Uma Comparação Entre Dois Algoritmos de Exclusão Mútua para Redes de Computadores*

Kêmio de Oliveira Couto† Marco Aurélio de Souza Mendes‡
Osvaldo S. F. Carvalho §

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
31270-010 Belo Horizonte, MG, Brasil

Resumo

Este trabalho faz uma comparação por simulação de dois algoritmos de exclusão mútua para sistemas distribuídos. Os algoritmos avaliados (Maekawa) e (Carvalho e Campos) utilizam um esquema de comunicação definido através de um plano projetivo finito a fim de minimizar o número de mensagens enviadas por pedido de exclusão mútua. A comparação aborda o número de mensagens gastas por cada pedido de exclusão mútua, o tempo de espera pelo recurso compartilhado, a carga sobre este e o tempo total de ciclo. Decorre deste trabalho que o algoritmo de Carvalho e Campos deveria ser usado em situações de carga alta sobre o recurso e sistemas não balanceados enquanto que o algoritmo de Maekawa é preferível nas situações de carga baixa e sistemas balanceados.

Abstract

This work compares two mutual exclusion algorithms for distributed systems by simulation. The two evaluated algorithms (Maekawa) and (Carvalho and Campos) use a communication scheme defined by a finite projective plane in order to minimize the number of sent messages by exclusion mutual request. The comparison takes into account the number of messages by mutual exclusion request, the waiting time and the load over the shared resource and the total cycle time. We conclude that the algorithm of Carvalho and Campos should be used in situations of high load over the resource and not balanced systems while the algorithm of Maekawa should be used in low load and balanced system situations.

*Este trabalho foi financiado com recursos da FAPEMIG (TEC 1113/90) e do CNPq (502353/91-O(NV))
†UFMG - email: chalub@dcc.ufmg.br UFMG - ICEX - DCC
‡UFMG - email: corelio@dcc.ufmg.br UFMG - ICEX - DCC
§UFMG - email: vado@dcc.ufmg.br UFMG - ICEX - DCC

1 Introdução

A exclusão mútua entre processos em um sistema distribuído é uma importante ferramenta para a resolução de muitos problemas de sincronização. Tais problemas definem, muitas vezes, a performance obtida por um sistema distribuído no gerenciamento de suas tarefas. Decorre deste fato a importância de uma análise comparativa entre os diversos algoritmos que se prestam a resolver o problema de exclusão mútua em sistemas distribuídos a fim de uma melhor utilização dos mesmos.

Neste artigo são comparados os algoritmos de Maekawa e de Carvalho e Campos. Tais algoritmos utilizam um esquema de comunicação definido através de um plano projetivo finito o que lhes garantem um esquema ótimo no que se refere ao número de mensagens enviadas por invocação de seção crítica por parte de um processo.

Este artigo primeiramente enuncia o conceito de um plano projetivo finito para a seguir descrever os algoritmos a serem comparados. A seguir, o modelo de simulação utilizado para a análise dos algoritmos é definido para então ser feita a análise dos resultados obtidos quando da simulação dos algoritmos. Por último, uma análise conclusiva indica as melhores situações de uso para ambos os algoritmos.

2 Descrição Geral

N processos de um sistema distribuído concorrem por um recurso comum. Os processos não compartilham memória nem utilizam um relógio físico comum. Quando um processo está utilizando este recurso diz-se este está em sua seção crítica. A propriedade de exclusão mútua deve garantir que nenhum outro processo esteja simultaneamente em sua seção crítica.

Os algoritmos supõem que a transmissão de mensagens não apresente erros e que a ordem de envio das mensagens seja mantida. Cada nodo do sistema deve, ainda, assegurar a serialização da demanda, i.e., no máximo uma demanda está ativa por vez e o recurso deve ser liberado antes que o nodo possa processar a próxima demanda.

