Algoritmo para sincronização de relógios físicos em sistemas distribuídos

Rônitti Juner da S. Rodrigues[1], Robson A. Lima[1], Diógenes Antonio M. José[1]

[1]Universidade do Estado de Mato Grosso - UNEMAT
Faculdade de Ciências Exatas e Tecnológicas - FACET
Coordenação de Ciência da Computação - CCC
Caixa Postal 92 - 78.390-000 - Barra do Bugres - MT - Brasil

{ronittijuner, robson.conq, dioxfile}@gmail.com

Abstract. *The synchronization of clocks in distributed systems allows the communication of processes in an orderly way, however its comprehension and execution are not a trivial task. Therefore, this article proposes an application capable of synchronizing physical clocks, considering Lamport's local time concept. The application was developed in Python and its evaluation was performed in a testbed with 16 computers. The results showed that the proposed application synchronized, in seconds, all computers clocks. In addition, it can be used as support in learning distributed systems since it allows to visualize the theoretical concepts of synchronization in practice.*

Resumo. *A sincronização de relógios em sistemas distribuídos permite a comunicação de processos de forma ordenada, todavia sua compreensão e execução não são uma tarefa trivial. Portanto, este artigo propõe um aplicativo capaz de sincronizar relógios físicos, considerando o conceito de hora local de Lamport. O aplicativo foi desenvolvido em Python e sua avaliação foi realizada em um testbed com 16 computadores. Os resultados mostraram que o aplicativo proposto sincronizou, em poucos segundos, todos os relógios dos computadores. Além disso, o mesmo pode ser usado como auxílio no aprendizado de sistemas distribuídos uma vez que ele possibilita visualizar na prática os conceitos teóricos de sincronização.*

1. Introdução

A sincronização de relógios em sistemas distribuídos consiste em manter os relógios dos computadores atualizados de modo que processos locais ou em diferentes computadores possam se comunicar de forma ordenada. Com o tempo, os temporizadores dos computadores, também conhecidos como relógios, sofrem uma defasagem. Neste contexto, é impossível garantir que todos os relógios, em todos os computadores, funcionem exatamente na mesma frequência [1]. Todavia, com um único computador e apenas um relógio não é necessário se preocupar com a defasagem do mesmo. Entretanto, em sistemas distribuídos, a defasagem dos relógios pode se tornar um sério problema. Por exemplo, o programa *make* do *Unix* é usado geralmente em projetos que possuem uma grande quantidade de códigos fontes a serem compilados. O funcionamento do programa é básico. Se o arquivo fonte *teste.c* foi modificado às *16:35* e o arquivo objeto *teste.o* marca a hora *16:30*, então o arquivo fonte *teste.c* foi alterado e precisa ser compilado. Dessa forma, o

make compilará apenas os arquivos fontes que foram modificados. Agora, imagine que a mesma situação aconteça em um sistema distribuído em que o fonte *teste.c* é editado em um computador com a seguinte marca de tempo *16:10*, já o arquivo objeto está em um computador remoto com a seguinte marca de tempo *16:50*. Assim, o *make* não compilará o código alterado e o arquivo objeto não funcionará como deveria.

Há muitos outros exemplos em que é necessário ter um horário global comum. Entretanto, o que importa é saber qual evento aconteceu antes do outro. No caso do *make*, basta apenas saber qual arquivo fonte foi alterado antes do arquivo objeto ser gerado. Há varias décadas cientistas desenvolvem algoritmos para a sincronização de relógios [2, 3, 4]. A sincronização é importante para vários tipos de aplicações, principalmente as que funcionam em sistemas distribuídos.

Diante do exposto, o objetivo deste trabalho consiste em desenvolver um aplicativo para realizar a sincronização de relógios utilizando o conceito de hora local desenvolvido por Lamport [2]. Além disso, como objetivo secundário a proposta de aplicativo visa auxiliar o processo de aprendizagem de sincronização de relógios na disciplina de sistemas distribuídos, uma vez que o aplicativo em questão permitirá visualizar na prática os conceitos que são vistos apenas em teoria. O aplicativo foi desenvolvido em *Python* e o mesmo foi avaliado em um *testbed* com 16 computadores no laboratório de redes da Universidade do Estado de Mato Grosso *(UNEMAT)*, Campus de Barra do Bugres. A solução proposta foi avaliada em sistemas *Unix*, todavia a mesma pode ser instalada em sistemas *Microsoft Windows*. Os resultados mostraram que a aplicação mantém todos os computadores sincronizados por meio da difusão de suas horas locais.

