



MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

(A constituent unit of MAHE, Manipal)

DISTRIBUTED SYSTEMS (CSE -3261) MINI PROJECT REPORT ON

**Distributed Implementation of
Clock Synchronization Algorithms**

SUBMITTED TO

Department of Computer Science & Engineering

by

Name	Registration Number	Roll Number	Semester	Section
Shyam Sundar Bharathi S	200905302	53	VI	B
Parv Jain	200905316	56	VI	B
Ajay Rajendra Kumar	200905390	61	VI	B
Ryan Sojan	200905396	64	VI	B

Name & Signature of Evaluator 1

Name & Signature of Evaluator 2

(Jan 2023 – May 2023)

Table of Contents			
			Page No
Chapter 1		INTRODUCTION	
	1.1	Introduction	3
	1.2	Motivation	4
Chapter 2		BACKGROUND THEORY	5
Chapter 3		METHODOLOGY	6
Chapter 4		RESULTS AND DISCUSSION	
	4.1	Results	8
	4.2	Discussions	10
Chapter 5		CONCLUSIONS AND FUTURE ENHANCEMENTS	
	5.1	Conclusions	11
	5.2	Future Enhancements	11
Chapter 6		REFERENCES	12

CHAPTER 1: INTRODUCTION

1.1 Introduction

Clock synchronization algorithms are used to synchronize the clocks of multiple devices or systems to a common time reference. In distributed systems, different devices may have slightly different clock speeds or may experience varying network latencies, which can cause inconsistencies in timestamping events and data. This can lead to incorrect sequencing of events, which can cause errors in the system. Therefore, clock synchronization is essential to ensure the correctness of distributed systems.

The Berkeley algorithm is a centralized algorithm that relies on a central time server to distribute time to other machines in the network. Each machine periodically sends its own clock time to the time server, and the time server calculates the average time among all machines and sends it back to each machine to adjust its clock accordingly. This algorithm is relatively simple and easy to implement, but it relies on a centralized time server and can be vulnerable to network outages or server failures.

The Lamport algorithm is a logical clock synchronization algorithm developed by computer scientist Leslie Lamport. In this algorithm, each device has its own logical clock that is incremented for every event that occurs on that device. When two devices communicate, the device with the lower clock value adjusts its clock to the higher value. This algorithm does not require a centralized time server and can handle network delays, but it assumes that events are completely ordered and may not work well in cases where there are causal relationships between events.

The Christian algorithm is a clock synchronization algorithm based on the Network Time Protocol (NTP), which uses a hierarchical system of time servers to synchronize clocks. The algorithm works by measuring the round-trip time between a client and a server and adjusting the client's clock based on the time difference. This algorithm is highly accurate and can handle network delays and disruptions, but it requires a reliable network connection and may not be suitable for systems with strict security requirements.

1.2 Motivation

The motivation behind developing clock synchronization algorithms is to ensure that different devices or systems have a common time reference for accurate timestamping of events and coordination of tasks. In many real-world applications, such as distributed systems, financial transactions, and industrial automation, accurate time synchronization is critical for maintaining consistency and integrity of data, avoiding conflicts and errors, and ensuring the proper functioning of the system. Clock synchronization algorithms aim to reduce clock drift and offset caused by various factors, such as hardware imprecision, network delays, and clock skew, and to maintain a common time base for all devices in the system. By achieving accurate clock synchronization, these algorithms enable efficient and reliable communication and coordination among different components, leading to improved performance, productivity, and user experience.

Chapter 2: BACKGROUND THEORY

Clock synchronization algorithms aim to ensure that different devices or systems have a common time reference, despite variations and drifts in their local clocks. The basic principle of clock synchronization is to exchange clock information between devices and adjust their clocks, accordingly, using various techniques and protocols.

One of the earliest clock synchronization algorithms is the Network Time Protocol (NTP), developed in the 1980s, which uses a hierarchical system of time servers to synchronize clocks over the internet. NTP employs a combination of polling and filtering techniques to estimate the offset and delay between clocks and adjust them to the most accurate source.

Another well-known algorithm is the Berkeley algorithm, proposed by Mills in 1989, which uses a central time server to distribute time to other machines in the network. Each machine periodically sends its own clock time to the time server, and the time server calculates the average time among all machines and sends it back to each machine to adjust its clock accordingly.

The Lamport algorithm, developed by Leslie Lamport in 1978, is a logical clock synchronization algorithm that assigns a unique logical timestamp to each event in the system, based on the causality relationship between events. This algorithm does not rely on a central server and can handle network delays and disruptions, but it assumes that events are completely ordered and may not work well in cases where there are causal relationships between events.

Other clock synchronization algorithms include the Precision Time Protocol (PTP), which is designed for high-precision synchronization in industrial and measurement applications, and the Global Positioning System (GPS) synchronization, which uses satellite signals to synchronize clocks in a geographic area.

