Revista Multidisciplinar do Nordeste Mineiro, v.12, 2024
ISSN 2178-6925

SINCPOINT UMA PROPOSTA DE FERRAMENTA DIDÁTICA PARA ENSINAR SINCRONIZAÇÃO DE RELÓGIOS EM SISTEMAS DISTRIBUÍDOS UTILIZANDO O ALGORITMO DE BERKELEY

SINCPOINT A PROPOSAL FOR A DIDACTIC TOOL TO TEACH CLOCK SYNCHRONIZATION IN DISTRIBUTED SYSTEMS USING THE BERKELEY ALGORITHM

Wilcker Richard Sierra Neckel
Undergraduate in Computer Science, Mato Grosso State University, Brazil
E-mail: wilcker.neckel@unemat.br
ORCID: https://orcid.org/0009-0007-0914-1538

Luís Gabriel Tavares Ferreira
Undergraduate in Computer Science, Mato Grosso State University, Brazil
E-mail: luis.gabriel@unemat.br
ORCID: https://orcid.org/0000-0001-8395-8844

Eduardo Campos de Oliveira
Undergraduate in Computer Science, Mato Grosso State University, Brazil
E-mail: eduardo.campos@unemat.br
ORCID: https://orcid.org/0000-0002-9707-6022

Diógenes Antonio Marques José
Master of Computer Science, Mato Grosso State University, Brazil
E-mail: dioxfile@unemat.br
ORCID: https://orcid.org/0000-0002-9707-6022

Resumo

Nos cursos de Ciência da Computação, a disciplina de sistemas distribuídos é essencial, e a sincronização de relógios é um dos tópicos centrais abordados nesse contexto. No entanto, devido à sua natureza abstrata, muitos alunos enfrentam dificuldades para compreender como a sincronização de relógios realmente funciona. Com isso, este artigo tem como objetivo apresentar uma ferramenta didática chamada ***SincPoint***, que visa facilitar o aprendizado de sincronização de relógios. O ***SincPoint*** é um aplicativo web desenvolvido em ***JavaScript***, projetado para simular de forma visual e interativa o funcionamento do ***Algoritmo de Berkeley***, um dos métodos mais importantes para sincronização de relógios em sistemas distribuídos. Após uma série de testes, observou-se que a ferramenta proposta contribui significativamente para o entendimento da sincronização de relógios, proporcionando uma experiência de aprendizado mais clara e acessível.

Palavras Chaves: Sistemas Distribuídos; Sincronização de Relógios; Ferramenta Didática; Algoritmo de Berkeley.

Abstract

In Computer Science courses, the study of distributed systems is fundamental, with clock

synchronization being one of its key topics. However, due to its abstract nature, many students find it challenging to grasp how clock synchronization operates in practice. This article introduces SincPoint, an innovative teaching tool designed to address this challenge by simplifying the concept of clocks synchronization. SincPoint is a web application built with JavaScript that provides an interactive and visual simulation of the Berkeley Algorithm, a widely-used method for clock synchronization in distributed systems. Through a series of tests, it was found that SincPoint significantly enhances students' understanding of clock synchronization, offering a more intuitive and accessible way to learn this complex topic.

Keywords: Distributed Systems; Clock Synchronization; Teaching Tool; Berkeley Algorithm.

1. Introduction

In distributed systems, one of the variables that requires the most attention is time, as ideally, it should always be as accurate as possible. However, time accuracy does not always occur due to the nature of Distributed Systems, as their components are geographically dispersed (Tanenbaum and Steen, 2017). In addition, clock synchronization in distributed systems is the main concept when it comes to recording events, for example, events involving sending and receiving messages.

In this context, reaching a time agreement is a challenge in distributed systems because, in some situations, there is no global clock that all processes agree on. Therefore, to solve this problem, several clock synchronization algorithms have been proposed, such as (Gusella and Zatti, 1989), (Cristian, 1989), (Mills, 1989), (Lamport, 1978), etc.

Consequently, one of the most widely used clock synchronization algorithms in distributed systems is the Berkeley Algorithm (Tanenbaum and Steen, 2017). This algorithm differs from the others because the time server is active, that is, the server, which is the reference, periodically asks the clients what time they have. In this way, each client sends its current time to the server, which calculates an average between it and the clients and from this returns to each client what they should adjust in their clocks, advancing or delaying their local times.

