A Distributed Performance Analysis Architecture for Clusters

Holger Brunst
Center for High Performance Computing (ZHR)
Dresden University of Technology

Email: brunst@zhr.tu-dresden.de
Tel.: +49 351 463-3 54 50

TECHNISCHE UNIVERSITÄT DRESDEN

Outline

- Research Background
- The Scalability Challenge
  - Overall Goals
  - Architecture Overview
  - First Results
- Summary

Research Background

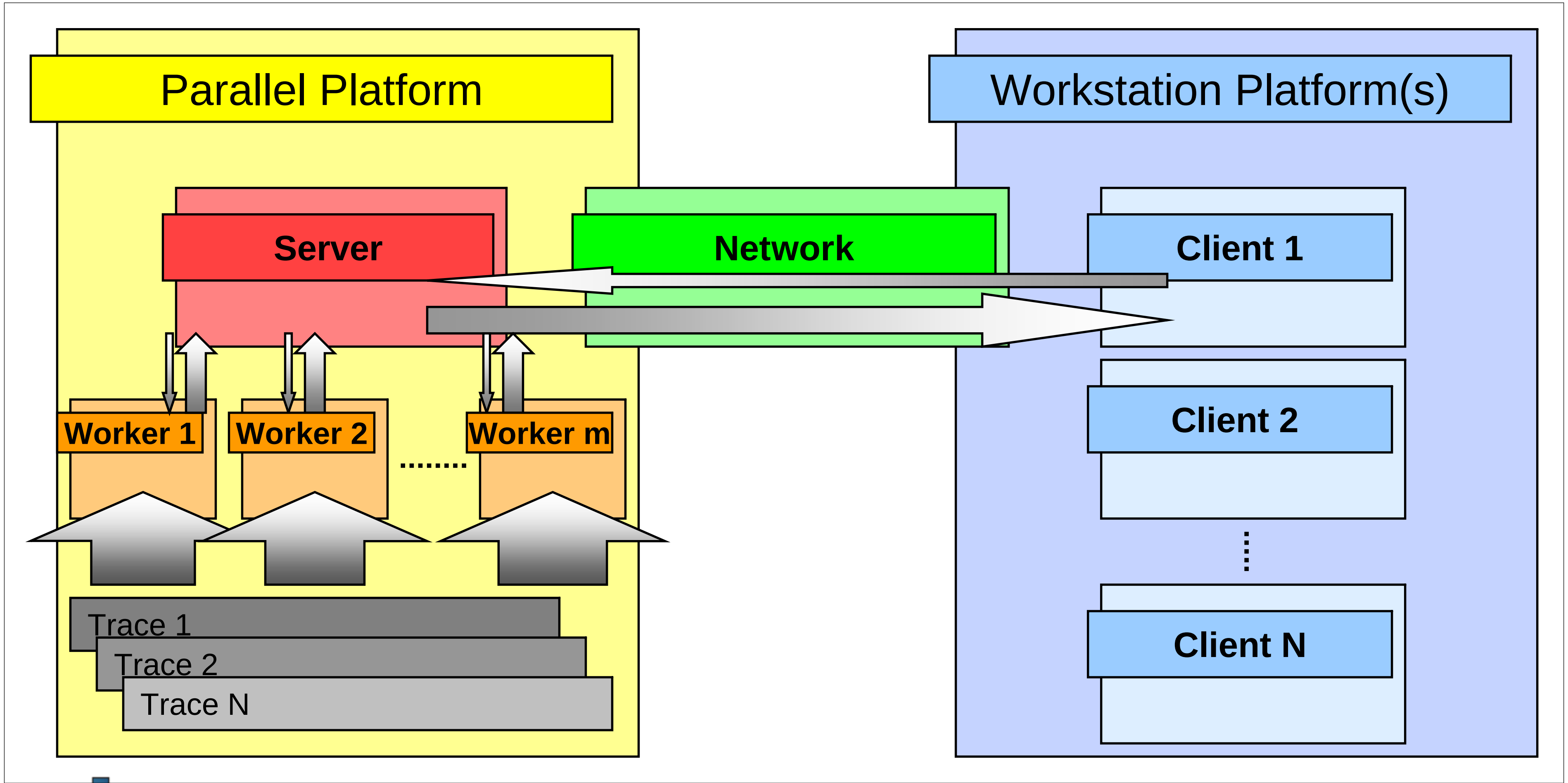
- Vampir: Visualization and Analysis of MPI Resources
- Development has started more than ten years ago at Research Center Jülich
- Today, developed at Dresden University of Technology
- From the beginning:
  - User driven
  - Main goal: getting insight into simulation runs with millions of events in a parallel environment
  - Support for message passing environments: 1994
- Commercialization by Pallas GmbH (now Intel™) since 1995
- Vampir 4.0 delivered in 3Q2003

