An Analysis of Berkeley's Algorithm and the Precision Time Protocol (PTP) on Simulated Distributed Systems Using SimGrid

Jonny Olswang
Joshua Reyes
Professor: Hailu Xu
CECS 327-02
California State University - Long Beach
Jonny.Olswang01@student.csulb.edu
Josh.Reyes@student.csulb.edu
May 11, 2024

***Abstract*—This paper will analyze the effectiveness of two distributed system clock synchronization algorithms based on factors such as network-wide clock accuracy, hardware requirements, network topology, and clock utilization. The selected algorithms were Berkeley's Algorithm and the Precision Time Protocol. SimGrid is a framework for simulating the development of distributed application executions on distributed platforms, which we used with Python for experimentation. Data will be provided to offer a pragmatic view of the different algorithms' uses.**

Keywords—Simulated Distributed Systems, Clock Synchronization, SimGrid, Berkeley's Algorithm, Precision Time Protocol (PTP)

I. INTRODUCTION

Clock accuracy is a vital feature of a functional distributed system. Given the tendency for individual clocks to drift over time, network clock synchronization is a necessity for operations like data consistency, task scheduling, resource management, and event logging.

Many algorithms exist for synchronizing clocks across a distributed system that differ depending on the required accuracy and how the system defines synchronization. For example, should network clocks be accurate to the order of milliseconds? Microseconds? Nanoseconds? Is the absolute time of the clocks important, or does the order of distributed events take precedence? How should outlier clocks be handled during the synchronization process?

For our testing, we chose to implement Berkeley's Algorithm and the Precision Time Protocol in Python, and ran them through the SimGrid simulation framework.

II. BACKGROUND

A. *SimGrid*

SimGrid provides comprehensive documentation in regards to its components and functionality. The framework advertises the ability to compare distributed designs for developers, debug real applications via SimGrid's reproducibility, and analyze distributed algorithms for research purposes [1]. This paper will be focused on utilizing the third of these three options.

Projects run with SimGrid have four major components [1]:

- ***Applications*** - one or more processes that implement distributed algorithms, and use SimGrid's API.
- ***Simulated Platform*** - a file description of the distributed system's hardware.
- ***Deployment Description*** - a file description of how the execution of the application is deployed on the platform.
- ***Platform Models*** - documented SimGrid models that can be configured to fit use cases.

Additionally, SimGrid provides an example project repository to pull from, with examples of these components available for reference or as a starting point for one's own project. We elected to code on Ubuntu distributions, which made

downloading Python's SimGrid dependencies relatively simple.

B. *Berkeley's Algorithm*

The clock identified as most reliable is given the role of master clock. The rest of the clocks are classified as slave clocks and aligned with the boundary clocks that are synchronized to the previously selected master clock. This is for network time synchronization for better coordinated operations.

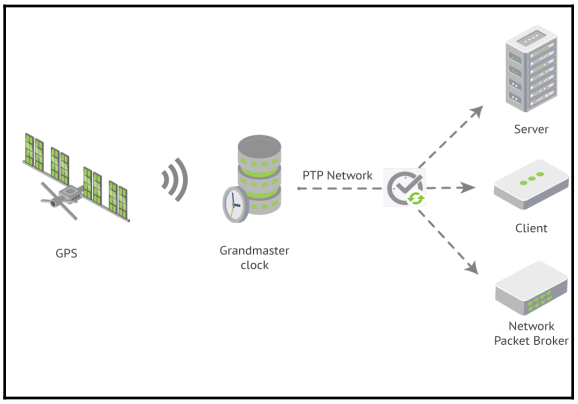

Fig. 1. Architecture of a PTP network. https://www.hyve.com/wp-content/uploads/sites/5/2023/04/hyve-PTP-infographic.png?w=1920&q=75

C. *Precision Time Protocol (PTP)*

PTP necessitates the network must be organized in a hierarchical structure. One clock, known as the grandmaster, acts as the undisputed source of correct time on the network. This status is maintained by its connection to a highly accurate Global Positioning System (GPS) or atomic clock. The grandmaster is connected to a set of boundary (aka master) clocks, which in turn are connected to a set of client (aka slave) clocks [2]. The boundary clocks have master-slave relationships with the client clocks, and the grandmaster has master-slave relationships to boundary clocks. To prevent a single point of failure, multiple grandmaster clocks can be prepared to step in should the operating time source fail. PTP has shown to consistently provide sub-microsecond accuracy [2], and is the standard for industries that deal with time-sensitive or life-threatening situations. However, it requires specialized hardware support to operate optimally, such as atomic/GPS clocks or hardware timestamping.

