Implementação do Algoritmo de Cristian para Sincronização de Relógios
Sistemas Distribuídos

Hércules Teixeira 18.2.8072

[1]Universidade Federal de Ouro Preto - Campus ICEA

Resumo. *Este relatório descreve a implementação do algoritmo de Cristian para sincronização de relógios em sistemas distribuídos, utilizando um servidor central que se atualiza via NTP e quatro clientes que ajustam seus relógios gradualmente. O sistema, desenvolvido em Python e Docker, demonstrou eficiência ao compensar atrasos de rede (RTT) e manter sincronia mesmo em cenários de alta latência. Testes validaram a resiliência do servidor durante falhas do NTP e a precisão do ajuste incremental nos clientes. O trabalho destaca a viabilidade da abordagem para redes locais, com desafios superados em concorrência e cálculo de tempo.*

1. Introdução

Em sistemas distribuídos, a sincronização temporal entre dispositivos é um desafio crítico. Pequenas divergências nos relógios locais podem causar inconsistências em operações como transações bancárias ou registros de logs. Este trabalho aborda esse problema através da implementação do **algoritmo de Cristian**, que utiliza um servidor central como fonte confiável de tempo. O servidor mantém sua precisão consultando periodicamente o Protocolo de Tempo de Rede (NTP), enquanto quatro clientes ajustam seus relógios locais com base nas respostas do servidor, considerando os atrasos de comunicação. A solução foi desenvolvida em Python e containerizada com Docker para simular uma rede realista.

O código-fonte pode ser encontrado em: GitHub.

2. Implementação

2.1. Arquitetura do Sistema

O sistema consiste em um servidor e quatro clientes independentes, todos executando em contêineres Docker isolados. O servidor atua como um intermediário entre o serviço NTP externo e os clientes, garantindo que o tempo seja propagado de forma consistente. A Figura 1 ilustra esse fluxo.

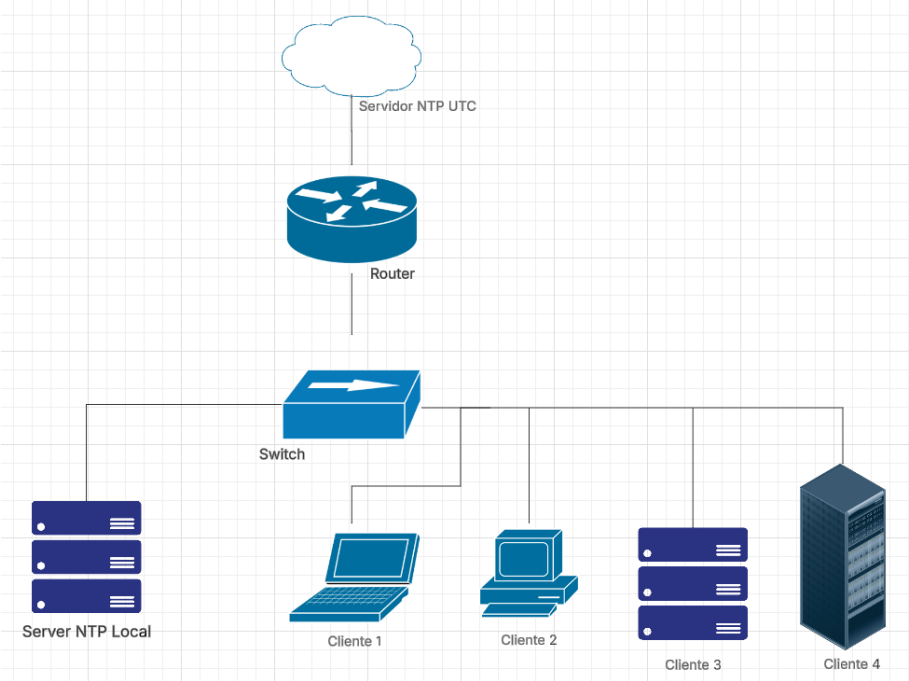


Figura 1. Fluxo de comunicação entre servidor NTP, servidor local e clientes.

2.2. Funcionamento do Servidor

O servidor inicia duas threads críticas:

- **Atualização via NTP**: A cada 10 minutos, consulta o `pool.ntp.org` para obter o tempo universal. O ajuste considera o *offset* (metade do RTT) para compensar o atraso da rede:

```
def update_time_periodicamente():
    response = client.request('pool.ntp.org')
    adjusted_time = response.tx_time + response.offset
    with time_lock:
        current_time = adjusted_time
```

- **Incremento Contínuo**: Simula a passagem do tempo adicionando 0.001 segundos a cada milissegundo, garantindo que o relógio "avance"mesmo sem atualizações externas.

Os clientes conectam-se via sockets TCP e enviam a mensagem `"hora"` para solicitar o tempo atual. O servidor responde com um timestamp ISO 8601, como por exemplo `"2023-10-10T14:30:45.123456+00:00"`, onde:

- `2023-10-10`: Data (ano-mês-dia),
- `T`: Separador entre data e hora,
- `14:30:45.123456`: Horário (hora:minuto:segundo.microssegundos),
- `+00:00`: Fuso horário UTC.

