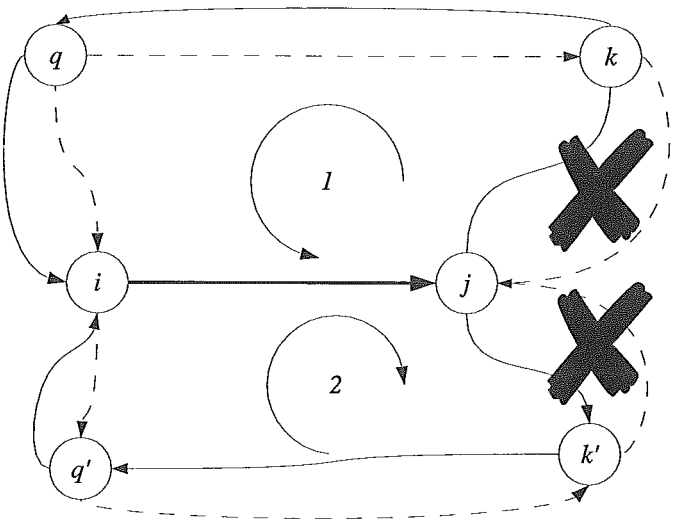


Figura 3.15: Ciclos hipotéticos em uma transição ξ^{HW}.

- um par concordante, orientado para i, resultante da mensagem $\mathbb{PP}$ que desencadeou ξ^{HW};
- pares discordantes do tipo

$$\omega((i,*,\oplus)) = i \wedge \omega((i,*,\ominus)) \neq i,$$

que não foram revertidos pelo procedimento executado por ξ^{HW};

- pares concordantes na forma

$$\omega((i,*,\oplus)) \neq i \wedge \omega((i,*,\ominus)) \neq i,$$

resultantes do envio da mensagem $\mathbb{MM}$ pelo procedimento executado por ξ^{HW}.

Vamos analisar os dois ciclos que possam existir em $\mathcal{G}^+$, indicados por 1 e 2 na Figura 3.15. Seja $q \in N_i^{viz}$, que através de um evento ξ^{RW} acordou o nó i e seja $j \in N_i^{viz}$, que teve sua aresta de cor $\ominus$ revertida pela transição ξ^{HW}. Se existe um ciclo em $\mathcal{G}^+$, como indicado por 1 na Figura 3.15, esse ciclo tem que obrigatoriamente

passar por um nó k que seja vizinho de q e tenha $s(k) = H$ (no limite, $k = j$), porque somente nesse caso a aresta de cor $\oplus$ estaria orientado para q. Então seja um ciclo que passe pelos nós i, q e j, tendo a seguinte seqüencia de nós:

$$i \rightarrow q \rightarrow ... \rightarrow k \rightarrow ... \rightarrow j \rightarrow i.$$

Nessa configuração, mostramos que não existe o referido ciclo analisando o que acontece no nó j em cada estado, como mostrado a seguir:

- Se $s(j) = R$, temos duas possibilidades. A primeira é que o nó j decide acordar o nó i. Nesse caso, a aniquilação de mensagens que ocorre no canal reverte o par de arestas para i, não gerando ciclos em $\mathcal{G}^+$. A segunda acontece quando j decide não acordar i, passando para o estado W, que veremos a seguir;
- Se $s(j) = W$, temos que o SER^{VT} executado no nó j reverte o par de arestas para i, não gerando ciclos em $\mathcal{G}^+$;
- Se $s(j) = H$, temos que o SER^{VT} executado no nó j reverte a aresta de cor $\oplus$ para i, não gerando ciclos em $\mathcal{G}^+$.

Além disso, nesse possível ciclo, não é gerado ciclo em $\mathcal{G}^-$ porque q se torna uma fonte em $\mathcal{G}^-$.

No possível ciclo mostrado por 2 na Figura 3.15, temos o seguinte raciocínio: seja um nó $q' \in N_i^{viz}$ no qual não tenha ocorrido reversões ocasionadas pelo procedimento de ξ^{HW}, mas que tenha o seu par discordante como mostrado na Figura 3.15. Seja um nó $j \in N_i^{viz}$, que tenha sua aresta de cor $\ominus$ revertida pela transição ξ^{HW}. Se existe um ciclo em $\mathcal{G}^+$, esse ciclo obrigatoriamente passa por um nó k' (no limite $k' = j$) que esta no estado de hibernação. Nesse caso, temos que o código do SER^{VT} executado no nó j reverte a aresta de cor $\oplus$ no par existente entre i. Além disso, a aresta de cor $\ominus$ tem sua orientação para o nó j, não gerando

ciclos em $\mathcal{G}^-$. □

Teorema 1 *A aciclicidade de $\mathcal{G}^+$ e $\mathcal{G}^-$ é preservada com a operação do SER^{VT}.*

Prova: A aciclicidade é mantida como demonstrada pelos lemas 5, 6 e 7. A transição de estado ξ^{WR} não gera modificações nos grafos, como mostrado pelo lema 4.□