Distributed Systems is a fundamental discipline in computer science courses at universities worldwide, and the subject of clock synchronization in distributed systems is one of the most important topics in this discipline. Therefore, given the

above, a problem in teaching clock synchronization in distributed systems is the lack of tools that enable learning the concepts of clock synchronization in a didactic and visual way, to facilitate the understanding of this concept that is, in most cases, seen by students as complicated and very abstract.

Thus, the objective of this work is to present a tool to aid in teaching clock synchronization in distributed systems called ***SincPoint***. ***SincPoint*** emerged due to the lack of didactic tools to teach clock synchronization, playfully, in distributed systems, and it consists of a Web page developed in Java Script that visually simulates the operation of the Berkeley Algorithm. Consequently, computer science teachers, when teaching clock synchronization in distributed systems, can use this proposal to facilitate students' learning on the topic “clock synchronization in distributed systems”.

Among the advantages of ***SincPoint***, the following stand out:

1. It was developed in ***JavaScript***, a very popular language used to develop web pages;
2. It has a user-friendly graphical interface, which provides meaningful and relevant experiences to users (UX);
3. It works on Linux, Windows, Android and Mac OS systems, and can be accessed from personal computers, notebooks, and smartphones;
4. It has installation and usage instructions, available on ***GitHub***[1];
5. It is completely free, ***Creative Commons 4.0 license***.

The proposed application, ***SincPoit***, was evaluated using Linux systems (Debian and Mint) and a cell phone with Android 11. The results showed a fast response in the manipulation of the simulator components (e.g., computers, master, and clients) and in the synchronization of these components on both tested platforms. Thus, students of distributed systems will be able to visualize, in practice, the concept of clock synchronization in distributed systems with the Berkeley Algorithm.

The remainder of this work was divided as follows: Section 2 presents the

[1] https://github.com.

literature review; Section 3 presents the related work; Section 4 presents the description of the proposal and methodology; Section 5 presents the results and discussion; and finally, Section 6 presents the conclusion and possibilities for future work.

2. Literature Review

2.1 Clock Synchronization Algorithms in Distributed Systems

This section presents some of the main works on clock synchronization, for example, Cristian's method, Lamport's algorithm, and the Berkeley Algorithm.

2.1.1 Cristian's Method

The method suggested by Flaviu Cristian in 1989 uses a time server using ***Coordinated Universal Time (UTC)*** data. As discussed by Cristian (1989), the client requests the time from the server, receiving the server's current time ***'T'*** as the response, so that the client can update its clock by adding ***'T'*** to the round-trip time divided by two ***(RTT/2)***.

For example, the formula used to calculate the client's clock is ***[T + RTT/2]***, where ***T*** is the clock time returned by the server and ***RTT***[2] is composed of the time in which the client receives the response from the server ***(t1)*** minus the time in which the client sends the request to the server ***(t0)***.

The precision is given by ***RTT ±(− Tmin)***, where ***Tmin*** is the minimum round-trip time from the source to the destination, that is, from the client to the server. If ***Tmin*** is unknown, it is assumed to be zero. The closer the ***RTT*** gets to ***Tmin***, the higher the accuracy.

2.1.2 Lamport's Algorithm

[2] ***RTT=(t1-t0).***

Lamport (1978) showed that there are cases in which synchronization does not need to be absolute, such as when two processes do not interact. In this way, only the order in which the events occur matters (Tanenbaum and Steen, 2007).

Thus, for the synchronization of logical clocks, Lamport defined the happens-before relationship, expressed by **a** → **b** or '**a happens before b**'. Following this logic, the algorithm assigns a number corresponding to its logical order to each event, so the following events have higher numbers than the previous ones.

For example:

1. **a** and **b** are events of the same process, and **a** occurs before **b**, so **a** → **b** is true;

2. **a** is the event of a message being sent by ***p1***, and **b** is the event of the same message being received by **i**, so **a** → **b** is true;

3. Furthermore, the relationship between the events **a** → **b** is transitive. Therefore, **a** → **b** and **b** → **c**, so **a** → **c**.

