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.

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 de servidores. Receba 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 tem início em dois problemas. 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 o 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 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. Quando uma sincronização é necessária, um coordenador (escolhido

previamente e que atende ao conceito de servidor dinâmico) solicita que os dispositivos enviem suas estampas de tempo. Ao receber, o mesmo avalia cada estampa com a própria 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.

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.
2º O coordenador envia sua estampa de tempo, em períodos fixos, para que todos os dispositivos mantenham-se sincronizados.

Desse modo, seguiu-se o critério de avaliar qual dispositivo possuía a maior estampa de tempo naquele momento

2.3. Implementação Cliente/Cliente

Para o desenvolvimento

3. Resultados

4. Conclusões

Referências

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

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

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