Commitment to the community:
Vampir remains available across ALL commercial platforms

Overall Goals

- Keep performance data close to its origin
- Perform event analysis in parallel
- Support up to 10000 processors
- Achieve speedups in the order of 10 to 100
- Fast & easy to use performance analysis
- Access data from remote platform
- Client/Server approach
- Client should look like a typical event analysis tool

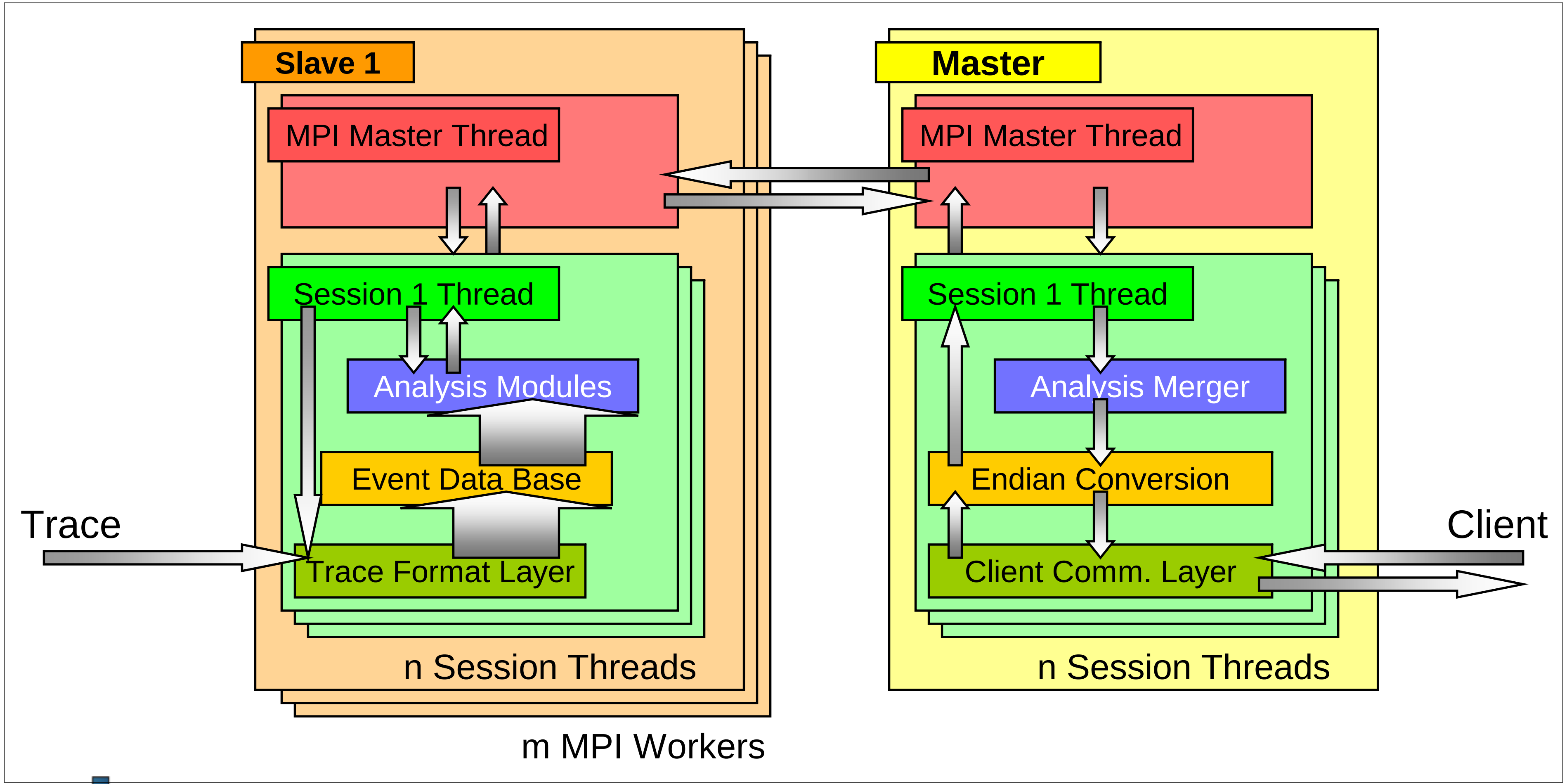
Architecture Overview



Analysis Server - Requirements

- Do *calculations in parallel*
- Support *distributed memory*
- Increase *scalability* for both long (regarding time) and wide (regarding number of processes) program traces
- Allow *preliminary cancellation* of requests, i.e. allow to cancel an analysis step when it takes to long
- *Limit the data* transferred to the client to a volume independent of the amount of actual trace data
- *Portability*
- *Extensibility (e. g. the different trace formats)*

Analysis Server - Overview



ZHR

TECHNISCHE UNIVERSITÄT DRESDEN

Analysis Server - Layout

- Provides analysis capabilities to external clients
- Supports multiple sessions i.e. multiple clients can connect and formulate requests
- Has a static number of m MPI processes
  (1 master and m-1 slaves)
- Has n session threads per slave
- Trace data is distributed among slaves
- Analysis is carried out concurrently on slave processes/threads
- Results are merged and exchanged by the master

Display Client

- Sequential program
