Time Stamp Synchronization for Event Traces of Large-Scale Message Passing Applications

D. Becker and F. Wolf
Forschungszentrum Jülich
Jülich Supercomputing Centre

R. Rabenseifner
High Performance Computing Center Stuttgart
Department Parallel Computing

Outline

- Introduction
- Event model and replay-based parallel analysis
- Controlled logical clock
- Extended controlled logical clock
- Timestamp synchronization
- Conclusion
- Future work

Non-Synchronized Clocks

- Investigation of dependencies between concurrent events
  - Timeline visualization
  - Wait states diagnosis
- Problem - local processor clocks are often non-synchronized
  - Clocks may vary in offset and drift
- Present approach - linear interpolation
  - Accounts for differences in offset and drift
  - Assumes that drift is not time dependant
- Inaccuracies and changing drifts can still cause violations of the logical event ordering

Synchronization method for violations not already covered by linear interpolation required

Idea

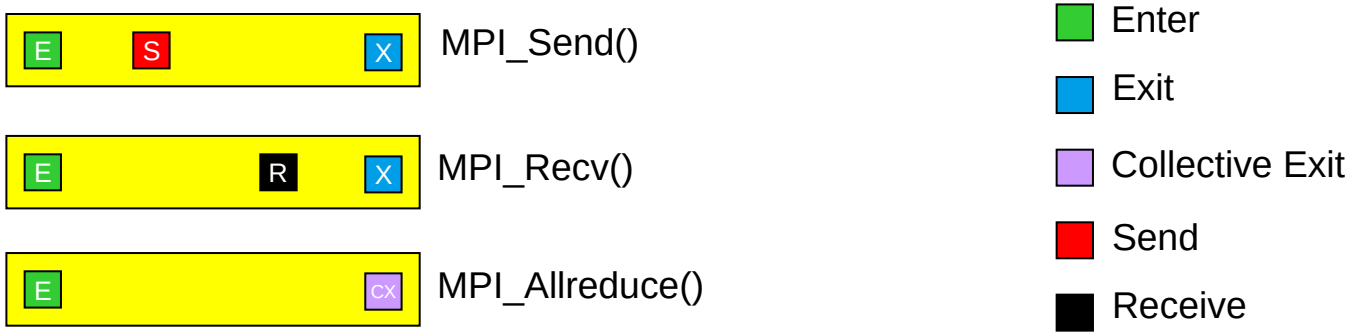
- Requirement - realistic message passing codes
  - Different modes of communication (P2P & collective)
  - Large numbers of processes
- Build on controlled logical clock by Rolf Rabenseifner
  - Synchronization based on Lamport's logical clock
  - Only P2P communication
  - Sequential implementation
- Approach
  - Extend controlled logical clock to collective operations
  - Define scalable correction algorithm through parallel replay

Event Model

- Event includes at least timestamp, location and event type
  - Additional information may be supplied depending on event type
- Event type refers to
  - Programming-model independent events
  - MPI-related events
  - Events internal to tracing library

