# Sincronização de ações de forma confiável para sistemas de tempo-real distribuídos

**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.*** *This paper proposes a solution for the actions synchronization issue on distributed systems for application on industrial control systems, robotic and other embedded real-time systems. The solution is based on wall clocks, synchronization and reliable group comunication protocols and Java real-time.*

***Resumo.*** *Este artigo propõe uma solução para o problema de sincronização de ações em sistemas distribuídos para aplicação em sistemas de controle industrial, robótica e outros sistemas embarcados de tempo real. A solução é baseada 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 sistemas mais complexos constituídos de vários controladores distribuídos em aplicações de robótica, tele-comando, tele-cirurgia, etc.. Sistemas de tempo real não-distribuídos são relativamente bem compreendidos. Entretanto, sistemas distribuídos de tempo real ainda apresentam-se como um desafio.

O propósito deste trabalho é encontrar soluções 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. Exemplos de aplicações são: controle de movimentos de robôs, automação industrial e controle de tráfego.

Requisitos de tempo real inevitavelmente levam à discussão de sistemas síncronos, nos quais a percepção do tempo é presente de forma marcante.

Pesquisas 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.

A disponibilização 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.

Adicionalmente, a dissiminação da computação embarcada nos últimos anos é outro fator importante que pode ser citado. Conseqüentemente a busca por plataformas

uniformes de desenvolvimento tem se intensificado. Um esforço neste sentido revela-se na recente publicação 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].

Soluções anteriores como o CASD [Cristian 1985] [Cristian 1990] foram propostas mas apresentavam problemas de consistência e mal-funcionamente para restrições muito agressivas [Birman 2005].

A solução proposta inclui um conjunto de protocolos e plataformas de desenvolvimento. A seção 2 descreve os protocolos de sincronização, a seção 3 os de comunicação em grupo, seção 4 a plataforma Java de tempo real e a seção 5 um protocolo próprio para o escalonamento de ações, o qual completa a solução como um todo.

## 2. 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.

Verissimo e Rodrigues [Verissimo 1992] e Clegg e Marzullo[Clegg 1996] sugeriram um 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.

Srikanth e Toueg [Srikanth 1987] propuseram outro método baseado no fato que a taxa de variação dos relógios tendem a ser constantes e tentam compensar as diferenças antecipadamente.

Estes protocolos de permitem que as várias máquinas do sistema estejam sincronizadas e que *timestamps* possam ser utilizados para o controle em sistemas distribuídos de tempo real, escalonando *threads* e sincronizando ações.

### 2.1. O Protocolo NTP

Atualmente o NTP (*Network Time Protocol*) [ntp 2007]é utilizado para manter o sincrinismo de rede. 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 e pode manter a diferença entre relógios entre 10ms, usando a Internet pública, até 200us ou mais, usando redes locais.

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.

## 3. Protocolos de Comunicação em grupo

Protocolos de comunicação em grupo - ou *multicast* - são extremamente úteis em sistemas distribuídos. Eles permitem que diversos processos pertencentes a um grupo rece-

bam mensagens de um transmissor. Pode-se classificá-los 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/dos 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, o grupo irá receber m1 e depois m2, independentemente de atrasos nos meios de comunicação.

### 3.1. *Spread Toolkit*

O *Spread Toolkit* [Spread 2007] é um conjunto de funções disponíveis para várias plataformas e sistemas operacionais. Entre as APIs disponíveis estão a Java, o que permite o uso em conjunto com as funções Java tempo real.

Os protocolos de comunicação em grupo disponíveis estão listadas na Figura 1. Como dito anteriormente, o interesse para a sincronização de ações recai sobre o protocolo FIFO_MESS que permite a ordenação das mensagens do iniciador das ações de forma confiável.

| Spread Service Type | Ordering | Reliability |
|---|---|---|
| UNRELIABLE_MESS | None | Unreliable |
| RELIABLE_MESS | None | Reliable |
| FIFO_MESS | Fifo by Sender | Reliable |
| CAUSAL_MESS | Causal (Lamport) | Reliable |
| AGREED_MESS | Total Order (Consistent w Causal) | Reliable |
| SAFE_MESS | Total Order | Safe |

**Figure 1. Serviços de comunicação disponíveis no *Spread Toolkit*.**

## 4. Escalonamento de ações locais

O escalonamento de ações em tempos definidos e preditíveis é um dos temas de pesquisa em sistemas distribuídos de tempo real. Dois problemas surgem: atrasos na rede e atrasos no escalonamento do sistema operacional.

