MI - Concorrência e Conectividade
Acertando os Ponteiros

Vinicius Pereira Santana[1]

[1]Graduando em Engenharia de Computação.
Universidade Estadual de Feira de Santana (UEFS), Feira de Santana, Brasil.

vpsantana@ecomp.uefs.br

Abstract.

Resumo. *O presente documento aborda o desenvolvimento do TimeSynchronizer, uma solução distribuída com o intuito de sincronizar, aproximadamente, um grupo de relógios físicos, garantindo a monotonicidade temporal. Para isto, conceitos pertinentes sobre Sistemas Distribuídos, Sincronização, Algoritmos de Eleição e arquitetura Cliente/Cliente foram aplicados.*

1. Introdução

Na modernidade, tempo tornou-se um elemento precioso. Não obstante, sua noção é bastante difusa e, por diversas vezes, é tema de discussões filosóficas e alvo de pesquisas na Física. O que se sabe é que ele norteia a vida das sociedades e constitui-se como moeda de troca nas relações de trabalho.

Nos sistemas computacionais não é diferente. Ter uma referência para a execução das tarefas é importante. Essas tarefas podem ser das mais variadas: desde a criação de um arquivo até a execução de um *back-up* agendado, por exemplo. Entretanto, por um aspecto construtivo, dispositivos diferentes (ou até mesmo iguais) podem mensurar os instantes de maneira diferente e, com o passar do tempo, naturalmente atrasar ou adiantar. Assim, é preciso buscar maneiras confiáveis de manter a sincronização dos relógios.

Neste aspecto, foi solicitado aos alunos do MI - Concorrência e Conectividade, da Universidade Estadual de Feira de Santana (UEFS), o desenvolvimento de uma solução distribuída para a sincronização aproximada de um grupo definido de relógios físicos com as seguintes características: utilização de um dos relógios como referência com o intuito de manter que o tempo sempre avance (monotonicidade) e a identificação e correção de possíveis falhas. A motivação é apresentar aos alunos os desafios e perspectivas quando é necessário que seja mantida a concordância entre os relógios.

Desse modo, o presente documento apresenta os conhecimentos necessários para a construção da solução solicitada (Seção 2), bem como apresenta os resultados obtidos (Seção 3). Por fim, são tecidas as conclusões (Seção 4).

2. Desenvolvimento

Diante da demanda estabelecida, é imediato iniciar o desenvolvimento do sistema com a definição da arquitetura. Se para que a sincronização dos relógios aconteça, os dispositivos devem conectar-se entre si. Desse modo, o modelo Cliente/Cliente deverá ser utilizado.

O modelo Cliente/Cliente é amplamente utilizado em arquiteturas distribuídas ou descentralizadas. Isto não significa que não exista um servidor, significa que a posse desta função é dinâmica, ao contrário do modelo Cliente/Servidor que é estática. Assim, a depender das configurações, estado e objetivos, por exemplo, há troca no cargo de servidores. Recebe também a denominação Peer-to-Peer (P2P) [Kurose e Ross 2010].

Outro ponto importante no projeto é a escolha da linguagem de programação. Para o desenvolvimento deste sistema foi escolhida a linguagem Java. A codificação em Java permite um ganho ímpar em portabilidade. Desse modo, independente da plataforma de Sistema Operacional (*Windows*, *Linux* ou *MacOS*), a aplicação funcionará, exigindo apenas a instalação da máquina virtual fornecida pelo Java. Em outras palavras, isso significa praticidade e redução nos custos do projeto [Caelum 2017].

Levando em consideração as informações apresentadas, as subseções seguintes apresentarão conceitos e tópicos necessários ao desenvolvimento desta aplicação.

2.1. Sistemas Distribuídos

Para [Coulouris et. al. 2012], um sistema distribuído é aquele cujos componentes localizam-se em computadores interligados em rede e que coordenam suas ações por meio da troca de mensagens. Por exemplo, inclui-se os jogos *multiplayer* online e os sistemas de buscas web. Esta definição consegue caracterizar bem a maioria dos sistemas existentes e faz menção às características de concorrência no uso dos componentes, tolerância às falhas e ausência de um relógio comum.

