

Tracing the MPI-IO Calls Disk Accesses



Thomas Ludwig

S. Krempel, J. Kunkel, F. Panse, D. Withanage

Ruprecht-Karls-Universität Heidelberg

Computer Science Department

Parallel and Distributed Systems

t.ludwig@computer.org



The Semantic Gap with I/O

- Program I/O is done in parallel applications with MPI-IO
- System I/O is performed by the servers of the I/O subsystem
- There is no tool that shows the causal relation between activities on both abstraction levels



Consequences

- No real insight into low level I/O activity in dependency of high level I/O activity
- No easy tuning of the application
- No easy tuning of the servers
- No optimal adaptation to the available resources



Project Goal

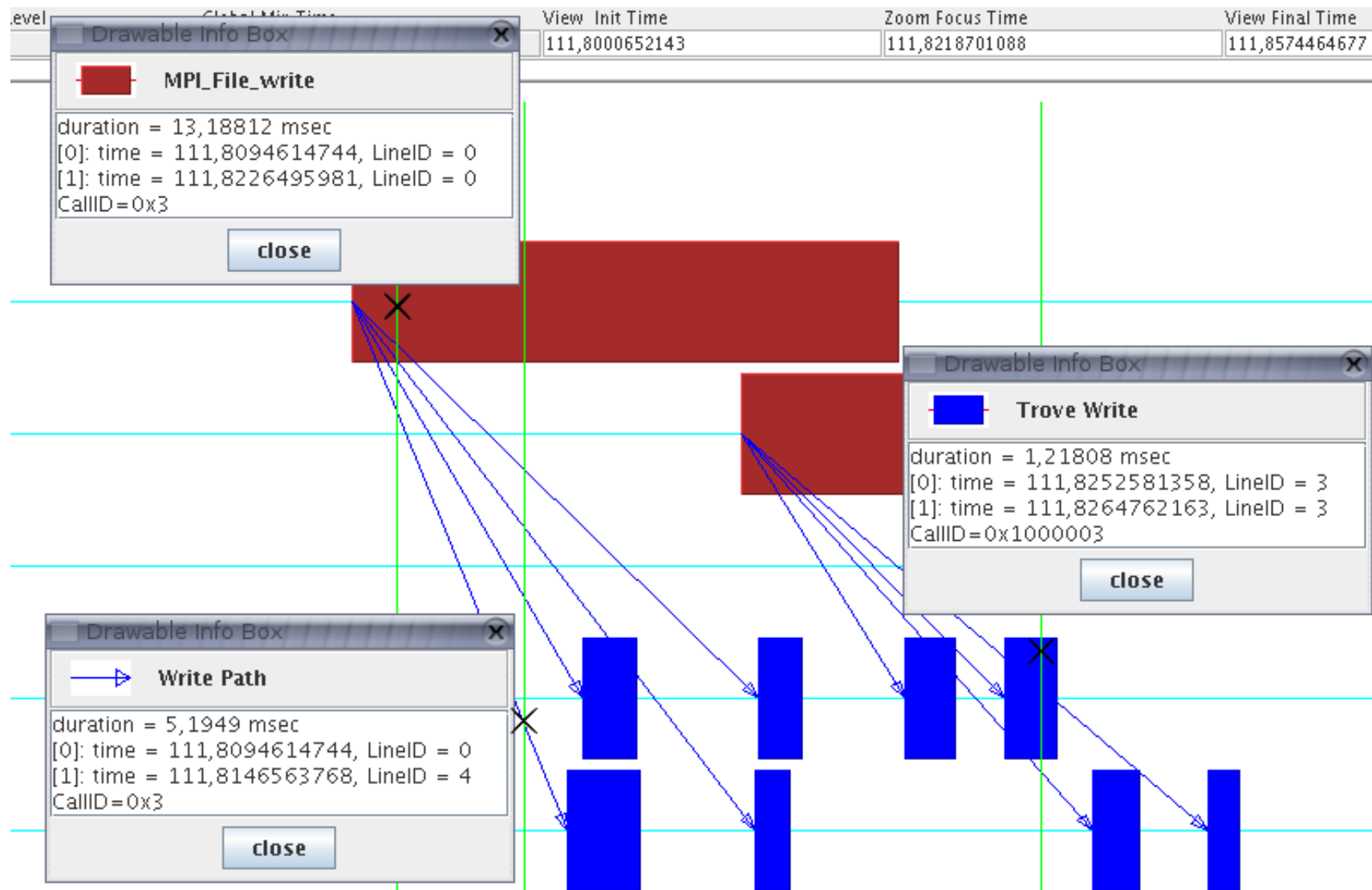
Design and implement a trace-based tool environment that visualizes three aspects:

- Client I/O activity
- Server I/O activity
- The relation of the two of them

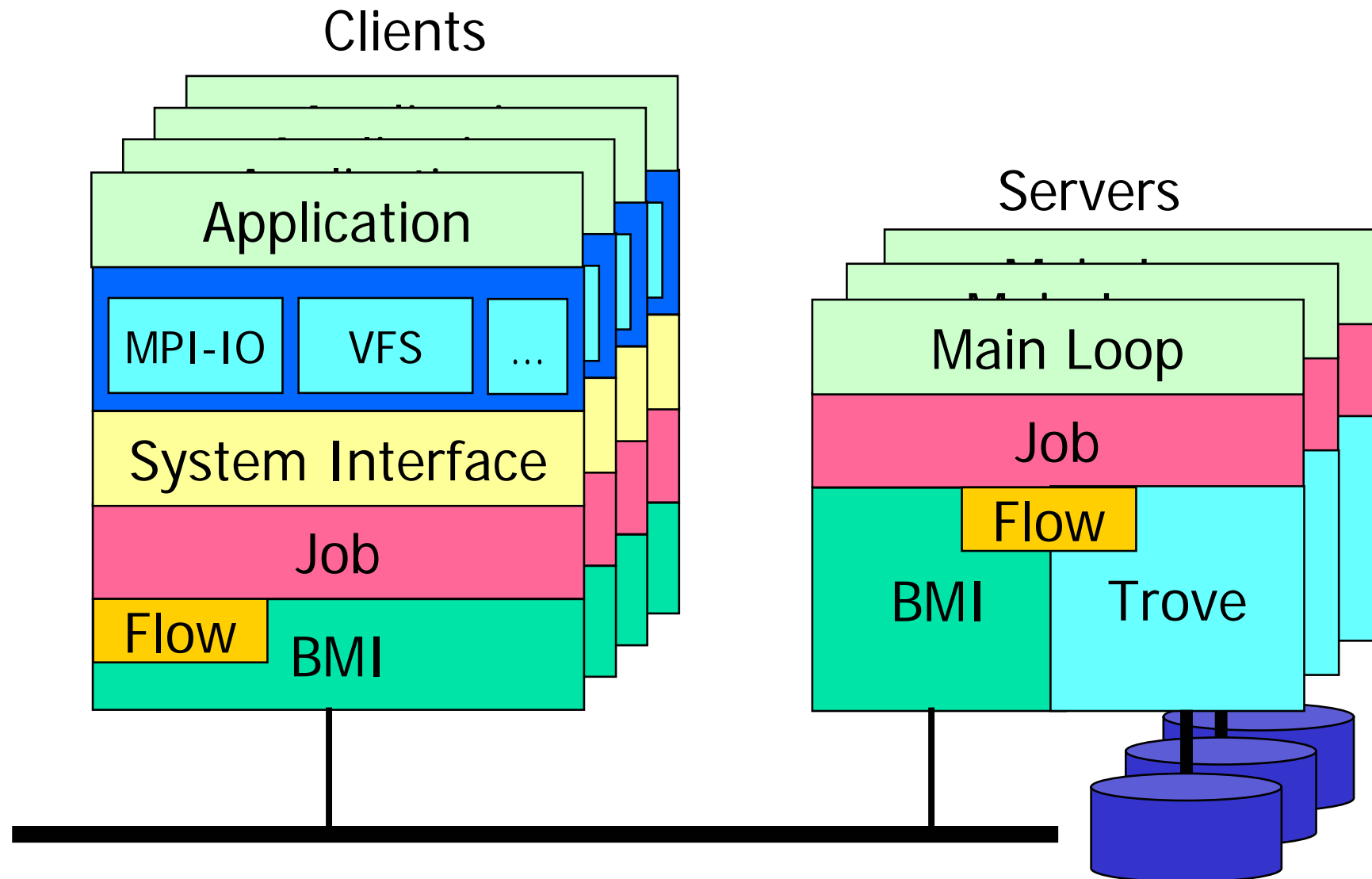
Environment

- MPICH2 for parallel programming
- PVFS2 as a parallel file system
- Jumpshot as visualization tool

An Example: MPI_File_write triggers PVFS2's Trove write