- Connects to analysis server via a communication-layer (sockets)
- Communication is endian independent
- Does NOT operate on events
- Receives pre-calculated information from analysis server
- Nevertheless, post processing of information is allowed
- The volume of exchanged information depends on the display resolution and NOT on the number of events in the trace
- Displays interact independently with analysis server

Message Passing + Multi Threading

- Communication between workers with MPI
  - no scalability restrictions on clusters or MPPs
  - portability
- Multithreading on workers with pthreads
  - was needed to support multiple sessions
  - allows to abort certain requests or sessions before they are completed
- Multithreading on master with pthreads
  - each session is handled by a thread
  - decouple the input/output process

Data Serialization and Unification Layer

- Basic types:
  - char, int, double,
  - vector<char>, vector<int>, vector<double>
- Derived structures from basic types
- Context independent endian conversion
- Serialization of requests
- Needed for:
  - client/server communication
  - master/slave communication

Multi Trace Format Layer

- Internal API to facilitate the development of drivers for different event based trace formats
- Supported formats:
  - STF (Pallas)
  - VTF3 (TU Dresden)
  - TAU (University of Oregon)
- STF allows to read subparts of the same trace independently (scalability has been proven already)
- Using VTF3, non-relevant parts of a trace will be ignored by the appropriate worker