Como citado anteriormente, o esquema de comunicação utilizado pelos algoritmos é definido através de um plano projetivo finito. Um plano projetivo finito consiste de um conjunto finito de pontos e linhas, cujas linhas são conjuntos de pontos, satisfazendo as seguintes propriedades:

- Dois pontos distintos estão em uma e somente uma linha comum.
- Duas linhas distintas passam através de um e somente um ponto comum.

Os planos projetivos finitos são usados a fim de determinar os conjuntos (S_i e R_i)[1] associados a cada nodo do sistema da seguinte maneira:

Cada nodo é considerado como um ponto e o plano projetivo finito é construído com todos os nodos do sistema. O conjunto S_i é definido como uma linha do plano. O conjunto R_i contém como elementos os nodos j tal que $i \in S_j$. O nodo i comunica-se, portanto, somente com os elementos de S_i e R_i. Uma redução no número de mensagens enviadas é alcançada se um nodo i qualquer do sistema fizer parte de seu conjunto S_i.

Segue a seguinte propriedade para os conjuntos S_i formados:

$$\forall i \forall j \; i \neq j \Rightarrow |S_i \cap S_j| = 1, |S_i| = |S_j| = k = O(\sqrt{n})$$

Cada nodo do sistema, portanto, age como um árbitro (mediando os pedidos de exclusão mútua dos elementos de seu conjunto R_i) e como cliente (enviando pedidos de exclusão mútua para os elementos de seu conjunto S_i).

Os algoritmos comparados são brevemente descritos a seguir.

[1] Nestes algoritmos, ao invés de uma comunicação direta entre um nodo e os nodos restantes dos sistema, é usado uma comunicação indireta com nodos agindo como representantes de um conjunto de outros nodos. O conjunto S_i indica os nodos aos quais um nodo i deverá enviar pedidos de exclusão mútua. O conjunto R_i define os elementos dos quais o nodo i receberá pedidos de exclusão mútua.

2.1 O Algoritmo de Maekawa

Cada nodo i executa um algoritmo idêntico. O algoritmo é baseado no fato de que, se um nodo i consegue conectar todos os membros de seu conjunto S_i, nenhum outro nodo consegue capturar todos os seus membros devido à propriedade de interseção não nula entre quaisquers S_i e S_j para qualquer par de nodos i e j do sistema. Portanto, quando este nodo requer exclusão mútua, ele tenta conectar todos os membros de S_i. Se isto ocorre, o nodo entra na sua seção crítica. Em caso de falha, o nodo espera até que todos os membros de seu conjunto estejam livres, em cujo ponto conecta todos eles.

Desde que existe o perigo de *deadlock* quando mais que um nodo requer simultaneamente exclusão mútua, um nodo precisará saber dos outros nodos se a prioridade de suas requisições é menor que qualquer outra requisição conflitante. A prioridade de requisições é determinada pela seqüência numérica (*TimeStamp*) [1] da requisição correspondente à mensagem de **REQUEST**.

A um **REQUEST** com menor seqüência numérica é atribuído uma prioridade maior e é dito preceder outras requisições com seqüência numérica maior. Se uma nova requisição recebida no nodo membro precede a requisição que gerou a atual conexão, então este nodo pergunta ao nodo conectado se este obteve sucesso em conectar todos os seus membros. O nodo conectado retornará uma mensagem de desistência se tornar-se aparente que este nodo não conseguirá conectar todos os seus membros. Por outro lado, se este nodo conseguir conectar todos os seus membros, então uma mensagem de **RELEASE** será enviada somente depois que este tiver completado sua seção crítica.

A descrição detalhada deste algoritmo é encontrada em [2].

2.2 O Algoritmo de Carvalho e Campos

Este algoritmo resulta de uma fusão de idéias dos algoritmos de Maekawa e Chandy-Misra [7]. Tal algoritmo é uma implementação do algoritmo de exclusão mútua dos filósofos à mesa de jantar usando um esquema de comunicação similar ao utilizado por Maekawa. Ao invés de uma comunicação direta entre qualquer par de filósofos, é utilizado uma comunicação indireta com nodos agindo como representantes para um conjunto de outros nodos. O esquema do plano projetivo finito é utilizado aqui, de modo similar ao algoritmo anterior, para minimizar o número de mensagens enviadas.

