PSAD - Protocolo de Sincronização de Ações Distribuídas para Sistemas Distribuídos de Tempo Real

Jorge Tortato Júnior[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – CEP 81531-990 – Curitiba – PR – Brasil

jorge.tortato@nsn.com

Abstract. *Real-time systems are present more and more in people's life and are being used in several applications. A lot of work can be found about that. The same cannot be said about real-time distributed systems, despite of the growing number of applications like industrial automation, robot control and motion control, and so on. This paper proposes a protocol for the action synchronization issue for real-time distributed system applications. The protocol, which is called PSAD (Distributed Action Synchronization Protocol), is based on wall clocks, synchronization and reliable group comunication protocols and Java real-time.*

Resumo. *Sistemas de tempo real estão cada dia mais presentes no cotidiano das pessoas sendo utilizados para as mais diversas aplicações. Vários trabalhos podem ser encontrados sobre este tema. O mesmo não se pode dizer sobre aplicações de tempo real em sistemas distribuídos, apesar do crescente número de aplicações como automação industrial, controle de robôs e controle de movimentos, entre outras. Este artigo propõe um protocolo para o problema de sincronização de ações distribuídos para aplicação em sistemas distribuídos de tempo real. O protocolo, chamado de PSAD (Protocolo de Sincronização de Ações Distribuídas), é baseado em relógios de parede, protocolos de sincronização de relógios e de comunicação confiável em grupos e em Java tempo real.*

1. Introdução

Sistemas de tempo real estão cada vez mais presentes no cotidiano das pessoas. Podemos encontrar processadores desde máquinas de lavar até em sistemas mais complexos constituídos de vários controladores distribuídos em aplicações de robótica, tele-comando, tele-cirurgia, entre outras. Sistemas de tempo real não-distribuídos são relativamente bem compreendidos. Entretanto, sistemas distribuídos de tempo real ainda apresentam-se como um desafio visto que o desempenho dos elementos do sistema e da rede de comunicação não podem ser previstos de forma determinística, o que pode inviabilizar a construção destes sistemas devido à falta de confiabilidade.

Requisitos de tempo real inevitavelmente levam à discussão de sistemas síncronos, nos quais a percepção do tempo é presente de forma marcante. Estudos iniciados na década de 80 expandiram este conceito com a implementação de protocolos sofisticados de sincronização que permitem a sincronização de relógios na ordem de milésimos

de segundo [Mills 1995], dependendo da distância entre os relógios que devem ser sincronizados e o relógio central. O surgimento do sistema de posicionamento global GPS (*Global Positioning System*) no início da década de 90 proporcionou novas oportunidades para a pesquisa de sistemas síncronos. A sincronização de máquinas antes restrita à redes próximas pôde ser expandida para áreas mais distantes.

Além disto, a dissiminação da computação embarcada nos últimos anos é outro importante fator que pode ser citado. Logo, a busca por plataformas uniformes de desenvolvimento que tem intensificado, especialmente se considerarmos plataformas de tempo real. Um esforço neste sentido é o desenvolvimento da plataforma Java de tempo real (RTSJ) [RTSJ Spec] e dos esforços para especificação da plataforma Java tempo real para sistemas distribuídos (DRTSJ)[Anderson 2006] [DRTSJ 2007].

Um dos problemas básicos de sistemas distribuídos em tempo real é a sincronização de ações, isto é, a capacidade de executarem-se ações coordenadas no tempo em diferentes processos. Estudos anteriores como o CASD [Cristian 1985] [Cristian 1990] foram propostos mas apresentavam problemas de consistência e mal-funcionamente quando utilizados com restrições de tempo muito agressivas [Birman 2005].

O objetivo deste trabalho é encontrar um protocolo para sistemas de tempo real distribuídos que permitam a sincronização de ações em pontos distintos de forma confiável, ou seja, que ações em pontos distantes possam ser tomadas de forma sincronizada a partir de um comando centralizado. Ele será baseado nos protocolos de sincronização de relógio, em protocolos confiáveis de *multicast* e em fundamentos de sistemas de tempo real que permitem a execução das ações em tempos determinados.

