State History Storage in Disk-based Interval Trees

Mid-project update

Alexandre Montplaisir
Michel Dagenais

December 8, 2010
École Polytechnique, Montreal

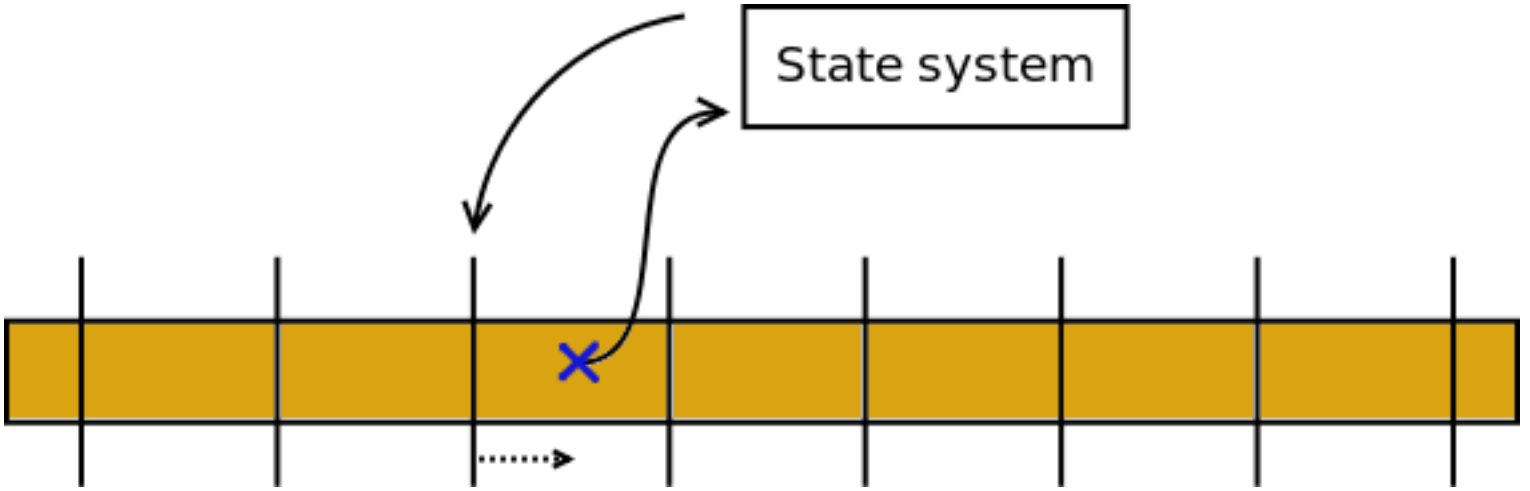
Contents

- Overview
  - State system
  - Current method, shortcomings
  - Proposed solution
- State History functionalities
- External API
- Design variants
- Future work
- Conclusion

State system in trace viewers

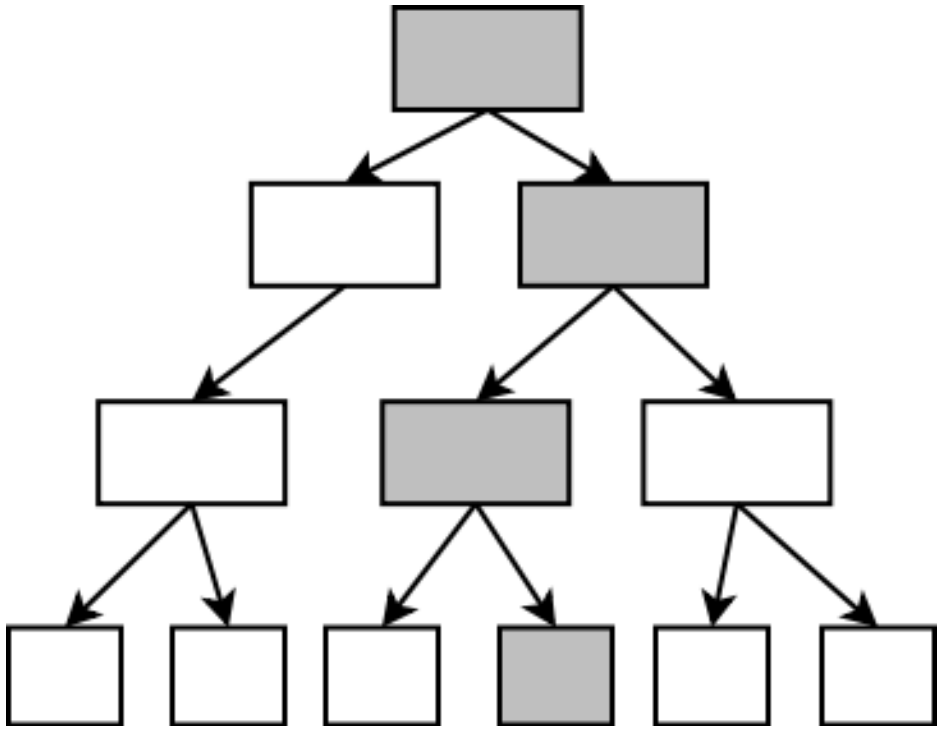
- Trace viewers need to be able to re-create the complete state the machine was in, at any given point in a trace.
- State information includes:
  - Running processes
  - Open file descriptors
  - State of CPUs, block devices, ...
  - etc.