Para melhor compreensão do conteúdo, o artigo foi organizado como segue: a Seção 2 apresenta os trabalhos relacionados. Na Seção 3, é apresentada a metodologia utilizada. A Seção 4 descreve, em detalhes, o aplicativo proposto. Na Seção 5, são apresentados os resultados e discussão. E por fim, na Seção 6, são apresentadas a conclusão e as possibilidades de trabalhos futuros.

2. Trabalhos relacionados

O objetivo desta seção consiste em descrever alguns dos principais trabalhos sobre sincronização de relógios, os quais servem como base para compreensão do aplicativo proposto.

No trabalho de Cristian *et al.* [4] é proposto um protocolo prático para a sincronização de relógios em ambientes distribuídos. A importância do desenvolvimento desse protocolo se deve ao fato da implementação de algoritmos que funcionam na presença de omissões, por exemplo, falhas do processador e/ou enlaces com auto atraso. Assim, o tempo é mantido sincronizado dentro de uma faixa de erro previamente definida e, dessa forma, é possível saber o tempo decorrido entre eventos de um mesmo processo. Por exemplo, um cliente solicita o tempo de um servidor. Após receber a solicitação o servidor envia uma resposta ao cliente anexando o valor de seu relógio T. Então, o cliente configura o tempo calculando $T + Round\ Trip\ Time\ (RTT)/2$.

Lamport em [2] trata da sincronização de eventos que ocorrem em processos locais e remotos levando em consideração a hora local. Segundo Lamport, a hora global não é importante, mas sim a relação *aconteceu antes*. A relação *aconteceu antes* é composta por três condições básicas: *(I)* Se a e b são eventos em um mesmo processo e a acontece

antes de b, então $a \rightarrow b$; *(II)* Se a é o envio de uma mensagem por um processo e b é o recebimento da mesma mensagem por outro processo, então $a \rightarrow b$; e *(III)* Se $a \rightarrow b$ e $b \rightarrow c$, então $a \rightarrow c$. Neste mesmo trabalho, também é definido o conceito de hora local através de relógios lógicos. Um relógio lógico é um número associado a um evento. Tal número representa o tempo no qual o evento aconteceu. A representação é feita da seguinte forma: cada processo P_i possui uma função C_i que determina um número $C_i(a)$ para cada evento a naquele processo. Assim, todo o processo é representado por meio da função C, que assimila a qualquer evento b o número $C(b)$, onde $C(b) = C_j(b)$ se b é um evento no processo P_j. Embora C_i seja uma representação lógica que não tem nenhuma relação com os relógios físicos, este conceito pode ser aplicado em redes que não estão conectadas à *Internet* ou que não possuem um servidor de hora local, pois estas podem usar o conceito de hora local para sincronizarem seus relógios.

Guzella e Zatti [5] propõem um algoritmo de sincronização denominado *Algoritmo de Berkeley*. Este algoritmo é ativo e consulta o tempo através de um *daemon*. Um servidor rodando um *daemon* de tempo pergunta aos outros computadores as horas de seus relógios, estes respondem, então o servidor calcula a hora média e informa a cada computador como deve acertar seu relógio.

O *Network Time Protocol (NTP) RFC 1059*, idealizado por David Mills da Universidade de Delaware (*USA*), consiste em um protocolo para sincronização de relógios dos computadores. O *NTP* usa o protocolo *User Datagram Protocol (UDP)* para receber solicitações na porta 123. Atualmente o *NTP* está na versão 4 *RFC 5905* e é amplamente utilizado para sincronizar computadores na *Internet* através de servidores de hora. Sua estrutura é composta por uma hierarquia em árvore e tem como fonte de referência a estação de rádio do *national standard time* [6]. Ele pode ser instalado no sistema operacional, *Windows* ou *Linux*, entretanto é necessário ter como base uma hora de referência para seu funcionamento.