O artigo está organizado da seguinte forma: seção 2 descreve os protocolos de sincronização, a seção 3 os de comunicação em grupo e protocolos de *multicast*, a seção 4 o problema de escalonamento de tarefas em tempo real, a seção 5 apresenta o PSAD, a seção 6 descreve os experimentos e os resultados e a seção 7 as conclusões.

2. Sincronização de relógios e protocolos de sincronização

Várias técnicas de sincronização de relógios em sistemas distribuídos foram pesquisadas durante a década de 80 [Birman 2005]. Os principais problemas consistem na variação dos relógios devido à imprecisão inerente à cada uma e à latência das redes de computadores que tornam a sincronização imprecisa.

Srikanth e Toueg [Srikanth 1987] propuseram um método baseado no fato que a taxa de variação dos relógios tendem a ser constante e tentam compensar as diferenças antecipadamente. Verissimo e Rodrigues [Verissimo 1992] e Clegg e Marzullo[Clegg 1996] sugeriram um outro método *a-posteriori* para a sincronização de relógios. Este método calcula o atraso entre os relógios do sistema periodicamente através da troca de mensagens entre os processos e faz os ajustes de acordo com os valores recebidos, assumindo que o atraso médio da rede é constante para os processos. Os protocolos mais utilizados atualmente são capazes de alcançar o sincronismo da ordem de poucas dezenas de milésimos de segundos ou menos[Mills 1995].

Outro método usado a partir da década de 90 é o sistema GPS. A partir dele é possível alcançar precisões ainda maiores que as atingidas pelos protocolos de

sincronização, na ordem de poucos milésimos de segundo, exceto em momentos nos quais as condições atmosféricas interfiram na recepção do sinal e os distorçam de forma significativa [Birman 2005]. Logo, com o uso do GPS, hoje é possível garantir uma boa precisão a um custo relativamente baixo.

Tanto os protocolos de sincronização quando o uso do GPS permitem que várias máquinas de um sistema estejam sincronizadas e que marcas de tempo possam ser utilizadas para o controle em sistemas distribuídos de tempo real, escalonando tarefas e sincronizando ações, problema cujo o protocolo proposto neste artigo pretende resolver.

3. Comunicação em grupo e protocolos de *multicast*

A comunicação em grupo é extremamente útil em sistemas distribuídos. Eles permitem que diversos processos integrem ou deixem um grupo dinamicamente, que sejam monitorados através de detetores de falhas, compartilhem estados e visões do grupo e que possam trocar mensagens *unicast* e *multicast*. Os protocolos de comunicação *multicast* são clasificados como básico, confiável, causal, FIFO (*First-in, first-out*) ou totalmente ordenado [Birman 2005].

Para a aplicação de sincronização de ações os protocolos mais relevantes são os dos tipos confiável e FIFO. Protocolos confiáveis garantem que todos os processos pertencentes ao grupos recebam cópias das mensagens mesmo na presença de falhas na comunicação entre e processos. Protocolos do tipo FIFO garantem que as mensagens enviadas por um processo sejam recebidas de forma ordenada pelos processos, isto é, se um processo *p1* envia uma mensagem *m1* para o grupo e então uma mensagem *m2*, todos os membros do grupo irão receber *m1* e depois *m2*, independentemente de atrasos nos meios de comunicação.

4. Escalonamento de tarefas em tempo real

O escalonamento de ações em tempos definidos e preditíveis é uma importante questão em sistemas distribuídos de tempo real. Dois problemas surgem: atrasos na rede e atrasos no escalonamento do sistema operacional.

No primeiro caso, as características da rede de comunicação podem ser controladas através da garantia de qualidade de serviço QoS e/ou prioridades de tráfego. Entretanto, pouco pode-se fazer quando redes complexas são usadas e garantias de qualidade geralmente não são facilmente alcançadas.