Current method: checkpoints



- Checkpoints stored in memory
- Number/frequency of checkpoints:
  trade-off between speed and space usage
- Scalability is a problem

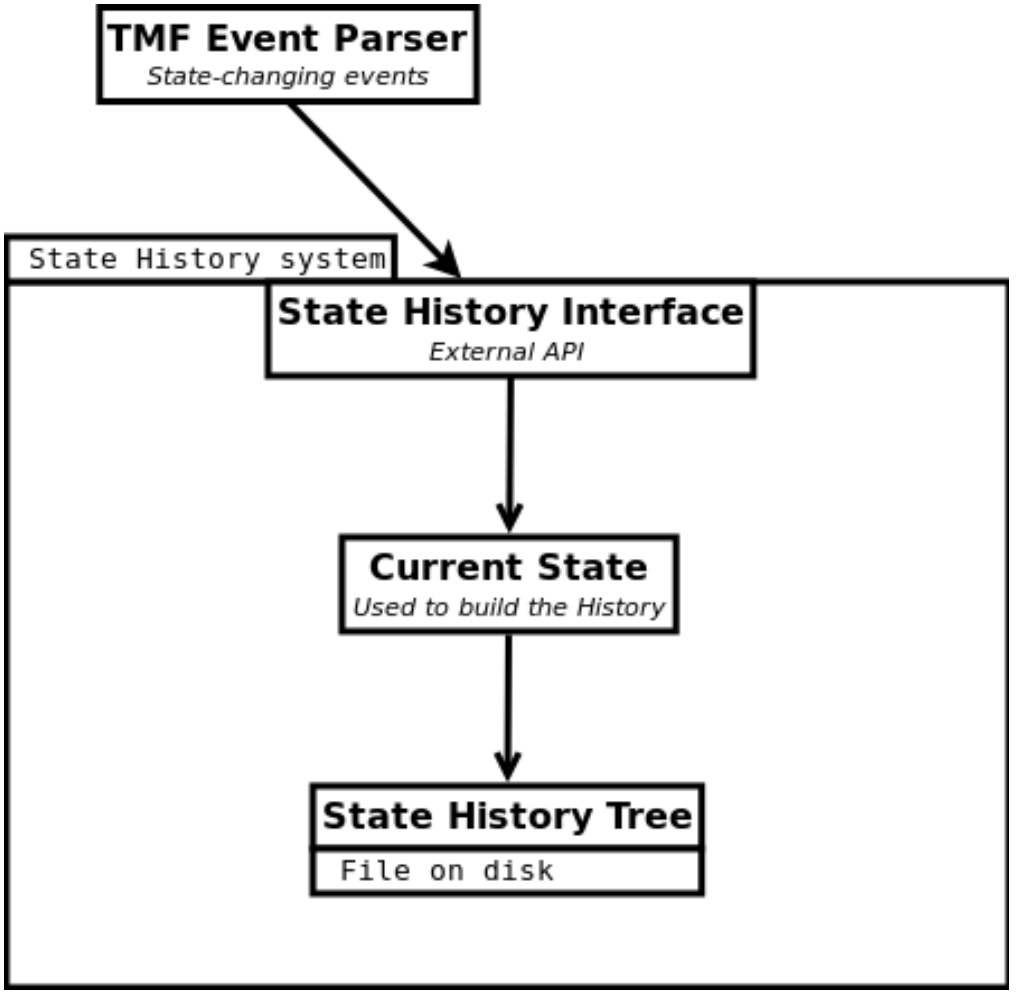
Proposed alternative: the State History



- State information stored as intervals
- Based on disk
- Fixed-size nodes (multiple of disk sector)
- Reading only one branch is required to rebuild the state at a given time.