A resolução de conflitos neste algoritmo é realizada de maneira bem diferente do algoritmo de Maekawa. Enquanto o algoritmo de Maekawa utiliza o conceito de *TimeStamps*, este algoritmo utiliza a profundidade de um nodo em um grafo de conflitos acíclico para a resolução de conflitos. A aciclicidade do grafo evita a ocorrência de *deadlocks* e o grafo é modificado de maneira tal que qualquer nodo alcança a profundidade zero no grafo (maior prioridade possível) depois de um número finito de conflitos perdidos.

Cada nodo age como um cliente (que envia pedidos de exclusão mútua aos nodos de seu conjunto S_i) e como um árbitro (que faz a mediação dos pedidos dos nodos de seu conjunto R_i). A autorização de um árbitro é dada através de um *permission token*, concedido a no máximo um de seus clientes por vez. Entre um cliente e cada um de seus árbitros circula um *request token*, usado por ambos para perguntar sobre a permissão.

A fim de representar o grafo de precedencia, utilizado para resolução de conflitos, é adicionado ao *permission token* um estado que pode ser *clean* ou *dirty*, e a cada árbitro uma lista de prioridade local que contém todos os seus clientes. Quando um cliente detém o recurso comum, então ele suja todos os seus *permission token*. Um cliente transmite um *permission token* no estado em que ele está, mas somente permissões limpas são enviadas pelos árbitros.

O protocolo completo para este algoritmo é encontrado em [3].

2.3 Decisões de Implementação

Os protocolos contidos em [2] e [3] foram usados para a implementação efetiva dos algoritmos. Cabe notar que o formato do protocolo descrito em [2] já se encontra em descrição procedimental e assim foi simplesmente transcrito. Já o algoritmo de Carvalho e Campos foi descrito de maneira reativa o que tornou o processo de implementação mais flexível. Foram observadas duas situações onde duas mensagens poderiam ser fundidas em uma. Tais mensagens foram denominadas:

- **CLEANPERMISSION-REQUEST**: ocorre quando um árbitro concede a permisão a um cliente, mas já existem **REQUESTS** pendentes neste árbitro. Então, um **REQUEST** poderá ser anexada à uma mensagem de **CLEANPERMISSION**.
- **DIRTYPERMISSION-REQUEST**: ocorre quando um cliente é obrigado a devolver o garfo[2] sujo, mas já se encontra em estado de nova solicitação do recurso (**HUNGRY**). Essa mensagem engloba um **REQUEST** e uma mensagem de **DIRTYPERMISSION** do cliente para o árbitro. Ou seja, o cliente devolve a sua permissão ao mesmo tempo que volta a solicitá-la.

3 Simulação

3.1 Descrição do Modelo Utilizado

A fim de uma melhor análise dos resultados obtidos na simulação de um algoritmo, faz-se necessário a criação de um modelo que represente os diversos parâmetros de variação do mesmo. O modelo utilizado para a comparação aqui efetuada é baseado no modelo descrito e utilizado em [4] para a comparação de outros dois algoritmos de exclusão mútua.

A cada nodo do sistema são associados três estados distintos:

- **at Rest**: Estado associado a um período de latência do nodo.
- **Hungry**: Estado associado ao pedido do recurso pelo nodo.
- **Using**: Estado associado à utilização do recurso compartilhado por parte de um nodo.

O tempo de troca de mensagens entre quaisquers dois nodos do sistema é descrito como uma variável distribuída e aleatória (d). Cada nodo i do sistema possui dois parâmetros (U_i e R_i), respectivamente o tempo de utilização do recurso compartilhado e o tempo de latência ou descanso. Assim como d, U e R são variáveis distribuídas e aleatórias.

O número de mensagens enviadas por um nodo por demanda é definido como μ. Um melhor parâmetro de simulação, no entanto, é a definição de um número de mensagens normalizado definido como:

$$M = \frac{\mu}{MAX}$$

MAX é por definição o número máximo possível de mensagens a serem enviadas por demanda.