Consequently, event a has a clock ***C(a)*** that everyone agrees on, e.g., ***a*** → ***b***, so ***C(a) < C(b)***. In this context, ***C*** always occurs forward, and the time is always positively corrected. Consequently, the Lamport algorithm is as follows:

• ***(Step 1 (Local process - pi))*** Before some event (e.g., sending a message to the network and delivering it to the application) ***pi*** executes ***Ci*** ← ***Ci + 1***;

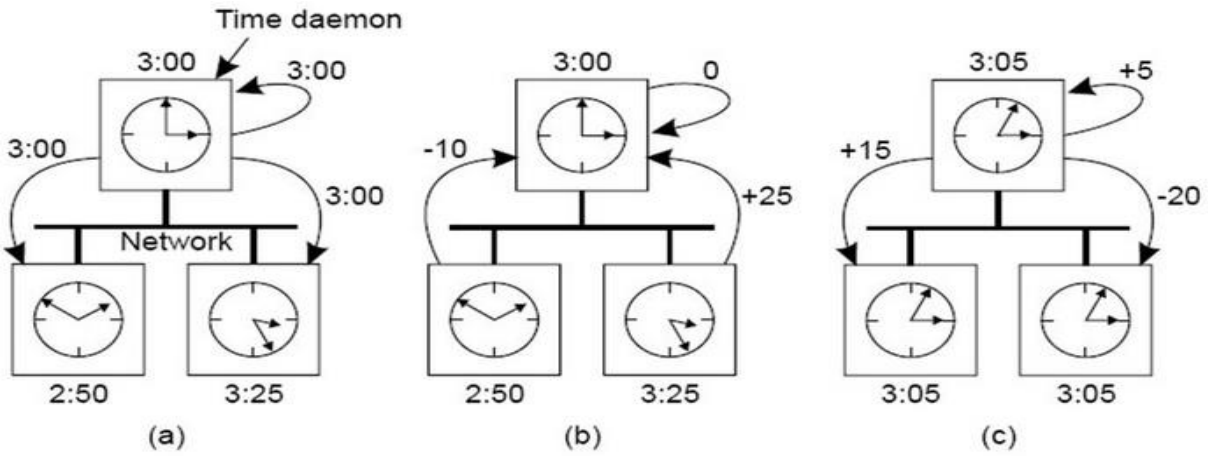
• **(Step 2 (Local process - pi)**) If ***pi*** sends a message m to ***pj***, then it sets the message timestamp ***m ts(m)*** to equal ***Ci***, after having executed ***Step 1*** (Local process, ***pi***);

• ***(Step 3 (Remote process - pj))*** Upon receiving ***m***, ***pj*** sets the local counter to ***Cj*** ← ***max{Cj, ts(m)}***, after which ***Step 1*** is executed, and the message is delivered to the application.

2.1.3 Berkeley Algorithm

Gusella and Zatti (1989) proposed an algorithm in which one computer is chosen as the master (server), and the others are called slaves (clients). In this context, the master talks to all the other computers, asking what time each one is keeping. After getting the answers, it calculates an average time of the hours, including its own, and responds individually with the time that each machine needs to advance or delay its clock, Figure 1.

Figure 1: Example of the execution of the Berkeley Algorithm.



As illustrated in Figure 3, panel (a) shows the time the server requesting the time from the clients, panel (b) shows the clients returning their respective times to the server, and panel (c) depicts the server calculating the average time and instructing each client on whether to adjust their clock by advancing or delaying it (Tanenbaum and Steen, 2007).

3. Related Work

This section presents simulators that are proposed and that implement the Berkeley Algorithm. These works were used as comparison parameters with our proposal, SincPoint.

In (GEEKSFORGEEKS, 2023) an implementation of the Berkeley Algorithm is presented, developed in Python3, and divided into two parts: one that imitates the client nodes and another that imitates the server node. The proposal in question presents a demonstration with four nodes, a master computer, and three slaves. The computers synchronize their times using sockets on port 8080 and a loopback interface (e.g., 127.0.0.1). In this context, the simulator demonstrates the correct functioning of the Berkeley Algorithm, however, it does not implement a graphical

interface that, in this case, could facilitate learning and understanding of how the algorithm works.

The proposal presented by (Moraes and Arakawa, 2020) describes a simulator of the Berkeley Algorithm that uses Docker containers. According to the authors, this is a simulator for the class of distributed systems that uses five containers as machines, and a daemon (e.g., server) that applies the algorithm to the other machines. However, on the proposal website (e.g., GitHub) there is no description of how to test the application. Furthermore, there is no information on whether it runs on the command line or has a graphical interface.

