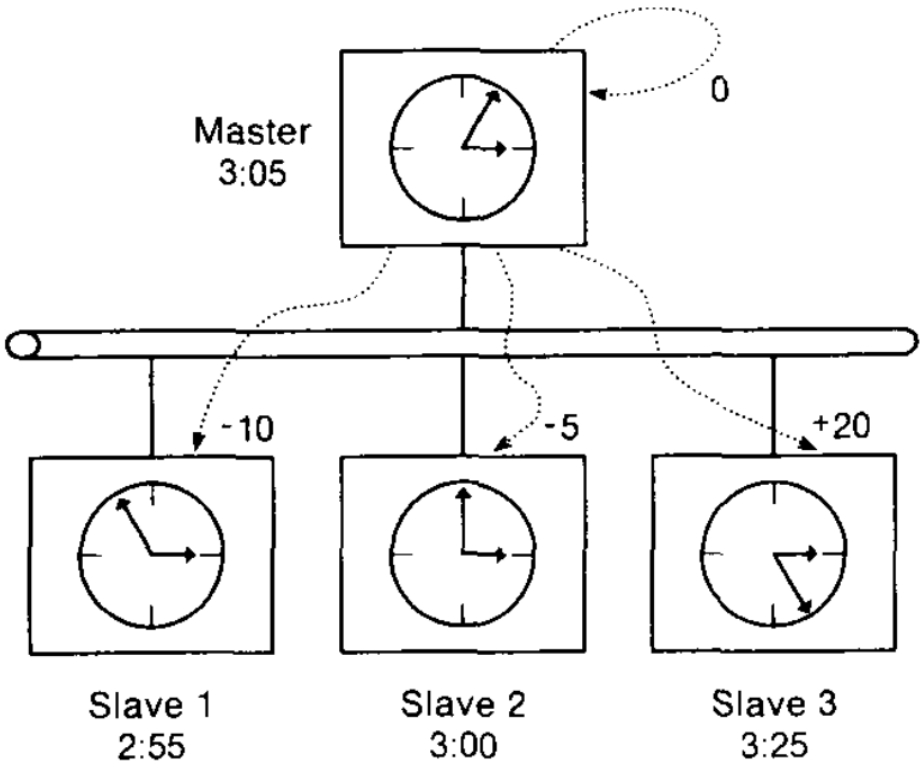
Projeto - Sincronização de relógios em um sistema distribuído

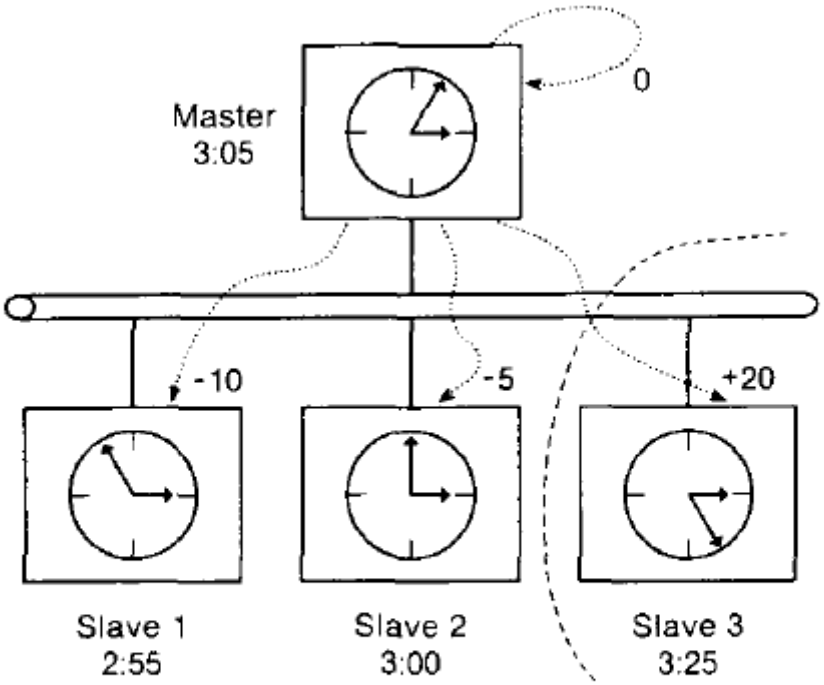
Este projeto proporcionará o aprendizado da sincronização de relógios em um sistema distribuído. Implementaremos o algoritmo de Berkeley

Descrição do Protocolo de Alto Nível

Existem dois tipos de nós no sistema: um único nó mestre, e um conjunto de nós escravos. Periodicamente, o mestre pergunta a todos os nós escravos o seu horário local e cada nó responde com sua hora atual.



