Berkeley's Algorithm

Iosif Vieru
DSWT 1-A

January 2026

1 Problem Description

Clock synchronization is the process by which all nodes in a distributed system update their clock values regularly and maintain information related to time so that the time shown by each node after appropriate transformation of clock value is nearly the same at any real time. Clock synchronization is essential in distributed systems, as it simplifies the analysis, design, and stable operation of such systems by enabling participating nodes to share a common notion of time. This common time base allows for synchronous data acquisition and simultaneous triggering of events. [1]

2 Solving ideas

There are different approaches to clock synchronization in distributed systems.

You can either have a centralized authority (a leader node), where a master server dictates the system time to the other nodes. *Christian's Algorithm* is a great example where each node requests the time from the server and adjust their clock with the estimated communication delay. Or you can have a decentralized approach where there is no authority and each node participate equally in the synchronization process example being *Average Consensus* [2] in which each node communicate to converge on a single value which is the average of all their initial values.

3 Algorithm

3.1 Description

The solution is divided into two consecutive phases:

- **Leader Election**: Processes are logically arranged in a ring topology. Each process generates a unique identifier (UUID) and participates in the LCR algorithm to elect the process with the highest UUID.

- **Clock Synchronization**: Once the leader is elected, *Berkeley's algorithm* [3] is used. The leader collects clock values from all processes, computes the average offset, and broadcasts the synchronized time. Each node will adjust its own clock based on the received offset.

3.2 Pseudocode

Algorithm 1 LCR leader election and Berkeley clock synchronization

Require: N processes in a ring, process id p
Ensure: elected leader L and synchronized clocks

$left \leftarrow (p - 1 + N) \bmod N$
$right \leftarrow (p + 1) \bmod N$
$uuid \leftarrow$ random integer
$maxUuid \leftarrow uuid$
LCR leader election
for $i \leftarrow 1$ to N **do**
  SEND($maxUuid$, $right$)
  $recv \leftarrow$ RECV($left$)
  if $recv > maxUuid$ **then**
    $maxUuid \leftarrow recv$
  end if
end for
if $uuid = maxUuid$ **then**
  $L \leftarrow p$ ▷ I am the leader and must tell everyone
  SEND(L, $right$)
else
  $L \leftarrow$ RECV($left$)
  if $right \neq L$ **then**
    SEND(L, $right$)
  end if
end if
Berkeley clock synchronization
$t_p \leftarrow$ wtime()
gather(t_p) to leader L
if $p = L$ **then**
  $avg \leftarrow \frac{1}{N-1} \sum_{i \neq L} (t_p - t_i)$
  $T_L \leftarrow t_p + avg$
end if
broadcast(T_L)
$t_p \leftarrow$ wtime() +($T_L -$ wtime)

3.3 Usage examples

3.4 Correctness

Leader Election Correctness:

- LCR (*Le-Lann, Chang, Roberts*) guarantees that the maximum UUID circulates through the entire ring
- Exactly one process owns this UUID
- A unique leader is elected

Clock Synchronization Correctness:

- The leader collects all clock values
- The computed average offset minimizes total clock deviation
- Broadcasting the final clock ensures all processes converge

The output confirms convergence, validating the correctness of the implementation.

3.5 Complexity analysis

Leader Election:

- Time Complexity: $O(n)$ rounds
- Message Complexity: $O(n^2)$ messages

Clock Synchronization:

- Time Complexity: $O(n)$
- Message Complexity: $O(n)$

3.6 Implementation

The algorithm is implemented in C using MPI primitives:

- `MPI_Send`, `MPI_Recv` for ring communication
- `MPI_Gather` for clock collection
- `MPI_Bcast` for time dissemination

3.7 Testing

The program was tested with multiple process counts. Observed results show:

- Correct leader election in all runs
- Clock values converging to a common synchronized time

Example of program execution:

```
mpirun -np 8 --oversubscribe main.out
pid: 3 my clock is : 0.576005
pid: 3, my UUID: 846
pid: 2 my clock is : 0.578270
pid: 2, my UUID: 566
pid: 4 my clock is : 0.577952
pid: 4, my UUID: 505
pid: 0 my clock is : 0.578752
pid: 5 my clock is : 0.576265
pid: 5, my UUID: 215
pid: 7 my clock is : 0.575147
pid: 7, my UUID: 454
pid: 0, my UUID: 128
pid: 1 my clock is : 0.574886
pid: 1, my UUID: 198
pid: 6 my clock is : 0.573728
pid: 6, my UUID: 326
PID 3 won the election with the value: 846!!!
pid: 3, my max_uuid is 846, my leader_id = 3
pid: 4, my max_uuid is 846, my leader_id = 3
pid: 6, my max_uuid is 846, my leader_id = 3
pid: 5, my max_uuid is 846, my leader_id = 3
pid: 7, my max_uuid is 846, my leader_id = 3
pid: 2, my max_uuid is 846, my leader_id = 3
pid: 0, my max_uuid is 846, my leader_id = 3
pid: 1, my max_uuid is 846, my leader_id = 3
LEADER: computed average is -0.000348
pid: 3 (LEADER): my adjusted clock is: 0.577159
pid 0: my adjustment is: -0.003793
pid 0: my adjusted clock is: 0.577164
pid 4: my adjustment is: -0.003006
pid 4: my adjusted clock is: 0.577165
pid 2: my adjustment is: -0.003368
pid 2: my adjusted clock is: 0.577164
pid 6: my adjustment is: 0.001336
pid 6: my adjusted clock is: 0.577163
pid 7: my adjustment is: -0.000432
```

```
pid 7: my adjusted clock is: 0.577164
pid 5: my adjustment is: -0.001558
pid 5: my adjusted clock is: 0.577164
pid 1: my adjustment is: -0.000173
pid 1: my adjusted clock is: 0.577165
```

4 Conclusions

The presented solution implements Berkeley's algorithm and successfully synchronizes the clocks of all participating processes. The results obtained from the MPI-based implementation confirm that the clock values converge to a common time after the synchronization phase.

The use of a leader-based approach simplifies the synchronization process, while the leader election mechanism ensures that a single coordinator is chosen. Experimental testing with multiple processes shows consistent behavior, validating the correctness of both the leader election and clock adjustment steps.

References

[1] *Clock Synchronization*. ScienceDirect Topics. URL: https://www.sciencedirect.com/topics/computer-science/clock-synchronization (visited on 01/07/2026).

[2] *Average Consensus*. ScienceDirect Topics. URL: https://www.sciencedirect.com/topics/computer-science/average-consensus (visited on 01/07/2026).

[3] R. Gusella and S. Zatti. *The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD*. Technical Report, Computer Science Division, University of California, Berkeley. 1987. URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/1987/CSD-87-337.pdf.