O Teorema 1 garante a não existência de *deadlocks* e que $\mathcal{G}^{ativo}$ permanece acíclico durante a execução do SER^{VT}. A aciclicidade em $\mathcal{G}^{ativo}$ garante que todos os resultados (eqüidade e concorrência) sobre o SER são válidos nessa parte do sistema. Além disso, todas as transições são efetuadas com um número finito de mensagens, que no pior caso é igual ao número de vizinhos de um nó. No Capítulo 4, vamos mostrar comparações com a solução do DPP proposta por Chandy e Misra, onde procuramos observar a eficiência do SER^{VT} em relação ao número de mensagens e como a obrigatoriedade de ser acordado por um nó vizinho ativo afeta a carga do sistema.

Capítulo 4

Utilização do SER^{VT}

O SER^{VT} apresenta um mecanismo de ativação e desativação diferente da solução do DPP apresentado por Chandy e Misra. No caso do DPP, o nó pode sair do estado *pensando* para com o estado *fome* em qualquer momento da execução. Já no SER^{VT}, um nó só pode sair do estado de hibernação através do pedido de um nó ativo que seja seu vizinho. Podemos visualizar essa diferença através de um simples modelo para simulação de um sistema síncrono. Na simulação do algoritmo de Chandy e Misra para a solução do DPP, os nós que se encontram no estado *pensando* decidem em cada ciclo do relógio se eles devem ir para o estado *com fome*, com uma determinada probabilidade p_h. O algoritmo entra em ação, garantindo a exclusão mútua entre os nós *com fome*. Contudo, no SER^{VT}, temos que na saída de um estado *executando* (R), o nó deve decidir, baseado em uma probabilidade p_r, se deve continuar ativo. Se ele não for hibernar, ele deve decidir, baseado em uma probabilidade p_w, se deve acordar um vizinho que está hibernando. Além disso, existem casos em que o nó não pode hibernar, porque ele é o único nó ativo cercado de vizinhos inativos. Nesse caso, ele é obrigado a acordar um vizinho e permanecer ativo.

Essa simulação síncrona foi executada e neste capítulo mostramos como p_r e p_w afetam o funcionamento do sistema (Seção 4.1) e fazemos uma comparação entre o SER^{VT} para demandas similares (Seção 4.2) com a solução de Chandy e Misra.

Na Seção 4.3, fazemos uma simulação na qual os nós operam sinais de trânsito de um cruzamento, onde apresentamos uma maneira de modelar o problema usando o SER^{VT} e diminuindo o impacto do mecanismo de ativação e desativação. Nessa simulação, os sinais de trânsito são os ursos do problema de Pangéia e as regiões do cruzamento são os lugares de pesca.

4.1 Ativação e desativação no SER^{VT}

Para a ilustração do impacto do mecanismo de ativação e desativação no SER^{VT}, fizemos uma simulação síncrona onde cada nó que passava pelo estado R, se existia pelo menos um vizinho ativo, decidia se iria para o estado W com uma probabilidade p_r. Se o nó continuasse ativo, ele deveria, para cada vizinho no estado H decidir acorda-lo baseado em um probabilidade p_w. Essa simulação, desenvolvida em C++, foi executada para um sistema que representava um clique de 7 nós (K_7) com $0 < p_r \leq 1$ e $0 < p_w \leq 1$. Cada ciclo da simulação foi quebrado em três fases (a, b e c), onde cada nó executava uma seqüencia de tarefas como reproduzido abaixo:

- Fase a:
    1. receber as mensagens que foram enviadas, aplicando o esquema de aniquilação de mensagens;
    2. atualizar os contadores de mensagem*;
- Fase b:
    1. atualizar o estado local;
    2. atualizar os contadores globais de estado;
- Fase c:

*o par de mensagens que é aniquilado em um canal é contabilizado na contagem global de mensagens.

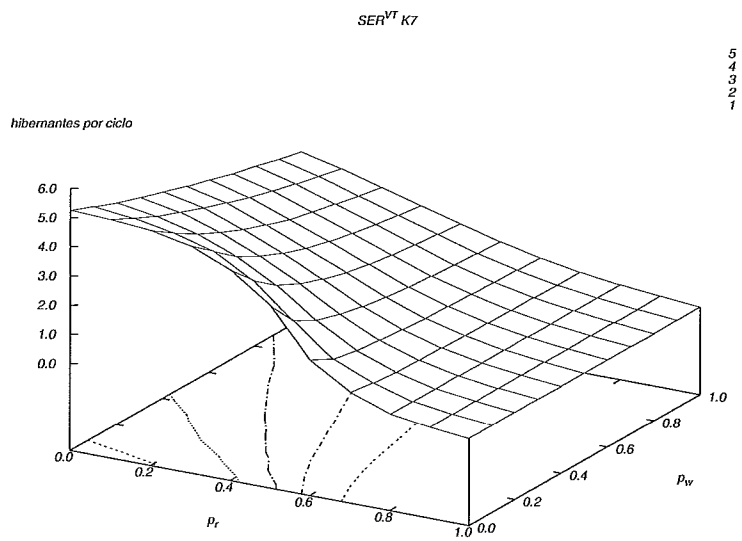


Figura 4.1: Número de nós hibernantes por ciclo no SER^{VT} para $G = K_7$, $0 < p_r \leq 1$ e $0 < p_w \leq 1$.