III. METHODOLOGY

A. *SimGrid*

SimGrid proved to be an excellent tool for experimentation. It was able to produce large amounts of results without the deployment of large distributed systems or networks, and the real world run-time was a fraction of the simulated time. This allowed for the easy repetition of experiments with large data sets.

The framework did have one unfortunate drawback: individual simulated processes did not have their own clocks. Instead, all processes operated off of a shared clock that ran for the duration of the simulation [1].

B. Berkeley's Algorithm

The methodology behind Berkeley's algorithm was to try and create an efficient clock synchronization model using a time synchronization algorithm. How this works is by a couple steps, Each computer in the system will check its local time which will be used later. The master which is selected at the beginning as the most efficient will collect all the times from the rest of the workers.With that info provided it will then calculate the offset of each one with their given local times. Offset is calculated as:

$$Offset = t_{remote} - t_{local}$$

Cristian's algorithm is also used to handle round trip time from the communication between the clocks while sending their time communication:

$$Offset = ((t_{serv} - t_{client}) + (t_{client} - t_{client}))/2$$

It will then use the offset that was calculated to adjust their respective clocks therefore synchronizing them back up with master.. This process is repeated throughout the runtime, ensuring they stay synchronized through it all. It's important to note no special hardware is required for either clock during the time retrieval and offset calculations which one advantage it holds over other models, which we will go over later. It is recommended and

needed for more precise synchronization if wanted, but again not required.

As mentioned above, SimGrid will run all its processes off of one simulated clock. In order to imitate clock desynchronization across the network, clients were preset with an offset variable, which when added to act as that individual process's clock. We also made sure to repeat our simulation 50 times to check that it produced the same results.

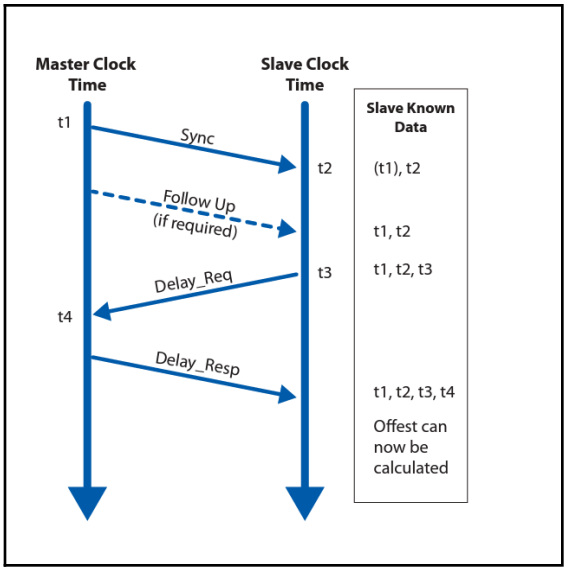

Fig. 2. PTP synchronization process. https://moniem-tech.com/wp-content/uploads/sites/3/2023/02/Master%E2%80%93Slave-Hierarchy-in-PTP.png

C. *Precision Time Protocol (PTP)*

Clocks in the master role of a relationship are responsible for initiating the syncing process. As shown in Fig. 2, four messages are exchanged between a master and slave. Once the slave has collected the four necessary timestamps, the offset can be calculated as:

$$Offset = \frac{((t_2 - t_1) - (t_4 - t_3))}{2}$$

In colloquial terms, the offset is the difference between the master and slave clock, minus the one-way delay of sending a message [3]. Intuitively the logic stands, as the easiest way to sync a network is to have a reliable time source send out its own time for all listening components to copy as their own. PTP's additional calculations are necessary because it is impossible to predict the state or latency of network connections.