3. Descrição da proposta

O aplicativo foi desenvolvido em linguagem *Python* com característica *multithreading* e utilizando programação orientada a objetos. Sua arquitetura é composta de três classes: *TSGUI*, *TSServer* e *TSClient*. A classe *TSGUI* é responsável por gerar e manipular a interface gráfica, que é apresentada ao usuário. Nesta interface, Figura 1, é possível fazer o controle de início da sincronização, visualizar hora atual e verificar quais computadores estão em sincronia. Ela faz uso da biblioteca *PyGTK 2.26*, disponível sob licença *LGPL*, que permite facilmente criar os componentes como menus, controles e *widgets* utilizados na interface.

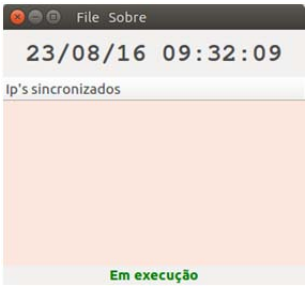


Figura 1. Interface gráfica do aplicativo.

A classe *TSServer* implementa as funcionalidades do servidor. Este é responsável por receber e analisar as mensagens originadas do cliente. Esta instância é executada em uma *Thread* e se mantém aguardando uma mensagem. Assim, quando o servidor recebe uma mensagem com o *timestamp* do relógio do cliente, ele compara o tempo recebido com o seu e, caso seja menor, envia por meio de uma mensagem *unicast* o seu relógio local para o cliente (e.g., um valor float em segundos e milissegundos). Se o valor recebido for igual, maior, ou originar do mesmo computador em que o servidor está executando, nenhuma resposta é retornada pelo servidor. O Algoritmo 1 apresenta as ações executadas no servidor.

Algorithm 1 Pseudocódigo da função *listen* da classe servidor.

```
Input: Relogio Remoto
Output: Relogio Local
1: procedure LISTEN
2:     while true do
3:         relogioremoto, ipcliente ← SOCK.RECVFROM()
4:         if relogioremoto < relogiolocal then
5:             SOCK.SENDTO(ipremoto , relogiolocal)
6:         end if
7:     end while
8: end procedure
```

A classe *TSCliente* implementa as funcionalidades do cliente. Este difunde uma mensagem com seu tempo atual, em segundos e milissegundos, para toda a rede (e.g., *broadcast*). A partir deste momento, o cliente aguarda o recebimento de uma mensagem de algum servidor. Ele aguarda por um tempo máximo de $1s$ e se nenhuma mensagem for recebida, uma exceção é gerada por estouro do *timeout*, e a espera da mensagem é interrompida. Todavia, quando uma mensagem é recebida, faz-se uma estimativa simples do atraso que possibilita a sincronização do relógio local, Algoritmo 2. Basicamente

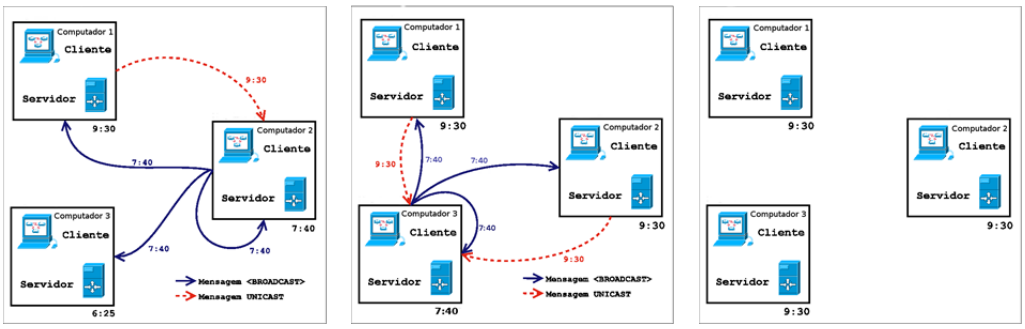
Algorithm 2 Pseudocódigo da função *sendmessage* da classe cliente.

```
Input: Relogio Remoto
Output: Relogio Local
 1: procedure SENDMESSAGE
 2:     SOCK.SENDTO(broadcast, relogiolocal)
 3:     datadeenvio ← relogiolocal
 4:     relogioremoto , ipservidor ← SOCK.RECVFROM( )
 5:     if relogioremoto then
 6:         atraso ← (datadeenvio − relogiolocal) ÷ 2
 7:         if relogioremoto > relogiolocal then
 8:             relogioremoto ← relogioremoto + atraso
 9:             CHANGETIME(relogioremoto)
10:         end if
11:     end if
12: end procedure
```