Example

- Event sequence recorded for typical MPI operations

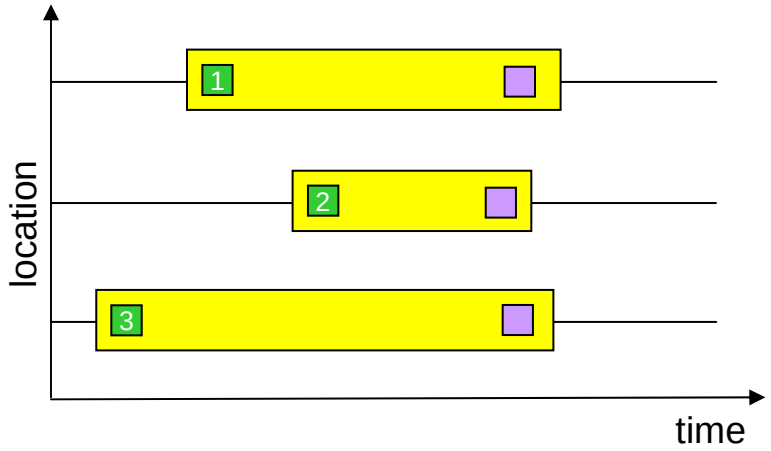


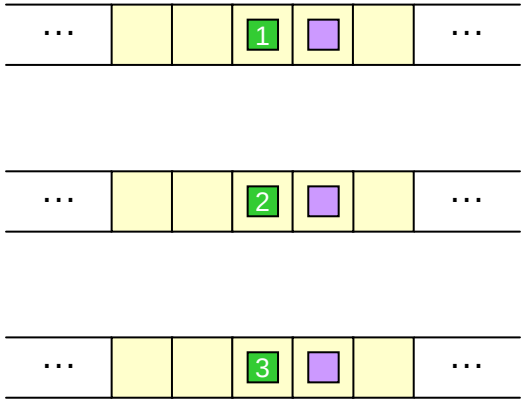
Replay-Based Parallel Analysis

- Parallel analysis scheme
  - SCALASCA toolset
  - Originally developed to improve scalability on large-scale systems
- Analyze separate local trace files in parallel
  - Exploits distributed memory & processing capabilities
  - Keeps whole trace in main memory
  - Only process-local information visible to a process
- Parallel replay of target application‘s communication behavior
  - Parallel traversal of event streams
  - Analyze communication with operation of same type
  - Exchange of required data at synchronization points of target application

Example

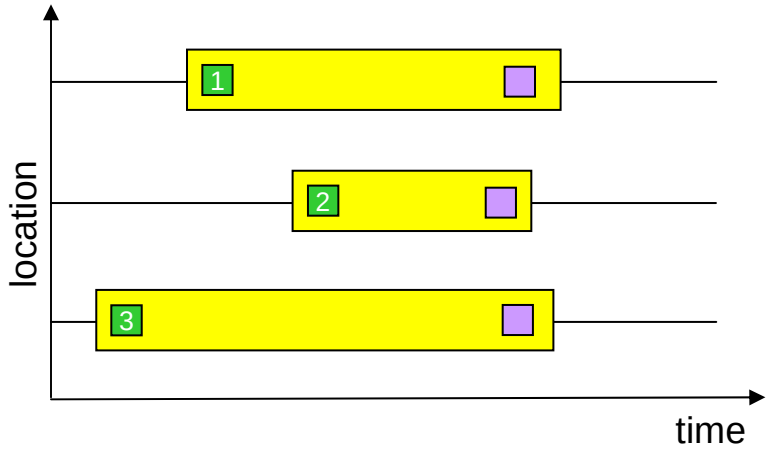
- Determine latest enter event using SCALASCA's parallel replay
- Collective operation: MPI_Allreduce()

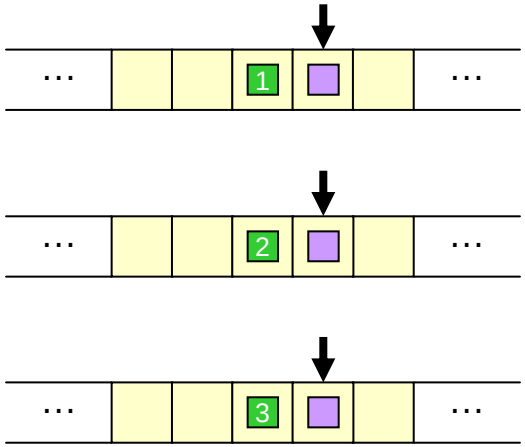




Example

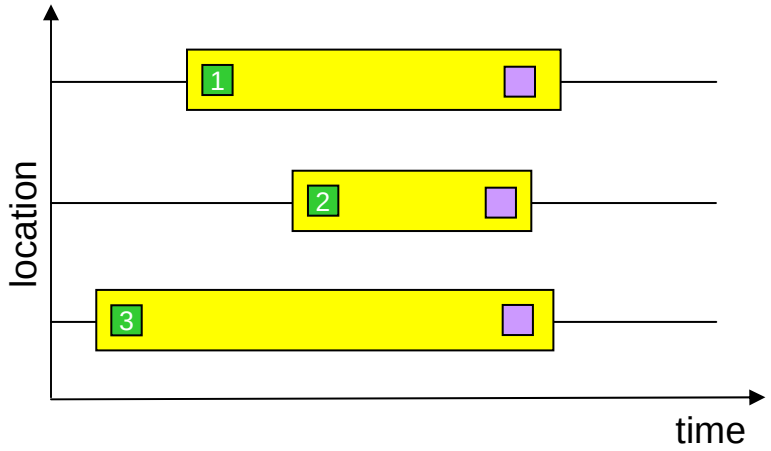
- Determine latest enter event using SCALASCA's parallel replay
- Collective operation: MPI_Allreduce()