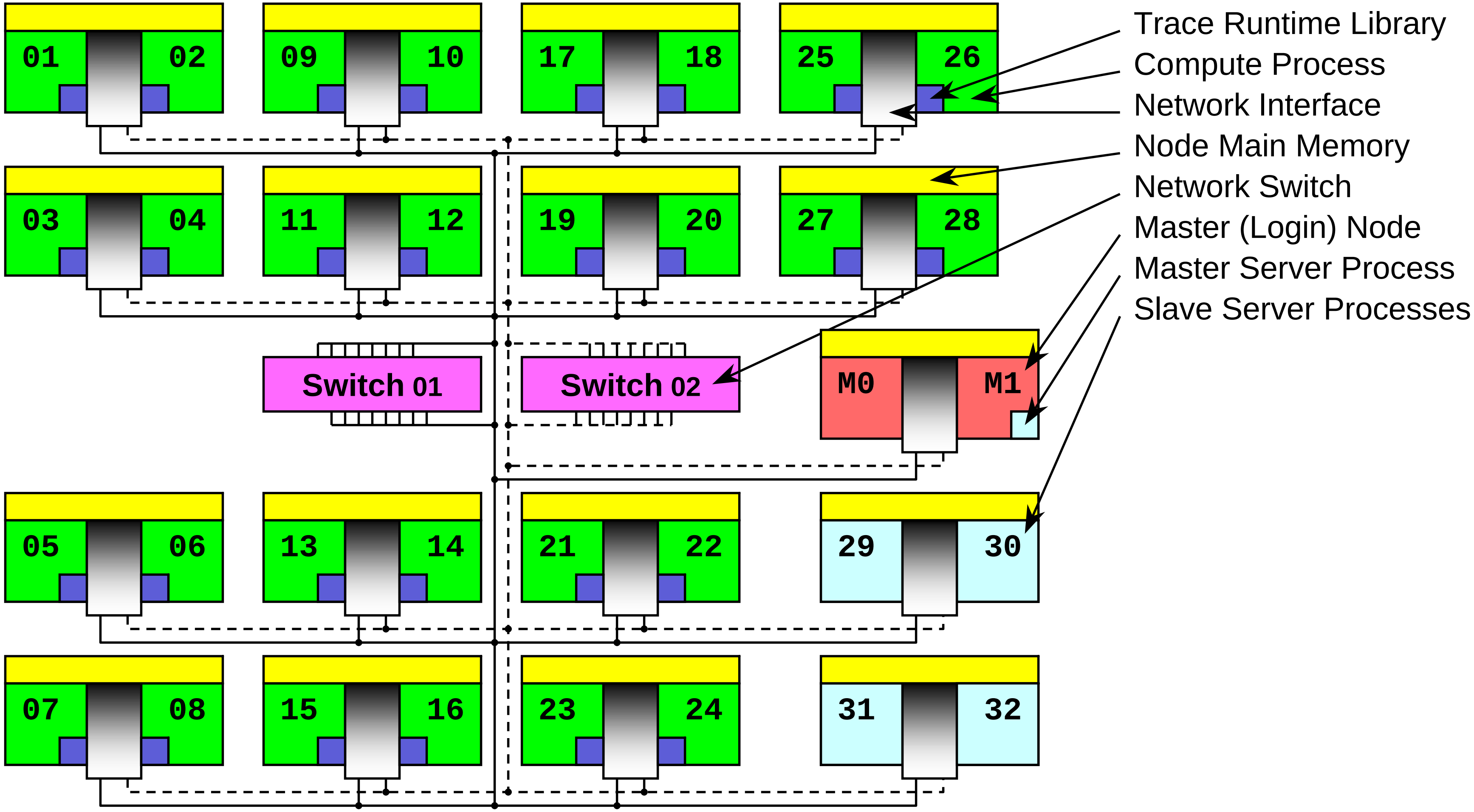
Event Analysis Process (Example)

1. Display client emits request (e.g. GetSummaryStatistics())
2. Analysis Server sends notification to client
3. Server forwards (MPI) request to slave processes
4. Slave processes hand over request to appropriate session threads
5. Session threads calculate statistics for their local event data subset
6. Results are handed back to MPI master process
7. Master merges results from slaves
8. Result is send to display client
9. Client receives reply
10. Client hands over reply to Summary Statistics display
11. Display does the drawing

Parallel Event Analysis

- Function analysis (enter / exit events)
  – two dimensional task distribution (time & rank)
  – result size is independent of the number of events
- Communication
  – event mapping problem needs to be solved first
  – similar distribution as enter / exit events
  – message summaries were introduced for timeline representation
- Performance Metrics
  – analog to function analysis

Runtime Environment Layout



System Setup

- Common data storage system for calculation & analysis process
- Security
  - Server runs in user space
  - Supports ssh tunneling
  - Firewall should shield the server
- Mode of operation
  - Interactive scheduling (batch does NOT suffice!)
  - Dedicated resources i.e. a few node should be reserved for analysis

First Results (1)