O tempo de espera pelo recurso compartilhado do sistema é definido como W. O tempo total de um ciclo, ou seja, a soma dos tempos de latência, tempo de espera pelo recurso e o tempo de utilização do mesmo é definido como Θ.

Cabe incluir também uma variável que indique o nível de utilização do recurso compartilhado. Este parâmetro, descrito como carga sobre o recurso compartilhado e representado como ρ é definido como se segue:

$$\rho = \frac{N * U}{\Theta}$$

Um ponto importante na utilização do modelo acima para a construção dos gráficos de simulação é o uso do valor de d como unidade. Isto implica que os valores de R, U e W representados nos gráficos a seguir são na realidade R/d, U/d e W/d.

[2]PERMISSION TOKEN

Na análise dos algoritmos feita a seguir, qualquer referência ao algoritmo de Maekawa é indicada por MA e as referências ao algoritmo de Carvalho e Campos são indicadas por CC.

Variável	
d	Tempo de troca de mensagens
R	Tempo de latência
U	Tempo de Uso do Recurso Compartilhado
μ	Número de mensagens
M	Número de mensagens normalizado
Θ	Tempo total de um ciclo
ρ	Carga sobre o recurso compartilhado

3.2 Análise dos Resultados

Número de Mensagens

Os gráficos 1 e 2 mostram o decréscimo da comunicacão média necessária em função do aumento do tempo de latência. Tais gráficos permitem fazer uma análise de M, que indica a carga de comunicação relativa.

Observa-se para valores pequenos de R, i.e, para uma carga grande sobre o recurso compartilhado, uma vantagem significativa do algoritmo de CC sobre o algoritmo de MA.

O algoritmo de MA apresenta um decréscimo do número de mensagens à medida que o valor de R aumenta. Para um valor de R tal que $R \gg d$, tem-se que $M \rightarrow 0.50$.

O algoritmo de CC exibe um interessante fenômeno para o número de mensagens enviadas. O valor de M cresce quando do crescimento de R até um determinado ponto e a partir daí apresenta um decréscimo. Nota-se, portanto, que o pior caso para o algoritmo de CC ocorre para redes cujos recursos compartilhados estejam em estados intermediários de uso. Além disso, observa-se que o valor máximo de M aproxima-se de 0.66 para este pior caso.

Tempo de Espera, Carga do Recurso e Tempo de Ciclo

Os gráficos de 3 a 8 mostram os resultados da simulação efetuada para tais variáveis em ambos os algoritmos.

Observa-se, para o tempo de espera (W), uma equiparação entre os algoritmos comparados. Ambos os algoritmos apresentam um mesmo tempo de espera para uma rede cujo recurso esteja sob forte utilização. À medida que o tempo de latência aumenta, nota-se um descréscimo mais acentuado para o algoritmo de MA. Quando o valor de R tende para valores muito grandes, ambos os algoritmos tendem a um mesmo limite, limite este particular a cada valor do tempo de uso do recurso (U) utilizado.

A carga, como observado nos gráficos 5 e 6, decresce consoante ao aumento de R. O tempo de espera (W) é o único fator dependente do algoritmo que interfere na carga. Como o comportamento dos dois algoritmos em relação ao tempo de espera é bastante semelhante é previsível a igualdade também na carga. Os resultados obtidos nesses gráficos mostram a igualdade esperada.

O tempo de ciclo apresenta grandes semelhanças para os dois algoritmos, visto que o tempo de espera é também o único fator variável para os algoritmos comparados.