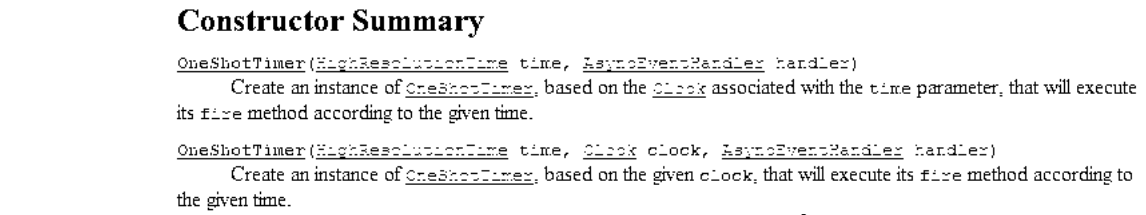
No primeiro caso, as características da rede de comunicação podem ser controladas através da garantia de qualidade de serviço QoS e/ou prioridades de tráfego. Entretanto, pouco pode-se fazer quando redes complexas são usadas e garantias de qualidade geralmente não são facilmente alcançadas.

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, o que limita a sua aplicação à sistemas heterogêneos, comuns nos sistemas distribuídos.

### 4.1. Java tempo real

A disponibilização do Java-RT (*Java Real Time*) [RTSJ Spec] permite um maior controle temporal dos programas escritos em Java. Estão disponíveis classes que implementam relógios de alta resolução (padrão Epcho). Estes mesmos relógios de alta resolução possibilitam o escalonamento de tarefas periódicas ou esporádicas (Figura 2) de forma preditiva.



**Constructor Summary**

`OneShotTimer(HighResolutionTime time, AsyncEventHandler handler)`
Create an instance of `OneShotTimer`, based on the `Clock` associated with the `time` parameter, that will execute its `fire` method according to the given time.

`OneShotTimer(HighResolutionTime time, Clock clock, AsyncEventHandler handler)`
Create an instance of `OneShotTimer`, based on the given `clock`, that will execute its `fire` method according to the given time.

**Figure 2. Classe Java-RT para tarefas esporádicas.**

O Java-RT também permite que as mesmas garantias temporais implementadas sobre uma plataforma ou sistema operacional possa ser facilmente adaptadas para as demais plataformas.

## 5. Solução proposta

A solução proposta neste trabalho é fortemente baseada em relógios de parede. Estes relógios permitem a sincronização de diversas máquinas distribuídas em rede através de um protocolo de sincronização padronizado, o NTP.

Considerando-se que o NTP permite a sincronismo dentro de limites da ordem de milésimos de segundo, ele pode ser empregado para a maior parte de aplicações de tempo real. Para sistemas nos quais a comunicação pode sofrer atrasos muito significativos, receptores GPS podem ser usados para permitir uma melhor precisão.

Uma vez propriamente sincronizados, o processo que controla as ações (processo mestre) e os processos remotos (processos escravos) são organizados em um grupo fechado. O mestre utiliza-se do protocolo de comunicação em grupo confiável e do tipo FIFO implementado no *Spread Toolkit*. Este protocolo permite que falhas de comunicação não afetem o sistema e que minimizem a tarefa da camada de escalonamento em tempo real.

As mensagens enviadas pelo mestre devem conter as ações requisitadas a cada escravo bem como o tempo requisitado para cada ação, T. A partir do envio da requisição, o mestre espera dentro de um limite de tempo seguro *T - 2*Lm* as confirmações de todos os escravos aceitando as requisições, onde *Lm* é o atraso médio da rede. Caso todas as confirmações sejam recebidas em tempo, nenhuma ação é requerida do mestre.

Caso algum processo escravo não responda dentro do tempo estipulado, ou responda negativamente, o mestre é responsável por enviar uma segunda mensagem para o grupo requisitando a anulação da mensagem anterior, evitando assim que falhas resultem em uma ação indesejada.