The PVFS2 Environment



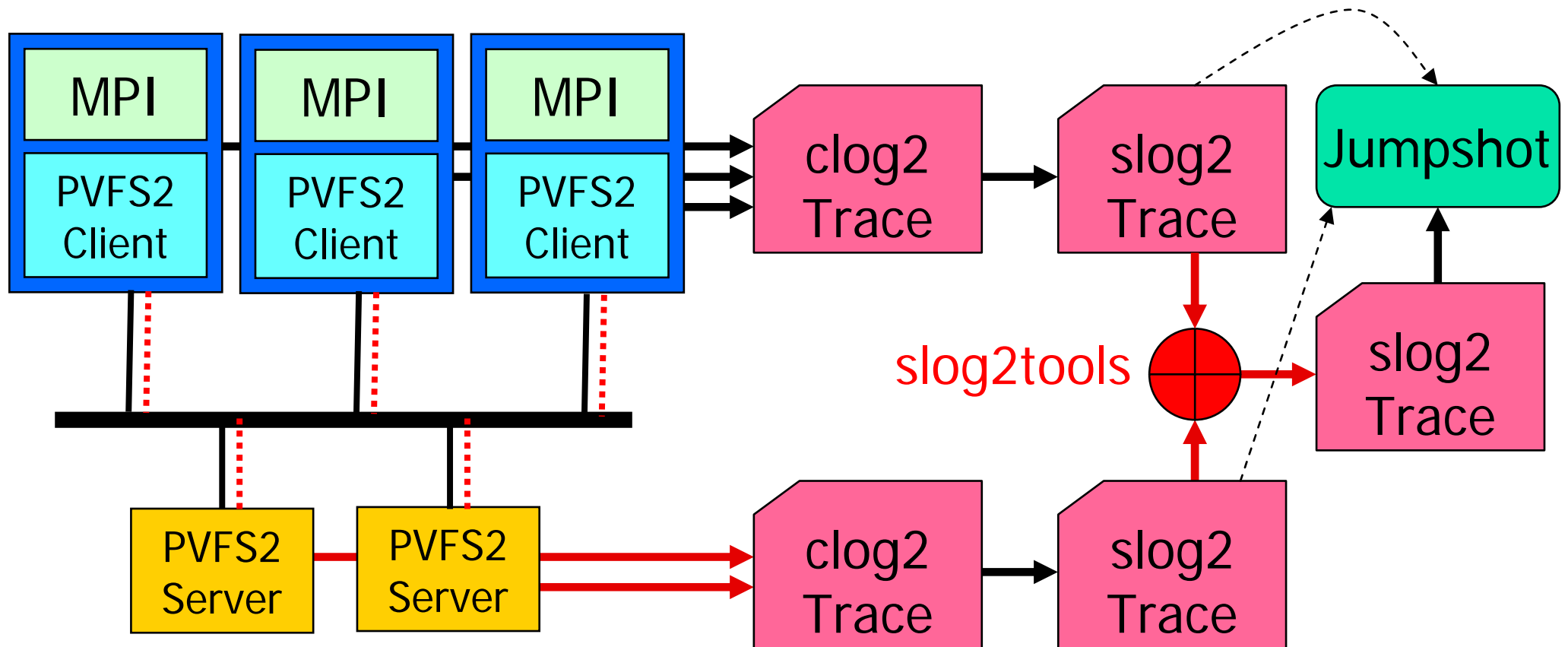


Concepts

1. Generate a trace for the servers
2. Merge the server trace with the client trace
3. Add arrows from MPI-IO calls to corresponding server activities
4. Visualize everything together in Jumpshot