As mentioned above, SimGrid runs all its processes off of one simulated clock. In order to imitate clock desynchronization across the network, client processes kept track of an offset float-type variable, which when added to the global simulation clock would act as that individual process's clock. The decided range for the offset would be within ± 5 seconds. As we lack the academic funding to purchase our own atomic clocks or implement hardware timestamping, there will be some uncertainty with our results compared to running PTP in the real world. This can be somewhat accounted for by adjusting the SimGrid platform file, but only to a limited extent.

In order to offer meaningful insight, the PTP simulation was run with 5, 50, then 100 clients, each for 50 rounds. It is worth noting repeating the experiments always produced the same results, thanks to SimGrid's advertised reproducibility. If different results are desired with the same number of rounds and clients, the platform file describing how processes are connected to each other in the distributed system must be altered.

IV. DISCUSSION

A. *Berkeley's Algorithm*

There are multiple factors that go into choosing a clock synchronization method but there are clear goals that come with Berkeley's Algorithm. As mentioned earlier this method does not require specialized hardware which is a huge plus for implementing it into your systems. Other big mentions would be the fact that it is a decentralized method, making it a great option for a wide range of distributed systems/ network environments. It also has a plus of not having reduced network load. This overall creates an algorithm made for most systems no worry about load capabilities or special hardware requirements. With the decentralized model there is no worry about the size of your system. It is important to note its faults which every system has. In Berkeley's case it has three big drawbacks. The biggest drawback being that the accuracy is heavily dependent on network conditions as it requires the workers to get local time and send it to the master node. Another big one is the vulnerability to

malicious nodes. Since there is no centralized authority to check the validity of the times sent back any malicious node can give back incorrect times to screw up the clock synchronization. This is a huge problem as there is no way for Berkeley's algorithm to check if it is receiving correct info from its nodes it will simply go through with calculating based on the time it is given. Lastly it has limited precision which is partly due to what we mentioned earlier in which specialized hardware is required for higher accuracy. This is because most machines have a limit to how granular it can measure time(milliseconds or microseconds). Other things that can decrease precision is the clock drift and network latency.

B. *Precision Time Protocol (PTP)*

PTP simulations did show sub-microsecond accuracy when performing the network-wide clock synchronization. Three experiments were run:

- 5 clients, 50 rounds (250 syncs)
- 50 clients, 50 rounds (2500 syncs)
- 100 clients, 50 rounds (5000 syncs)

The clients are pulled from the same pool of worker nodes, so the first five clients are the same for all three experiments, the first 50 clients are the same for experiment 2 and 3, etc.

TABLE I. FIRST 10 ROUNDS OF 5 CLIENT SIM

Round #	Mean (sec)	Standard Deviation (sec)
1	$-6.4286047e^{-8}$	$1.1357125e^{-8}$
2	$-6.4286047e^{-8}$	$1.1357126e^{-8}$
3	$-6.4286047e^{-8}$	$1.1357125e^{-8}$
4	$-6.4286049e^{-8}$	$1.1357129e^{-8}$
5	$-6.4286047e^{-8}$	$1.1357122e^{-8}$
6	$-6.4286048e^{-8}$	$1.1357124e^{-8}$
7	$-6.4286047e^{-8}$	$1.1357127e^{-8}$
8	$-6.4286049e^{-8}$	$1.1357120e^{-8}$
9	$-6.4286049e^{-8}$	$1.1357120e^{-8}$
10	$-6.4286049e^{-8}$	$1.1357120e^{-8}$

TABLE II. 5 CLIENT SIM DATA

Expected Value of Mean	$-6.4286045e^{-8}$
Expected Value of Standard Deviation	$1.1357121e^{-8}$
Min. Offset	$-5.2472785e^{-8}$
Max. Offset	$-7.5070261e^{-8}$

The 5 client simulation ran for 565.9 simulated seconds.