O escalonamento do sistema operacional, que é dependente da carga do processador, pode ser resolvido através do uso de prioridades de processos e de características de tempo real do próprio sistema operacional. Diversos sistemas operacionais de tempo real podem ser encontrados na literatura. A maioria restringe-se a uma arquitetura específica, x86 ou PowerPC, por exemplo, o que limita a sua aplicação à sistemas heterogêneos, comuns nos sistemas distribuídos, nos quais cada elemento do sistema pode consistir de um diferente sistema operacional, família de processador, características de processamento e armazenamento. Desta forma, uma solução independente de plataforma parece ser o caminho mais viável para a implementação do escalonamento de tarefas no contexto de sistemas distribuídos.

5. PSAD - Protocolo de Sincronização de Ações Distribuídas

O PSAD é um protocolo projetado para atender de forma confiável ao problema de sincronização de ações em sistemas distribuídos em tempo real. Ou seja, ações devem ser iniciadas em todos os processos corretos do sistema ou em nenhum dos processos. Ele é baseado no fato que é possível obter-se um nível mínimo de sincronização entre os diversos processos em um sistema distribuído, como descrito na seção 2. A seguir são descritos o modelo e o algoritmo propostos.

5.1. Modelo do sistema

O sistema distribuído é definido como *S* e composto por *n* processos $\{p1, p2, ... , pn\}$ sincronizados entre si, cada qual com seu próprio relógio $\{C1, C2, ... , Cn\}$, sendo ϵ a maior diferença entre todos os relógios *Cn*.

Os processos $\{p1, p2, ... , pn\}$ comunicam-se através de uma rede com as seguintes propriedades temporais:

PT1 .Δ_{max} : atraso máximo das mensagens na rede.

PT2. Δ_{med} : atraso médio das mensagens na rede.

PT3. Δ_{opmax}: tempo máximo para uma operação ser completada na rede.

A troca de mensagens é feita através de duas primitivas de comunicação:

PC1. *R_FO_UNICAST* : mensagem *unicast* confiável e FIFO ordenada.

PC2. *R_FO_MULTICAST*: mensagem *multicast* confiável e FIFO ordenada. ,sendo as mensagens enviadas através de PC1 e PC2 FIFO ordenadas entre si.

Dois tipo de falhas são previstos. Falhas por *crash* são definidas como falhas nas quais um processo não responde mais à mensagens. Falhas de temporização da rede são definidas como atrasos superiores aos esperados para o recebimento de uma mensagem.

5.2. O algoritmo PSAD

O algoritmo proposto é fortemente baseado em relógios de parede. Uma vez propriamente sincronizados, o processo que controla as ações (coordenador) e os processos remotos (cliente) são organizados em um grupo fechado. O coordenador utiliza-se do primitivas de comunicação PC1 e PC2 em grupo confiável. Uma vez que a comunicação é feita de forma FIFO ordenada, não existem restrições aos papéis que cada processo desempenha dentro do grupo. Logo, cada ação pode ter um processo diferente como coordenador.

Execuções típicas do protocolo é mostrada na Figura 1. No primeiro exemplo, o coordendar inicia a requisição de uma ação sincronizada enviando uma mensagem para os clientes através de PC2. A menssagem contém a marca de tempo de envio, o identificador de seqüência das requisições enviadas pelo coordenador, o tipo de ação desejada e o tempo alvo para a execução da mesma. Os processos clientes a recebem, escalonam a ação e enviam confirmações através de PC1 dentro do tempo máximo estipulado, o qual é baseado no tempo alvo e do atraso da rede. Assume-se que cada cliente possui o controle de quanto tempo é requerido para cada ação. Desta forma, o cliente pode rejeitar requisições caso existam ações pendentes conflitantes ou cujo tempo de escalonamento seja muito próximo do tempo corrente, pois escalonamentos muito restritivos podem resultar na execução de

ações que deveriam ter sido anuladas mas cujas mensagens de anulação não foram recebidas em tempo. Nenhuma outra troca de mensagens é necessária neste caso.