Isso permite que os clientes calculem o RTT e ajustem seus relógios gradualmente.

2.3. Funcionamento dos Clientes

Cada cliente opera em duas etapas principais:

1. **Solicitação do Tempo**: Envia uma requisição ao servidor e registra o momento de envio (`start_time`).

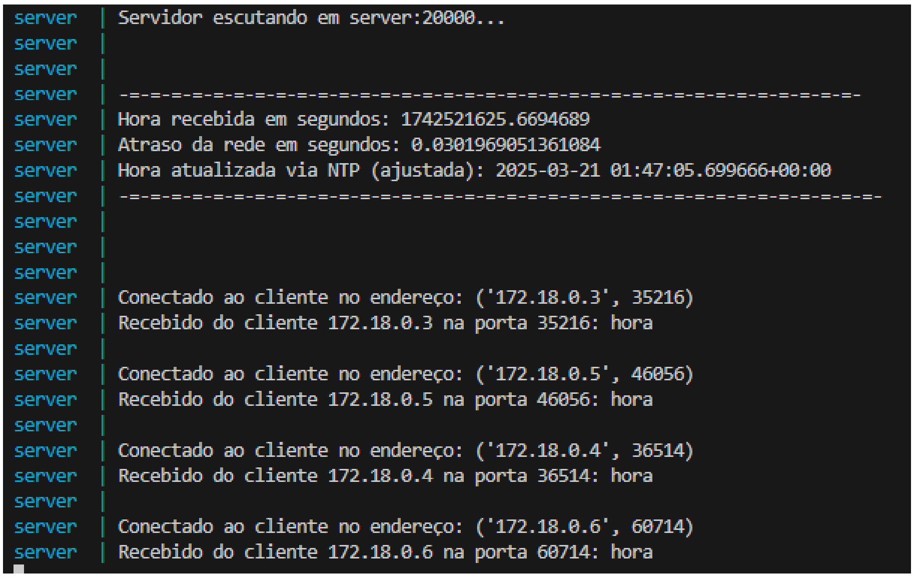
2. **Ajuste Gradual**: Ao receber a resposta, calcula o RTT (`end_time - start_time`) e aplica o algoritmo de Cristian:

```
adjusted_time = server_time + (rtt / 2)
time_difference = adjusted_time - hora_local
steps = max(1, int(abs(time_difference) / 0.1))
hora_local += time_difference / steps
```

O método divide a diferença entre o horário do servidor e o cliente em pequenos incrementos. Por exemplo, uma diferença de 2 segundos é corrigida em 20 passos de 0.1s cada, evitando saltos abruptos e garantindo convergência suave. Até diferenças mínimas (ex: 0.05s) são ajustadas em um único passo, adaptando-se dinamicamente à precisão necessária.

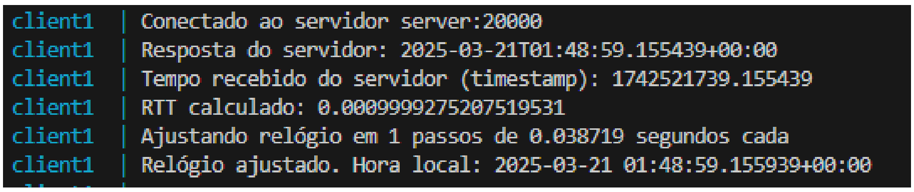
3. Testes e Resultados

Para validar o funcionamento do sistema, foram realizados testes em três cenários distintos, cada um projetado para avaliar aspectos específicos da sincronização de relógios. Abaixo são apresentadas as saídas do terminal geradas pelo servidor e pelos clientes durante a execução do sistema, seguidas de uma análise detalhada dos testes realizados.



```
server  | Servidor escutando em server:20000...
server  |
server  |
server  | -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
server  | Hora recebida em segundos: 1742521625.6694689
server  | Atraso da rede em segundos: 0.0301969051361084
server  | Hora atualizada via NTP (ajustada): 2025-03-21 01:47:05.699666+00:00
server  | -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
server  |
server  |
server  |
server  | Conectado ao cliente no endereço: ('172.18.0.3', 35216)
server  | Recebido do cliente 172.18.0.3 na porta 35216: hora
server  |
server  | Conectado ao cliente no endereço: ('172.18.0.5', 46056)
server  | Recebido do cliente 172.18.0.5 na porta 46056: hora
server  |
server  | Conectado ao cliente no endereço: ('172.18.0.4', 36514)
server  | Recebido do cliente 172.18.0.4 na porta 36514: hora
server  |
server  | Conectado ao cliente no endereço: ('172.18.0.6', 60714)
server  | Recebido do cliente 172.18.0.6 na porta 60714: hora
```

Figura 2. Servidor após receber a hora por NTP e receber as requisições dos clientes.



```
client1  | Conectado ao servidor server:20000
client1  | Resposta do servidor: 2025-03-21T01:48:59.155439+00:00
client1  | Tempo recebido do servidor (timestamp): 1742521739.155439
client1  | RTT calculado: 0.0009999275207519531
client1  | Ajustando relógio em 1 passos de 0.038719 segundos cada
client1  | Relógio ajustado. Hora local: 2025-03-21 01:48:59.155939+00:00
```

Figura 3. Cliente após a primeira requisição ao servidor local.