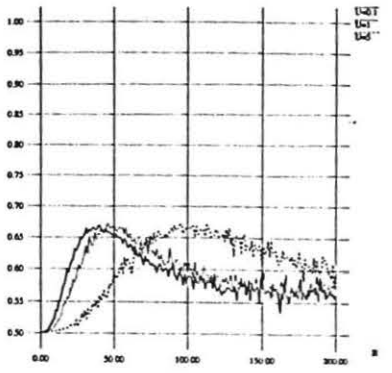

Figura 1: Número de Mensagens Normalizado pelo aumento do tempo de latência — Carvalho e Campos

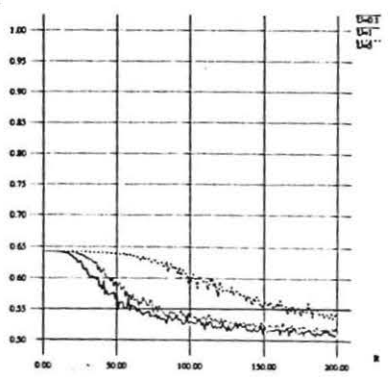

Figura 2: Número de Mensagens Normalizado pelo aumento do tempo de latência — Maekawa

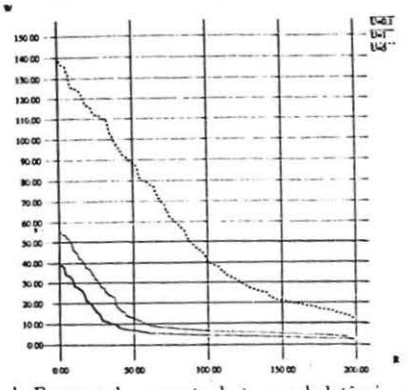

Figura 3: Tempo de Espera pelo aumento do tempo de latência — Carvalho e Campos

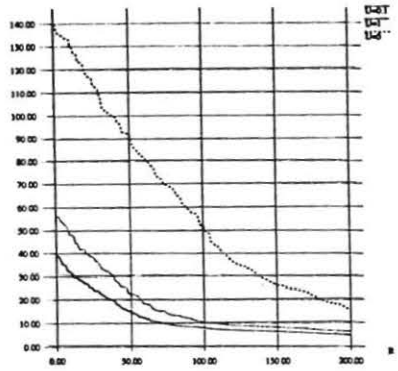

Figura 4: Tempo de Espera pelo aumento do tempo de latência — Maekawa

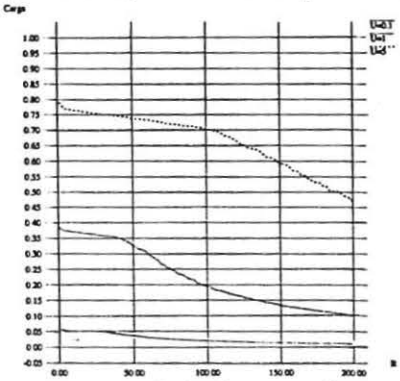

Figura 5: Carga sobre o recurso compartilhado pelo aumento do tempo de latência — Carvalho e Campos

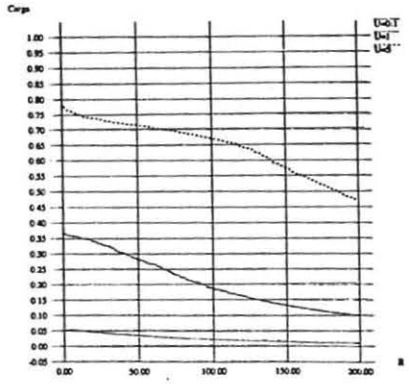

Figura 6: Carga sobre o recurso compartilhado pelo aumento do tempo de latência — Maekawa

