

A Performance Monitor based on Virtual Global Time for Clusters of PCs

Michela Taufer

Scripps Institute
& UCSD Dept. of CS
San Diego, USA

Thomas Stricker

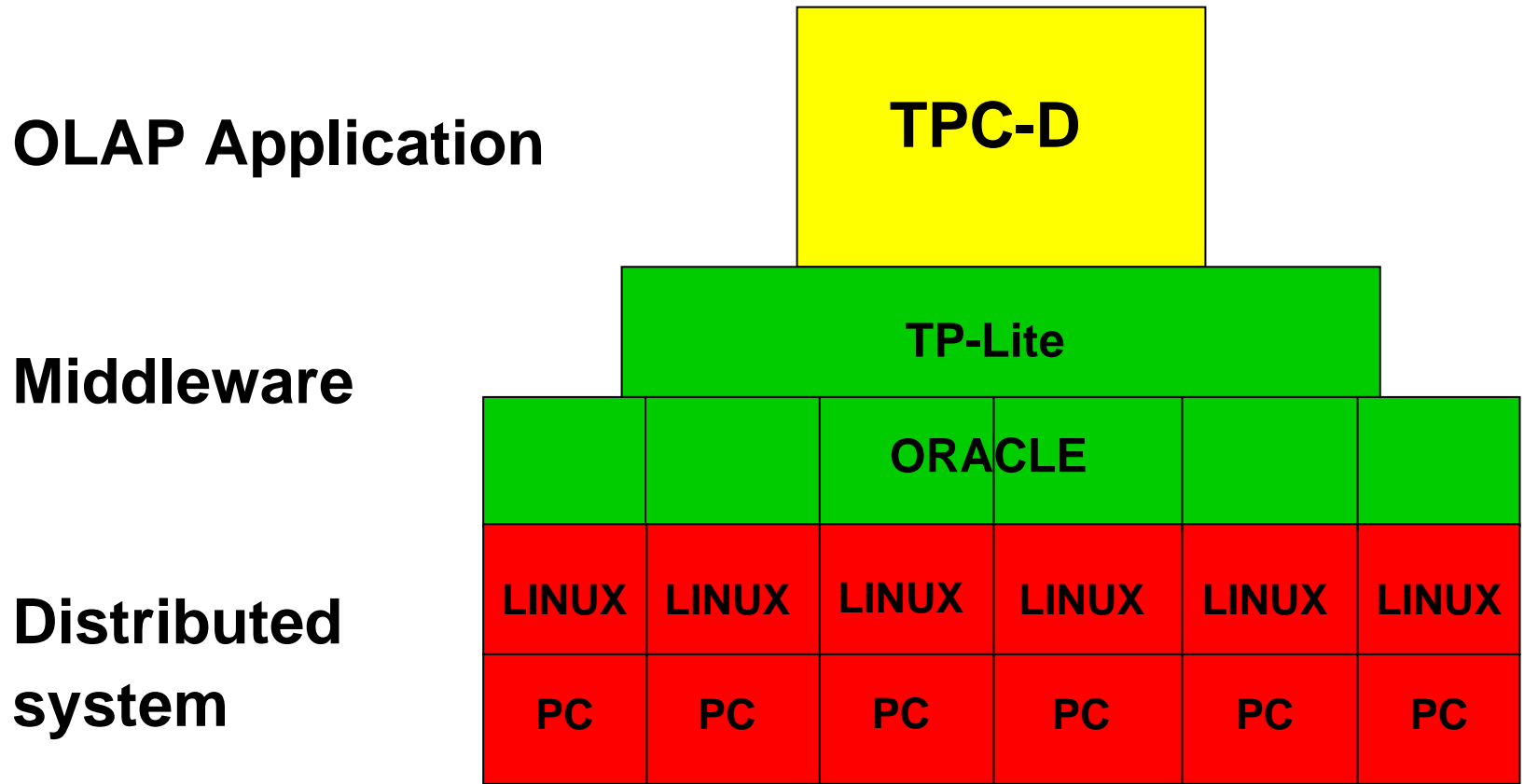
Lab. for Computer Systems
Swiss Institute of Technology
ETH Zürich, Switzerland

Cluster 2003, 12/2/2003
Hong Kong, SAR, China

Slides of this talk: www.cs.inf.ethz.ch/CoPs/talks/1223.pdf



Work motivated by a study of distributed DB on cluster of commodity PCs

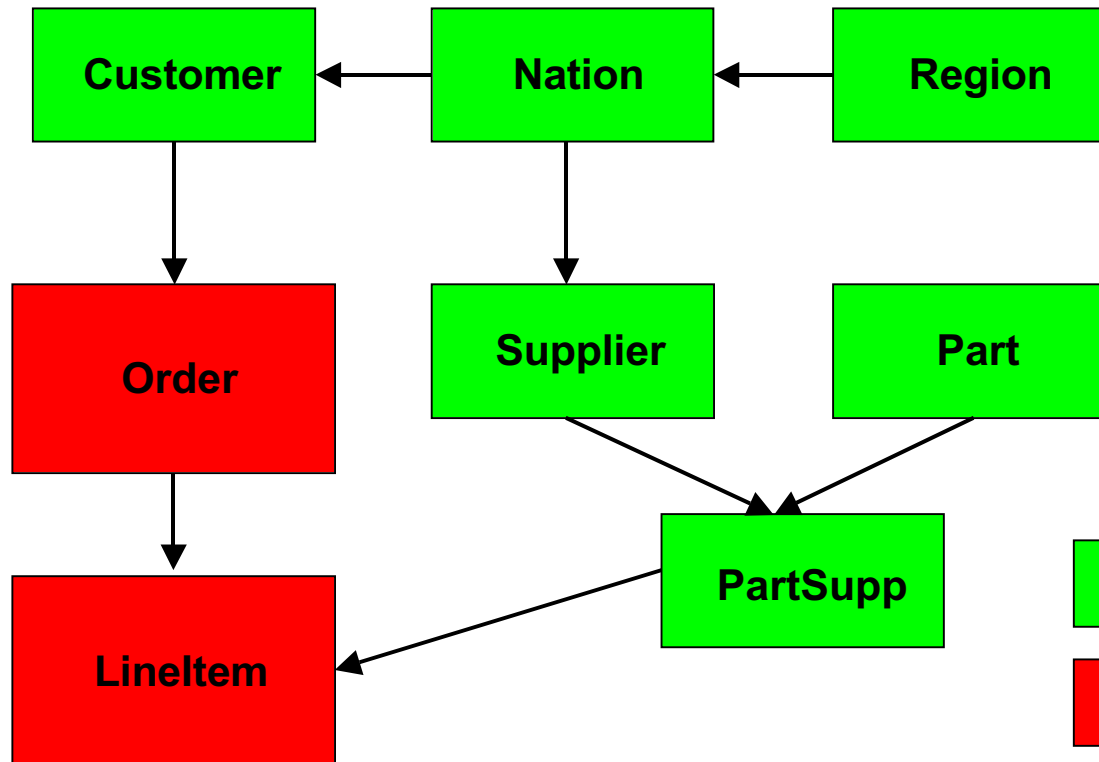


3

2

TPC-D benchmark for OLAP (data-mining)