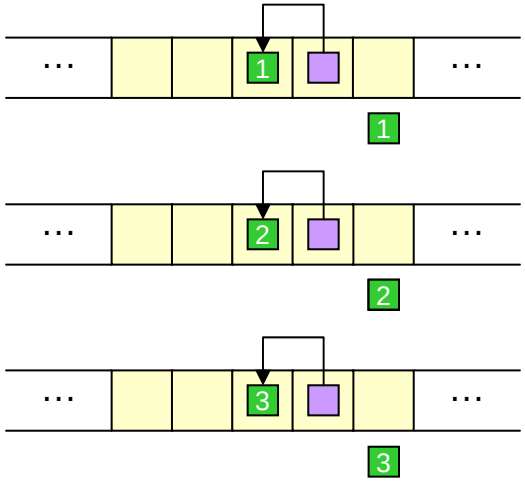




Example

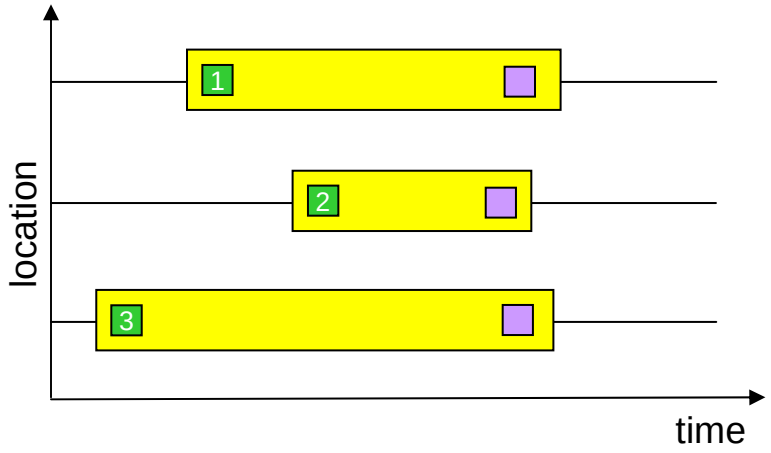
- Determine latest enter event using SCALASCA's parallel replay
- Collective operation: MPI_Allreduce()

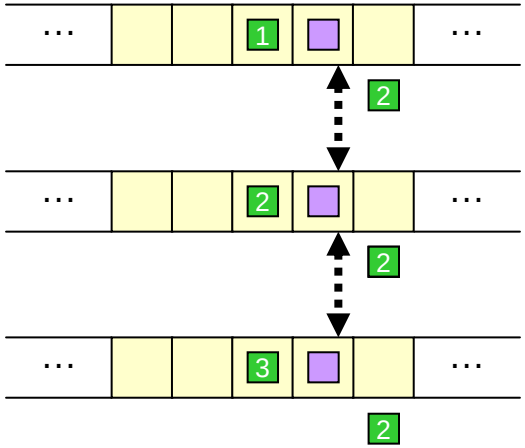




Example

- Determine latest enter event using SCALASCA's parallel replay
- Collective operation: MPI_Allreduce()