O mestre então calcula a diferença de horário entre o seu relógio e a de cada um dos escravos (derivando um conjunto de valores delta, um por escravo). A seguir o mestre calcula **avg**, uma média que utiliza um critério de tolerância a falha: uma média sobre o conjunto dos maiores valores dos valores delta que diferem um do outro no máximo por um valor pré-definido **d**.

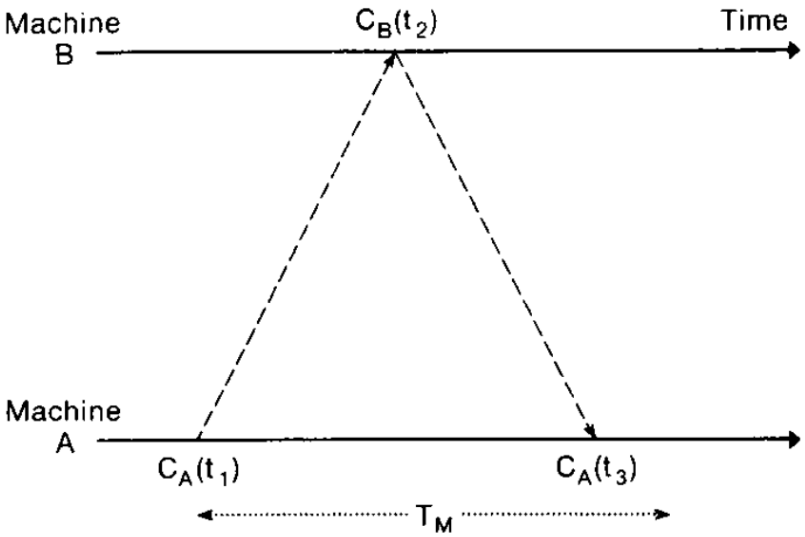


$$Av = \frac{0 - 10 - 5}{3} = -5$$

Na figura acima, o delta da máquina Slave 3 foi descartado por estar acima da tolerância, ou seja, foi considerado *com falha*.
Então, para cada escravo, o mestre calcula um valor de correção do relógio pela diferença entre o **avg** e o valor delta para o escravo.
Finalmente, o mestre envia este valor de correção de cada escravo para os escravos ajustarem seus relógios.

Descrição do protocolo de baixo-nível:

Considere um daemon de tempo rodando em uma máquina A, que mede a diferença entre os relógios das máquinas A e B através de um carimbo de tempo em uma mensagem no instante $C_A(t_1)$ e então envia esta mensagem à máquina B. O kernel da máquina B “bate” um carimbo de tempo na mensagem no instante $C_B(t_2)$ e a envia de volta. Quando a mensagem é recebida, vindo da máquina B, o daemon de tempo a lê no instante $C_A(t_3)$. Este processo é representado na figura abaixo:



O daemon de tempo poderá então estimar $\Delta_{AB}(t_3)$ que é a diferença entre os relógios das máquinas A e B, dado por:

$$\frac{C_A(t_1) + C_A(t_3)}{2} - C_B(t_2)$$

Como indicado, Δ_{AB} é uma função do tempo, mas nós assumimos que sua variação no intervalo t_3 - t_1 é tão pequena que podemos escrever:

$$\Delta_{AB}(t_3) = \Delta_{AB}(t_1) = \Delta_{AB}$$

Note também que $\Delta_{BA} = -\Delta_{AB}$

Para fins de aprofundamento, veja na figura abaixo, vinda do paper original, esta mesma explicação. É recomendado fazer a leitura do paper original.

III. The Clock Difference Measurement Algorithm

A time daemon program on machine A measures the time difference between the clock of machines A and B by timestamping a message at time $C_A(t_1)$ and sending it to machine B. The kernel of machine B timestamps that messages at time $C_B(t_2)$ and sends it back.[1] Upon receipt of the message from machine B, the time daemon reads the time $C_A(t_3)$. This process is represented in Fig. 5. As derived in Theorem 1 below, the time daemon can then estimate $\Delta_{AB}(t)$, the difference between the clocks of machines A and B, as