In (Kumar, 2023), a simulator of the Berkeley Algorithm developed in Python is presented. The code is organized into two primary functions: **def master_main()**, which simulates the behavior of the server, and **def client_main()**, which simulates the behavior of the client. This implementation serves as a basic example, but it lacks both a graphical interface and exception handling.

In (BE-DISTRIBUTED-SYSTEMS, 2023), a Java-based simulator of the Berkeley Algorithm is presented, consisting of six classes: **TimeServer.java**, **ClockOne.java**, **ClockTwo.java**, **ClockThree.java**, and **MainClock.java** for the client-side code. The simulator uses Java **RMI** for distributed communication between the clients and the server. However, the GitHub repository for this project does not include test examples. Besides that, it also lacks a graphical interface. This absence of a visual interface can make it more difficult for beginners to understand and learn the concepts of distributed systems.

In (Tan, 2021) a simulator of the Berkeley Algorithm is proposed, developed in C++. The proposal in question is complete, as it has real network communication made in a TCP socket. In addition, it has a video tutorial showing the execution of the simulator, on a Linux system, with four nodes, three clients, and one server. In this context, the author explains in detail the Berkeley Algorithm and how to use the simulator. However, despite being a great reference for students looking for an implementation of the Berkeley Algorithm, the simulator does not have a graphical interface.

4. Proposal Description

SincPoint is a web application developed in JavaScript to simulate the clock synchronization process in distributed systems using the Berkeley Algorithm. It is available for download on the ***GitHub***[3] development platform. Thus, the project structure is organized into four main parts: **Components**, **Contexts**, **Assets**, and the **App.jsx** file. The **Components** directory contains reusable components used throughout the application, while **Contexts** handles the global states and contexts. The **Assets** folder stores images, icons, and other static resources. The **App.jsx** file serves as the central hub of the application, containing key functions such as ***applyBerkeleyAlgorithm***, ***handleSync***, and others. Consequently, the Berkeley Algorithm is implemented in the ***applyBerkeleyAlgorithm*** function, as described in Table 1. This function's primary role is to calculate and record the time adjustments (either increment or decrement) for each clock in the list.

Table 1: Description of the actions of the applyBerkeleyAlgorithm(clockList) function.

Steps	Action
1	Convert the times of the clocks in the clockList list to manipulatable objects.
2	Calculate the average time for each clock: For each clock in the list: a) Get the current time in milliseconds; b) Add up all the times; c) Divide the total sum by the number of clocks.
3	For each clock in the list: a) Get the current time in milliseconds; b) Calculate the difference between the average time and the current time; c) Record the necessary adjustment in milliseconds.

Source: Authors.

The ***handleSync*** function described in Table 2 was used to call ***applyBerkeleyAlgorithm***, so that the clocks in the list can be updated. The start and end times of execution are also recorded to provide the time it took for the simulated computers to synchronize with the server. Since ***SincPoint*** is a simulator, the round

[3] https://github.com.

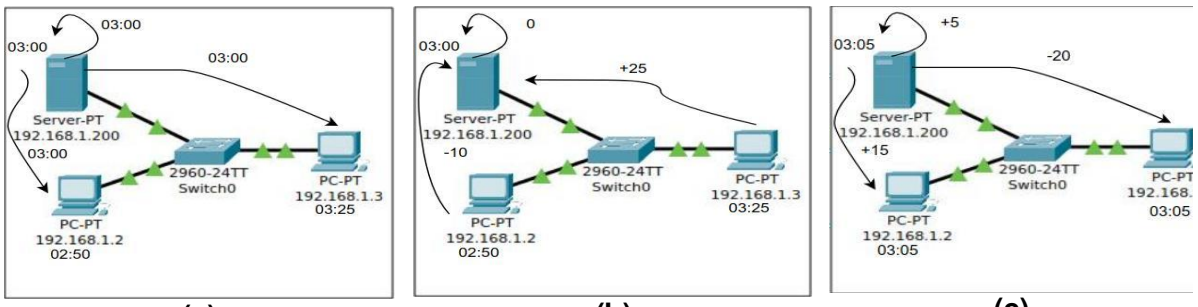
trip time from the client to the server was not considered in the ***applyBerkeleyAlgorithm*** function, nor was the total execution time of the program to be presented to the user at the end. For example, considering 3 computers, as shown in Figure 2, one server and two clients with clocks that are out of sync. Thus, the server/master with Internet Protocol (IP) address 192.168.1.200 asks the clients/slaves for their date/time stamps. Based on this data, the server calculates the average time and sends each of the clients/slaves, including itself, the time they need to adjust their clocks.