1. nos nós com estado R, sortear o próximo estado e sortear vizinhos para serem acordados, se necessário;

2. enviar mensagens;

3. atualizar os contadores de mensagens.

Como primeiro resultado da simulação, apresentamos a superfície da Figura 4.1. Essa superfície é formada pelo pontos obtidos na simulação, onde no eixo z, temos o número médio de processos no estado de hibernação por ciclo, que foi obtido pela contagem em cada ciclo do número de nós no estado H (efetuada na Fase b), divido pelo número de ciclos da simulação[†]. No eixo x temos a probabilidade de um processo continuar ativo (p_r) depois de utilizar os recursos e no eixo y temos a probabilidade de um processo acordar um vizinho hibernante (p_w).

Para a análise da Figura 4.1, dividimos o plano $p_r p_w$ em quatro regiões distintas, de acordo com a demanda do sistema, como mostrado na Tabela 4.1. Na Região

[†]foram executados 10^3 ciclos de aquecimento, onde não foi feita a contagem, seguidos de 10^4 ciclos efetivos.

Tabela 4.1: Regiões de demanda.

	$p_r \leq 0,5$	$0,5 < p_r \leq 1$
$p_w \leq 0,5$	I	II
$0,5 < p_w \leq 1$	III	IV

I, ou região de baixa carga, podemos observar o principal impacto no sistema de ativação e desativação do SER^{VT}. Apesar da baixa demanda do sistema, temos o número médio de nós *ativos* elevado (sempre maior que 1,0), i.e., o número de nós no estado de hibernação H é sempre maior que 5. Esse fato é decorrente do nó só poder hibernar se existe pelo menos um vizinho ativo no sistema. Não acontecendo essa pré-condição, o nó permanece ativo no sistema e tenta acordar um vizinho, baseado na probabilidade p_w. Na Região II, temos que p_r é mais influente no número de hibernantes do que p_w, pois existe um número menor de vizinhos que podem ser acordados. Na Região III, o sistema funciona como um *flip-flop*. Os nós vão hibernando até que o último ativo acorde todos novamente para mais uma seqüencia de desativação. Na Região IV, o sistema opera com alta demanda, onde o número médio de processos inativos é próximo de zero.

Para verificar o comportamento do número de hibernantes em relação ao número de nós, executamos a mesma simulação para uma clique de 30 nós ($K30$). Os resultados se encontram na Figura 4.2. Nesse caso, podemos observar que o comportamento *flip-flop* do sistema passa a ocorrer na Região I, porque existe um número maior de nós que pode ser acordado, por causa do maior número de sorteios efetuados, mesmo com valores baixos de p_w. Além disso, temos que as regiões II e IV tem comportamentos similares, pois rodando com uma clique maior, o mecanismo de exclusão mútua faz que os processos ativos passem por um número maior de ciclos para poder ter a oportunidade de hibernar novamente.

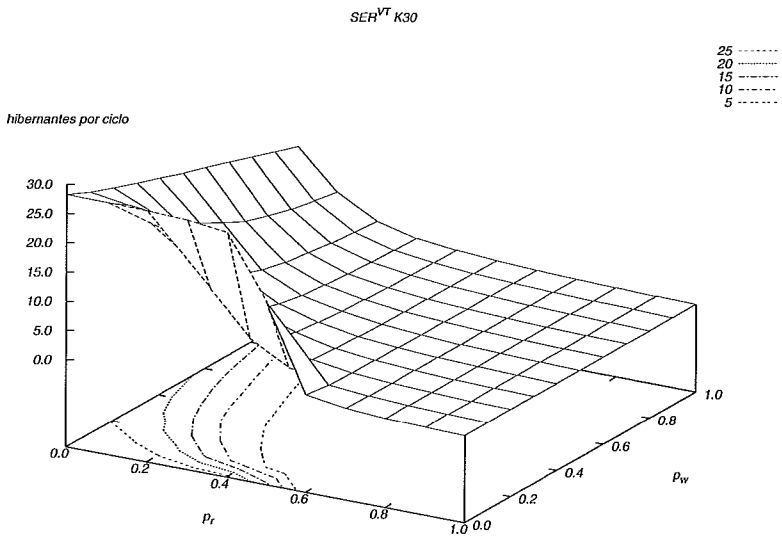


Figura 4.2: Número de nós hibernantes por ciclo no SER^{VT} para $G = K_{30}$, $0 < p_r \leq 1$ e $0 < p_w \leq 1$.