No segunda troca de mensagens, o coordenador inicia a troca de mensagens como anteriormente. Neste caso, porém, uma mensagem de confirmação não é recebida a tempo pelo coordenador, que inicia o processo de cancelamento da requisição enviando através de PC2 uma mensagem de cancelamento contendo o identificador da ação requisitada e o tempo alvo. O cliente é então responsável por cancelar a ação anteriormente escalonada dentre as ações pendentes para serem executadas.

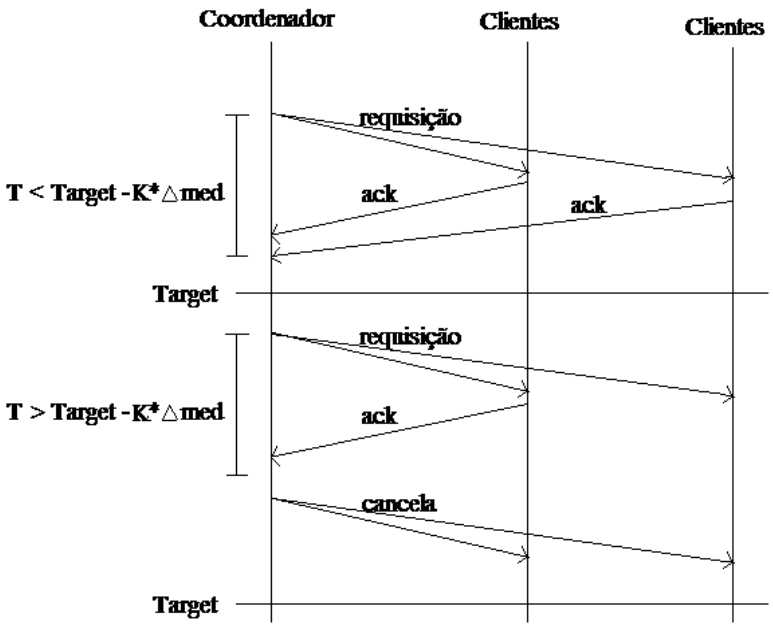

Figura 1. Execução típica do protocolo PSAD.

O algoritmo de comunicação é mostrado na Figura 2. Nas linhas 2 e 3 as variáveis *action_number* e *ack_number* são inicializadas em zero e representam o contador seqüencial de ações e o número de confirmações recebidas, respectivamente. Nas linhas 5 e 6 os atrasos iniciais são configurados.

Na transmissão de uma nova requisição, o tempo alvo é calculado na linha 8, o número de seqüência é incrementado e o número de confirmações recebidas reinicializado. Uma mensagem com a marca de tempo, o tipo de pacote, número de seqüência, tempo alvo e a mensagem da aplicação de nível superior é montada e envia através de *multicast* (PC2). O transmissor então espera na linha 15 até que todas as mensagens tenham sido recebidas ou que o tempo para confirmações se esgote. O tempo é função do tempo alvo e dos atrasos observados na rede. Caso as confirmações não tenham sido recebidas, uma mensagem de cancelamento especificando a ação a ser cancelada é enviada através de multicast para todos os processos. O tempo de Δ_{opmax} é então atualizado de acordo com o tempo utilizado para efetuar a troca de mensagens.