TPC-D data model



Database size:
10 GByte

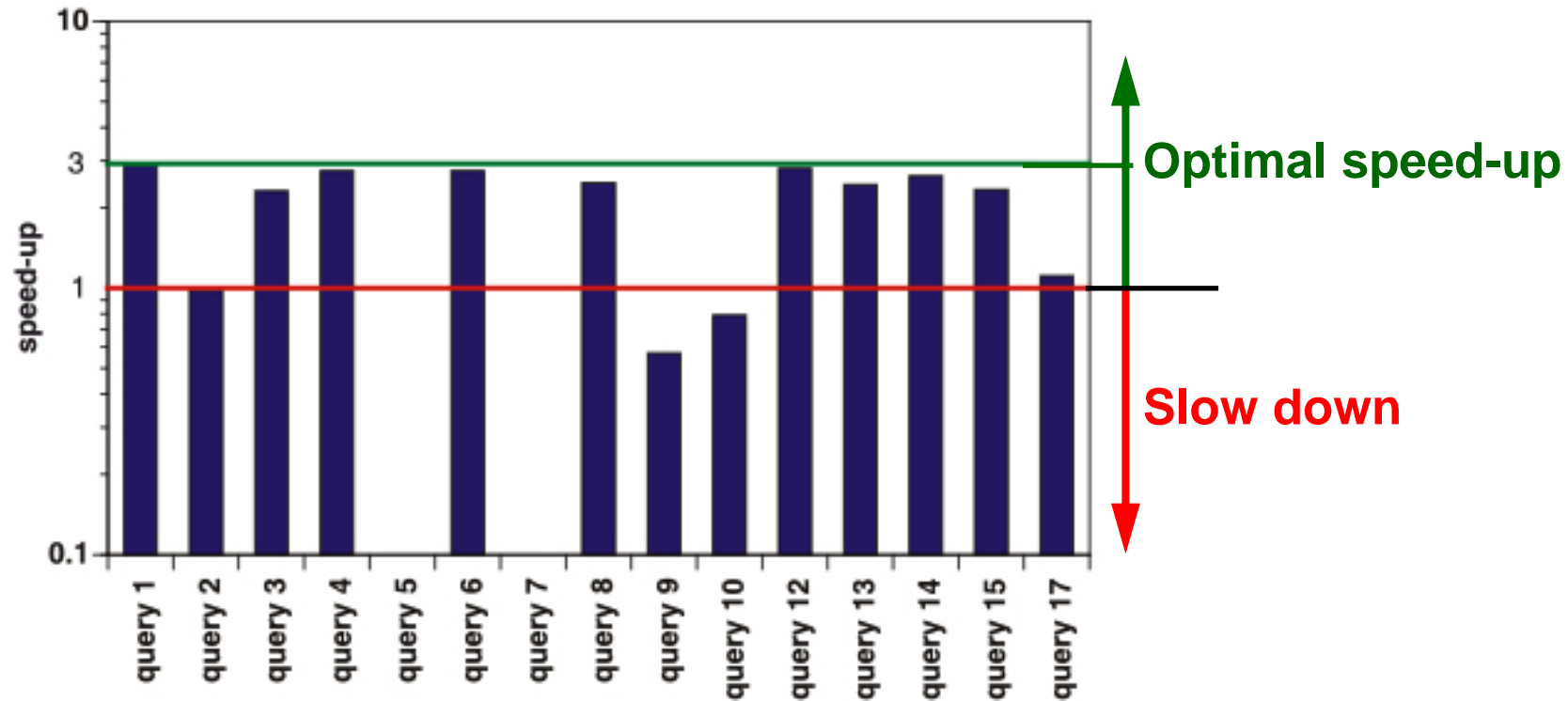


Fully replicated data



Disjointedly distributed data

Results in Performance and Scalability



We look into the [resource usage](#) of the TPC-D benchmark to explain the scalability picture

Outline

- **Motivation**
- Keeping a **performance monitor** in distributed systems with middleware packages
- Complementing **middleware** with **inverted middleware**
- Issues addressed and problems solved in the inverted middleware performance monitoring approach
 - application **performance model** based on **execution time** and machine **resource usage**
 - keeping track of **computation phases** in the application using a notion of **global virtual time**
 - dealing with **monitoring intrusion** by **dropping samples** (UDP/IP)
- **Experimental results - see our tool at work**
- **Conclusions**