Além do número de processos inativos, fizemos a contagem do número de mensagens trocadas pelo algoritmo, levando em consideração as mensagens extras que são necessárias para a implementação da aniquilação de mensagens[‡]. Os resultados são apresentados na Figura 4.3. Nela podemos observar que o máximo de mensagens ocorre na região definida por $0{,}4 \leq p_r \leq 0{,}6$ e $0{,}4 \leq p_w$ e encontramos o mesmo número de mensagens utilizado no SER quando $p_r = 1$. Outro ponto interessante é que operando com probabilidades $0.6 < p < 1$, o SER^{VT} gasta mais mensagens que o SER, mesmo tendo nós que estão desligados do sistema. Esse fato mostra que o número de mensagens usadas nas ativações foi bastante significativo. Isso ocorre pela a forma que compomos as probabilidades e restrições do sistema. Teríamos um desempenho melhor se o problema resolvido fosse um grafo com uma conectividade menor e com partes significativas desligadas por grandes períodos de tempo. De uma maneira geral, temos esse comportamento mostrado pela a seguinte equação:

$$m_{total} = m_{\mathcal{G}^{ativo}} + m_{\mathcal{G}^0},$$

[‡]mostraremos na simulação do cruzamento o protocolo de sincronização de mensagens

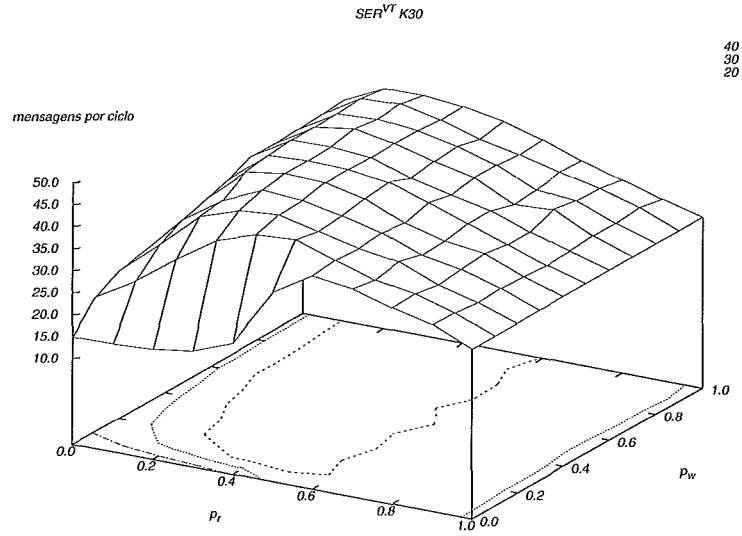


Figura 4.3: Número de mensagens por ciclo no SER^{VT} para $G = K_{30}$, $0 < p_r \leq 1$ e $0 < p_w \leq 1$.

onde o número total de mensagens é igual as mensagens trocadas em $\mathcal{G}^{ativo}$ que pode ser considerado como um sistema reduzido operando em SER, mais as mensagens enviada pelos nós em ativação que se encontram na fronteira.

4.2 Comparação entre o SER^{VT} e a solução de Chandy e Misra

A comparação do SER^{VT} com o algoritmo apresentado por Chandy e Misra não pode ser feita diretamente por causa da diferença existente entre os mecanismos de hibernação como vimos na seção anterior. No caso dos filósofos, eles podem sair expontaneamente do estado pensando para o estado com fome. No SER^{VT}, os ursos que se encontram nas cavernas devem ser acordados pelos ursos em vigília que devem ser seus vizinhos.

Utilizando a mesma infraestrutura de simulação desenvolvida para o SER^{VT}, foi implementado o algoritmo proposto por Chandy e Misra para efetuarmos uma comparação. Nessa versão, cada filósofos pode sair do estado *pensando* para o estado

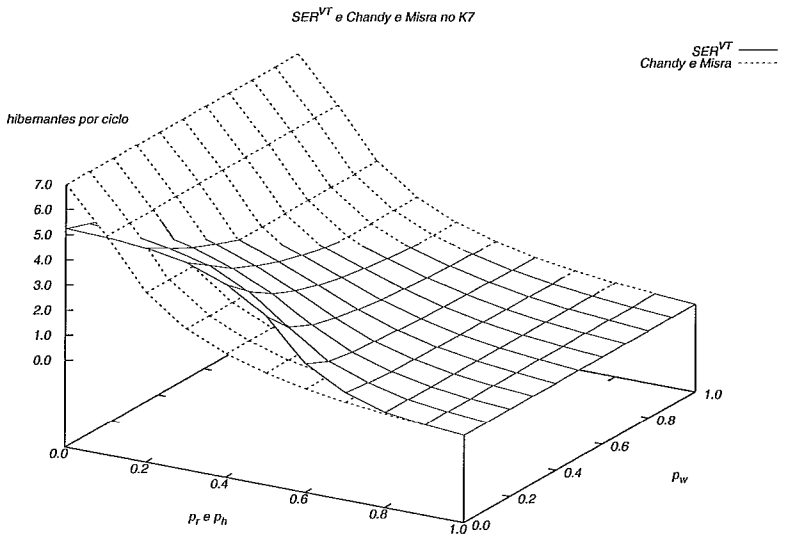


Figura 4.4: Número de hibernantes por ciclo no SER^{VT}e na solução de Chandy e Misra para $G = K_7$, $0 < p_r, p_h \leq 1$ e $0 < p_w \leq 1$.

com fome com uma determinada probabilidade p_h.