A concorrência no uso dos componentes é uma realidade quase que inevitável em uma rede de computadores. Se um grupo de estudantes, nas suas residências, compartilham arquivos e escrevem o trabalho simultaneamente, é importante que o sistema tenha a capacidade de coordenar as ações da melhor forma.

Falhas podem acontecer a qualquer instante e é importante que seja dimensionado três itens: identificação das falhas, ações de emergência e os impactos. Assim, por exemplo, mensagens podem não ser recebidas ou atrasarem e os processos podem finalizar de maneira inesperada (*crash*). Em qualquer situação, o sistema deve continuar funcionando corretamente.

A ausência de um relógio comum justifica-se em dois aspectos. O primeiro, como abordado anteriormente, é um aspecto construtivo: os dispositivos tendem a contabilizar diferente o tempo. Além disso, como é provável que os dispositivos estejam em locais distintos do mundo, questões de fuso horário são pertinentes. Neste cenário, é natural que aconteçam diferenças. Em sistemas nos quais tempo é um elemento crucial, isto pode inviabilizar o funcionamento. Neste contexto, o uso de mensagens e a noção de causalidade (um evento só pode acontecer após o acontecimento de um determinado evento anterior) é indispensável. Mas, se a sincronização for inevitável, é importante lançar mão de alguma estratégia.

2.2. Sincronização Temporal em Sistemas Distribuídos

O grande trunfo da sincronização temporal, nos sistemas distribuídos, é fazer com que os componentes concordem com uma determinada hora, não necessariamente a hora tida como a correta. Não obstante, para haver consistência temporal, é importante que a política seja mantida.

Neste plano, o algoritmo contido no *Berkley Software Distribuition* (BSD) é uma solução elegante (Figura 1). Quando uma sincronização é necessária, um coordenador (escolhido previamente e que atende ao conceito de servidor dinâmico) envia sua estampa aos outros dispositivos e estes respondem com a diferença. Por fim, o coordenador avalia as diferenças recebidas e responde com o ajuste que cada dispositivo deve fazer para manter-se sincronizado. Assim, a concordância temporal é alcançada [Coulouris et. al. 2012]. Entretanto, esta solução permite que tempo possa avançar ou retroceder, pois o ajuste mantém relação de dependência com quem ocupa o cargo de coordenador. Em aplicações críticas, tal comportamento temporal é indesejado.

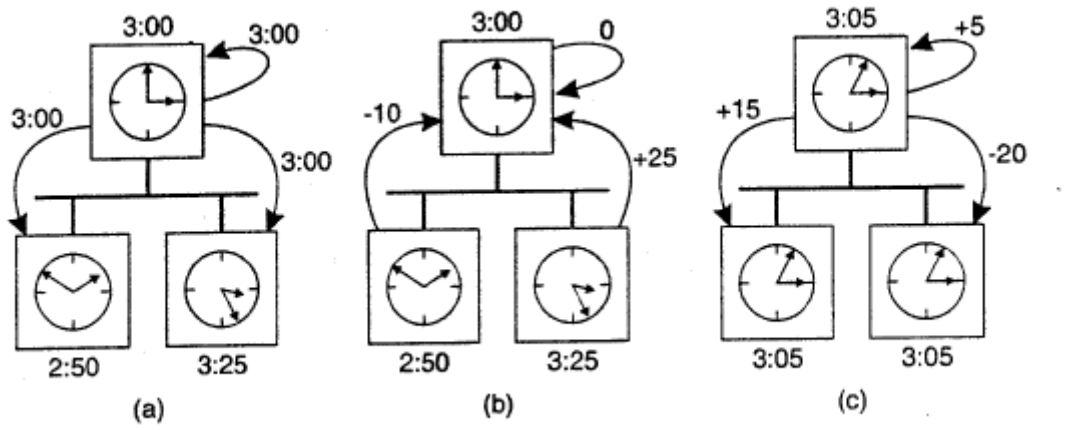


Figura 1. (a) O coordenador envia seu horário aos outros dispositivos. (b) Os dispositivos respondem com a diferença. (c) O coordenador envia qual ajuste deve ser feito [Tannembaum e Steen 2007].

Com o objetivo de adaptar o algoritmo para o desenvolvimento do sistema proposto, duas modificações foram feitas:

1º) A escolha do coordenador é baseada em critérios que garantem a monotonicidade temporal. Em outras palavras, coordenadores (servidores dinâmicos) são aqueles cuja estampa de tempo é a maior entre as demais, momentaneamente.
2º) O coordenador envia sua estampa de tempo em períodos fixos. Deste modo, todos os dispositivos constantemente são sincronizados.

