Logging Kernel Events on Clusters

Jürgen Reuter and Walter F. Tichy

Institut für Programmstrukturen
und Datenorganisation

Universität Karlsruhe, Germany

Motivation

- Understand & improve Linux based cluster OS:
  - Cooperative caching
  - Distributed parallel file systems
  - Checkpointing
  - Gang scheduling
  - …

Approach

- Track and analyse OS events on clusters
- Find bottlenecks
- Patch kernel
- Verify effect of patch

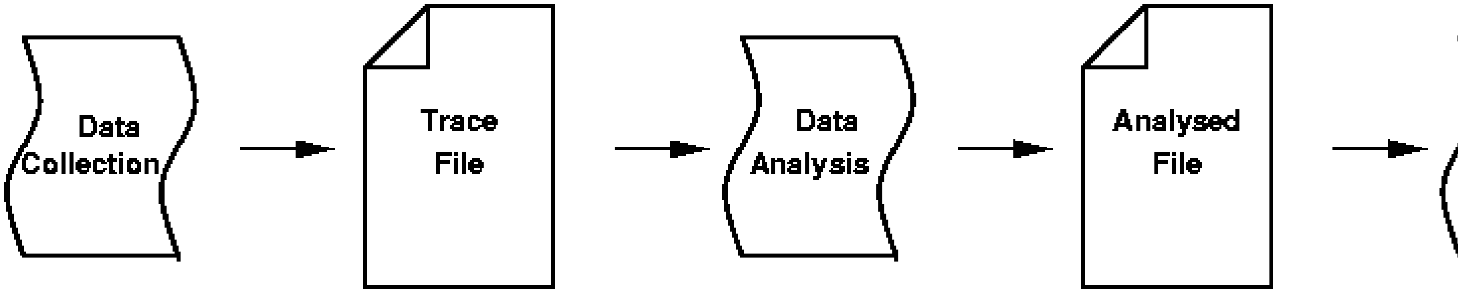
Example: Scheduling

- Log task switch events
- Find and analyse idle times
  - problem: waiting for de-scheduled process on remote node
- Co-schedule interactive processes via remote scheduling feature
- Verify effect of remote scheduling feature

→ provide tools for OS-level logging & analysis

Tools Overview

- **Data collection:** instrumented kernel code
- **Data analysis:** offline filtering
- **Data display:** graphical tool



Data Collection

- Mechanisms
  - Provide fast, non-blocking kernel event logging function for
    - Task switch events
    - Receive packet events
    - Enter/leave handler events
    - …
  - Instrument kernel
    - Manually insert call to logging function

Data Collection

- Steps
  - – Log event type & time in kernel
  - – Transfer data kernel → user level
    - Producer – consumer paradigm
  - – Save log data to disk in user level

Data Collection – Scalability

- **Cluster level:**
  - Logging performed locally per node
  - Data saved to local disk

→ Scalability is not an issue at cluster level.

Data Collection – Scalability

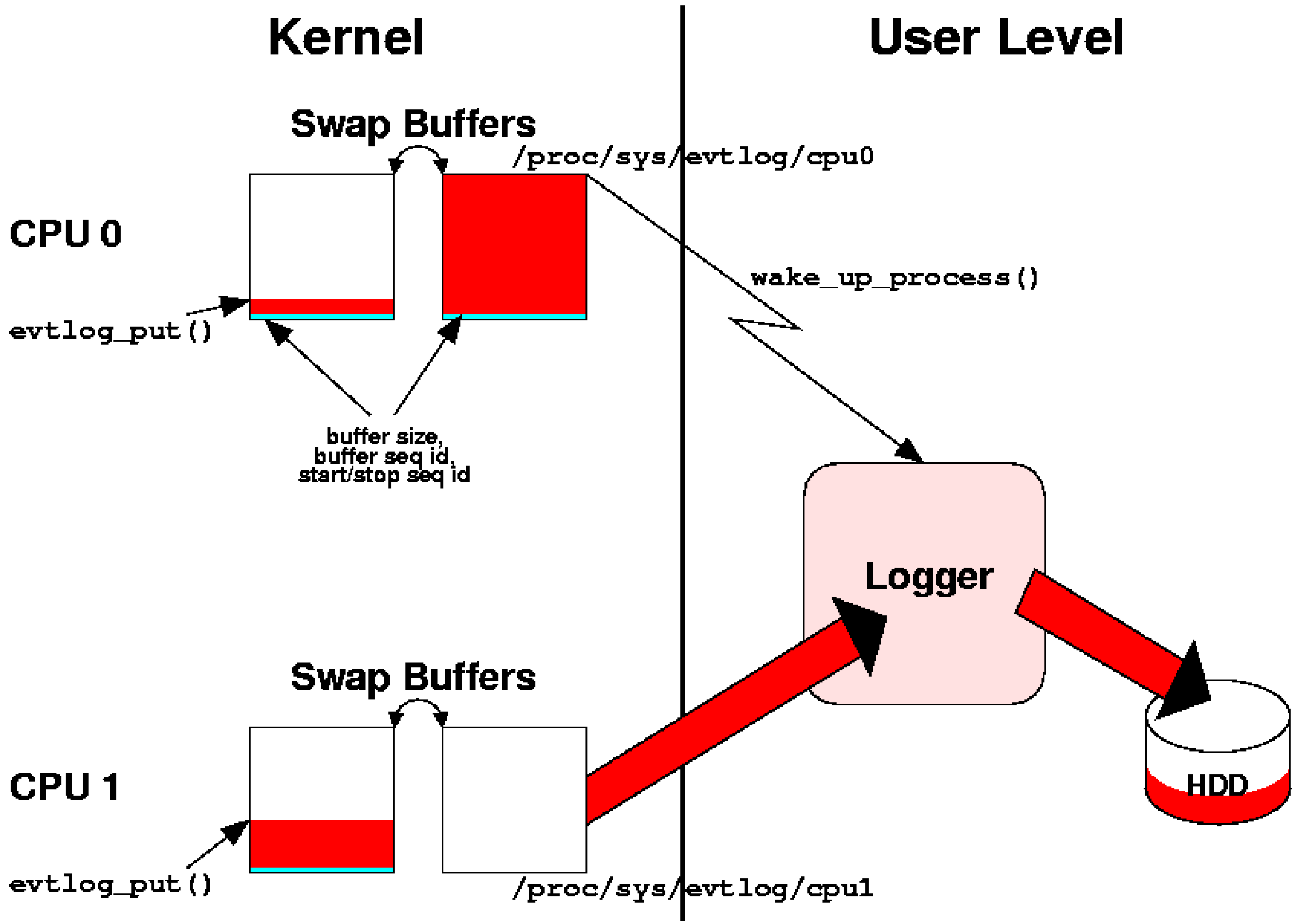
- **Node level (SMP):**
  - Bottleneck log buffer
    - Use per CPU log buffers