Using Middleware in Clusters

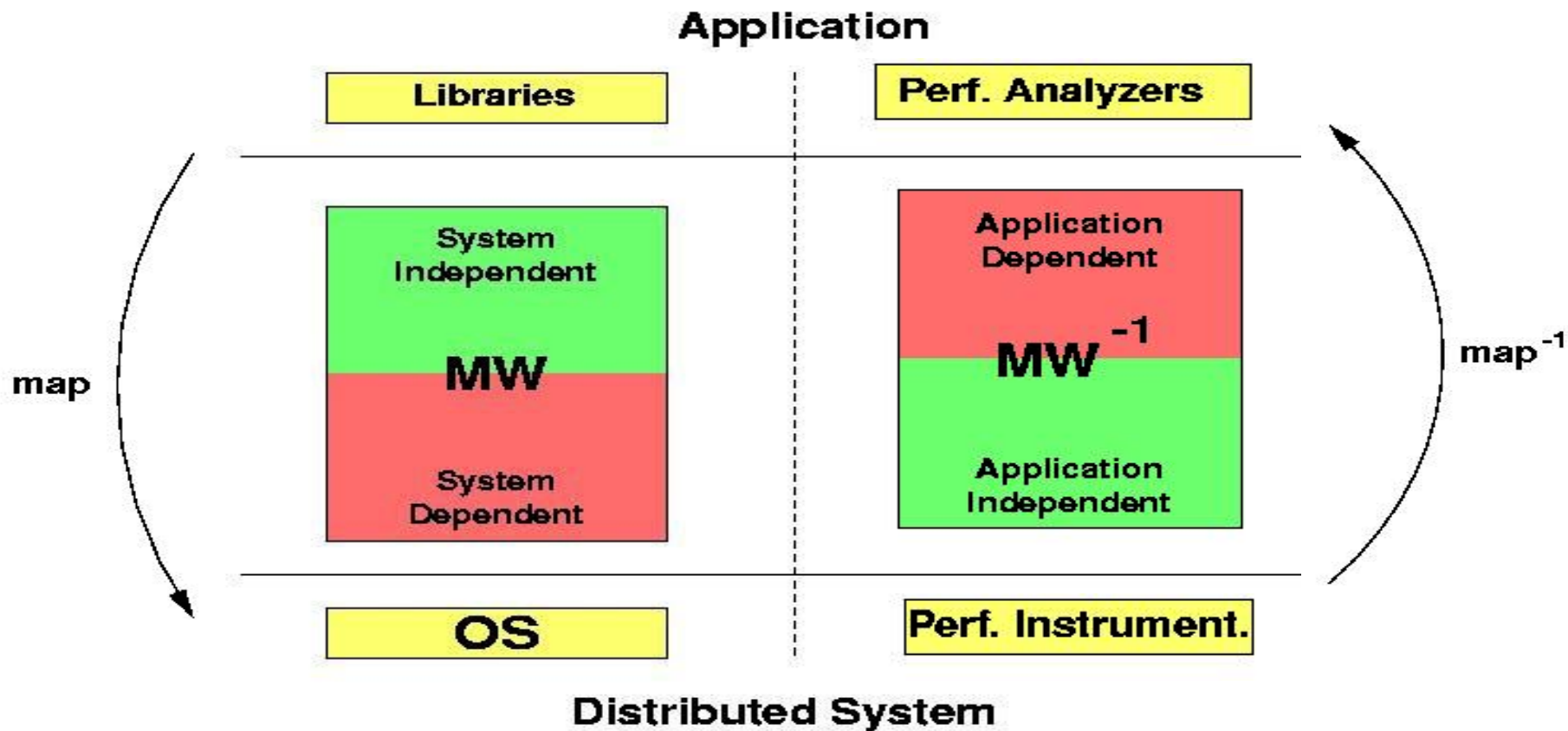
Benefits of middleware packages (DBMS in this case):

- provide higher level of abstraction for programmer
- address the problem of distribution to nodes
- hide system dependencies

Drawbacks of using middleware packages:

- make instrumentation for performance analysis related to distribution of computation in cluster difficult
- obstruct the detection of performance bottlenecks and architectural problems in the distributed system

Our Approach: Complementing middleware (MW) with inverted middleware (MW⁻¹)



Inverted Middleware as a Performance Monitoring Tool

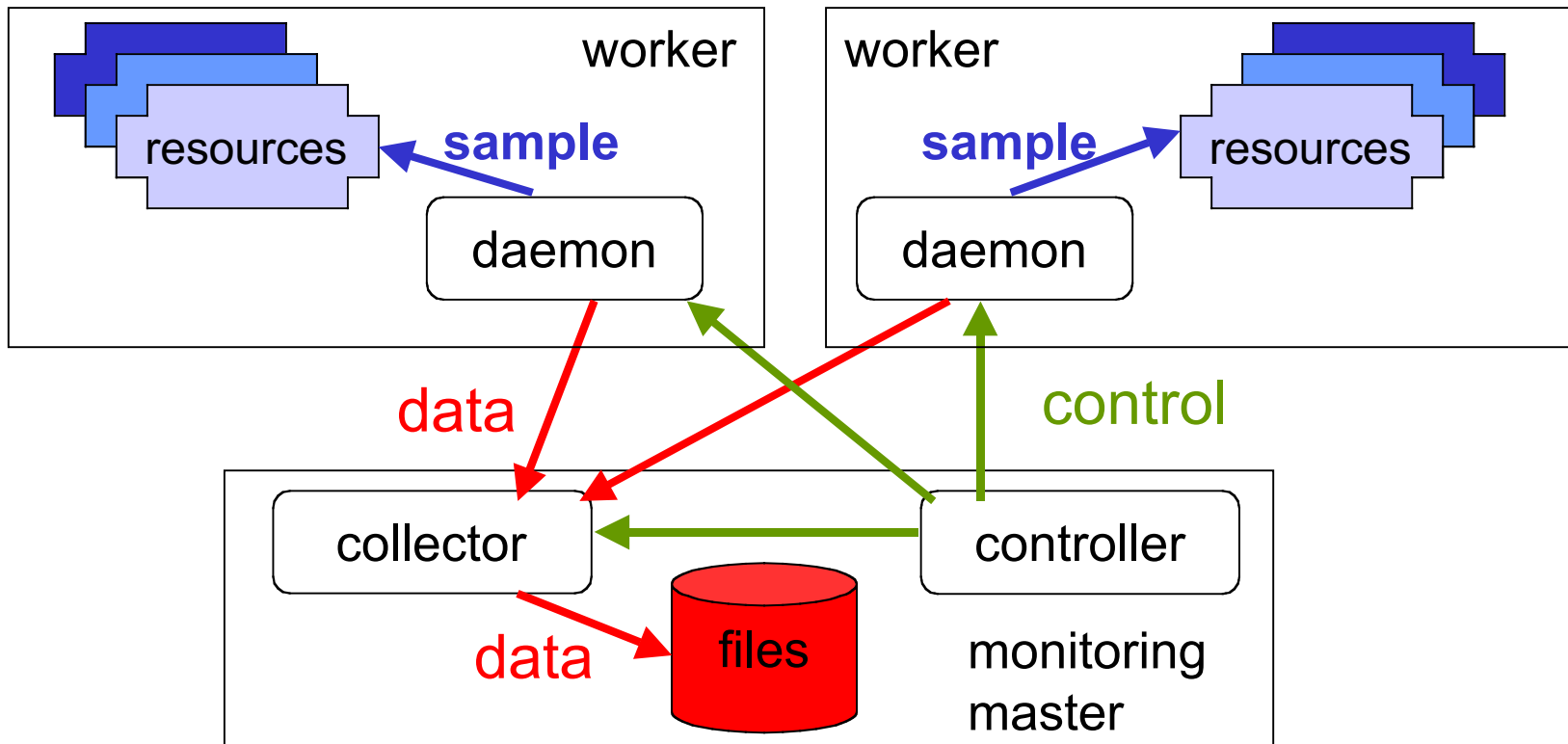
The relation between **middleware** and **inverted middleware** as a performance monitoring tool is similar to the relation between **compiler** and **debugger**.