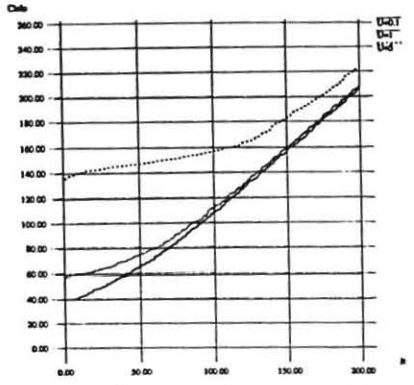

Figura 7: Tempo de ciclo pelo aumento do tempo de latência — Carvalho e Campos

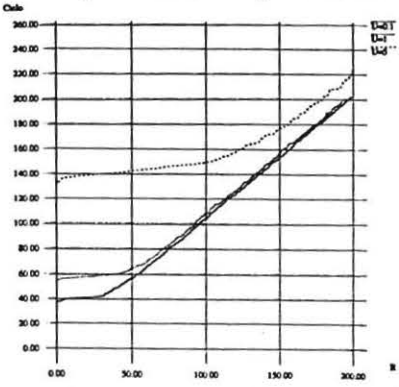

Figura 8: Tempo de ciclo pelo aumento do tempo de latência —Maekawa

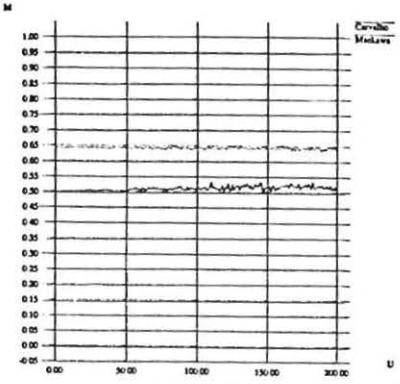

Figura 9: Número de Mensagens Normalizado pelo tempo de Utilzação do recurso — $R \rightarrow 0$

Análise de Casos Limites

A idéia principal desta seção é explicar como se comportam ambos os algoritmos para configurações da rede que levem a utilização muito alta do recurso e no caso oposto levem a uma baixa utilização do recurso compartilhado. A partir disso, um melhor entendimento dos gráficos é alcançado. Vale lembrar que k é tamanho do conjunto ao qual um nodo enviará pedidos de **REQUESTS**.

- **Alta utilização:**

  Para uma situação de alta utilização, quando um processo solicita por um recurso, normalmente a prioridade de sua requisição é a menor possível.

  - O algoritmo de MA exige que um nodo faça a emissão de k **REQUESTS** para a obtenção do recurso. Como estes **REQUESTS** possuem baixa prioridade, k mensagens de **FAILED** são retornadas. Quando os **REQUESTS** chegarem a um estado de alta prioridade, k mensagens de **LOCKED** serão enviadas. Após o uso do recurso, k mensagens de **RELEASE** serão enviadas. Neste extremo, $4k$ mensagens são necessárias por nodo. Este resultado é observado no gráfico 2 para valores pequenos de R.
  - O algoritmo de CC faz com que um cliente faça o envio de k **REQUESTS** para a obtenção do recurso (normalmente, ele não possui nenhum garfo consigo). A prioridade de seus **REQUESTS** nesta situação é normalmente baixa. Quando os **REQUESTS** atingem a maior prioridade na fila de prioridades de cada nodo, um total de k **CLEANPERMISSION** (com um **REQUEST** embutido) são enviadas[3] devido à alta probabilidade de um nodo que está devolvendo os garfos que carrega estar solicitando uma nova demanda. Após o uso do recurso, o nodo devolve todos os seus garfos para os seus k árbitros. Isto contabiliza um total de $3k$ mensagens. Este limite (0.50) é observado no gráfico 1 para valores pequenos de R.

- **Baixa Utilização:**

  - Para uma situação de sub-utilização do recurso observa-se para o algoritmo de MA o seguinte comportamento: Um nodo qualquer envia k mensagens de **REQUESTS** para os nodos de seu conjunto S_i. Normalmente nesssa situação, as mensagens de **REQUESTS** terão uma alta prioridade devido à não concorrência e portanto k mensagens de **LOCKED** serão retornadas a este nodo. Após o término de sua seção crítica, então k mensagens de **RELEASE** serão enviadas. Este total de mensagens ($3k$) pode ser observado no gráfico 2 para valores muito grandes de R.
  - No algoritmo de CC uma análise teórica para esta situação se torna mais complexa devido à característica sonegadora deste algoritmo.