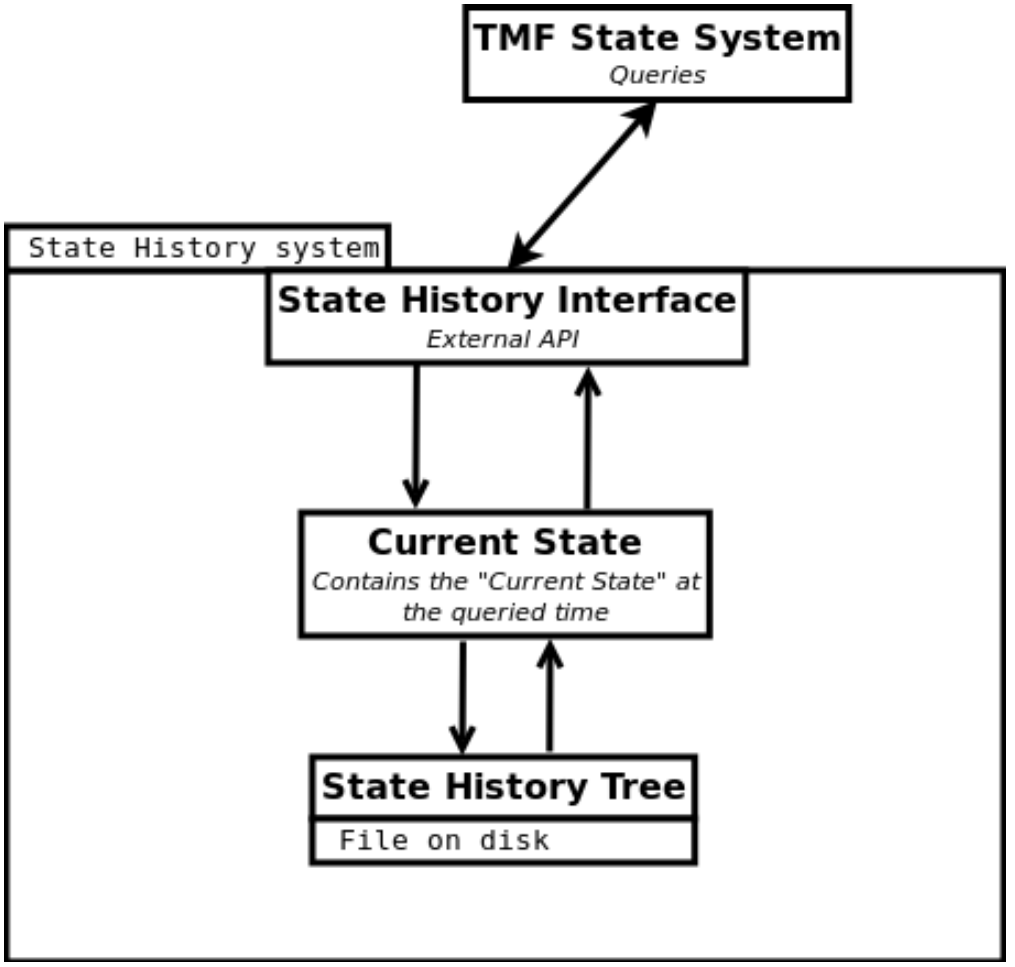
State History functionalities

Building the tree

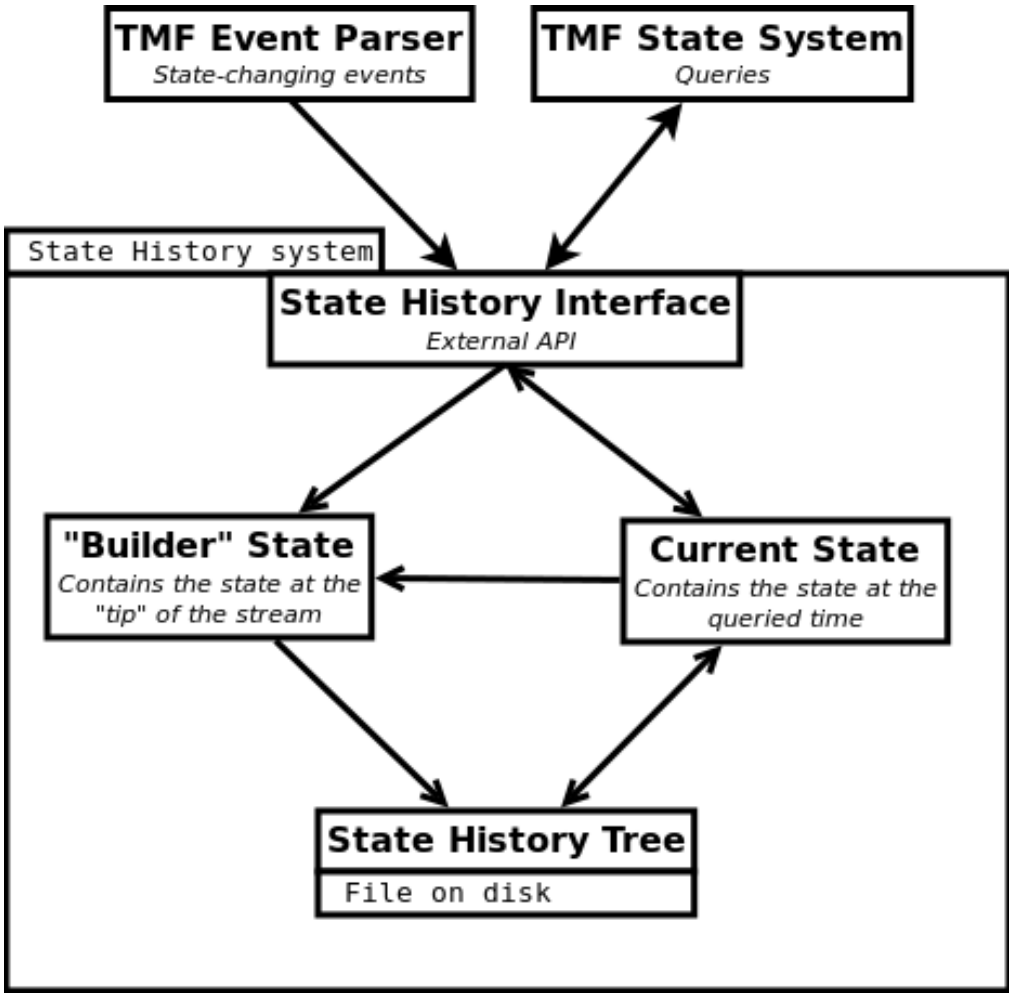


State History functionalities

Querying the tree
(offline trace)



State History functionalities



Live trace reading

External API

Basic methods:

- ModifyAttribute(attribute, value, timestamp)
- Query(attribute, timestamp), returns the value

But it also allows for more nifty stuff:

- Get next/prev. state change(attrib., timestamp),

  returns the next/previous state or timestamp of that change

Design variants

- Find optimal Block size & Max. nb of children per node

- Storing intervals at all levels vs. Storing only in leafs
- Using disjointed nodes vs. overlapping ones (R-Trees)

- Sorting the intervals in a node when we close it
- Sorting the nodes/blocks in the file once we're done writing

Future work

- Implement the complete State History system (DONE)
- Connect into TMF's event parser (ONGOING)
- Return state queries information to TMF (TODO)
- Test different implementation and design variants, benchmark, compare (TODO)
- Add support for re-opening existing History files (Nice to have)
- Fix bugs!

Conclusion

This new storage algorithm:

- Solves the scalability problem
- Generic

  Will not require modifications if we add new states
- Allows streaming
- Allows dependency analysis

Thank you

Questions?