Table 2: Description of the handleSync() function actions.

Steps	Action
1	Record the start time of execution.
2	Call the applyBerkeleyAlgorithm function: a) Return the list of adjusted clocks; b) Update the adjusted clocks in the list.
3	Record the execution completion time: a) Calculate the total execution time.

Source: Authors.

Figure 2: Synchronization model with two clients and one server.



(a) **(b)** **(c)**

Figure 2 shows (a) the server requesting time, (b) the clients responding to the server, and (c) the server calculating the adjustment (average) and returning this adjustment to the clients. Source: Authors.

4.1. Materials and Methods

This work is based on a comprehensive bibliographic review, drawing on key references related to clock synchronization, including works by Colouris (2013), Tanenbaum and Steen (2007), Lamport (1978), Gusella and Zatti (1989), Flaviu Cristian (1989), and Rodrigues (2017). The following technologies were utilized in the development of the application:

- **Software:**
    - **Programming Language:** JavaScript (v18.19.0)
    - **Libraries:** React (v18.3.1) for building the user interface, Material-UI (v6.1.6) for styling components and ensuring responsive layout, Day.js for date and time manipulation, and Vite (v5.4.10) to streamline the development process;
    - **Web Server:** Nginx (v1.24.0) to serve the application;
    - **Operating Systems:** Ubuntu 24.04.1 LTS (64-bit) for hosting the application, Debian GNU/Linux 12 (64-bit) for development, Linux Mint 21.2, and Android 11 for testing.
- **Development Hardware:**
    - Intel Core i5-7200U CPU (2.50GHz), 8GB DDR4 RAM, GeForce 940mx GPU, 500GB SSD.
- **Web Server Hardware:**
    - AMD EPYC 7543P processor with 32GB RAM.
- **Testing Hardware:**
    - **Notebook:** Intel Pentium Gold 7505 (2.00GHz), 12GB DDR4 RAM, 120GB SSD.
    - **Smartphone:** MediaTek P22 octa-core (up to 2 GHz), 2GB RAM, 32GB storage.
- **Application Testing:** The application was tested on two devices - a notebook and a smartphone - within a class C network (192.168.1.0/24). The testing process involved adding clients, performing clock synchronization using the Berkeley Algorithm, and analyzing the results.

5. Results and Discussion

The proposal in question consists of an interactive application, called ***SincPoint***, capable of running online (e.g., https://sincpoint.nexsyn.com.br) and whose objective is to simulate the operation of the Berkeley Algorithm to aid in teaching clock synchronization in distributed systems in a practical and visual way.

Given the above, the main functionalities of SincPoint are:

1. Adding devices with customized IP addresses and times;
2. Monitoring the current time of each device directly in the interface;
3. Synchronizing the times of all computers with a single click;
4. Responsive and interactive interface compatible with mobile devices and Desktops.

Therefore, when accessing the application link https://sincpoint.nexsyn.com.br, the user will see a Server, Figure 3(a), in the center of the screen, with the IP address 192.168.1.200 and a display of the current time below. On the right side of the page, there is a bar that, when pressed, displays a small form, Figure 2(b), for entering client machines, Figure 2(c), with their respective IP addresses, time, and date.

Figure 3: SincPoint web interface with two clients and one server.



(a) **(b)** **(c)**

Source: Authors.

Throughout the development process, several tests were performed to define the appropriate color palette and define the icons that best fit the purpose of the simulator. In addition, a multitude of bug fixes and updates were performed to improve the interface and operation of ***SincPoint***.