Apresentamos nas figuras 4.4 e 4.5 os resultados da simulação para uma clique de 7 nós ($K7$). Na Figura 4.4, podemos observar que os dois sistemas operam com o mesmo número de nós inativos na região onde $0.7 < p_h, p_r < 1$. Nessa Região, podemos notar que o SER^{VT} trabalha sempre com um número menor de mensagens, que pode ser até 50% menor quando $p_h, p_r = 1$, como mostrado na Figura 4.5. Esse fato ocorre porque o DPP necessita de duas mensagens entre cada nó vizinho para a passagem do direito de uso do recurso. No SER^{VT} o direito do uso do recurso é enviado com o uso de uma única mensagem entre cada vizinho ativo.

Os comportamentos médios similares que nos referimos acima não garantem que as seqüências de ativação e hibernação sejam similares, quando utilizamos os algoritmos na solução de um problema real. Para essa comparação aplicamos o SER^{VT} e o algoritmo de Chandy e Misra na solução de um mesmo problema onde analisaremos os resultados quando submetidos a mesma carga.

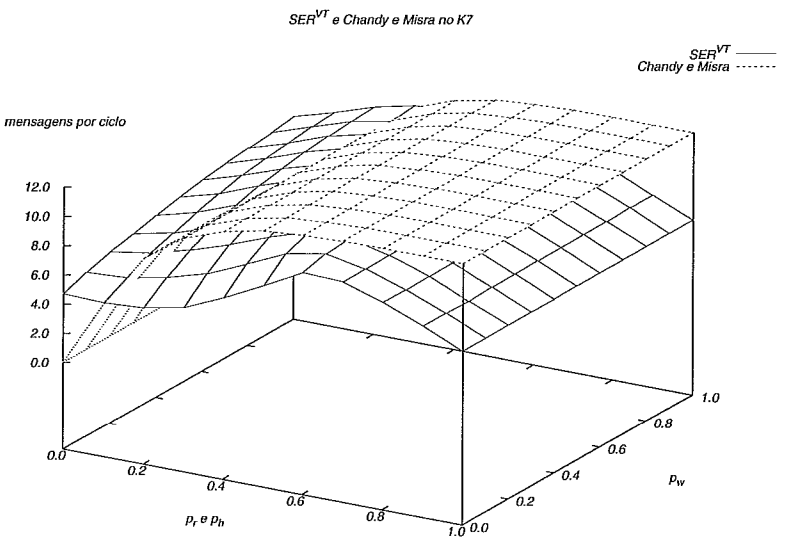


Figura 4.5: Número de mensagens por ciclo no SER^{VT} e na solução de Chandy e Misra para $G = K_7$, $0 < p_r, p_h \leq 1$ e $0 < p_w \leq 1$.

4.3 Problema do cruzamento viário

Neste problema, modelamos o cruzamento da figura 4.6 utilizando os dois algoritmos de exclusão mútua. Para tal, utilizamos um sistema assíncrono desenvolvido em linguagem C e bibliotecas MPI ([31] e [32]), que foi executado em um *cluster* com dois computadores bi-processados. A modelagem consiste de um grafo onde os nós são os sinais de trânsito, que na figura são representados pelos círculos etiquetados pelas letras $P_i, i \in N$, e os recursos compartilhados são as regiões marcadas na figura pelas etiquetas R. As arestas do grafo foram construídas seguindo a regra apresentada na seção 2.2. O grafo resultante é apresentado na figura 4.7.

Além de um processo por sinal, o sistema tinha um processo, chamado de Zero e não desenhado nas figuras, responsável pela distribuição de carga entre os processadores que funcionava da seguinte maneira. Primeiro, o processo Zero, utilizando probabilidades pré-estabelecidas para cada pista ou ponto de travessia de pedestres, calculava o número de carros ou pedestres que chegam nas respectivas filas