Chapter 3: METHODOLOGY

Berkeley Algorithm: -

The Berkeley algorithm is a centralized clock synchronization algorithm that uses a central time server to distribute time to other machines in the network. Each machine periodically sends its own clock time to the time server, and the time server calculates the average time among all machines and sends it back to each machine to adjust its clock accordingly. This algorithm requires a reliable network connection to the time server and assumes that the time server is highly accurate. The method involves sending messages from each machine to the time server, calculating the average time, and distributing the adjusted time back to the machines. The method is relatively simple and easy to implement but is vulnerable to network outages or server failures.

Christian Algorithm: -

The Christian clock synchronization algorithm is a method for synchronizing the clocks of computers on a network. To implement this algorithm, the first step is to select a server that will act as the time reference for the network. Next, the Christian clock synchronization software should be installed on each computer that needs to be synchronized, and then configured to use the chosen server as the time reference. The software will periodically synchronize the clock on each computer with the server, ensuring that all clocks are kept accurate. It is important to monitor the synchronization process to ensure that all computers on the network are synchronized and that there are no errors or discrepancies. By following this methodology, the Christian clock synchronization algorithm can help ensure that all computers on a network are synchronized, allowing for accurate timekeeping and improved network performance.

Lamport Algorithm: -

Lamport clock synchronization algorithm is a technique used to order events in a distributed system. The algorithm was proposed by Leslie Lamport in 1978. In this algorithm, each process maintains a logical clock, which is updated whenever an event occurs. An event can be a process receiving a message, sending a message, or performing a local computation.

To synchronize the clocks, whenever a process sends a message, it includes its current logical clock value in the message. When a process receives a message, it updates its logical clock to be the maximum of its current value and the value in the received message plus one. This ensures that the logical clocks of different processes always move forward, and events are ordered correctly.

By using this algorithm, we can order events in a distributed system even if the events occur concurrently at different processes. The algorithm is simple and efficient, but it does not guarantee that the clocks will be perfectly synchronized, as clock drift can still occur.

Chapter 4: RESULTS & DISCUSSIONS

4.1 Results

Lamport Algorithm

```

).
Updated Lamport clock to 1682970627891.
Time difference: -0.586181640625 ms

Server connected to: ('127.0.0.1', 52342)

Received message "sync:1682970632270" from ('127.0.0.1', 52342)
).
Updated Lamport clock to 1682970632271.
Time difference: 0.269775390625 ms

Server connected to: ('127.0.0.1', 52343)

Received message "sync:1682970632896" from ('127.0.0.1', 52343)
).
Updated Lamport clock to 1682970632897.
Time difference: 0.242919921875 ms

Server connected to: ('127.0.0.1', 52344)

Received message "sync:1682970637275" from ('127.0.0.1', 52344)
).
Updated Lamport clock to 1682970637276.
Time difference: 0.052978515625 ms

Server connected to: ('127.0.0.1', 52345)

Received message "sync:1682970637899" from ('127.0.0.1', 52345)
).
Updated Lamport clock to 1682970637900.
Time difference: -0.343017578125 ms

Sent message "sync" to server.
Updated client clock to 1682970592852.
Received acknowledgement: ack

Sent message "sync" to server.
Updated client clock to 1682970597858.
Received acknowledgement: ack

Sent message "sync" to server.
Updated client clock to 1682970602864.
Received acknowledgement: ack

Sent message "sync" to server.
Updated client clock to 1682970627890.
Received acknowledgement: ack

Sent message "sync" to server.
Updated client clock to 1682970632896.
Received acknowledgement: ack

Sent message "sync" to server.
Updated client clock to 1682970637899.
Received acknowledgement: ack

```


Berkeley Algorithm

```
Last login: Tue May  2 01:21:59 on ttys009
shyamsundarbharathi@Shyams-MacBook ~ % /Users/shyamsundarbharathi/Sem-6/DSL/MiniProject/berkeley_server.py ; exit;
Server started
Connected by ('127.0.0.1', 52371)
Connected by ('127.0.0.1', 52372)
```

```
Last login: Tue May  2 01:23:29 on ttys006
shyamsundarbharathi@Shyams-MacBook ~ % /Users/shyamsundarbharathi/Sem-6/DSL/MiniProject/berkeley_client.py ; exit;
Connected to server
Time difference: 1.0034749507904053 seconds
Connected to server
Time difference: 1.0031449794769287 seconds
Connected to server
Time difference: 1.0048589706420898 seconds
Connected to server
Time difference: 1.0050277709960938 seconds
Connected to server
Time difference: 1.0016169548034668 seconds
Connected to server
Time difference: 1.0012080669403076 seconds
Connected to server
```

```
Time difference: 1.004822015762329 seconds
Connected to server
Time difference: 1.0015628337860107 seconds
Connected to server
Time difference: 1.0046279430389404 seconds
Connected to server
Time difference: 1.0008668899536133 seconds
Connected to server
Time difference: 1.0017011165618896 seconds
Connected to server
Time difference: 1.0048420429229736 seconds
Connected to server
Time difference: 1.0049550533294678 seconds
Connected to server
Time difference: 1.0024609565734863 seconds
Connected to server
```