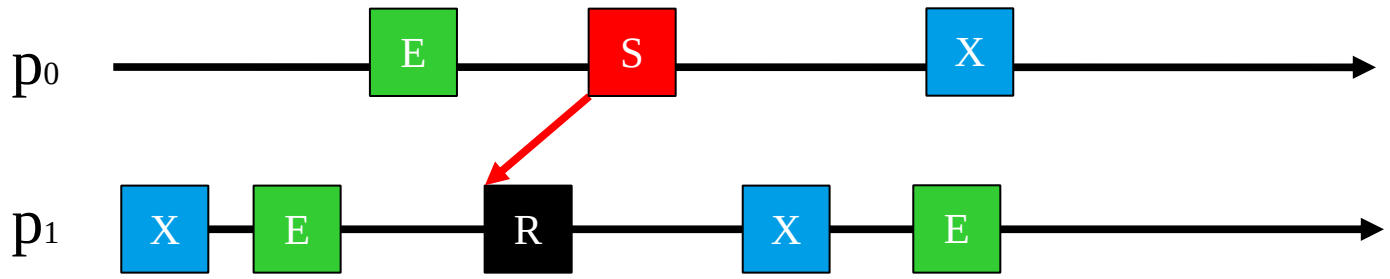




Controlled Logical Clock

- Guarantees Lamport's clock condition
  - Use happened-before relations to synchronize timestamps
  - Send event always earlier than receive event
- Scans event trace for clock condition violations and modifies inexact timestamps
- Stretches process-local time axis in the immediate vicinity of affected event
  - Preserves length of intervals between local events
    - Forward amortization
  - Smoothes discontinuity at affected event
    - Backward amortization

Example

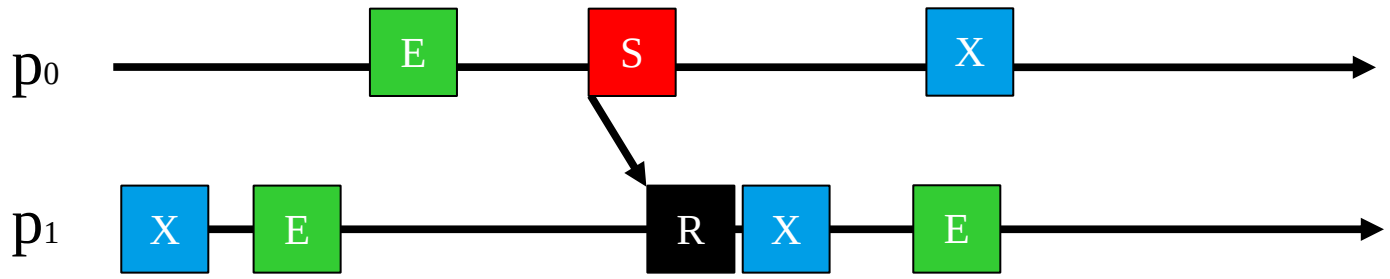


Problems:

- Clock condition violation

Solutions:

Example



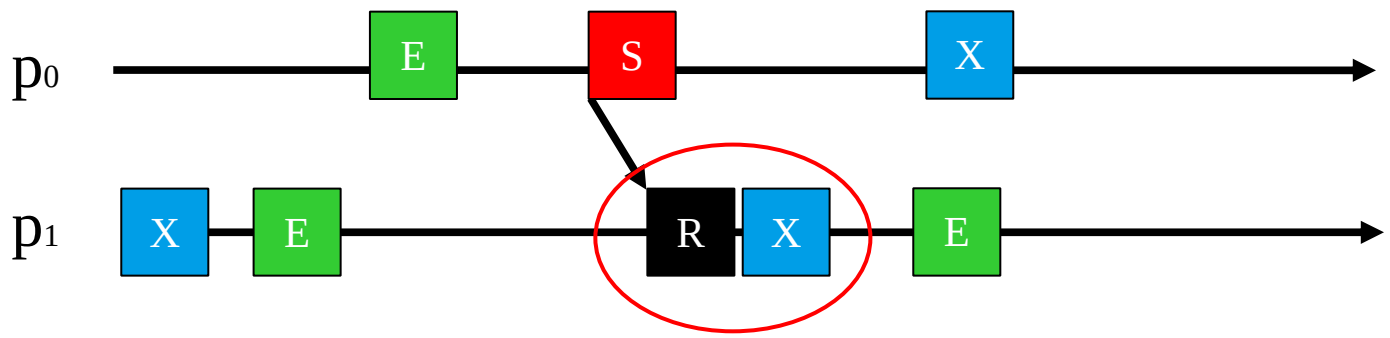
Problems:

- Clock condition violation

Solutions:

- Local correction

Example



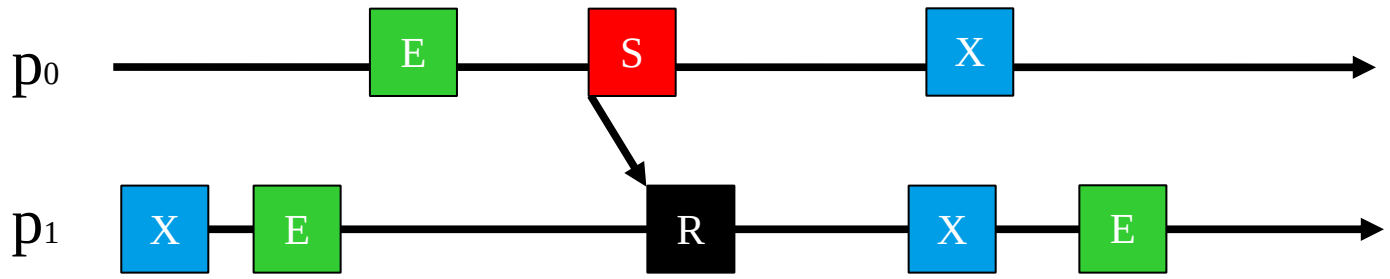
Problems:

- Clock condition violation
- Time period between subsequent events

Solutions:

- Local correction

Example



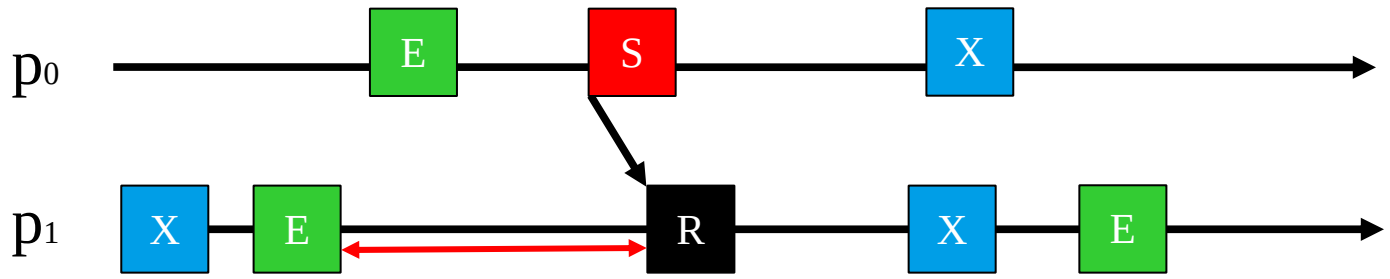
Problems:

- Clock condition violation
- Time period between subsequent events

Solutions:

- Local correction
- Forward amortization

Example



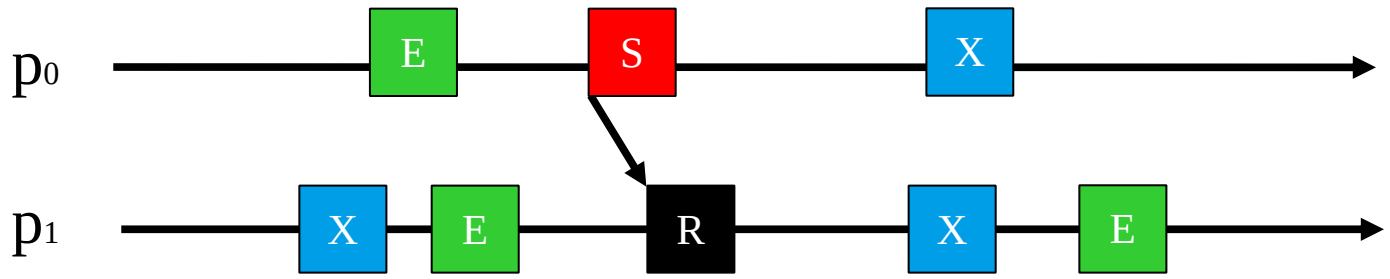
Problems:

- Clock condition violation
- Time period between subsequent events
- Time period between preceding events

Solutions:

- Local correction
- Forward amortization

Example



Problems:

- Clock condition violation
- Time period between subsequent events
- Time period between preceding events

Solutions:

- Local correction
- Forward amortization
- Backward amortization

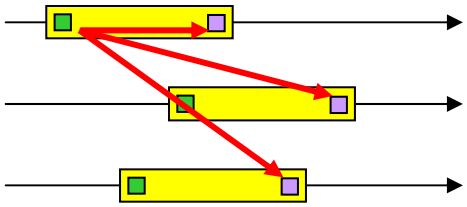
Extended Controlled Logical Clock

- Advance local clock if clock condition violation occurred
- Consider single collective operation as composition of many point-to-point communications
- Use different types of event semantics
  - Point-to-point communication
  - Collective communication

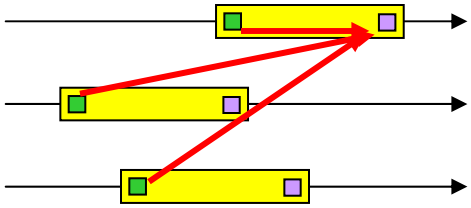
- Determine send and receive events for each type
- Define happened-before relations based on decomposition of collective operations

Decomposition of Collective Operations

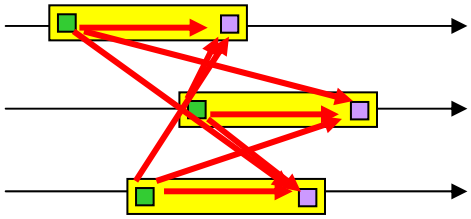
- 1xN: Root sends data to N processes



- Nx1: N processes send data to root



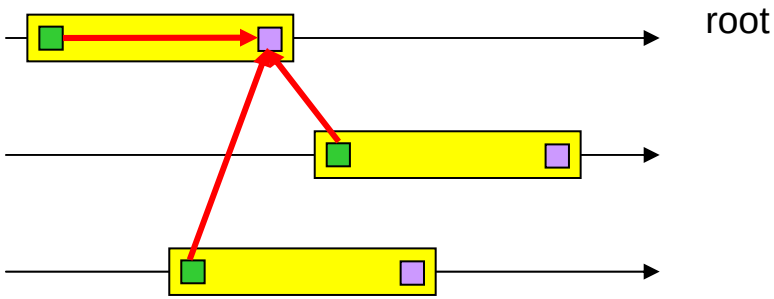
- NxN: N processes send data to N processes



Happened-Before Relation

- Send event always earlier than receive event
- Synchronization needs one send event timestamp
- Operation may have multiple send and receive events
- Multiple receives used to synchronize multiple clocks
- Latest send event is the relevant send event

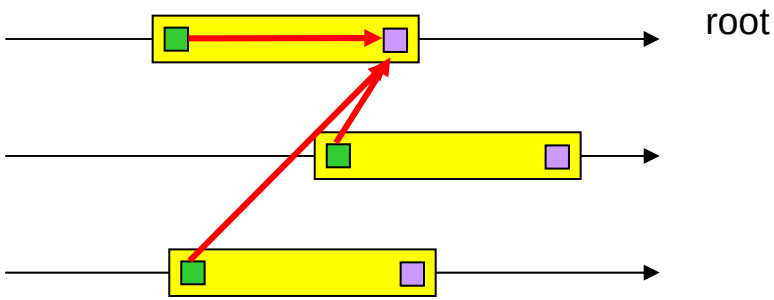
- Example: N-to-1



Happened-Before Relation

- Send event always earlier than receive event
- Synchronization needs one send event timestamp
- Operation may have multiple send and receive events
- Multiple receives used to synchronize multiple clocks
- Latest send event is the relevant send event