$$\frac{C_A(t_1) + C_A(t_3)}{2} - C_B(t_2).$$

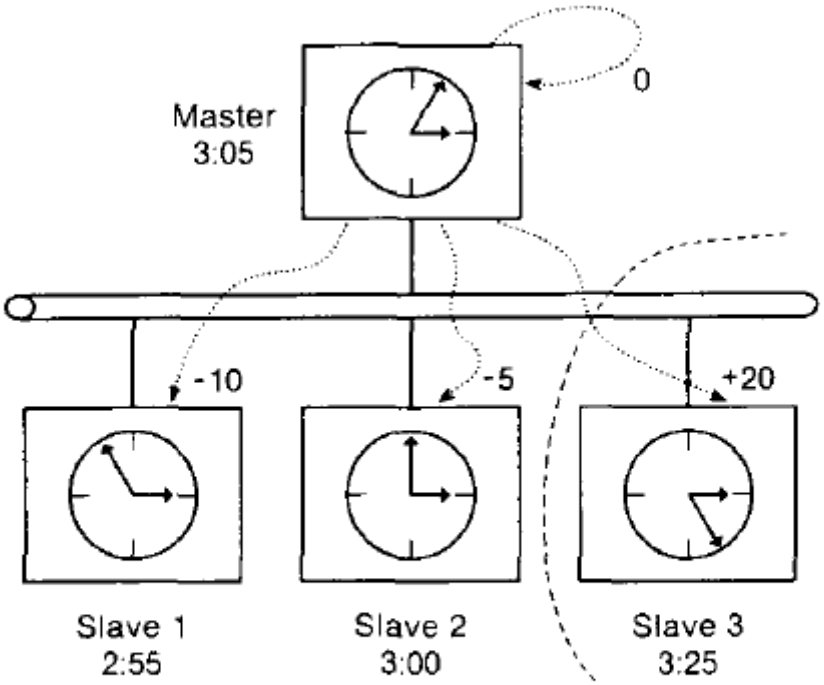
As indicated, Δ_{AB} is a function of time, but we assume that its variation in the interval $t_3 - t_1$ is so small that we can write

$$\Delta_{AB}(t_3) = \Delta_{AB}(t_1) = \Delta_{AB}.$$

Also, notice that $\Delta_{BA} = -\Delta_{AB}$.

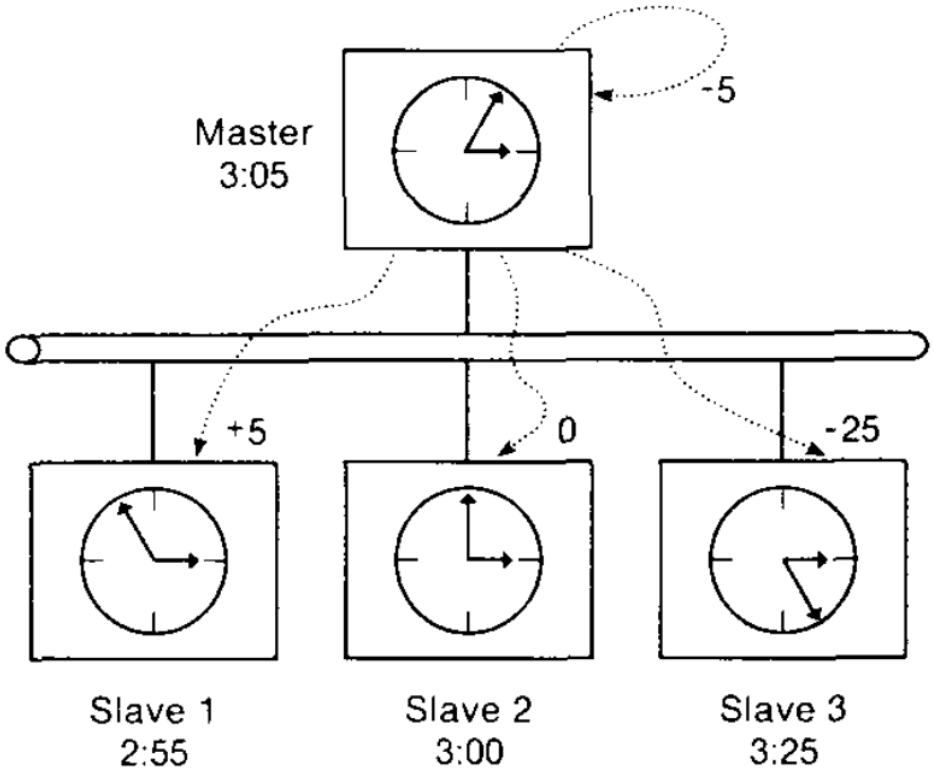
Algoritmo de Sincronização

O daemon de tempo master, utilizando o algoritmo de estimativa da diferença dos relógios, calcula as diferenças entre o seu relógio e os relógios das máquinas escravas. Para se prevenir contra relógios com mal funcionamento bem como contra relógios com valores anormalmente grandes de diferença, e como estes afetam adversamente os outros relógios, o daemon utiliza uma função de cálculo de média com tolerância a falha. Esta função seleciona o maior conjunto de relógios que não diferem uns dos outros por mais do que uma pequena quantidade γ e então calcula a média destes relógios. Por exemplo, na figura abaixo, assumindo que γ seja 10 minutos, a função de cálculo com tolerância a falha seleciona o conjunto consistindo do relógio do Master, o relógio do Slave 1 e o relógio do Slave 2. Relógios que não são selecionados pela função com tolerância a falha são considerados **falhos**.