Na recepção (linha 25), o tempo médio de atraso de mensagens Δ_{med} é atualizado a partir da marca de tempo da mensagem recebida. Na seqüência a mensagem é processada de acordo com o seu tipo. Caso seja uma nova requisição (linha 27), a possibilidade

de escalonar a mensagem é verificada com o escalonador de tarefas. Caso seja possível, uma confirmação positiva *unicast* é mandada para o coordenador. Caso contrário, uma confirmação negativa. Caso seja uma mensagem de cancelamento de ação (linha 39), a ação correspondente é imediatamente cancelada através de um pedido ao escalonador. Se a mensagem for de confirmação positiva (linha 41), o número de confirmações é incrementado se a mesma se referir à ação sendo coordenada.

```
1   inicialização:
2   action_number = 0;
3   ack_number    = 0;
4   member_num    = 0;
5   Δmod          = default Δmax;
6   Δopmax        = default Δopmax;

7   transmissão:
8   target_time = now + Δop max;
9   action_number = action_number + 1;
10  ack_number = 0;
11  pkt_type = action_type;
12  msg = timestamp & ptk_type & action_number & target_time & app_msg;
13  R_FO_MULTICAST(msg);
14  start_time = now;
15  wait until (now < target_time – k*Δmax) or (ack_number == member_num);
16  if (ack_number == member_num)
17    return sucesso;
18  else
19    pkt_type = cancel_type;
20    msg = timestamp & ptk_type & action_number & target_time;
21    R_FO_MULTICAST(msg);
22  end if;
23  Δopmax = update_Δopmax(start_time);

24  recepção de msg_rcv:
25  Δmod = update_Δmod (msg_rcv.timestamp));
26  if msg_rcv.pkt_type = action_type
27    send_msg_RT(msg_rcv);
28    wait until rcv_msg_RT(msg_RT);
29    if (msg_RT = sucesso)
30      pkt_type = ack_type;
31      msg = timestamp & ptk_type & action_number & target_time & msg_rcv;
32      R_FO_UNICAST(msg);
33    else
34      pkt_type = nack_type;
35      msg = timestamp & ptk_type & msg_rcv.action_number & msg_rcv.target_time;
36      R_FO_UNICAST(msg);
37    end if;
38  else if msg_rcv.pkt_type = cancel_type
39    send_msg_RT(msg_rcv);
40  else if msg_rcv.pkt_type = ack_type
41    if (msg_rcv.action = action_number) && (msg_rcv.target_time = target_time)
42      ack_number = ack_number + 1;
43    end if;
44  end if;
```

Figura 2. Algoritmo do protocolo PSAD.

O algoritmo de escalonamento é mostrado na Figura 3. Quando uma ação deve ser escalonada e ela é possível (linha 3), a ação é agendada e confirmada para o algoritmo de comunicação. Caso contrário (linha 7), ela é descartada e não confirmada. Quando uma ação é requisita para ser cancelada, se ela ainda não foi executada, ela é cancelada. Caso contrário, uma falha crítica é disparada para a aplicação, responsável em recuperar o sistema.

```
1   recepção de msg_RT:

2   if msg_RT.pkt_type = action_type
3     if isfeasible(msg_RT.target_time)
4       sched_action(msg_RT.action, target_time);
5       send_com(sucesso);
6     else
7       send_com(falha);
8     end if;
9   else if msg_RT.pkt_type = cancel_type
10    if notfired(msg_RT.action)
11      dis_sched_action(msg_RT.action)
12    else
13      recover_action(msg_RT.action)
14    end if;
15  end if;
```

Figura 3. Algoritmo do escalonador PSAD.

6. Experimentos e Resultados

Os experimentos foram realizados usando-se um computador Intel dual core 1.66GHz, com 1GB de memória, com sistema operacional Windows XP. Os processos foram simulados usando-se máquinas virtuais Linux, mas especificamente Fedora Core 6.

A sincronização dos relógios foi implementada usando-se NTP (*Network Time Protocol*) [ntp 2007]. A versão atual é a 3, de março de 1992, embora a versão 4 tenha sido desenvolvida e esteja em processo de padronização pelo IETF. O NTP usa o algoritmo de Marzullo para manter a sincronização da ordem de poucas dezenas de milésimos de segundos ou menos [Mills 1995]. Os contadores de segundos do NTP são mantidos com campos de 32bits para parte inteira e 32bits para a parte fracionária, o que resulta em um intervalo de contagem de 136 anos (a partir de janeiro de 1900) e com uma resolução teórica de 0.233ns.