- Example: N-to-1

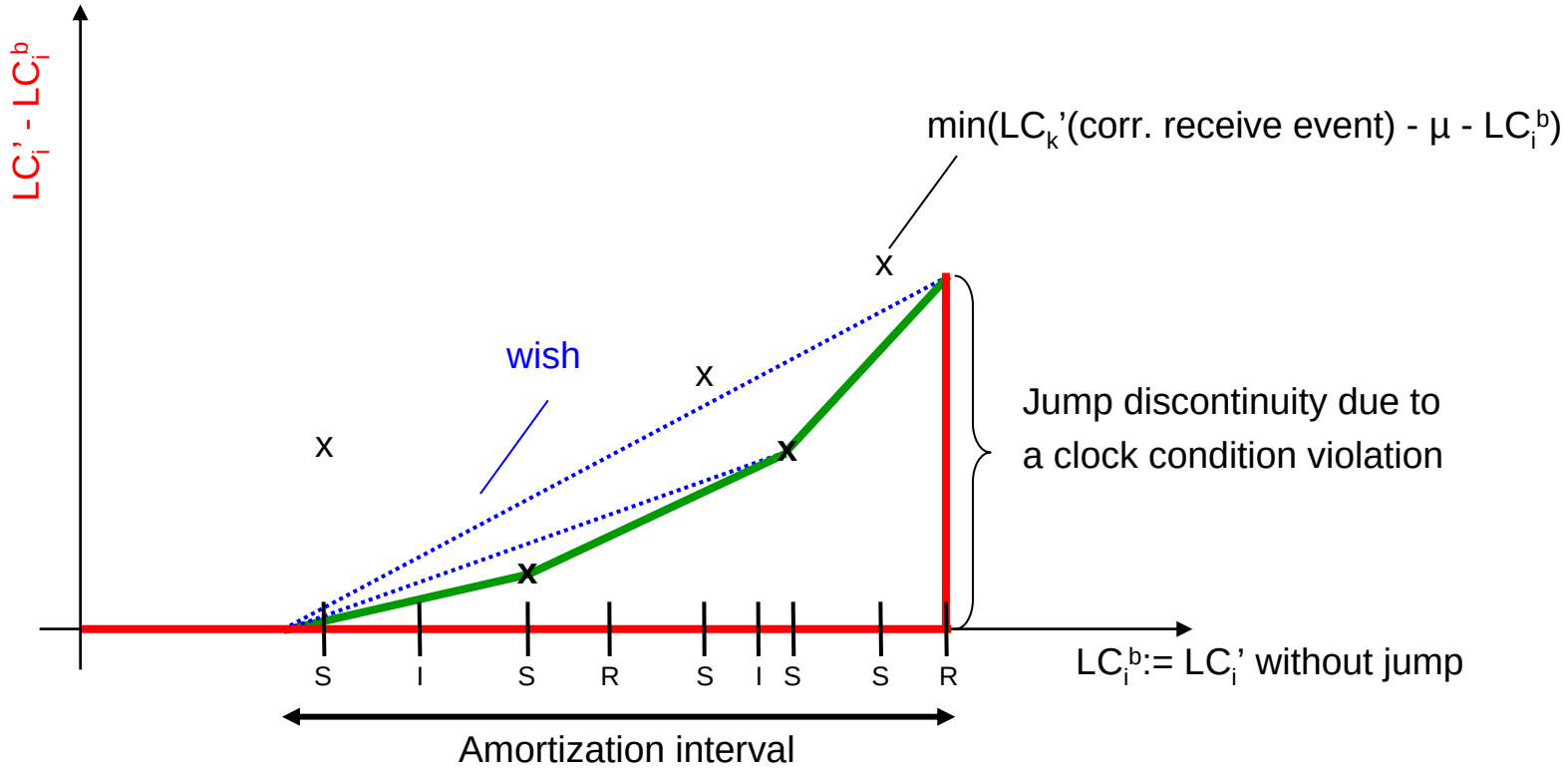


Forward Amortization

- New timestamp LC' is maximum of
  - Max(send event timestamp + minimum latency)
  - Event timestamp
  - Previous event timestamp + minimum event spacing
  - Previous event timestamp + controlled event spacing

- Controller
  - Approximates original communication after clock condition violation
  - Limits synchronization error
  - Bounds propagation during forward amortization
  - Requires global view of the trace data