$$Av = \frac{0 - 10 - 5}{3} = -5$$

Por último, o daemon de tempo master solicita a cada escravo para corrigir seu relógio por uma quantidade igual à diferença entre o valor da média provido pela função de cálculo com tolerância a falha e o valor anteriormente medido da diferença entre os relógios do mestre e do escravo. O próprio daemon de tempo master também corrige seu relógio da mesma forma.
Este processo é repetido a cada ***T*** segundos.
É importante notar que o daemon de tempo master produzirá valores de correção apropriados para cada relógio, inclusive aqueles considerados **falhos**, conforme mostrado na figura a seguir:



Requisitos de Implementação:

- O mestre precisa também sincronizar o seu relógio juntamente com os escravos, conforme descrito no paper original
- Durante todo o tempo em que um escravo estiver vivo, o seu relógio deve ser eventualmente sincronizado (com os tempos dos outros escravos vivos e do mestre).
- Deve ser utilizado o protocolo UDP para as mensagens.
- Sua implementação deve ser robusta com relação a perda de mensagens.
- Sua implementação deve ser robusta com relação a atrasos de mensagens (p.explo, escravos que estejam respondendo a uma rodada anterior de sincronização devem ser ignorados)
- Sua implementação deve ser robusta com relação às falhas de travamento de escravos (o mestre deve prosseguir com a sincronização conquanto que pelo menos um escravo responda com o seu tempo)
- Cada nó em sua implementação deve manter e sincronizar o relógio de processo local (não o relógio do sistema).
- Cada nó deve periodicamente “logar” (registrar) seu horário local, particularmente antes e depois da sincronização.
- Você pode utilizar **Java**, **Python** ou **Go** como linguagens de programação.

Hipóteses que você pode fazer:

- O nó mestre não falha.
- O conjunto de nós escravos (vivos ou falhos) não muda.

Dicas:

- Evite tratar explicitamente a perda de mensagens.
- Limite a duração de uma rodada de sincronização.

Especificações da solução:

Escreva um único programa que age como um mestre de tempo ou um escravo de tempo, dependendo das opções de linha de comando passadas ao programa.
Dado uma flag **-m** o programa deve se comportar com como um **mestre do tempo** e esperar os seguintes argumentos adicionais:

- ip:port : endereços que o mestre deve utilizar para se comunicar com todos os escravos.
- time : o relógio local de processo que o mestre deve inicializar.
- d : o limite **d** que é utilizado no cálculo da média com tolerância a falha (veja explicação acima).
- slavesfile : um arquivo com tantas linhas quanto necessárias, cada linhas contém um endereço de escravo no formato: “ip:port”.
- logfile : nome do arquivo no qual devem ser gravadas as mensagens de log.

Dado um flag -s, o programa deve se comportar como um escravo do tempo e esperar os seguintes argumentos:

- ip:port : endereço que o mestre deve escutar, esperando pelas mensagens de servidores.
- time : o relógio local de processo que o escravo deve ser inicializado.
- logfile : nome do arquivo para o qual as mensagem

Considerações Finais

Este projeto vai permitir o aprendizado de sincronização de relógios em um ambiente distribuído. Implementaremos o **Algoritmo de Berkeley**.

Submissão

O projeto deverá ser entregue na data especificada no Moodle e deve ser submetido eletronicamente pelo Moodle, com todos os arquivos zipados. Os seguintes arquivos devem ser submetidos:

- Todos os **códigos fonte**
- Um arquivo **Readme**, que descreve brevemente todos os arquivos submetidos. No arquivo Readme você deve prover também informação acerca do ambiente de compilação, passos para a compilação, instruções de execução, etc. Você deve prover tanta informação detalhada quanto você julgue que possa ajudar o processo de correção e execução do seu código.
- Um arquivo **Amostra**, que descreve os testes que você executou no seu programa para se certificar de sua correção. Também descreva quaisquer casos para os quais o seu programa não esteja funcionando corretamente. Enviar os arquivos de logs dos servidores (dados e negócio).
- Um arquivo **Projeto**, que descreve o projeto global do programa, uma descrição verbal de "como ele funciona", incluindo as bases do que o sistema está fazendo e as deciões de projeto consideradas e tomadas. Também descreva possíveis melhorias e extensões ao seu programa (e rascunhe como elas poderiam ser feitas).

Notas

As notas serão (mais ou menos) baseadas nos seguintes elementos:

- Todos os arquivos necessários foram submetidos? (5%)
- O código fonte é fácil de entender? (i.e., boa estrutura e comentários) (25%)
- O sistema roda corretamente? (25%)
- O sistema foi testado por você com sucesso? (prints e logs das execuções) (20%)
- Cobertura dos casos de teste (5%)
- Documentação e Relatório do Projeto (20%)

COMECE O QUANTO ANTES. É UM PROJETO DESAFIADOR.
Leia atentamente o artigo "How Not to Go About a Programming Assignment". Busque no Google.