Trace Generation

Use MPICH/MPE clog2/slog2 traces





Trace Generation Problems

- Servers run permanently
 - Start and stop tracing
- Client and server traces are not synchronized
 - Add special events to adjust them
- Server has no tracing facility
 - Start servers as MPI program and produce a trace as with an MPI application
- Client and server traces are separate
 - Merge them



Add Arrows to the Trace

- Arrow semantics
 - We want to show which MPI-IO call results in which PVFS server activities (data, metadata)
 - 1:n relation between program and I/O-system
- What do we need?
 - We need a unique ID for every MPI-IO call
- Concept
 - Trace activities at the client (create IDs)
 - Transfer IDs to the servers
 - Trace activities at the servers (store IDs in trace)
 - Connect activities with identical IDs at both levels



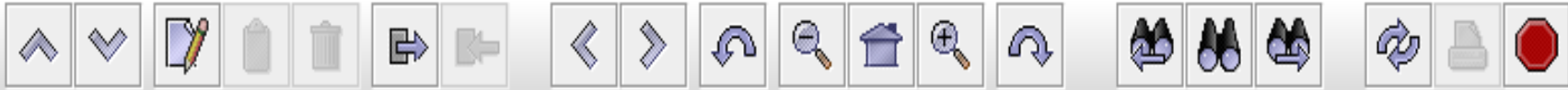
Visualization

- In theory: no problem at all
 - Jumpshot just takes the elements of an slog2 file and visualizes them – has no semantics integrated
- In practice
 - Too much information for the individual time lines (overlapping events because of asynchronous server behavior)
 - Solution: Distribute events to several time lines such that there is no overlap of events



First Results

- We can see all PVFS2 activities from the server modules Job, BMI, Trove, Flow
- We can relate Trove activities to the corresponding MPI-IO calls that triggered them
- Jumpshot visualizes all this in a nice way