o tempo de atraso T_{atraso} do Algoritmo 2 é calculado com base no método de Cristian apresentado em [4]. O tempo T_{envio} é registrado no momento do envio da mensagem e quando uma mensagem é retornada ao cliente obtém-se $T_{recebido}$. Tem-se então o tempo $T_{RTT} = T_{recebido} - T_{envio}$. Assumindo que o tempo é igualmente dividido, obtém-se: $T_{atraso} = T_{RTT}/2$. Com T_{atraso} obtido, então é feita a soma com o tempo recebido do servidor T_{serv} e esta é comparada com a hora atual do cliente. Se a hora do cliente for

menor, o relógio do cliente é configurado com $T_{serv} + T_{atraso}$. Para manter a sincronia, o cliente envia a cada $5s$ uma mensagem na rede.

Por exemplo, considere três computadores interligados por uma rede, como mostra a Figura 2. Eles têm seus relógios fora de sincronia. Note que na Figura 2(a), o cliente no computador 2 envia uma mensagem com o *timestamp*, por *broadcast*, na rede. Observa-se também que os computadores 1, 2 e 3 recebem a mesma mensagem. Entretanto, somente o computador 1 possui seu relógio maior, adiantado, que o da mensagem recebida. Assim, somente o servidor deste computador envia ao computador 2 uma mensagem *unicast* contendo seu relógio. Deste modo, o computador 2 atualiza seu relógio com o valor recebido. Na Figura 2(b), o cliente do computador 3 envia uma mensagem via *broadcast* e os computadores 1 e 2 recebem a mesma. Os servidores dos dois computadores verificam que o valor de relógio recebido é menor. Dessa forma, cada um envia uma mensagem com o valor de seu relógio para o computador 3. A primeira mensagem que chega até ele é recebida, e seu relógio é atualizado. Assim, observa-se na Figura 2(c) que os relógios de todos os computadores foram sincronizados.



(a) Computador 2 sincronizando. (b) Computador 3 sincronizando. (c) Computadores sincronizados.

Figura 2. Modelo de sincronização com três computadores.

4. Materiais e métodos

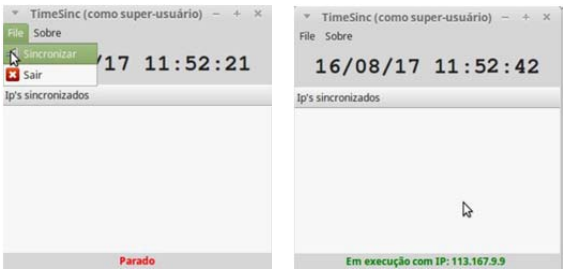
A técnica de pesquisa utilizada neste trabalho foi a revisão bibliográfica, tendo como base referenciais teóricos que tratam de sincronização como o apresentado por Lamport. Portanto, para o desenvolvimento do aplicativo e execução dos testes, a metodologia foi dada por:

- **Softwares utilizados**: linguagem de programação *Python 2.7*, biblioteca de interface gráfica *PyGTK 2.26*, IDE *Eclipse 4.6.0 (Neon)* com o *plugin PyDev 5.1*, Sistema Operacional *Ubuntu 16.04 LTS 64bits*, e Sistema Operacional *Linux Mint 17 32bits*;
- **Tipo de socket**: *socket UDP*, pois as mensagens são trocadas por *broadcast* e isso não seria possível em um *socket* orientado à conexão como o *TCP*;
- **Hardware para desenvolvimento**: *Intel (R) Core (TM) i5-3337U CPU 1.80GHz × 4*, *8GB* de memoria *RAM*, *HD* híbrido *500GB + 32GB SSD*;
- **Hardware do ambiente de teste**: *Intel (R) Pentium 4 (TM) 3.0GHz*, *1GB* de memoria *RAM*, *HD 40 GB*, rede *Fast Ethernet 10/100*;
- **Testes realizados**: os testes do aplicativo foram feitos em um *testbed* com 16 computadores, *Laboratório de Redes de Computadores da (UNEMAT)*, e todos os computadores do laboratório foram configurados em uma rede classe

A 113.167.9.0/24. Para que a sincronização ocorre-se, foi adotado como hora correta a maior hora. Dessa forma, os relógios de 15 computadores foram atrasados e o 16º foi configurado com a maior hora local. Além disso, foi verificado se todos os computadores atualizavam corretamente. Entre os testes realizados, um dos computadores teve sua hora adiantada para perceber se todos os outros também seriam atualizados, além disso, cada computador foi atrasado manualmente para verificar se ele entrava novamente em sincronia.