Os serviços de comunicação em grupo foram implementados a partir das bibliotecas JGroups 2.4.1 SP3 [jgroups 2007]. Foram usados o serviço de criação de grupos e os protocolos *unicast* e *multicast* confiável ambos FIFO ordenados.

O algoritmo PSAD foi implementado em Java versão JDK1.4 e Java-RT (*Java Real Time*) [RTSJ Spec] versão RI 1.1 Alpha 2 da TimeSys. Toda parte do algoritmo dependente da biblioteca JGroups foi codificada e executada sobre a plataforma JDK1.4. As partes dependentes de funções de tempo real, como o escalonamento das tarefas, foram codificadas e executadas a partir da plataforma Java-RT. A comunicação entre os processos JDK1.4 e Java-RT foi realizada através de uma conexão *socket* local. Esta abordagem foi necessária devido à restrições da máquina virtual da TimeSys. Apesar do máquina Java-RT RI da TimeSys ser específica para Fedora Core 6, outras versões comerciais estão disponíveis para outros sistemas operacionais. Logo, a escolha do Java foi devido à independência de plataforma, como mencionado na secção 4. Todos dados coletados foram obtidos através da instrumentação do código fonte. A Figura 4 mostra a estrutura do código.

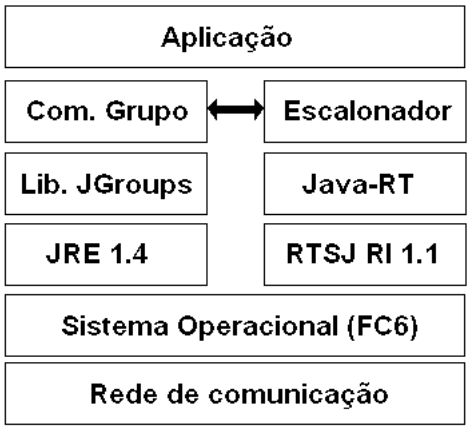

Figura 4. Estrutura da implementação do PSAD.

Dois tipos de experimentos foram realizados. No primeiro, em uma mesma máquina virtual FC6 foram disparados 1, 2 ou 3 processos, sendo que apenas um gerava requisições. Por estarem todos sendo executados em uma mesma máquina virtual, um ambiente perfeitamente sincronizado é simulado. A Figura 5 mostra os resultados de desempenho e falhas. Em (a), (b) e (c) o intervalo entre as requisições feitas pela aplicação foi de 5ms, com Δ_{opmax} = 2s; em (d) o intervalo foi de 100ms e Δ_{opmax} era a variável do experimento. Em (a) temos que o taxa média de requisições reduziu linearmente com o número de processos. (b) mostra a variação da taxa no tempo. Em (c) e (d)

apenas um processo foi executado, este atuando como coordenador e como cliente. Em (c) temos o número de requisições que não obtiveram sucesso e foram canceladas. Em (d), variando-se o tempo alvo através da modificação do parâmetro Δ_{opmax}, vemos que o número de requisição que foram canceladas tende a 100% na medida em que o tempo de envio da requisição mais o da confirmação aproxima-se do valor do atraso da rede, neste caso variando em desde 4 até 50ms.