    → enables concurrent logging on SMP

    → no SMP lock required
  - Bottleneck disk access
    - Co-schedule all user level logger processes

    → hides effect for cluster as a whole

Data Collection – Architecture



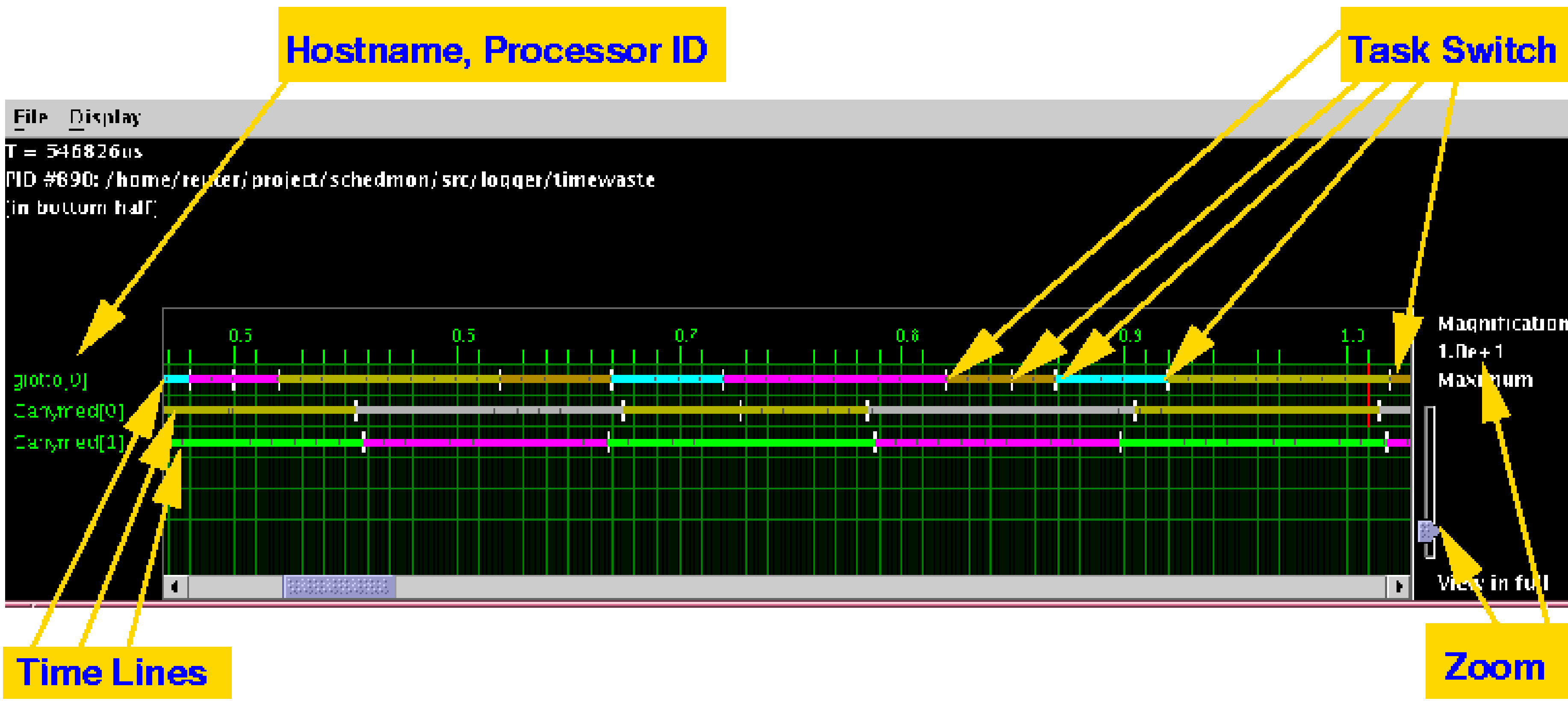
Data Collection – Costs

- User level process
  - Log task switch events → record costs
- Kernel code
  - Logging function: short & quick
  - Main costs: time measurement

Data Collection – Details

- Locking buffer access
  - Not an issue (non-preemptive kernel)
  - Exception: interrupts
- Cluster-wide synchronization
  - Start/stop via broadcast

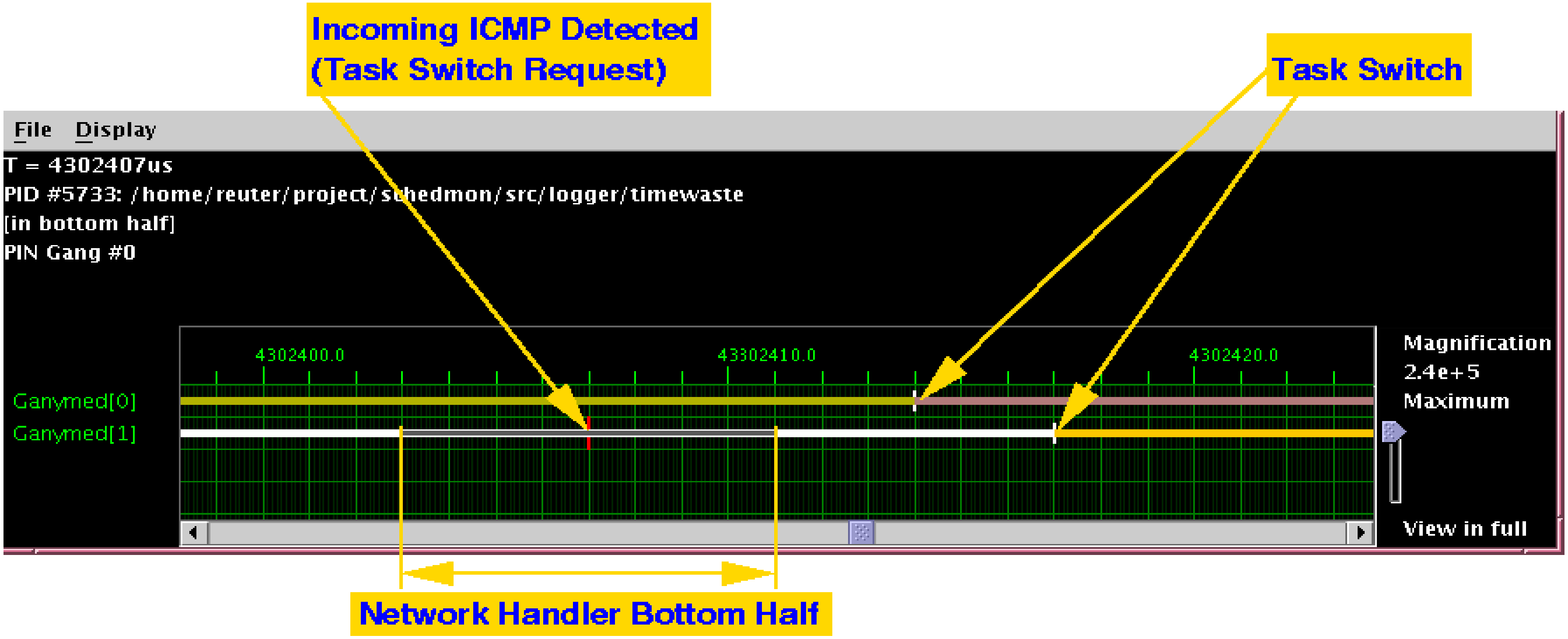
Data Analysis and Display



Example: Scheduling

- Remote schedule feature via ICMP
- Short control packets
- Broadcast/multicast applicable
- Short response time
  - Early handling in kernel
  - All in kernel
- ICMP dispatch over function table $\rightarrow$ no slow down for other traffic

Remote Schedule Feature



Conclusions

- Provided set of tools for OS level Performance analysis:
  - Kernel logging
  - Analysis and graphical display
- Proved usefulness by remote scheduling implementation

Future Work

- Verify scalability (fat SMPs, many nodes)
- Need filters for huge amount of data
- Enhance synchronization
- Better support relation between kernel and user level events