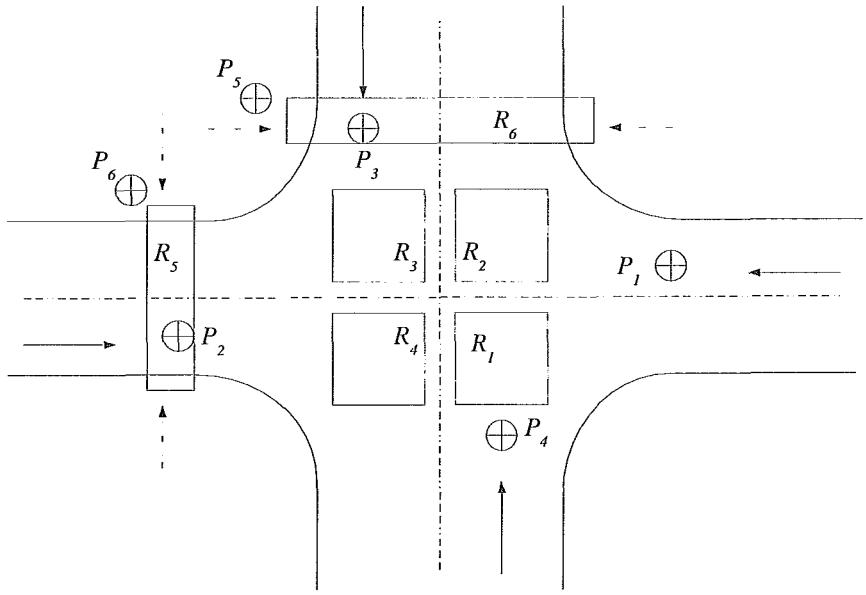


Figura 4.6: Cruzamento de Trânsito.

no próximo intervalo de tempo da simulação. Essa informação era enviada para os processadores dos sinais e eles trocavam mensagens de acordo com o algoritmo que estava sendo executado, fazendo a movimentação das filas. Em cada operação dos sinais, somente um número fixo de carros ou pedestres se movimentavam pelas vias. Se elas não fossem esvaziadas, o restante ficava para o intervalo de tempo seguinte. Todo o sistema tinha um número máximo de ativação por ciclo que determinava quantas vezes os sinais poderiam abrir por ciclo. Então, os processadores dos sinais retornavam o estado do sistema, liberando o processo Zero para a interação do próximo intervalo de tempo. As primeiras interações foram descartadas por serem consideradas como *aquecimento* do sistema.

Na execução do Chandy e Misra, os sinais que tinham suas filas com tamanho maior que zero trocavam para o estado com fome e esperavam o seu momento de operação. No SER^{VT}, foi adicionado um nó ligado a cada nó de sinal como mostrado na figura 4.8. Esse nó funciona como os ursos de vigília na porta das cavernas. Eles acordavam os sinais quando a fila era maior do que zero, mas os sinais só

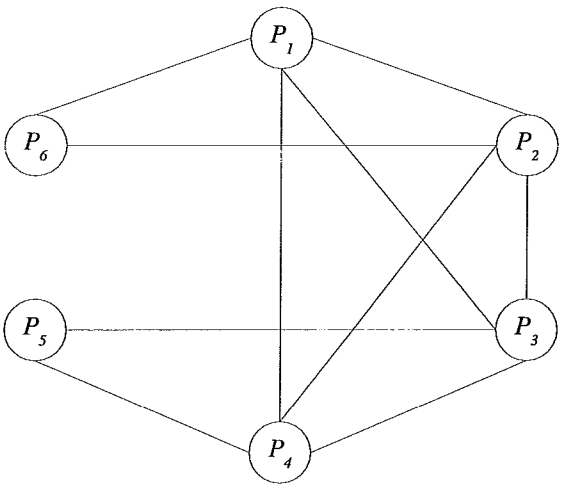


Figura 4.7: Grafo utilizado na modelagem do cruzamento.

hibernavam somente quando as filas fossem iguais a zero ou quando o número de ativações globais no ciclo tivesse se esgotado. O mapeamento dos novos nós foi feito nos mesmos processadores de cada sinal. O mapeamento do sistema nos processadores dos computadores foi feito segundo [33], que trata sobre o mapeamento de nós/processadores no SER. Além disso, o aniquilamento de mensagens foi feito com a introdução de uma nova mensagem de confirmação, permitindo a identificação do envio de mensagens simultâneas.

As mensagens dos tipos PP e MM eram trocadas com a utilização de um protocolo de *hand-shake*, ou seja, toda a vez em que um nó desejava transmitir alguma mensagem, ele primeiro enviava uma mensagem especial de sincronização. O outro nó deveria responder com uma mensagem especial de aceite e finalmente o nó enviava a mensagem desejada (PP ou MM). No caso em que dois nós iniciavam a comunicação simultaneamente, no lugar da mensagem de aceite, os nós recebiam a mensagem especial de sincronização. Nesse caso, os nós transmitiam os seus dados e aplicavam as regras de aniquilação de mensagens. Esse mecanismo se faz necessário