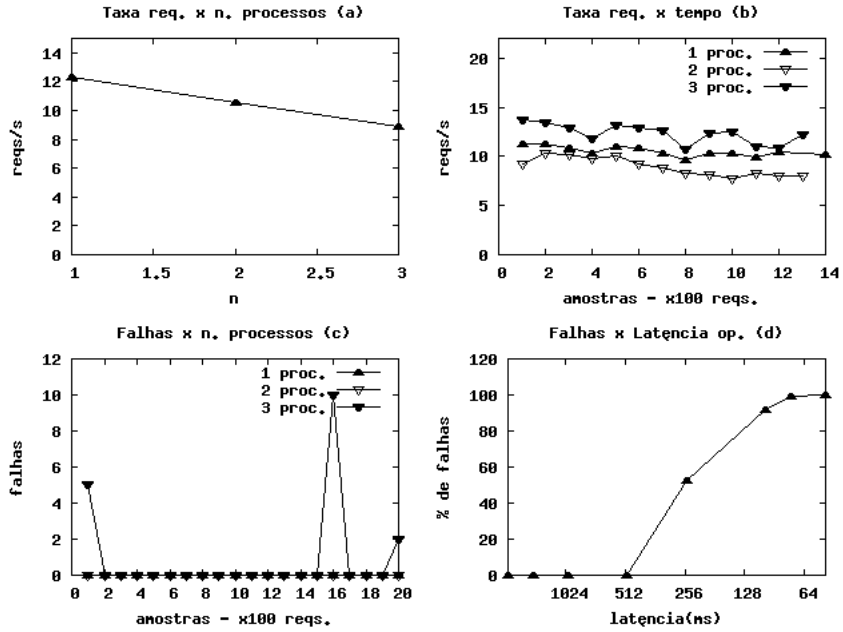

Figura 5. Resultados de desempenho para sistema perfeitamente sincronizado.

A Figura 6 mostra os resultados de *jitter* no tempo, ou seja, a diferença entre o tempo esperado para o escalonamento e o realmente observado. Verifica-se que a na maior parte dos casos o *jitter* foi menor que 15ms.

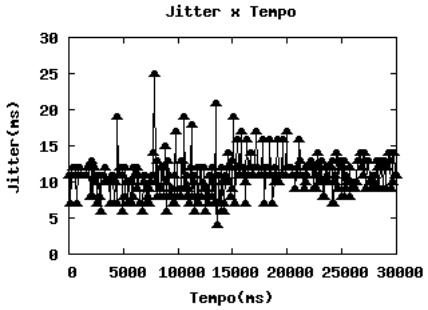

Figura 6. Resultados de jitter para sistema perfeitamente sincronizado.

No segundo experimento, duas máquinas virtuais FC6 foram disparadas, com um processo em cada uma delas, sendo que apenas um gerava requisições. As máquinas foram sincronizadas usando-se o NTP. Os parâmetros de intevalo e Δ_{opmax} foram os mesmos utilizados para os respectivos experimentos anteriores. A Figura 7 mostra resultados qualitativamente semelhantes ao experimento 1.

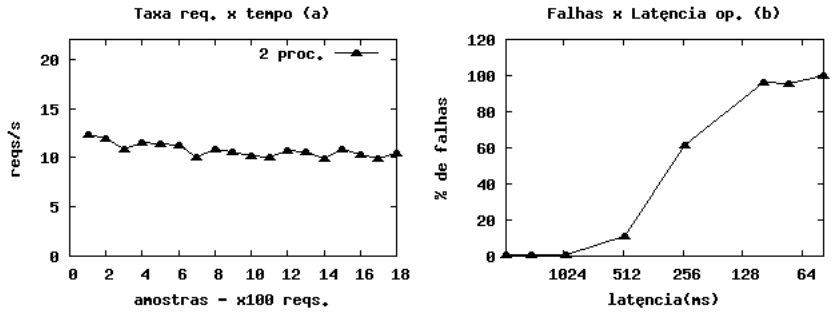

Figura 7. Resultados de desempenho para sistema sincronizado via NTP.

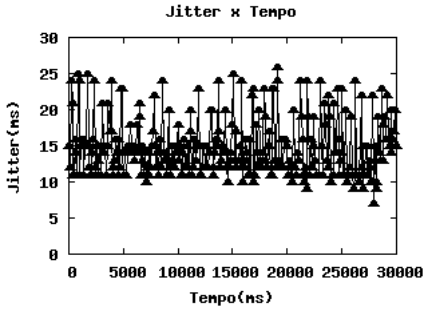

Figura 8. Resultados de jitter para sistema sincronizado via NTP.