A fim de explicar o interessante fenômeno observado para o algoritmo de CC em situações de uso intermediário do recurso, temos o seguinte:

Um cliente, ao solicitar o recurso, envia k mensagens de **REQUETS** para a obtenção do recurso pois durante o tempo de latencia os seus garfos foram cedidos aos árbitros. Normalmente os **permissions tokens** de seus árbitros estarão com os últimos clientes que solicitaram o recurso aos seus respectivos árbitros. K **REQUESTS** são então enviados aos seus clientes e estes clientes retornam k mensagens de **DIRTYPERMISSION**. Neste ponto os árbitros retornam k mensagens de **CLEANPERMISSION** ao cliente que solicitou o recurso e este então entra em sua seção crítica. Um total de $4k$ mensagens são necessárias, como se observa no gráfico 1 para valores intermediários de R (66% do máximo de mensagens enviadas).

O gráfico 9 nos mostra um resultado importante em situações de alta utilização do recurso compartilhado ($R \rightarrow 0$). Observa-se para esse gráfico que o algoritmo de CC mostra um desempenho superior ao de MA para qualquer valor de U. Observa-se, entretanto, que o número de mensagens enviadas pelo algoritmo de CC e MA apresentam um comportamento estável quando há um grande aumento de U.

[3] Esta nova mensagem é definida como **CLEANPERMISSION-REQUEST**

Sistemas Não Balanceados

Cabe esclarecer que todos os testes realizados foram efetuados com redes totalmente balanceadas, ou seja, redes cujos nodos apresentam basicamente um número igual de pedidos do recurso durante um tempo muito grande de simulação. Em redes não balanceadas é fácil perceber que um algoritmo sonegador[4] como o algoritmo de Carvalho e Campos apresenta uma performance superior ao algoritmo de Maekawa.

4 Conclusão

Os experimentos realizados mostraram facetas dos algoritmos não muitos claras de perceber numa visão analítica. Em primeiro lugar, observa-se um comportamento superior para o algoritmo de Carvalho e Campos para sistemas ditribuídos cuja utilização do recurso seja alta. O algoritmo de Maekawa, entretanto, apresentou uma estabilidade maior para as várias situações de demanda o que levou a resultados ligeiramente melhores nos gráficos do tempo de espera, carga do recurso e do tempo de ciclo. Cabe ressaltar que uma análise para sistemas não-balanceados não é apresentada neste trabalho. Além disso, um estudo analítico mais elaborado de cada gráfico não foi realizado.

Como conseqüência deste trabalho conclui-se que o algoritmo de Carvalho e Campos deveria ser utilizado em situações de carga sobre o recurso alta para uma rede e para sistemas não balanceados. O algoritmo de Maekawa mostrou-se mais robusto e deveria ser utilizado nas demais situações.

Referências

[1] Lamport, L. *Time, clocks, and ordering of events in a distributed system.* Commun. ACM 21,7 (July 1978), 558-565.

[2] Maekawa, Mamoru. *A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems.* ACM trans. Comp. Syst. 3,2 (May 1985), 145-159.

[3] Carvalho, Oswaldo S. F. and Campos, Sérgio V. A. *A $0..\sqrt{n}$ distributed mutual exclusion algorithm.*

[4] Dupuis, Alan and Hebuterne, Gérard and Pitie, Jean-Marc *A Comparison of Two Mutual-Exclusion Algorithms for Computer Networks.* Note Technique CNET/LAA/SLÇ 1985.

[5] Albert, A.A., and Sandler, R. *An introduction to finite projective planes.* Holt, Rinehart, and Winston, New York, 1968.

[6] Carvalho, O.S.F., and Roucairol, G. *On mutual exclusion in computer networks.* Commun. ACM 26,2 (Feb. 1983), 146–147.

[7] Chandy, K.M., and Misra J. The drinking philosophers. *ACM Trans. Program. Lang. Syst.* 6,4 (Oct. 1984), 632–646.

[4] O algoritmo de CC é dito sonegador pelo fato de manter consigo os garfos **PERMISSION TOKEN** após sua seção crítica caso nenhum outro nodo esteja solicitando essas permissões.