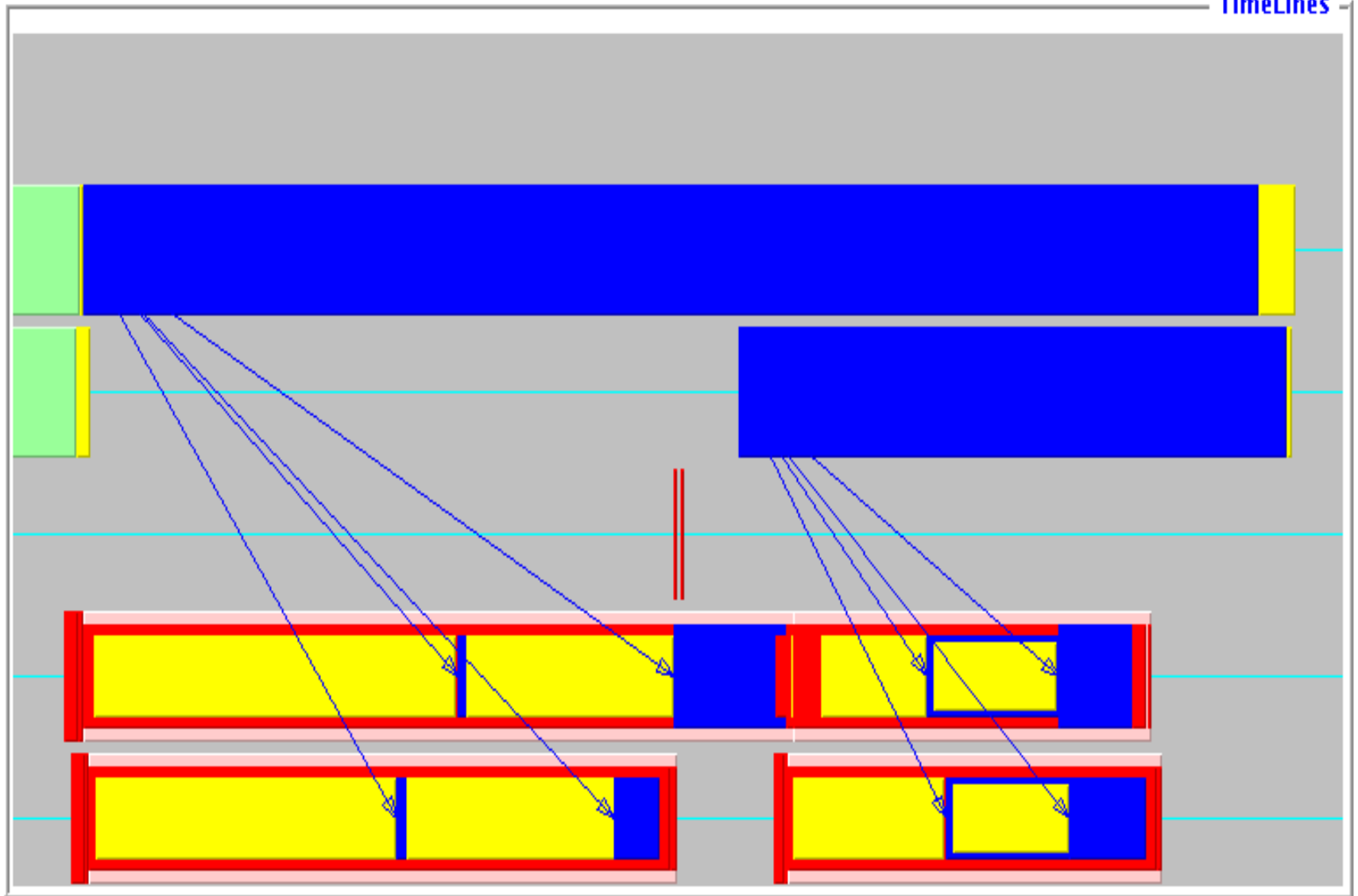
Lowest / Max. Depth
0 / 0

Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time	Time Per Pixel
11	29,2024400234	84,1354582045	84,1519966456	84,1628834592	94,5503048897	0,0000362289



FitMostLegends

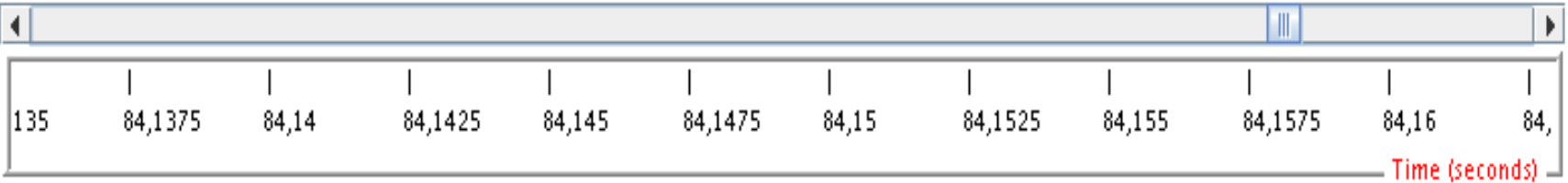
- SLOG-2
- client 1
- 1
- md server
- server 1
- server 2



Row
Row Count
6,0



@ LineID



Fit All Row



Open Problems

- We did not yet consider the standard problems
 - Size of traces
 - Correct time synchronization of events
- We will „adopt“ solutions provided by colleagues 😊



Next Steps

- We will (re-)run various programs and produce traces
 - Synthetic test programs
 - systematic analysis of all(!) MPI-IO calls
 - build knowledge about how the behavior looks like
 - Real programs
- A trace analysis will give more insight into sources of performance problems