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 Leader election (LCR) + Berkeley synchronization (RTT + outlier exclusion)

```
Require: $N$ processes in a ring, process id $p$
Ensure: leader $L$ and local adjustment $\Delta_p$
  $left \leftarrow (p - 1 + N) \bmod N,\;\; right \leftarrow (p + 1) \bmod N$
  $uuid \leftarrow$ unique value generated locally
  $max \leftarrow uuid$
  LCR leader election
  for $k \leftarrow 1$ to $N$ do
      send $max$ to $right$; receive $x$ from $left$
      $max \leftarrow \max(max, x)$
  end for
  $L \leftarrow p$ if $uuid = max$ else receive $L$ from ring and forward
  Berkeley synchronization (leader-based)
  $t_p \leftarrow$ localTime()
  if $p = L$ then
      for each process $i \neq L$ do
          measure RTT to $i$ and receive its timestamp $t_i$
          $\hat{t}_i \leftarrow t_i + \mathrm{RTT}/2$                    ▷ one-way delay estimate
      end for
      choose threshold $\tau$ (maximum allowed drift)
      $S \leftarrow \{\, i \mid i = L \,\vee\, |\hat{t}_i - t_L| \le \tau \,\}$          ▷ exclude outliers
      $avgOffset \leftarrow \frac{1}{|S|} \sum_{i \in S} (\hat{t}_i - t_L)$
      $T_L \leftarrow t_L + avgOffset$
  end if
  broadcast $T_L$ from leader
  $\Delta_p \leftarrow T_L -$ localTime()
                                        ▷ Adjusted time is localTime() $+\Delta_p$
```

3.3 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.

3.4 Complexity analysis

Leader Election:

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

Clock Synchronization:

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

3.5 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

3.6 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: 7 my clock is : 0.559591
pid: 7, my UUID: 3507
pid: 5 my clock is : 0.560775
pid: 5, my UUID: 21305
pid: 1 my clock is : 0.561513
pid: 1, my UUID: 90601
pid: 0 my clock is : 0.561882
pid: 0, my UUID: 5900
pid: 4 my clock is : 0.561940
pid: 4, my UUID: 62404
pid: 6 my clock is : 0.561513
pid: 6, my UUID: 38806
pid: 3 my clock is : 0.561855
pid: 3, my UUID: 58803
pid: 2 my clock is : 0.562063
pid: 2, my UUID: 57902
PID 1 won the election with the value: 90601!!!
pid: 3 I got random drift: 1.926000, my clock now: 0.563507
pid: 5 I got random drift: 5.653000, my clock now: 0.562484
pid: 0 I got random drift: 7.574000, my clock now: 0.563577
LEADER: calcualted RTT time for node: 0 is 0.000009
LEADER: calcualted RTT time for node: 2 is 0.000001
LEADER: calcualted RTT time for node: 3 is 0.001463
LEADER: calcualted RTT time for node: 4 is 0.000011
LEADER: calcualted RTT time for node: 5 is 0.000011
LEADER: calcualted RTT time for node: 6 is 0.000010
LEADER: calcualted RTT time for node: 7 is 0.000007
LEADER: pid 0 is excluded due to high clock drift: 7.574384
LEADER: pid 3 is excluded due to high clock drift: 1.928513
LEADER: pid 5 is excluded due to high clock drift: 5.653804
LEADER: computed average is 0.000696
pid: 1 (LEADER): my adjusted clock is: 0.563900
pid 2: my adjustment is: -0.001787
pid 2: my adjusted clock is: 0.563908
pid 6: my adjustment is: -0.001354
pid 6: my adjusted clock is: 0.563908
pid 0: my adjustment is: -7.575762
pid 0: my adjusted clock is: 0.563908
pid 4: my adjustment is: -0.001785
pid 4: my adjusted clock is: 0.563908
pid 5: my adjustment is: -5.653733
pid 5: my adjusted clock is: 0.563907
pid 3: my adjustment is: -1.927762
pid 3: my adjusted clock is: 0.563908
pid 7: my adjustment is: 0.000429
pid 7: my adjusted clock is: 0.563909
```

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.