5. Resultados e Discussão

Para efeitos de simplificação, esta seção apresenta os resultados dos testes realizados em apenas três computadores. Os testes de sincronização foram feitos nos computadores 113.167.9.9, 113.167.9.13 e 113.167.9.14. O primeiro passo consistiu em executar o aplicativo, para isso foi necessário instalar a biblioteca *PyGTK* na versão 2.26. Como é necessário alterar a hora do sistema, o aplicativo foi iniciado como administrador, *root*. Ao ser iniciado, o aplicativo apresenta a data e hora corrente e a sincronização é acionada através do menu *"file"*, Figura 3.

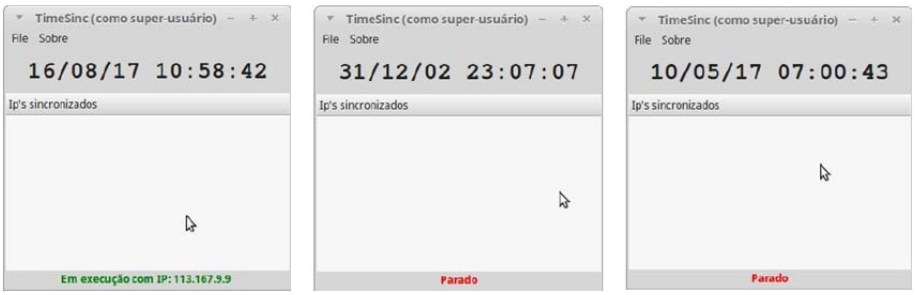


(a) Tela 1, parado. (b) Tela 2, em execução.

Figura 3. Iniciando a sincronização no computador 113.167.9.9.

A interface gráfica desenvolvida para o aplicativo é simples, intuitiva e de fácil uso. Nela é apresentado o relógio local com a data e hora, eliminando assim a necessidade de verificar a hora na bandeja do sistema. O aplicativo foi iniciado em três computadores, de modo que o computador 1 teve sua hora correta mantida e os computadores 2 e 3 tiveram suas horas atrasadas, Figura 4(a)(b)(c). Conforme mostra a Figura 4(d)(e), os computadores 2 e 3 foram atualizados com base no computador 1. Observa-se que há um espaço de tempo entre as horas, em função do tempo que foi levado para logar em cada computador e executar o aplicativo em cada um deles. Observa-se também que o aplicativo no computador 3, Figura 4(e), foi executado antes de sua execução no computador 2, Figura 4(d). Por causa disso, existe uma diferença de tempo nas horas apresentadas: *10:48:42* (computador 1), *11:10:32* (computador 3) e *11:14:38* (computador 2).

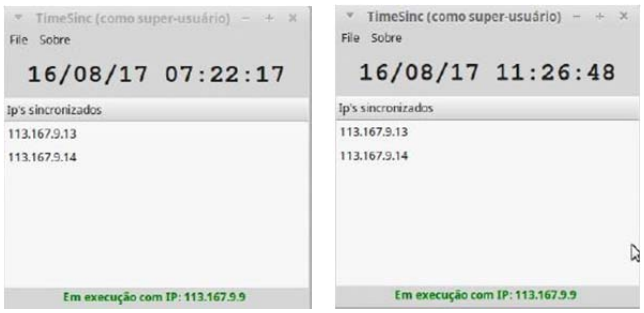
A Figura 5(a) apresenta outro teste no qual a hora do computador 1 é atrasada. Logo após o atraso, ele entrou novamente em sincronia com os computadores 2 e 3, Figura 5(b). Em outro teste, o computador 3 teve a hora adiantada, Figura 6(a). Assim, todos os outros computadores tiveram suas horas sincronizadas com o mesmo, por exemplo, o computador 2, Figura 6(b). A aplicação também gera dois arquivos de *log* (e.g., *server.log* e *client.log*), que possibilitam analisar as mensagens enviadas e recebidas por cada entidade, Figura 7.



(a) Computador 1. (b) Computador 2 atrasado. (c) Computador 3 atrasado.