Backward Amortization



- Results of the extended controlled logical clock with jump discontinuities
- Linear interpolation with backwards amortization
- Piecewise linear interpolation with backwards amortization

$$\text{Amortization interval} = \frac{\text{jump}}{\text{accuracy}}$$

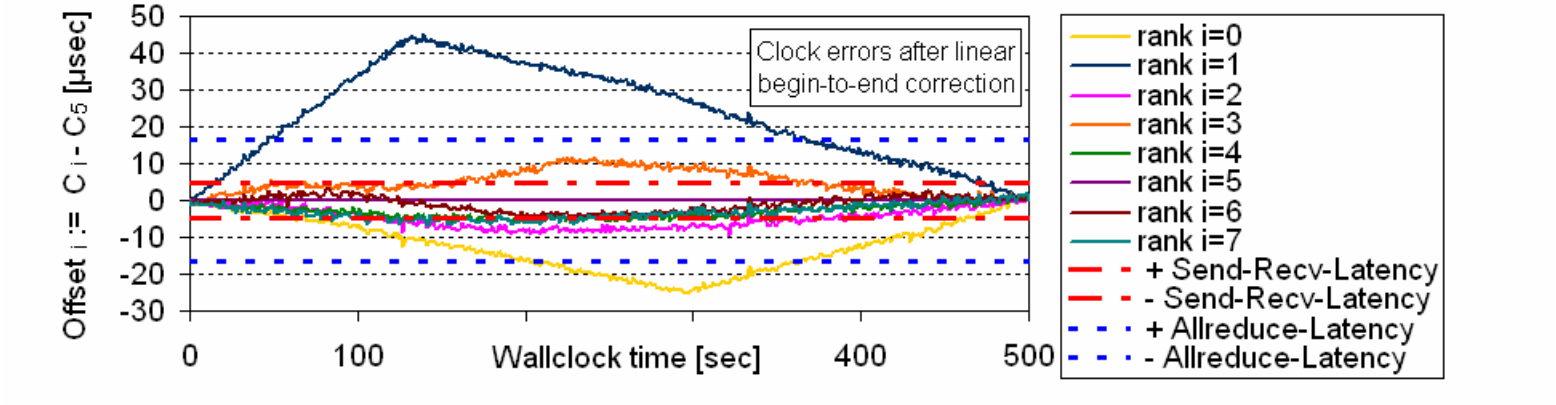
Timestamp Synchronization

- Event tracing of applications running on thousands of processes requires scalable synchronization scheme
- Proposed algorithm depends on accuracy of original timestamps

- Two-step synchronization scheme
  - Pre-synchronization
    - Linear interpolation
  - Parallel post-mortem timestamp synchronization
    - Extended controlled logical clock

Pre-Synchronization

- Account for differences in offset and drift
- Assume that drift is not time dependant
- Offset measurement at program initialization and finalization
  - Among arbitrary chosen master and worker processes
- Linear interpolation between these two points



Parallel Timestamp Synchronization

- Extended controlled logical clock
- Parallel traversal of the event stream
  - Forward amortization
  - Backward amortization

- Exchange required timestamp at synchronization points
- Perform clock correction
- Apply control mechanism after replaying the communication
  - Global view of the trace data
  - Multiple passes until error is below a predefined threshold

Forward Amortization

- Timestamps exchanged depending on the type of operation

Type of operation	timestamp exchanged	MPI function
P2P	timestamp of send event	MPI Send
1-to-N	timestamp of root enter event	MPI Bcast
N-to-1	max(all enter event timestamps)	MPI Reduce
N-to-N	max(all enter event timestamps)	MPI Allreduce

Backward Amortization

- Timestamps exchanged depending on the type of operation

Type of operation	timestamp exchanged	MPI function
P2P	timestamp of receive event	MPI Send
1-to-N	min(all collective exit event timestamps)	MPI Reduce
N-to-1	timestamp of root collective exit event	MPI Bcast
N-to-N	min(all collective exit event timestamps)	MPI Allreduce

Conclusion

- Extended controlled logical clock algorithm takes collective communication semantics into account
  - Defined collective send and receive operations
  - Defined collective happened-before relations

- Parallel implementation design presented using SCALASCA's parallel replay approach
  - Exploits distributed memory & processing capabilities

Future Work

- Finish actual implementation
- Evaluate algorithm using real message passing codes
- Extend algorithm to shared memory programming models
- Extend algorithm to one sided communication

Thank you...

For more information, visit
our project home page:

http://www.scalasca.org