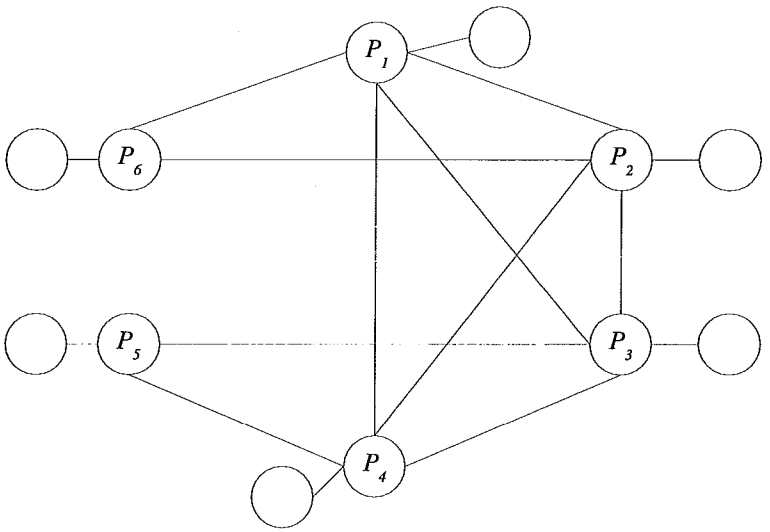


Figura 4.8: Grafo utilizado na modelagem do cruzamento com SER^{VT}.

Tabela 4.2: Resultados da simulação do cruzamento viário.

	Fator de carga pedestre	Fator de carga via	mensagens (SER^{VT}/C&M)
I	0,1	0,8	0,83
II	0,5	0,8	0,79
III	0,8	0,8	0,72

por que não temos uma base de tempo comum ou outro mecanismo para quebrar a simetria existente nessa situação. Todos os valores de quantidades de mensagens apresentados nesse capítulo levam em consideração as mensagens extras que são necessárias na sincronização.

Fizemos três execuções do sistema. Em cada uma, foi variado o valor do fator de carga nos sinais de pedestre, como mostrado na Tabela 4.2. O fator de carga é a probabilidade da chegada de um novo grupo de veículos ou pedestres na fila do respectivo sinal. Nessa simulação, utilizamos grupos de 4 carros e 8 pedestres. Em cada execução, os tempos de operação dos sinais comportava a passagem de 4 carros ou 8 pedestres, dependendo do caso, e cada ciclo da simulação permitia 6 operações dos sinais de trânsito.

A Tabela 4.2 apresenta também a razão entre o número de mensagens utilizadas por Chandy e Misra e o número de mensagens no SER^{VT}. Deve ser notado que o SER^{VT} sempre utiliza um número menor de mensagens que o algoritmo referência. Além disso, notamos que a introdução do nó auxiliar consegue reproduzir um mecanismo de hibernação similar ao Chandy e Misra. Esse fato, sugere que para a modelagem de outros problemas, devemos colocar um nó responsável somente para acordar os outros nós que necessitam se adaptar a demandas variáveis.

Capítulo 5

Conclusões e trabalhos futuros

Nesse Trabalho, apresentamos o Problema dos Ursos de Pangéia, onde os ursos, ou processadores, passam por períodos de hibernação e por regimes de operação em alta carga. Essa situação particular dos ursos pode servir de modelo para diversos problemas, como por exemplo:

- circuitos digitais para micro-arquitetura de processadores, onde unidades funcionais podem ficar no estado de hibernação, economizando energia, até que ocorra a necessidade de processamento, momento no qual elas necessitam acessar registradores e caminhos de dados compartilhados;
- circuitos digitais para comutação de alto desempenho, onde caminhos secundários podem ficar hibernando até a ocorrência de um surto de alta demanda, fazendo que não ocorra interrupção de serviço;
- equipamentos para aquisição de dados, onde as informações dos eventos de aquisição podem variar em taxa, obrigando que outras partes dos detectores entrem em ação para adaptação ao novo regime de demanda.

Para a solução do problema dos ursos, foi proposto o algoritmo de escalonamento por reversão de arestas com variação de carga ou topologia (ou, em inglês, *Scheduling by Edge Reversal with Varying Topology* – SER^{VT}, que teve a sua apresentação

formal e corretude mostrados no Capítulo 3. Além disso, foi mostrado que por projeto, o algoritmo tem um mecanismo de ativação e desativação mais restrito que os algoritmos normalmente encontrados na literatura. Os efeitos desse mecanismo foram estudados no Capítulo 4, além das comparações com o algoritmo proposto por Chandy e Misra para a solução do problema dos filósofos. Nessas comparações, foi mostrado que o SER^{VT} opera com um número de mensagens inferior que o algoritmo de Chandy Misra, na faixa de demanda para a qual ele foi projetado.

Esse trabalho despertou o interesse no estudo de adaptação do SER^{VT}ao SMER [34], permitindo que os processadores possam operar com freqüências diferentes no período, quando operando em alta demanda. Além disso, estamos interessados no estudo do comportamento das aplicações do SER^{VT}, nas economias proporcionadas pelo seu emprego e em novas modelagens que permitam o seu uso em sistemas mais diversos.