Inverted middleware comprises:

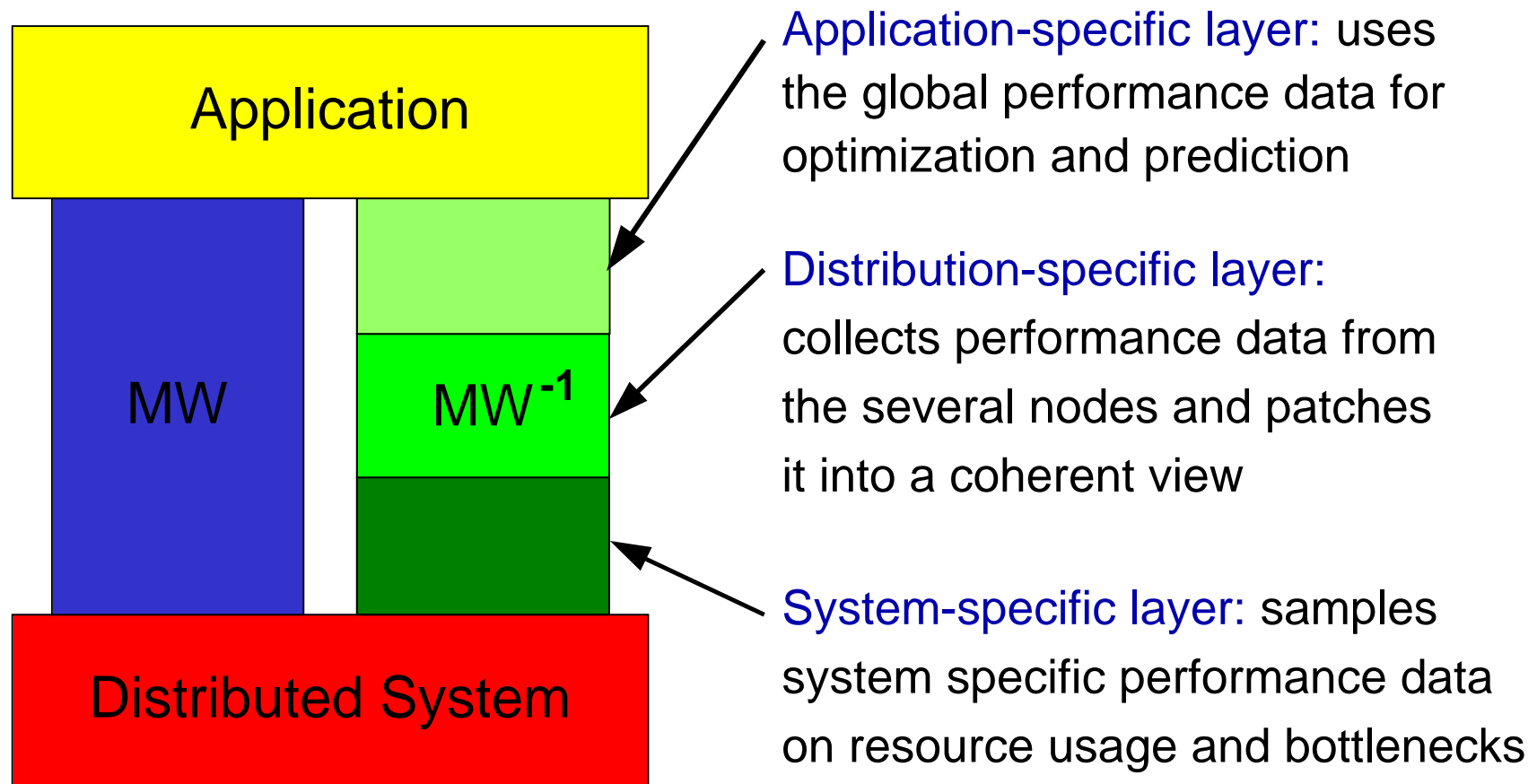
- software instrumentation at the OS level to gather data at the nodes of a cluster.
- infrastructure for the collection of performance data.
- analytical model for reconstructing a global picture of application performance.

Architecture of *Inverted Middleware* as Cluster Monitoring Tool

Our monitoring tool has a master-worker setting



Internal Structure of Inverted Middleware



Performance Data Collected

Packet identification:

- Worker_id, packet_id (counter)

CPU performance data:

- Total number of **user instructions** over the interval of time t
- Total number of **system instructions** over the interval of time t

Disks performance data:

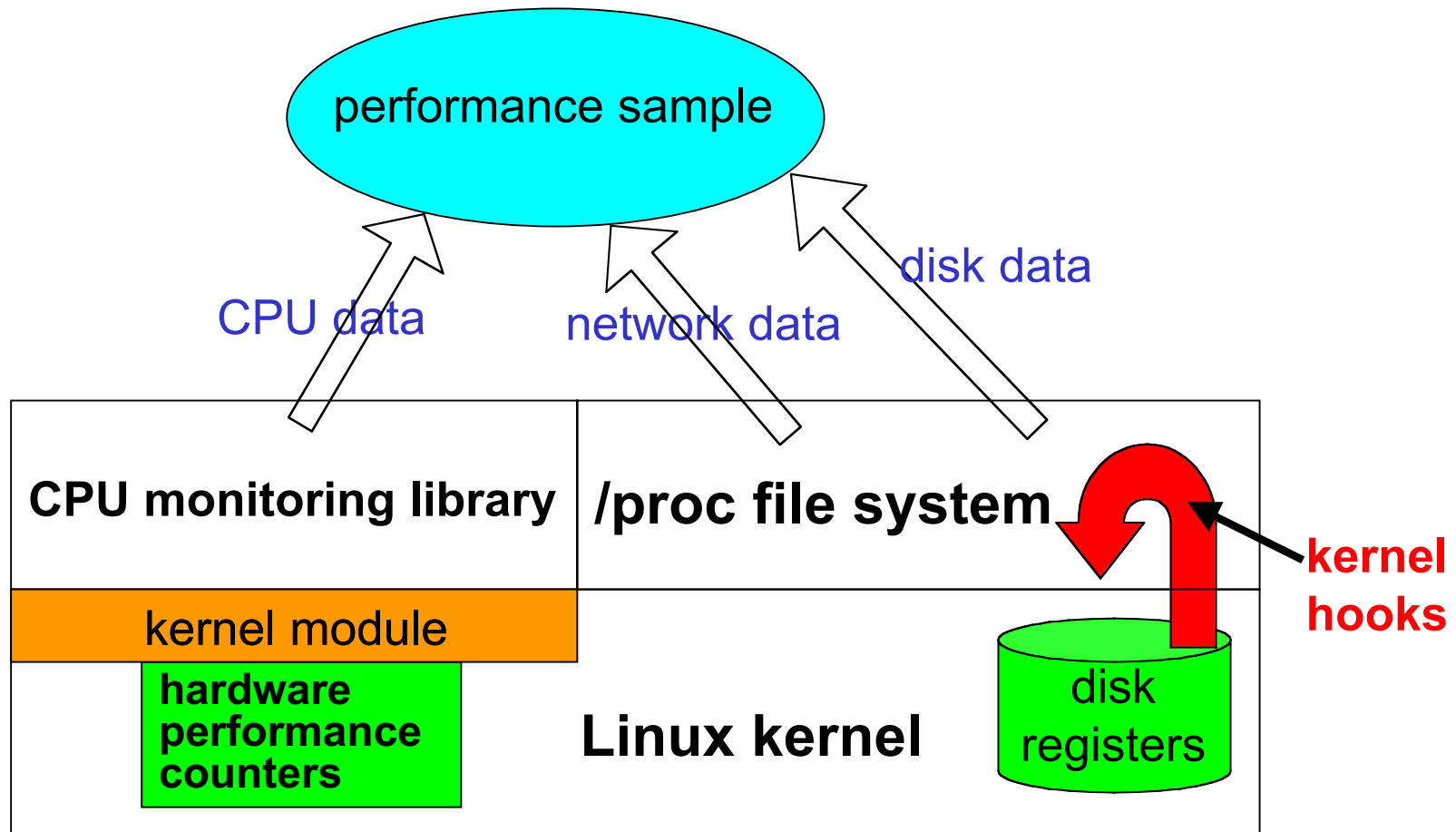
- Total number of **bytes read and written per disks** (we have three disks) over the time interval t
- Total number of **bytes accessed sequentially/bytes accessed not sequentially per disks** over the time interval t

Network performance data:

- Number of **packets/bytes sent and received on the several network connections** over the interval of time t

System-Specific Layer

(sampling local performance data)



System-Specific Layer

Sampling mechanism

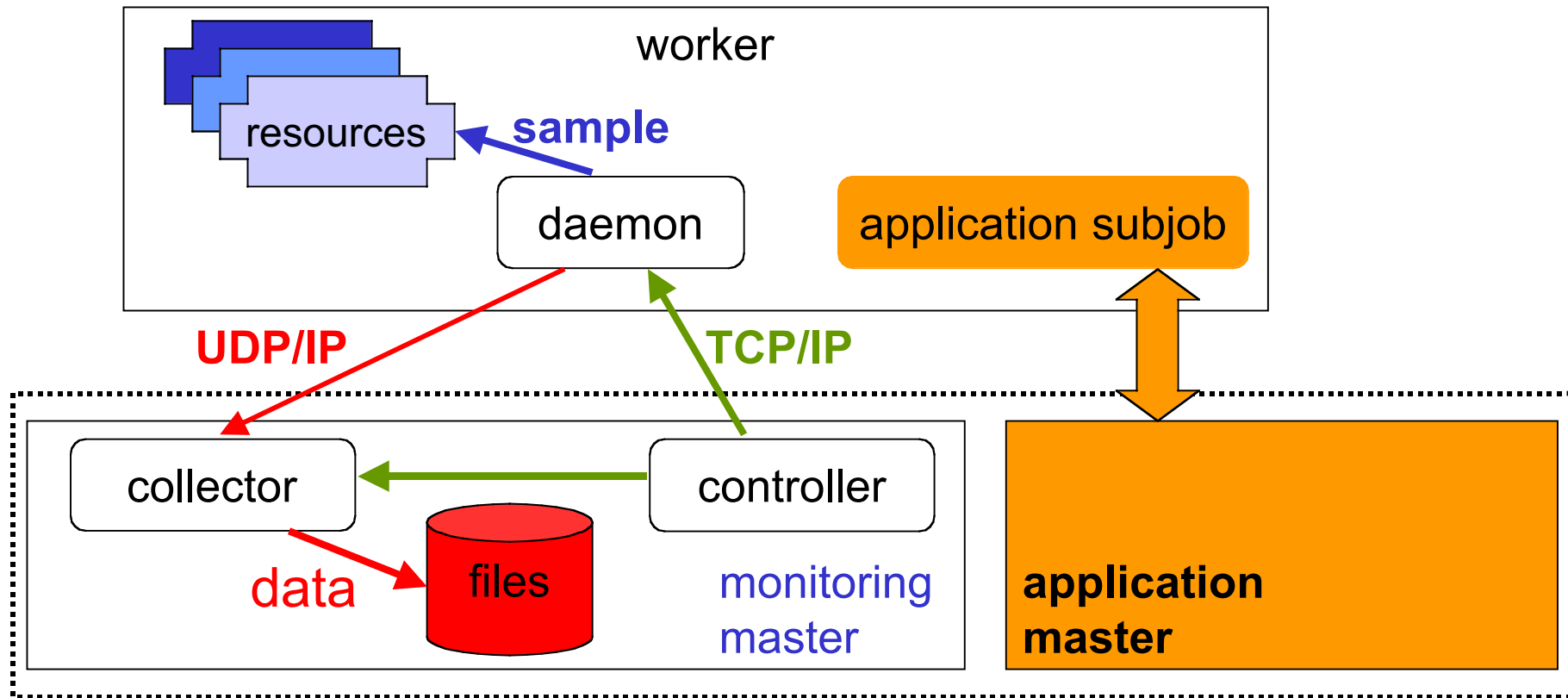
- Dynamic sample of resource information (e.g. floating point operations, amount of traffic over the network or to the disks) at regular intervals