Em relação ao processo escravo, este aguarda mensagens do mestre para escalonar suas ações. A cada requisição recebida, a mensagem é analisada em relação as tarefas

pendentes de cada escravo. Caso seja possível realizá-la dentro no tempo previsto, uma mensagem de confirmação positiva deve ser enviada.

Assume-se que cada escravo possui o controle de quanto tempo é requerido para cada ação. Desta forma, o escravo pode rejeitar requisições caso existam ações pendentes conflitantes ou cujo tempo de escalonamento seja muito próximo do tempo atual. Este valor deve ser tipicamente menor que *2*Lm*. Isto porque escalonamentos muito restritivos podem resultar na execução de ações que deveriam ter sido anuladas mas cujas mensagens de anulação não foram recebidas em tempo.

Confirmações positivas devem ser enviadas de forma *unicast* para minimizar o tráfego na rede. Confirmações negativas devem ser enviadas através de serviços *multicast* confiável, da mesma forma que as requisições. Desta forma, falhas podem ser reconhecidas mais rapidamente pelo processos escravos e de forma confiável.

O escalonamento das ações nos processos escravos são implementados utilizando-se as funcionalidades providas pelo Java-RT, mas especificamente pelo escalonamento de tarefas realizadas por processos não esporádicos. Mensagens do mestre pedindo anulação de ações são facilmente bloqueadas através da desabilitação imediata dos *timers* esporádicos do Java-RT.

Todos os processos, sejam mestre ou escravos, mantêm a média dos atrasos das mensagens na rede com o intuito de verificar as regras de aceitação de escalonamento descritas acima. Para tanto, todas as mensagens e confirmações devem conter *timestamps*.

A avaliação da solução será realizada utilizando-se de várias cópias de máquinas virtuais Linux, permitindo a simulação de uma rede. Atrasos na rede poderão ser simulados atrasando-se o recebimento das mensagens tanto no mestre quando nos escravos. A avaliação do desempenho e da corretude do sistema poderá ser avaliada através do *log* das mensagens/aplicações e do respectivo *timestamp*, uma vez que a sincronização é a premissa principal da solução.

## 6. Conclusões

A aplicação de protocolos de sincronização permitem que regras de escalonamento sejam aplicadas à requisições de um processo central para vários processos distribuídos.

Combinado-se o fato dos processos estarem sincronizados no tempo, protocolos clássicos de comunicação em grupo e novas technologias emergentes voltadas para sistemas de tempo real como o Java-RT, é possível garantir de forma confiável que ações sincronizadas sejam tomadas.

A falha inerente à solução é associada à premissa do sincronismo entre os processos, cujas falhas pequenas de relógio não podem ser facilmente percebidas. Falhas mais pronunciadas, da ordem de grandeza do atraso da rede, podem ser reconhecidas através de *timestamps* incoerentes.

O tráfego associado ao protocolo de escalonamento deve ser O(n2), ou seja, da mesma ordem do protocolo de *multicast* confiável, o que permitiria a escalabilidade do sistema.

## 7. Cronograma

O cronograma proposto é mostrado na tabela 1.

**Table 1. Cronograma de execução proposto.**

| Semana | 16/04 | 23/04 | 30/04 | 7/05 | 14/05 | 21/05 | 28/05 | 4/06 | 11/06 |
|---|---|---|---|---|---|---|---|---|---|
| Atividade | | | | | | | | | |
| Instalação e configuração de máquinas virtuais linux. | X | | | | | | | | |
| Instalação e configuração do protocolo NTP. Verificação do funcionamento correto do protocolo. | | X | | | | | | | |
| Instalação do ambiente de desenvolvimento Java-RT. Testes básicos. | | X | X | | | | | | |
| Instalação das APIs do Spread Toolkit e verificação do funcionamento com interface Java. | | | X | X | | | | | |
| Implementação do protocolo de escalonamento em Java. | | | | X | X | X | | | |
| Teste do sistema e coleta dos resultados. | | | | | | X | X | | |
| Escrita do relatório/artigo final. | | | | | | | | X | |
| Preparação da apresentação. | | | | | | | | | X |

## 8. References

## References


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.

*Spread Toolkit*. www.spread.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.