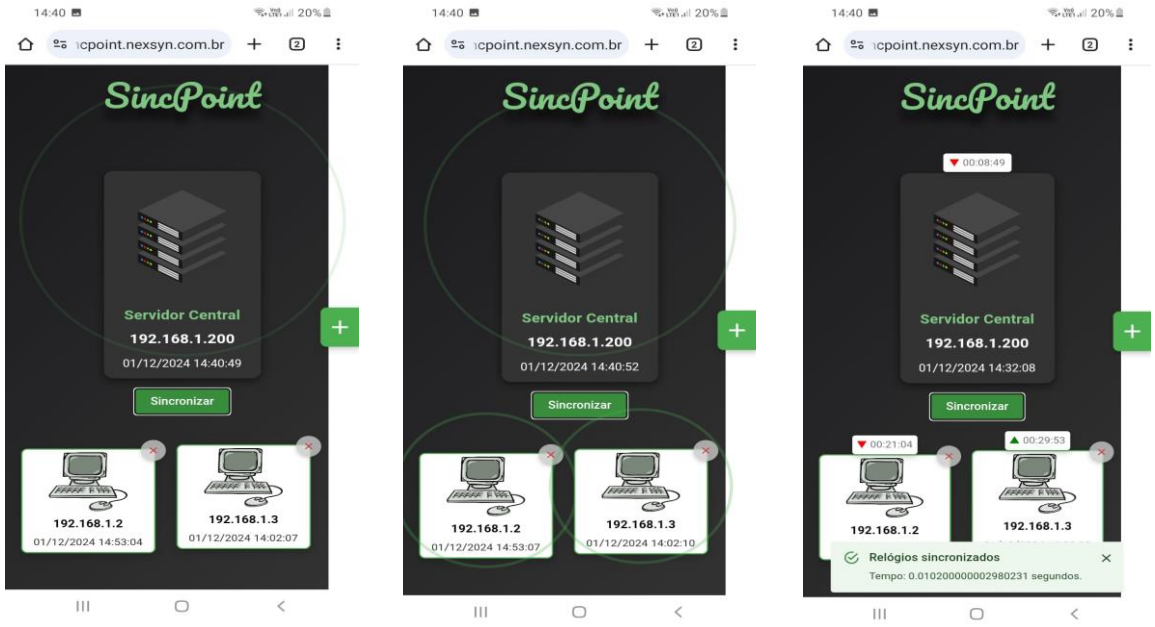
To represent the communication that occurs between the clients and the server, according to the model described previously in Figure 2, instead of arrows, a

wave-shaped animation was used that starts with a light green color and fades away until the messages between the server and the clients are delivered.

For example, considering a scenario with two computers and a server, the first wave is emitted by the server, then the client computers also emit a wave, and finally, the server ends the communication with the final wave.

Figure 4 shows the test performed with two client computers on an Android Smartphone with a MediaTek P22 processor. Thus, we configured the server time with the following timestamp 14:40 and IP 192.168.1.200. The clients were configured with the following timestamps 14:53:04/14:02:07 and IP addresses 192.168.1.2/192.168.1.3 respectively.

Figure 4: SincPoint test performed with a Smartphone.



(a) (b) (c)

Source: Authors.

Figure 4(a) illustrates the initial state, and Figure 4(b) shows the communication between clients and server through circular waves. Figure 4(c) shows all computers with their times updated (e.g., “Synchronized Clocks” message). The value that was advanced or delayed on the machines’ clock is displayed at the top of the server and computer icons. In addition, the application displays a message with the time needed to perform the synchronization. In the case of the test with the Smartphone, the time for synchronization was 10 milliseconds.

However, the display time of the wave, which simulates the transmission of the message, is not counted. Figure 5 illustrates the test performed on the notebook with Linux Mint and Intel Pentium Gold 7505 processor. Consequently, Figure 5(a) shows the initial state of the synchronization, Figure 5(b) shows all computers with their updated times, showing the value that was advanced or delayed on each client's clock, and, finally, Figure 5(c) shows the time that was necessary to perform the synchronization. In this context, the synchronization time using the notebook was 1.3 milliseconds.

Figure 5: SincPoint test performed with a Notebook.



(a) **(b)** **(c)**

Source: Authors.

Therefore, after several tests on the application, some adjustments were made to improve the user experience, such as, for example, increasing the duration of the messages that are displayed as a result at the end of the execution and some color adjustments to make the process more pleasing to the users' eyes.

6. Conclusion and Future Work

This paper presented ***SincPoint***, a web application based on the Berkeley Algorithm (Gusella & Zatti, 1989), designed to enhance the learning experience of clock synchronization in distributed systems. The tool aims to assist computer science educators in visually demonstrating the operation of one of the most widely used clock synchronization algorithms, making it easier for students to understand this complex concept.