TABLE III. FIRST 10 ROUNDS OF 50 CLIENT SIM

Round #	Mean (sec)	Standard Deviation (sec)
1	$-6.3990800e^{-8}$	$4.6949277e^{-8}$
2	$-6.3990798e^{-8}$	$4.6949278e^{-8}$
3	$-6.3990797e^{-8}$	$4.6949281e^{-8}$
4	$-6.3990800e^{-8}$	$4.6949275e^{-8}$
5	$-6.3990800e^{-8}$	$4.6949275e^{-8}$
6	$-6.3990803e^{-8}$	$4.6949269e^{-8}$
7	$-6.3990805e^{-8}$	$4.6949274e^{-8}$
8	$-6.3990801e^{-8}$	$4.6949278e^{-8}$
9	$-6.3990805e^{-8}$	$4.6949274e^{-8}$
10	$-6.3990805e^{-8}$	$4.6949274e^{-8}$

TABLE IV. 50 CLIENT SIM DATA

Expected Value of Mean	$-6.3990804e^{-8}$
Expected Value of Standard Deviation	$4.6949287e^{-8}$
Min. Offset	$-1.5786099e^{-8}$
Max. Offset	$-2.0950847e^{-7}$

The 50 client simulation ran for 4419.9 simulated seconds.

TABLE V. FIRST 10 ROUNDS OF 100 CLIENT SIM

Round #	Mean (sec)	Standard Deviation (sec)
1	$-7.0787094e^{-8}$	$5.2001844e^{-8}$
2	$-7.0787094e^{-8}$	$5.2001843e^{-8}$
3	$-7.0787094e^{-8}$	$5.2001847e^{-8}$
4	$-7.0787099e^{-8}$	$5.2001847e^{-8}$
5	$-7.0787099e^{-8}$	$5.2001847e^{-8}$
6	$-7.0787098e^{-8}$	$5.2001847e^{-8}$
7	$-7.0787090e^{-8}$	$5.2001842e^{-8}$
8	$-7.0787090e^{-8}$	$5.2001842e^{-8}$
9	$-7.0787090e^{-8}$	$5.2001842e^{-8}$
10	$-7.0787090e^{-8}$	$5.2001842e^{-8}$

TABLE VI. 100 CLIENT SIM DATA

Expected Value of Mean	$-7.0787120e^{-8}$
Expected Value of Standard Deviation	$5.2001870e^{-8}$
Min. Offset	$-1.5786099e^{-8}$
Max. Offset	$-2.0950847e^{-7}$

The 100 client simulation ran for 8560.8 simulated seconds.

As depicted above, the 5 client experiment produced the lowest standard deviation by far, but as the sample size was so small, some extreme values can be expected. The 50 client experiment produced the smallest mean offsets, making it the most accurate of the three. Interestingly, the 100 client

experiment showed a decrease in mean offset accuracy, even lower than that of the 5 client experiment. The expected standard deviation is the largest as well. More research can be done to determine where the optimal amount of client clocks for 50 round clock synchronization lies.

Given the nature of SimGrid to produce the same results from the same inputs, it is not worthwhile to compare the minimum and maximum offsets. They exist to form a baseline range for the post-synchronization offsets.

V. CONCLUSION

Berkeley's Algorithm and PTP barely occupy the same pseudo-ecological niche in the dangerous environment that is a distributed system. They serve different purposes, and both have their benefits and drawbacks.

When it comes to Berkeley's Algorithm, it easily settles itself as a go-to system for usability and scalability. Its simplicity and lack of required extra hardware makes it perfect for systems who don't have the privilege for special tools, yet still works for when/if you ever decide to grow your system. Disregarding its lower accuracy, Berkeley's works perfectly for self-contained systems that are more concerned with the order of distributed events than with absolute time accuracy. It is only when it comes to security and pushing lower latency is it that you should consider other options from Berkeley's.

The Precision Time Protocol is many degrees more accurate than Berkeley's Algorithm, and ensures the network is synchronized to a legitimate time source. Conversely, it requires hardware support that may not be viable for every distributed system, and the synchronization process can produce non-negligible amounts of network traffic. However, PTP remains the industry standard for clock synchronization. Its accuracy can be life-saving when implemented in real-time systems.

References

[1] The SimGrid Team. "The Modern Age of Computer Systems Simulation." SimGrid. https://simgrid.org/doc/latest/index.html (Accessed May 11, 2024).

[2] "PTP - Precision Time Protocol." Perle Systems. https://www.perle.com/supportfiles/precision-time-protocol.shtml (Accessed May 11, 2024).

[3] "How works the Precision Time Protocol (PTP)?" Bodet Time. https://www.bodet-time.com/resources/blog/1774-how-works-the-precision-time-protocol-ptp.html (Accessed May 11, 2024).