Sampling performance data

- Outside the kernel as [daemon processes](#)
- Performance hooks into extended LINUX */proc* file system and hardware performance counters provided by the processors

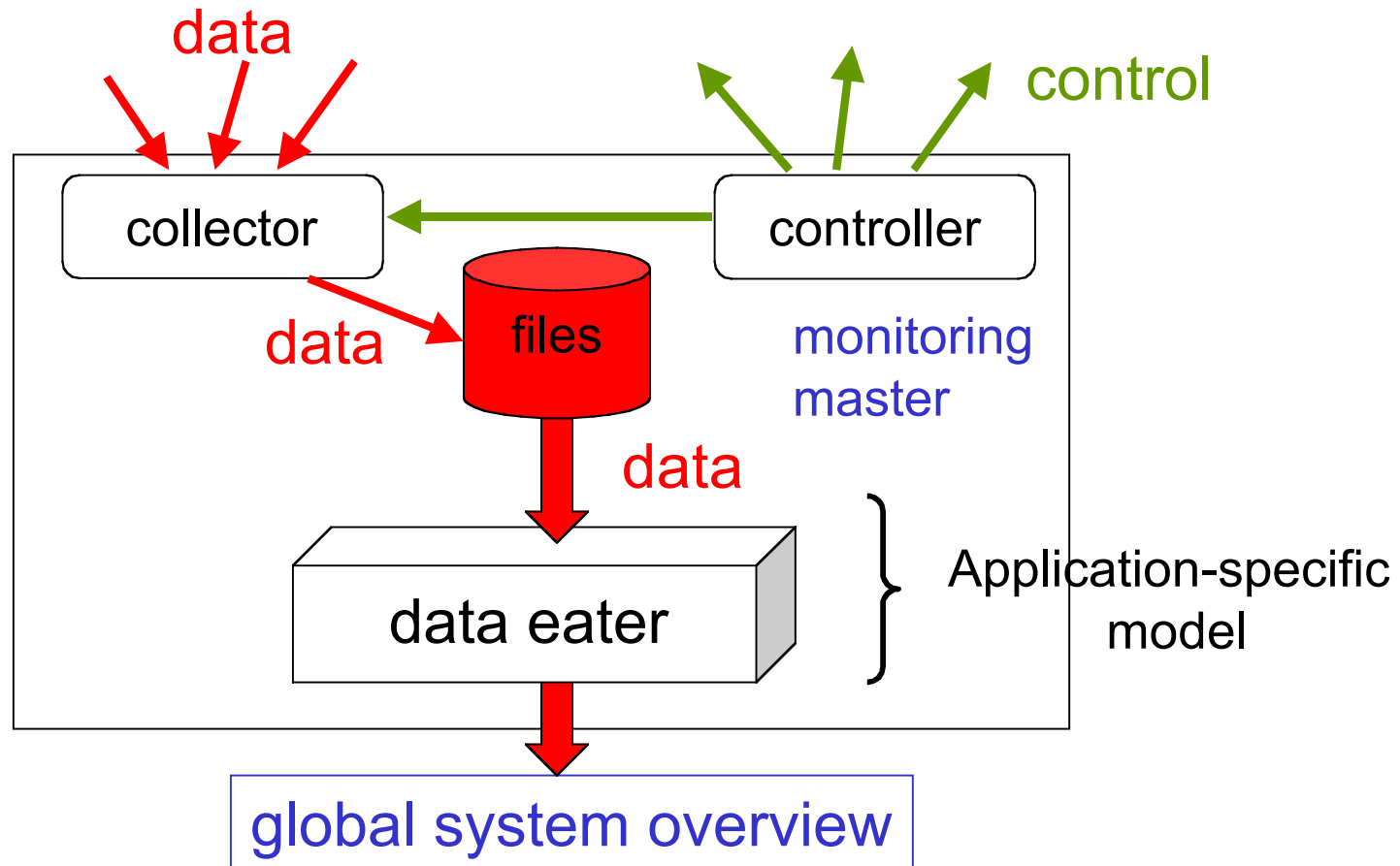
No re-engineering of application code or middleware

Distribution-Specific Layer



Monitoring and application masters can reside on different PCs

Application-Specific Layer



Application-Specific Layer

Performance model of the application:

- translates the **elementary knowledge** about the resource usage **into high level answers** to performance questions and bottlenecks suitable for **suggesting optimizations** to the user
- is a simple **set of formulas** which allows the calculation of the **individual execution times** due to the usage of each machine resource
- needs some **calibration and validation**

Our Simple Performance Model

Different approach to system performance:

Application user: Total execution time	System engineer: Usage of system resources
---	---

We claim:

- Simple relationship between the machine resources required by the application and its execution time
- Total execution time can be decomposed into parts
- Each part is largely determined by the usage of one single critical machine resource

Execution time is linked to **resource usage!**

Metrics in Performance Model

Everything is translated into execution time:

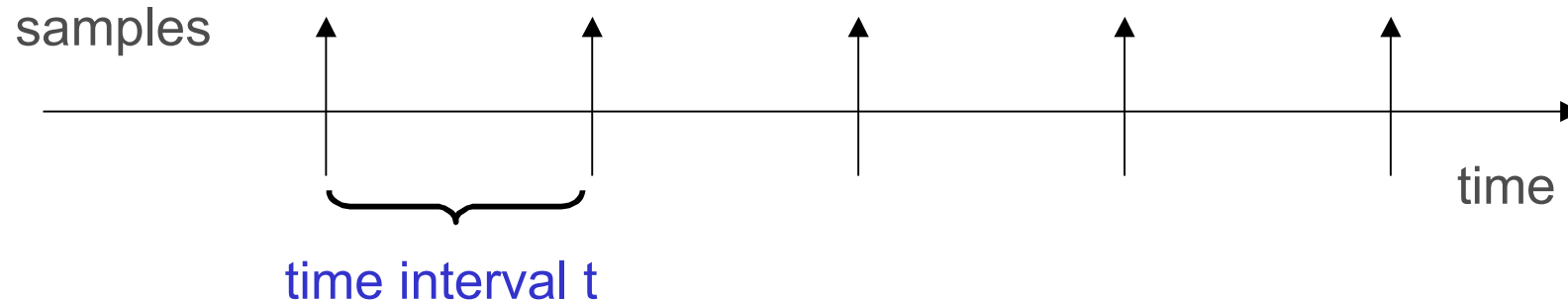
- CPU: Number of FIOp * Processing Rate (FIOp/s)
- Disk: Number of Bytes read/written * Data Rate
- Disk: Number of Random Access * Average Access Time
- Network: Number of Bytes transferred * Bandwidth
- Network: Number of Messages sent/rcvd * Latency/Overhead