Referências Bibliográficas

[1] E. W. Dijkstra. Hierarchical ordering of sequencial processes. *Acta Informatica*, 1(2):115–138, 1971.

[2] Valmir Barbosa and Eli Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Transactions on Programming Languages and Systems*, 11(4):562–584, October 1989.

[3] Valmir Carneiro Barbosa. *Concurrency in Systems with Neighborhood Constraints.* Ph.D. thesis, UCLA Computer Science Department, Los Angeles, 1986.

[4] Valmir C. Barbosa. *An Introduction to Distributed Algorithms.* MIT Press, Cambridge, MA, 1996.

[5] Claude Berge. *Graphes et Hypergraphes.* Dunod, Paris, 1970.

[6] Paulo Oswaldo Boaventura Netto. *Grafos: Teoria, Modelos, Algoritmos.* Edgard Blücher, São Paulo, 1996.

[7] L. Lamport. Time, clocks, and the ordering of events in a distributed system. In *Communications of the ACM*, pages 558–565, July 1978.

[8] K. M. Chandy and Jayadev Misra. The drinking philosopher's problem. *ACM Transactions on Programming Languages and Systems*, 6(4):632–646, October 1984.

[9] M. Velazquez. A survey of distributed mutual exclusion algorithms. *Technical Report CS-93-116, Colorado State University*, 1993.

[10] Michel Raynal. A simple taxonomy for distributed mutual exclusion algorithms. *Operation Systems Review*, 25:47–50, 1991.

[11] D. Agrawala and A. El Abbadi. An efficient solution to the distributed mutual exclusion problem. In *Proceedings of the* 8^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 193–200, August 1989.

[12] D. Agrawala and A. El Abbadi. Exploiting logical structures in replicated databases. *Inf. Proc. Letters*, 33:255–260, 1990.

[13] O. S. F. Carvalho and G. Roucariol. On mutual exclusion in computer networks. *Communications of ACM*, 26(2):145–147, 1983.

[14] H. Garcia Molina and D. Barbara. How to assign votes in a distributed system. *Jornal of the ACM*, 32(4):150–159, 1985.

[15] K. D. Gifford. Weighted voting for replicated data. *Proceedings of the* 7^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 150–159, 1989.

[16] N. Plouzeau J. M. Helary and M. Raynal. A distributed algorithm for mutual exclusion in an arbritary network. *The Computer Journal*, 31(4):289–295, 1988.

[17] M. A. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, 1985.

[18] Injong Rhee. A fast distributed modular algorithm for resource allocation. In *Proceedings og the* 15^{th} *International Conference on distributed Computer Systems (ICDCS '95)*, pages 161–168, 1995.

[19] G. Le Lann. Distributed systems: towards of formal approach. In *IFIP Congress*, pages 155–160, North-Holland, 1977.

[20] N. A. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. *Proceedings of* 7^{th} *ACM Symposium on Operating Systems Principles*, pages 137–151, 1987.

[21] J. A. Martin. Distributed mutual exclusion on a ring of processors. *Science of Computer Programming*, 5:256–276, 1985.

[22] M. Naimi and M. Treiiel. A distributed algorithm for mutal exclusion based on data structures and fault tolerance. *Proceedings of* 6^{th} *IEEE International Conference on Computer and Communications*, pages 35–39, 1987.

[23] K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.

[24] J. L. Welch and N. A. Lynch. A modular drinking philosophers algorithm. *Distributed Computing*, 6:233–244, 1993.

[25] Yuh-Jzer Joung. Asynchronous group mutual exclusion. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing (PODC '98)*, pages 51–60, New York, June 1998. Association for Computing Machinery.

[26] Yuh-Jzer Joung. The congenial talking philosophers problem in computer networks. *DISTCOMP: Distributed Computing*, 15, 2002.

[27] Walter, Welch, and Vaidya. A mutual exclusion algorithm for ad hoc mobile networks. *Wireless Networks: The Journal of Mobile Communication, Computation and Information, Kluwer*, 7, 2001.

[28] Eli M. Gafni and Dimitri P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 29(1):11–18, 1981.

[29] J. Kramer and J. Magee. The Evolving Philosophers Problem: Dynamic Change Management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, November 1990.

[30] Kenneth Goldman and Joe Hoffert. A modification to the chandy-misra dining philosophers algorithm to suport dynamic resource conflict graphs. submetido para Information Processing Letters.

[31] Jeffrey M. Squyres and Andrew Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, Venice, Italy, September/October 2003. Springer-Verlag.

[32] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, Message Passing Interface Forum, 1994.

[33] F. M. G. França and L. Faria. Optimal mapping of neighbourhood-constrained systems. In *Proceedings, IRREGULAR '95*, volume 980 of *Lecture Notes in Computer Science*, pages 165–170, Lyon, France, 1995. Springer-Verlag.

[34] V. C. Barbosa, M. R. F. Benevides, and F. M. G. França. Sharing resources at nonuniform access rates. *Theory Comput. Systems*, 34:13–26, 2001.