2.3. Algoritmo de Eleição

As partes integrantes de um sistema distribuído necessitam manter certo nível de organização. Um modo de manter a organização é a existência de hierarquia. Desse modo, é preciso determinar, mesmo que por um instante ou não, quem será responsável por enviar as estampas e quem as receberá. Na literatura, esta coordenação é provida por um algoritmo de eleição.

O objetivo mais simples desta classe é permitir que alguém possa conduzir a realização de uma determinada tarefa. Não obstante, é fornecido também uma modo para identificar possíveis falhas e providências. Aqui, falha é compreendida como a não resposta do coordenador corrente ou estampa de tempo enviada sem assegurar a

monotonicidade do tempo. Na arquitetura proposta, foi escolhido a utilização do Algoritmo de *Bully* (tradução livre, Valentão), com modificações [Coulouris et. al. 2012], [Tannembaum e Steen 2007].

O algoritmo define três tipos básico de mensagens. A primeira é uma convocatória para a realização de uma eleição. A segunda consiste de uma resposta para um pedido de eleição. Por fim, uma mensagem que notifica o vencedor da eleição que, em outras palavras, será o coordenador.

Para avaliar o funcionamento, é importante considerar a existência de um processo P que pode ter se recuperado de uma falha ou identificou a ocorrência de uma falha. As seguintes ações são possíveis [Coulouris et. al. 2012], [Tannembaum e Steen 2007], [Wikipedia, 2017]:

- Se P possui a maior identificação (neste caso, estampa de tempo), ele envia uma mensagem de vitória. Caso contrário, somente os que possuem maiores identificações são notificados sobre a eleição.
- Se P não recebe nenhuma resposta após o início de uma eleição, então ele envia uma mensagem de vitória.
- Se P recebe uma resposta de um processo com maior identificação, P não enviará mais mensagens e aguardará uma mensagem de vitória. (Caso não haja mensagem de vitória, o processo é reiniciado).
- Se P recebe uma mensagem de eleição de um processo com menor identificação, P envia uma resposta e inicia uma nova eleição.
- Se P recebe uma mensagem que possui uma estampa de tempo, P trata o remetente como coordenador.

Na implementação corrente, quando P identifica a ocorrência de falha, este assume o papel de coordenador, sem iniciar uma eleição. Caso P seja de fato o coordenador, não há modificações. Caso contrário, o sistema vai ajustando-se até encontrar o verdadeiro coordenador. Tal configuração dá mais dinâmica e permite sincronizar mais rapidamente os dispositivos.

2.4. Implementação Cliente/Cliente

Para realizar a comunicação entre os dispositivos, com o auxílio do Java, foi definido um grupo de *multicast*. Este é capaz de reunir os interessados em enviar e receber determinadas mensagens. Além disto, *Threads* e o protocolo de transporte UDP foram utilizados também. No aspecto comunicativo, é importante que os dispositivos possam se comunicar de maneira eficiente e padronizada. Desse modo, é importante definir um protocolo de comunicação. As informações estão na Tabela 1.

Tabela 1. Protocolo de comunicação estabelecido neste *software*.

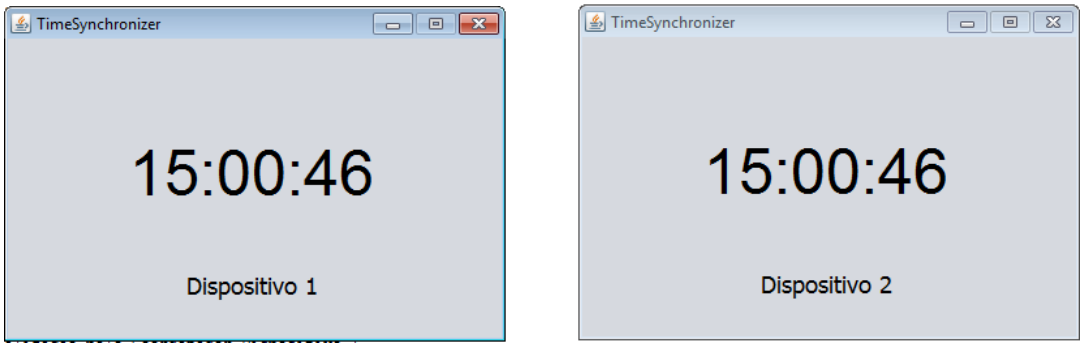
Operação	Código da Operação	Quantidade de Parâmetros
Enviar Hora (Eleição)	1001	4
Enviar Hora	1002	3
Realizar Eleição	1003	4
Responder Eleição	1004	2
Vencedor Eleição	1005	1

Fonte: Autor.