All metrics are transmitted from node to master:

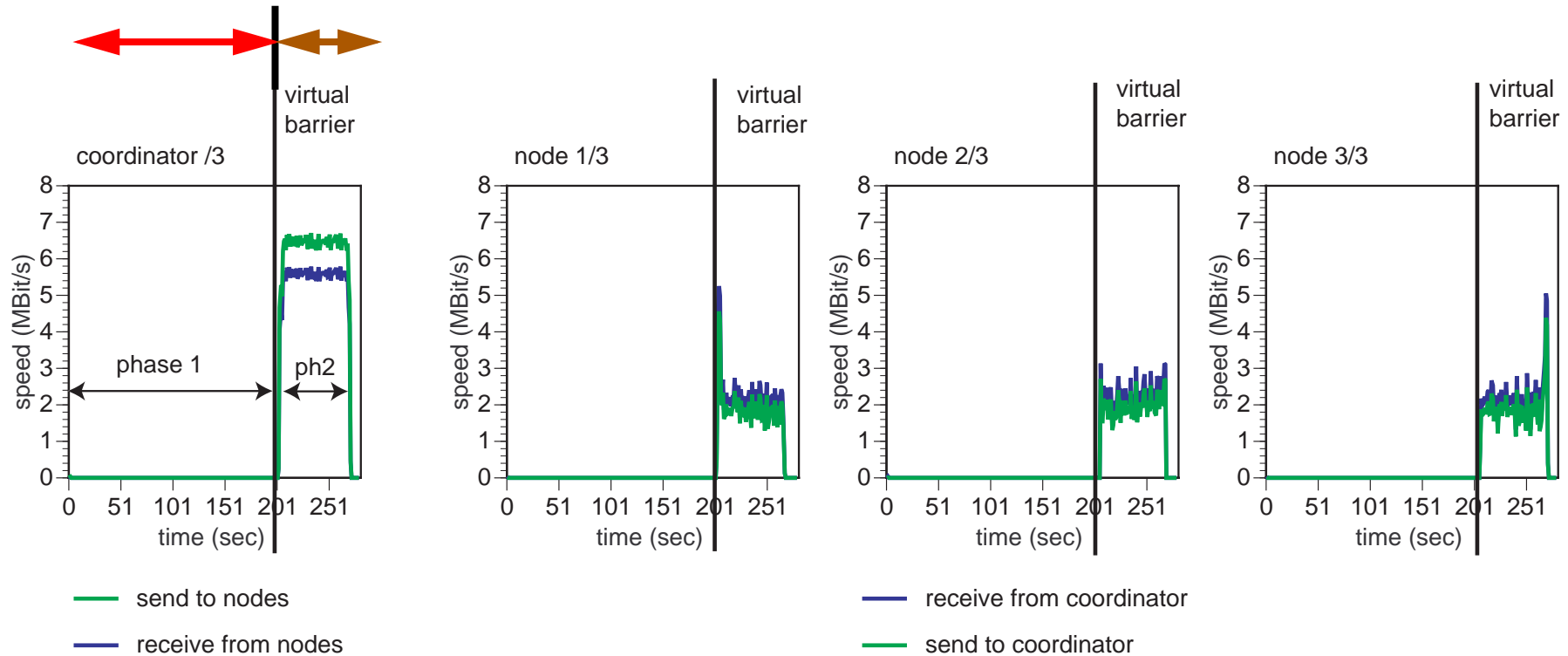
- with timestamps to allow reconstruction of time variant load.
- as total count to allow reconstruction of dropped samples.
- per node samples to allow reconstruction of global count.

Virtual Time - Phases of an Application

- Coarse granularity samples for long time simulations
- Daemon samples resource data **at regular intervals**
- User decides the **time interval t** for each single node
- Each sample collects **total resource data** within the time interval t
- Size of the sample packet is small: less than 126KByte



Virtual Time - Phases of an Application



Computation Phase: CPU limited

Communication Phase: network limited

Managing Intrusion of Monitoring Traffic

Protocol for the monitoring traffic: UDP/IP

- Loss of monitoring data packets
- Treatment of lost information as sample errors

Loose notion of time [Fox]

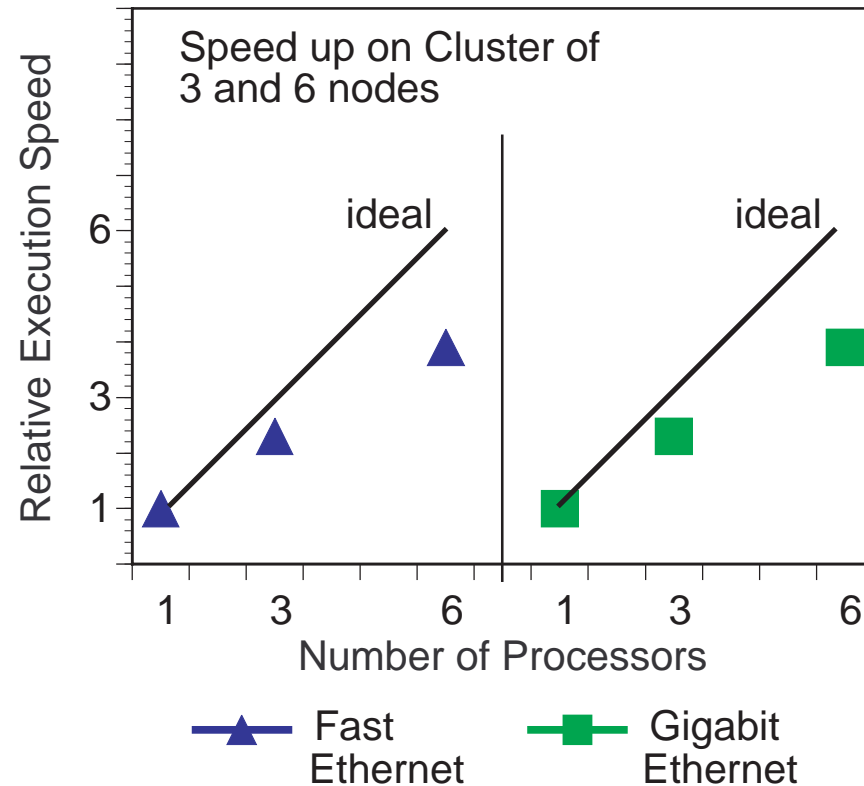
- Strong synchronization at the start of monitoring session
- Build-in cycle counters as **virtual barriers** for the synchronization of the samples