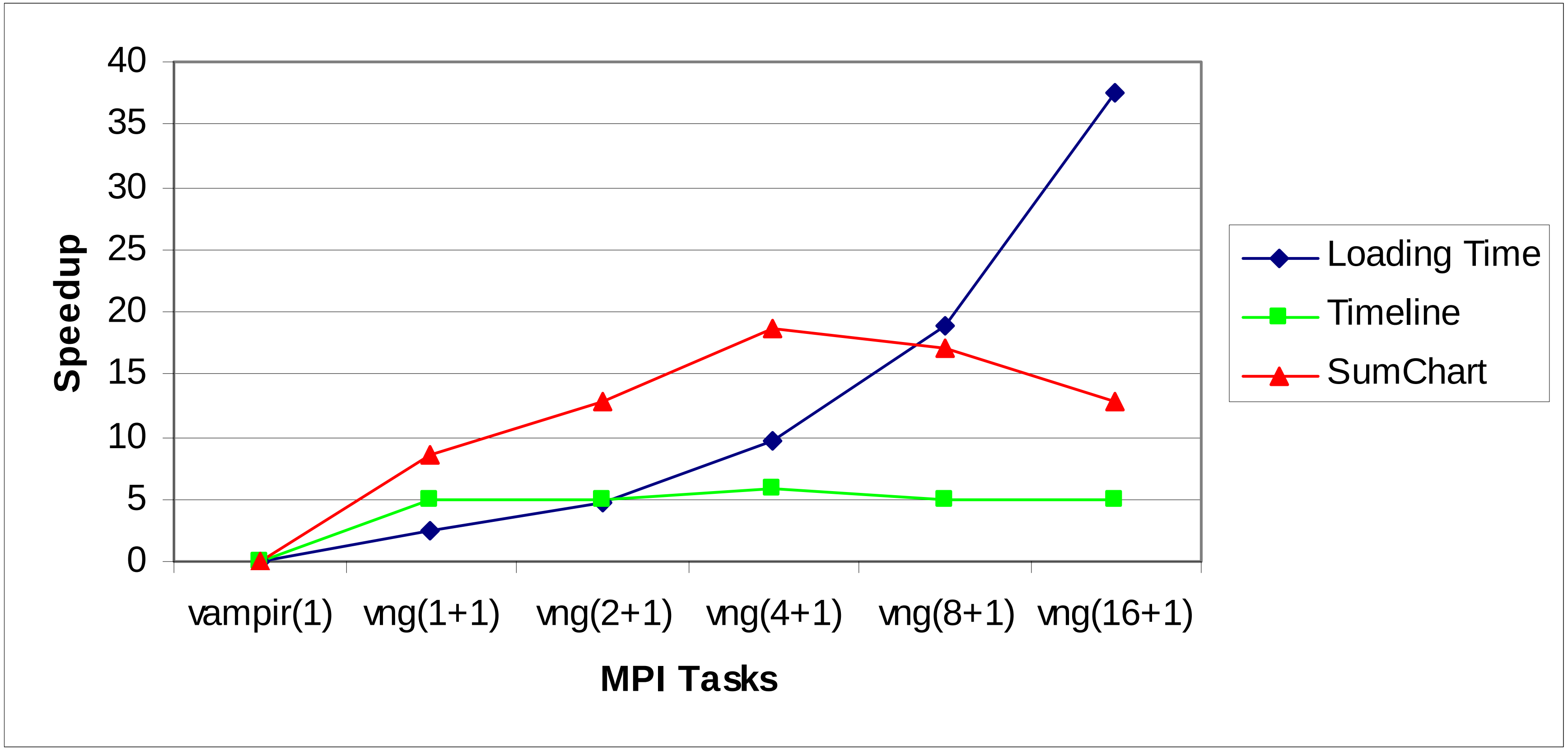
- Benchmarks
  - Time needed to fully load trace
  - Calculation of a timeline representation
  - Calculation of arbitrary profiles

- Example Code
  - sPPM benchmark (ASCI)
  - 327 megabytes, approximately 22 million events
  - still small

First Results (2)

- Platform A: **Linux Cluster**
  - Processors: 32
  - CPU: Intel Xeon, 2.80 GHz
  - Memory: 4GB per node
  - Scheduling: exclusive
- Platform B: **SGI Origin 3800**
  - Processors: 64
  - CPU: MIPS R12000, 400 MHz
  - Memory: 64GB total
  - Scheduling: exclusive