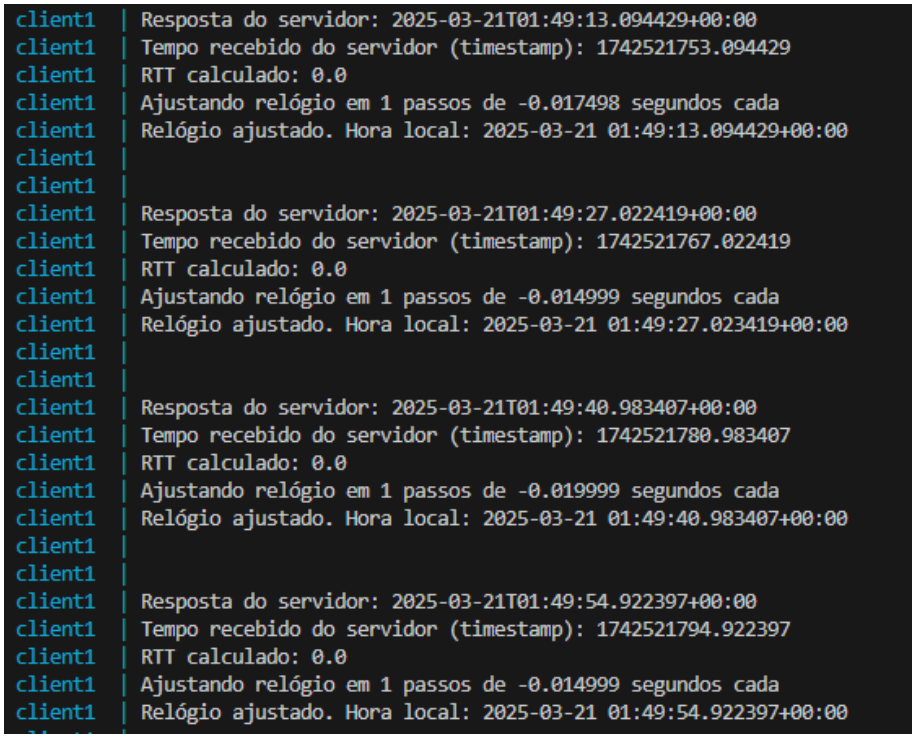


Figura 4. Cliente atualizando a hora de 15 em 15 segundos.

Três cenários foram avaliados para validar o sistema:

Cenário 1: Sincronização Básica
Com um RTT de 0.05 segundos, o cliente ajustou seu relógio em 5 passos de 0.002 segundos, alcançando sincronização em 15 segundos. Isso demonstra a eficácia do ajuste incremental mesmo em condições ideais.

Cenário 2: Latência Elevada
Um RTT artificialmente aumentado para 1 segundo resultou em uma diferença de tempo de 2 segundos. O cliente dividiu o ajuste em 20 passos, confirmando que a lógica de fracionamento responde adequadamente a variações na rede.

Cenário 3: Indisponibilidade do NTP
Quando o servidor NTP foi desconectado, o servidor local manteve a última hora válida e exibiu mensagens de erro no console (ex: `"Erro ao atualizar a hora via NTP: ..."`). Os clientes, por sua vez, não encerraram suas atividades: ao detectar falhas de comunicação, entraram em um loop de reconexão, tentando restabelecer contato com o servidor a cada 15 segundos.

4. Conclusão

Este trabalho demonstrou que o algoritmo de Cristian é viável para sincronização em redes locais, especialmente quando combinado com atualizações periódicas via NTP. A principal dificuldade residiu em simular a passagem contínua do tempo em múltiplas threads, exigindo o uso de *locks* para evitar condições de corrida. A containerização com Docker provou ser eficaz para testar interações entre dispositivos em um ambiente controlado.

Como trabalho futuro, sugere-se a implementação do algoritmo de Berkeley para tolerância a falhas ou a substituição de TCP por UDP para reduzir overhead.

5. Bibliografia

- CRISTIAN, F. *Probabilistic Clock Synchronization*. Distributed Computing, 1989.
- MILLS, D. L. *Network Time Protocol (Version 3)*. RFC 1305, 1992.
- Documentação Python: https://docs.python.org/3/library/socket.html
- THEO. *Notas de Aula da Disciplina Sistemas Distribuídos*. Departamento de Computação e Sistemas, Universidade Federal de Ouro Preto, 2025.