Christian Algorithm

```
Last login: Tue May 2 01:19:17 on ttys012
shyamsundarbharathi@Shyams-MacBook ~ % /Users/shyamsundarbharathi/Sem-6/DSL/MiniProject/cristian_server.py ; exit;
Socket successfully created
Server is listening.
Server connected to ('127.0.0.1', 52354)
Server connected to ('127.0.0.1', 52355)
Server connected to ('127.0.0.1', 52356)
Server connected to ('127.0.0.1', 52357)
Server connected to ('127.0.0.1', 52358)
Server connected to ('127.0.0.1', 52359)
Server connected to ('127.0.0.1', 52360)
Server connected to ('127.0.0.1', 52361)
Server connected to ('127.0.0.1', 52362)
Server connected to ('127.0.0.1', 52363)
Server connected to ('127.0.0.1', 52364)
Server connected to ('127.0.0.1', 52365)
█

Last login: Tue May 2 01:21:58 on ttys006
shyamsundarbharathi@Shyams-MacBook ~ % /Users/shyamsundarbharathi/Sem-6/DSL/MiniProject/cristian_client.py ; exit;
Time returned by server: 2023-05-02 01:21:59.050170
Process Delay latency: 0.002702415999999999 seconds
Actual clock time at client side: 2023-05-02 01:21:59.051731
Synchronized process client time: 2023-05-02 01:21:59.051521
Synchronization error : 0.00021 seconds

Time returned by server: 2023-05-02 01:22:04.059426
Process Delay latency: 0.0033688749999999605 seconds
Actual clock time at client side: 2023-05-02 01:22:04.061623
Synchronized process client time: 2023-05-02 01:22:04.061110
Synchronization error : 0.000513 seconds

Synchronization error : 0.000288 seconds Time returned by server: 2023-05-02 01:22:19.078918
Process Delay latency: 0.00212041700000004297 seconds
Actual clock time at client side: 2023-05-02 01:22:19.079899
Synchronized process client time: 2023-05-02 01:22:19.079978
Synchronization error : -7.9e-05 seconds

Time returned by server: 2023-05-02 01:22:24.086270
Process Delay latency: 0.0015136249999976314 seconds
Actual clock time at client side: 2023-05-02 01:22:24.087387
Synchronized process client time: 2023-05-02 01:22:24.087027
Synchronization error : 0.00036 seconds
```

4.2 Discussions

Discussions on clock synchronization algorithms often focus on the trade-offs between accuracy, reliability, and complexity, as well as the specific requirements and constraints of the system in question. Some common topics of discussion include the choice between centralized and distributed algorithms, the impact of network delays and disruptions, the use of logical clocks versus physical clocks, and the integration of security mechanisms.

Another important topic of discussion is the evaluation and comparison of different clock synchronization algorithms, which may involve benchmarking, simulation, or empirical testing. Metrics such as clock accuracy, convergence time, overhead, and scalability are often used to evaluate the performance of different algorithms under various conditions.

Finally, discussions on clock synchronization algorithms may also involve the development of new algorithms or improvements to existing ones.

Chapter 5: CONCLUSIONS & FUTURE ENHANCEMENTS

5.1 Conclusions

In conclusion, clock synchronization algorithms play a crucial role in distributed systems, where multiple processes communicate and execute tasks concurrently. These algorithms enable us to order events correctly, ensuring consistency and coherence of the system. The Lamport clock synchronization algorithm, proposed by Leslie Lamport in 1978, is a fundamental technique used in many modern systems. This algorithm uses logical clocks to synchronize the clocks of different processes, ensuring that events are ordered correctly. While this algorithm is efficient and simple to implement, it does not guarantee perfect synchronization, as clock drift can still occur. Overall, clock synchronization algorithms are essential for the efficient and reliable functioning of distributed systems.

5.2 Future Enhancements

Clock synchronization algorithms are crucial for ensuring consistency and coherence in distributed systems where multiple processes communicate and execute tasks concurrently. The Lamport clock synchronization algorithm, proposed by Leslie Lamport in 1978, uses logical clocks to synchronize the clocks of different processes, ensuring that events are ordered correctly. While the Lamport algorithm is simple and efficient, there are several future enhancements that can improve its accuracy, reliability, and security. These enhancements include hybrid clock synchronization algorithms, adaptive clock synchronization, fault-tolerant clock synchronization, blockchain-based clock synchronization, and quantum clock synchronization. These enhancements can make clock synchronization algorithms more suitable for modern distributed systems, providing better accuracy, reliability, and security.

Chapter 6: References

- <https://www.geeksforgeeks.org/synchronization-in-distributed-systems/>
- <https://www.tutorialspoint.com/index.html>
- https://en.wikipedia.org/wiki/Clock_synchronization
- <https://www.thecode11.com/2022/06/clock-synchronization-in-distributed-system.html>