A Figura 8 mostra os resultados de *jitter* no tempo. De forma similar, verifica-se que a na maior parte dos casos o *jitter* foi menor que 15ms-20ms. Neste caso, porém, a este valor deve ser acrescentado a diferença de sincronismo entre as máquinas.

7. Conclusões

A aplicação de protocolos de sincronização permitem que regras de escalonamento sejam aplicadas à requisições de um processo central para vários processos distribuídos e que ações possam ser executadas por todos os processos de forma sincronizada e em concenso. Usando-se controle histórico de atrasos na rede pode-se conseguir com que a taxa de sucesso das operações de sincronização sejam altas e escalonáveis até o limite de desempenho da rede. Eventos esporádicos de falhas críticas nos quais o critério de concenso não é atendido devido à picos de atraso em mensagens de cancelamento podem ser minimizados verificando-se a consistência dos parâmetros de escalonamento e configurando-se a rede para priorizar o tráfego de cancelamentos. O *jitter* e o desempenho observados permitem concluir que as execuções podem ser sincronizadas na ordem de poucas dezenas de milésimos de segundo e emitidas a uma taxa em torno de uma centena de milésimos de segundo.

Para futuros trabalhos podem ser previstos estudos mais aprofundados das funções de cálculo do atraso médio de mensagens e do tempo máximo para completar uma operação, o aprimoramento da verificação das marcas de tempo dos pacotes para a detecção de falhas de sincronismo entre as máquinas, a integração do código para a execução em uma única máquina virtual Java com o intuito de aumentar o desempenho do protocolo e a realização de experimentos em uma rede real.

Referências

Birman, Kenneth P. (2005). *Reliable Distributed Systems*. Springer, 1st edition.

Jensen, E. Douglas. (2004). *Timeliness in Mesosynchronous Real-Time Distributed Systems*. Proc. 17th ISORC'04. *The Computer Society*. IEEE Press.

Verissimo, P. and L. Rodrigues (1992). *A-posteriori Agreement for Fault-Tolerant Clock Synchronization on Broadcast Networks*. Proc. 22nd International Symposium on Fault-Tolerante Computing. Boston, July 1992.

Clegg, M. and K. Marzullo (1996). *Clock Synchronization in Hard Real-Time Distributed Systems*. Technical Report. Department of Computer Science, University of California, San Diego. March 1996.

Srikanth, T. K. and S. Toueg (1987). *Optimal Clock Synchronization*. Journal of the ACM 34:3 (July 1987):626-645.

Network Time Protocol. www.ntp.org. 2007.

RTSJE Group (2005). *The Real Time Specification for Java*. www.rtj.org. July 2005.

Anderson, Jonathan S. and E. Douglas Jensen (2006). *The Distributed Real-Time Specification for Java*. Status Report. 4th International workshop on Java Technologies for Real-Time Embedded Systems (JTRES 2006).

Distributed Real-Time Java. www.real-time.org. 2007.

Java Groups. www.jgroups.org. 2007.

Cristian, F. , D. Dolev, R. Strong, and A. Aghili. (1990). *Atomic Broadcast in a Real-Time Environment*. Fault-Tolerant Distributed Computing. Springer-Verlag Lecture Notes in Computer Science, Vol.448, 1990, 51-71.

Cristian, F. , D. Dolev, R. Strong, and A. Aghili. (1985). *Atomic Broadcast: From Simple Message Difusion to Byzantine Agreement*. Proc. of the 15th Fault-Tolerant Distributed Computing.

Mills, D. L.. (1995). *Improved Algorithms for Synchronizing Computer Network Clocks*. IEEE/ACM Trans. Networks 3,3 , 245-254. June 1995.