The main contributions of this work are as follows:

1. **Uniqueness:** ***SincPoint*** is a novel tool, as no other educational tools for teaching clock synchronization in distributed systems offer the same functionality.
2. **Technology Stack:** The application was developed in JavaScript, a widely used language for creating web applications, ensuring broad accessibility and compatibility.
3. **Interactive Interface:** ***SincPoint*** features an intuitive and engaging graphical interface that enhances the user experience (UX), making the learning process more interactive and informative.
4. **Multiplatform Compatibility:** The tool is cross-platform, running on Linux, Windows, Android, and macOS, and is accessible from desktops, laptops, and smartphones.
5. **Documentation:** ***SincPoint*** comes with a comprehensive installation and user guide, available on ***GitHub***, making it easy for users to get started.
6. **Open-Source:** The application is completely free to use, licensed under Creative Commons 4.0, allowing users to modify and share it.

Following extensive testing, several adjustments were made to improve the user experience, including extending the duration of messages displayed at the end of the execution and adjusting color schemes for better visual appeal and readability.

Therefore, based on the experiments performed, students can learn in detail how the Berkeley algorithm works visually and interactively, which facilitates the learning of clock synchronization in distributed systems. Consequently, the most significant contribution of this work is that through ***SincPoint*** it is possible, as a didactic in the distributed systems discipline, to develop practical and laboratory activities.

In future work, we will expand ***SincPoint*** to allow the user to also test the algorithms of Cristian (1989) and Lamport (1978). In addition, in new updates, we intend to correct some gaps in the proposal, for example: 1) the application as it is does not allow changing the server time, which can frustrate users who want to set up a specific scenario for calculating the time; and 2) create a version of ***SincPoint*** that synchronizes the clocks of physical machines instead of just simulating.

7. References

BE-DISTRIBUTED-SYSTEMS. **Berkeley Algorithm.** GitHub Plataform, 8 Dec, 2023. Available at: https://github.com/BE-Distributed-Systems/BerkeleyAlgorithm. Accessed on: Oct 18, 2024.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. **Sistemas Distribuídos: Conceitos e Projetos.** 5ª Edição, Porto Alegre, Editora Bookman, 2013.

CRISTIAN, F. **Probabilistic Clock Synchronization.** Distributed Computing, Spring Verlang, Volume 3, p. 146–158, 1989, DOI: https://10.1007/BF01784024.

GEEKSFORGEEKS. **Berkeley's Algorithm.** Tutorial Site Geeks for Geeks, 15 Mar, 2023. Available at: https://www.geeksforgeeks.org/berkeleys-algorithm/. Accessed on: Nov 28 , 2024.

GUSELLA, R.; ZATTI, S. **The Accuracy of the Clock Synchronization Achieved by Tempo in Berkeley Unix 4.3bsd.** IEEE Transactions on Software Engineering, Volume 15, Issue 7, p. 847–853, 1989, DOI: https://10.1109/32.29484.

KUMAR, S. **Berkley Algorithm Implementation.** Site Tutorialspoint, 8 Feb, 2023. Available at: https://www.tutorialspoint.com/berkeley-s-algorithm. Accessed on: Jun 11, 2024.

LAMPORT, L. **Time, Clocks, and the Ordering of Events in a Distributed System.** Communication of ACM, Volume 21, Issue 7, p. 558–565, 1978, DOI: https://10.1145/359545.359563.

MORAES, M.; ARAKAWA, K. **Berkley Algorithm Simulator.** GitHub Plataform, 9 Dec, 2020. Available at: https://github.com/MicaelBarreto/Node-Berkeley-Algorithm. Accessed on: Dec 1, 2024.

RODRIGUES, J. S. R.; LIMA, R. A.; JOSÉ, D. A. M. **Algoritmo para Sincronização de Relógios Físicos em Sistemas Distribuídos.** Anais da 15ª ERRC, p. 42-49, Setembro 2017.

TAN, D. **Berkley Algorithm Simulator.** GitHub Plataform, 7 Dec, 2021. Available at: https://github.com/DayuanTan/berkeley-algorithm-implementation. Accessed on: Oct 20, 2024.

TANENBAUM, A. S.; STEEN, M. **Distributed Systems.** Create Space Independent Publishing Platform, 3rd Edition, 2017.

TANENBAUM, A. S.; STEEN, M. V. **Sistemas Distribuídos: Princípios e Paradigmas.** São Paulo: Pearson Prentice Hall, 2º ed., 2007.