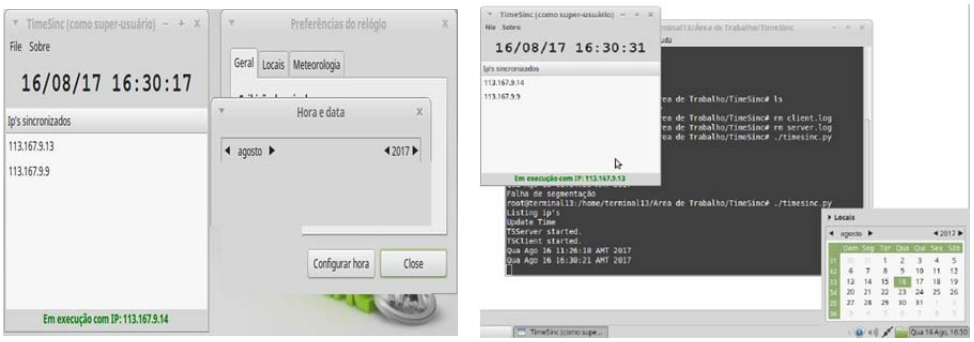
(d) Computador 2 sincronizado. (e) Computador 3 sincronizado.

Figura 4. Configuração da sincronização em três computadores.



(a) Computador 1 atrasado. (b) Computador 1 sincronizado.

Figura 5. Atraso da hora no computador 1.



(a) Computador 3, hora adiantada. (b) Computador 2 sincronizado.

Figura 6. Computador 3 com a hora adiantada para as 16:30.

```
Server: 113.167.9.14 is delayed. I'm answering.
Server: 113.167.9.14 sent to me 1502896240.8533, mine is 1502896240.8156
```

(a) Server *log*.

```
Client: Sending my time. 1502896364.0157
Response:  ServerTime 1502896235.9353 - MyTime 1502896364.0177 - Delay0.0010
```

(b) Client *log*.

Figura 7. Arquivos de *log* das entidades: servidor e cliente.

6. Conclusão

Este trabalho apresentou um aplicativo baseado no conceito de hora local proposto por Lamport, para sincronização de relógios físicos de *n* computadores ligados em rede. Além disso, o aplicativo proposto tem como um de seus objetivos servir como auxílio no aprendizado de sistemas distribuídos no que diz respeito à sincronização de relógios. Dentre as contribuições deste trabalho destacam-se: a proposta de um algoritmo de sincronização de relógios físicos em computadores legados; o desenvolvimento de um aplicativo em *Python* para sincronização de relógios; a utilização de programação *multithreading* e orientada a objetos; a aplicação dos conceitos de sincronização em sistemas distribuídos na resolução de um problema do mundo real; a utilização e promoção de *Software Livre*, pois todas as ferramentas usadas são livres; e o desenvolvimento de atividade prática e de laboratório como um exercício didático da disciplina de sistemas distribuídos.

Dessa forma, concluiu-se que o aplicativo proposto atingiu o objetivo esperado que consiste na sincronização de relógios em redes de computadores. Como trabalhos futuros, pode-se melhorar a eficiência do envio de mensagens, diminuindo o tráfego das mesmas na rede e consequentemente a economia de energia no caso de ambientes *wireless*. Entre as diversas possibilidades é possível implementar soluções tolerantes a falhas, por exemplo, falhas bizantinas, nós faltosos, queda de processos. Também, pode ser proposto a ordenação total dos eventos no aplicativo utilizando relógios vetoriais.

Referências

[1] A. S. Tanenbaum and M. V. Steen, *Sistemas Distribuídos: princípios e paradigmas*. São Paulo: Pearson Prentice Hall, 2th ed., 2007.

[2] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, pp. 558–565, Jul 1978.

[3] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *Journal of the Association for Computing Machinery*, vol. 32, p. 27, January 1985.

[4] F. Cristian, H. Aghili, and R. Strong, "Clock synchronization in the presence of omission and perfomance failures, and processor joins," in *16th IEEE Int. Symp. on Fault-tolerant Computing System*, (Vienna), 1986.

[5] R. Gusella and S. Zatti, "The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd," *IEEE Transactions on Software Engineering*, vol. 15, pp. 847–853, Jul 1989.

[6] D. Mill, "Network Time Protocol version 1." `https://tools.ietf.org/html/rfc1059`, 1989. [Último acesso: 18-Agosto-2017].