A tabela possui 3 colunas: Operação, Código da Operação e Quantidade de Parâmetros. Operação diz respeito as ações que podem ser executadas pelos dispositivos. A coluna seguinte denota o código que identifica a operação. Por fim, a última coluna informa sobre quantidade de parâmetros enviados em uma dada operação.

3. Resultados

A primeira versão do sistema, denominado *TimeSynchronizer*, foi construída, levando em consideração os requisitos solicitados. Testes foram executados e os resultados obtidos foram satisfatórios. As Figuras 3(a) e 3(b) apresentam dois dispositivos que possuem as estampas de tempo sincronizadas.



(a) Relógio do Dispositivo 1 (b) Relógio do Dispositivo 2

Figura 2. Imagens do *TimeSynchronizer*.

Ao iniciar a execução, é solicitado que o usuário informe uma identificação para o novo relógio. Logo após, a estampa de tempo inicial (formato HH:MM:SS) e o *drift*. O *drift* corresponde ao quão rápida ou lenta é uma contagem. Este parâmetro serve para modelar as características construtivas dos dispositivos. Quanto mais próximo de 0, mais rápida é contabilização do tempo. Para uma contagem regular, o *drift* deve ser 1.

Após as configurações iniciais, a sincronização tem inicio. Ele é efetuada a cada 5 segundos. A *thread* associada é responsável por: identificar falhas, modificar coordenador e enviar/receber estampas. Outra *thread* é responsável por efetuar a contagem e exibição da estampa. Por conta da interface minimalista, o estado do sistema é exibido por meio do console para rastrear as ações.

Um outro aspecto considerado na sincronização é o tempo entre o envio e o recebimento da resposta de uma mensagem - *Round-Trip Time* (RTT). Nesta solução, o cálculo do RTT é simulado e pode estar entre 100 e 1100 milissegundos (típico de redes locais e das de longas distâncias). A simulação é feita para avaliar o comportamento do sistema quando os atrasos de rede são pertinentes. Por aspectos de projeto, quando o RTT é maior ou igual a 1000 milissegundos, há compensação na estampa de tempo enviada (adiciona-se mais um segundo).

4. Conclusões

A formação de um profissional de Engenharia de Computação deve buscar ao máximo o desenvolvimento de habilidades e competências que o auxiliem em qualquer ambiente

adverso. Neste contexto, insere-se a construção e/ou administração de Sistemas Distribuídos.

Não obstante, foi desenvolvido o *TimeSynchronizer*. Este é uma solução distribuída para a sincronização aproximada de um grupo definido de relógios físicos que é capaz de tratar falhas (*crashes* de coordenador, atrasos de redes, dentre outros). As informações sobre o estado do sistema são apresentados nas interfaces gráfica e de comando. De modo geral, os resultados obtidos foram satisfatórios. No aspecto de melhorias, é sugerido fazer o cálculo real do RTT.

Referências

Caelum (2017). Java e Orientação a Objetos - Curso FJ - 11. Disponível em: <www.caelum.com.br/apostilas>. Acesso em: 8 julho 2017.

Kurose, F. e Ross, K. (2010). *Redes de computadores e a Internet: uma abordagem top-down*. 5 ed. São Paulo: Pearson.

Coulouris, G., Dollimore, J., Kindberg, T. e Blair, G. (2012). *Distributed Systems: Concepts and Design*. 5 ed. Boston: Pearson.

Wikipedia (2017). Bully algorithm. Disponível em: <https://en.wikipedia.org/wiki/Bully_algorithm>. Acesso em: 26 junho 2017.

Tanenbaum, A. S., Steen, M. V. (2007) *Distributed Systems: Principles and Paradims.* 2 ed. Upper Saddle River: Pearson-Pretice Hall.