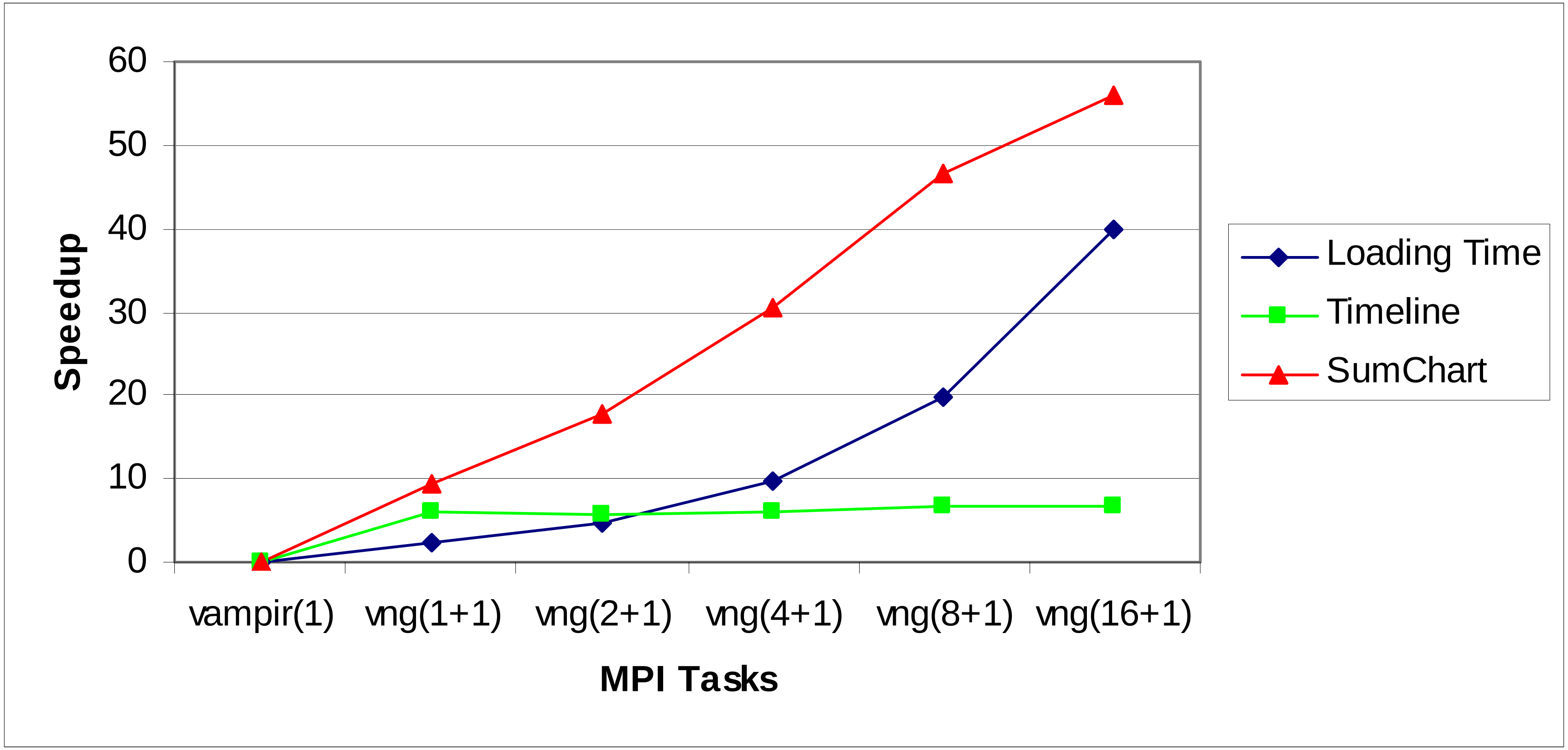
Speedups Platform A



Interpretation A

- Timings:
  - Load Op:
    41 s (vampir); 16.63 s (vng 1) - 1.09 s (vng 16)
  - Timeline:
    0.35 s (vampir); 0.07 s (vng 1) - 0.07 s (vng 16) !!!! (1)
  - Profile:
    2.05 s (vampir); 0.24 s (vng 1) - 0.16 (vng 16) !!!! (2)
- (1) No real speedup for Timeline because of O(log n) algorithm in Vampir and VNG
- (2) Calculation is too fast! We basically see network communication to client

Speedups Platform B



TECHNISCHE UNIVERSITÄT DRESDEN

Interpretation B

- Timings:
  - Load Op:
    208 s (vampir); 91.5 s (vng 1) - 5.20 s (vng 16)
  - Timeline:
    1.05 s (vampir); 0.17 s (vng 1) - 0.16 s (vng 16) !!!
  - Profile:
    7.86 s (vampir); 0.82 s (vng 1) - 0.14 (vng 16)
- No real speedup for Timeline because of O(log n) algorithm in Vampir and VNG

Summary

- Performance analysis on large systems is a “Must“
- Focus at our center:
  - Scalability
    - the methods of the past will not work for long
    - you need parallelism to work with large data
    - our approach seems to scale at least quite a while
  - Automatic performance analysis
  - I/O

It is a Challenge Anyway!