Master-slave paradigm of the performance data collection:

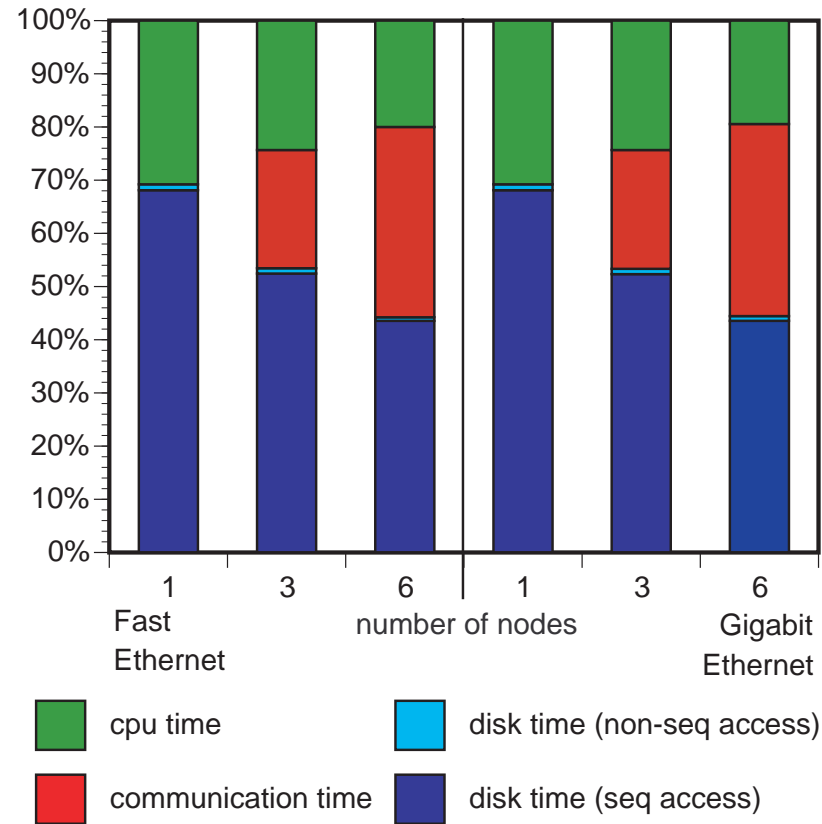
- Free allocation of monitoring master

Experiments with the Tool at Work

(Explaining a Scalability Problem of TPC-D query 3 as Resource Bottleneck)



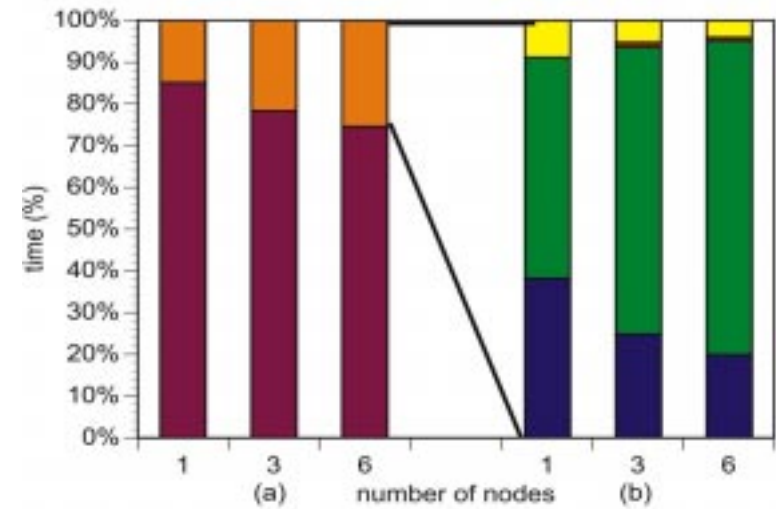
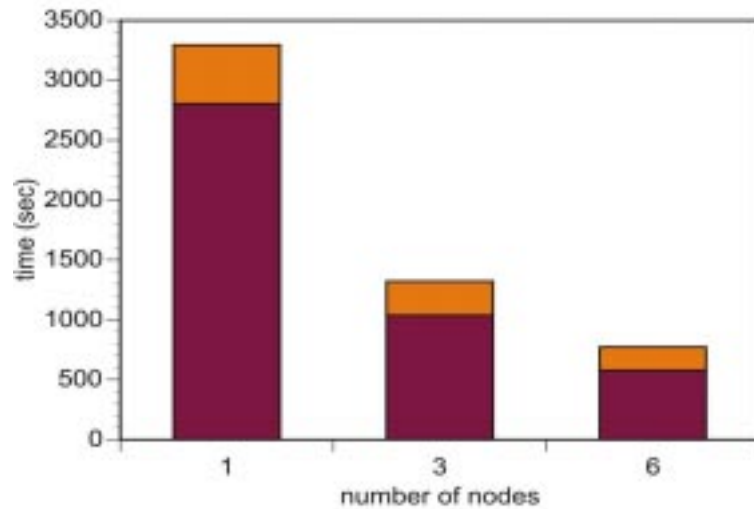
Communication-Limited Queries: Resource Usage



Fast Ethernet vs. Gigabit Ethernet

Does it always work...

(what about queries with disastrous performance?)



- allocated to machine-resources
- non-allocated to machine-resources

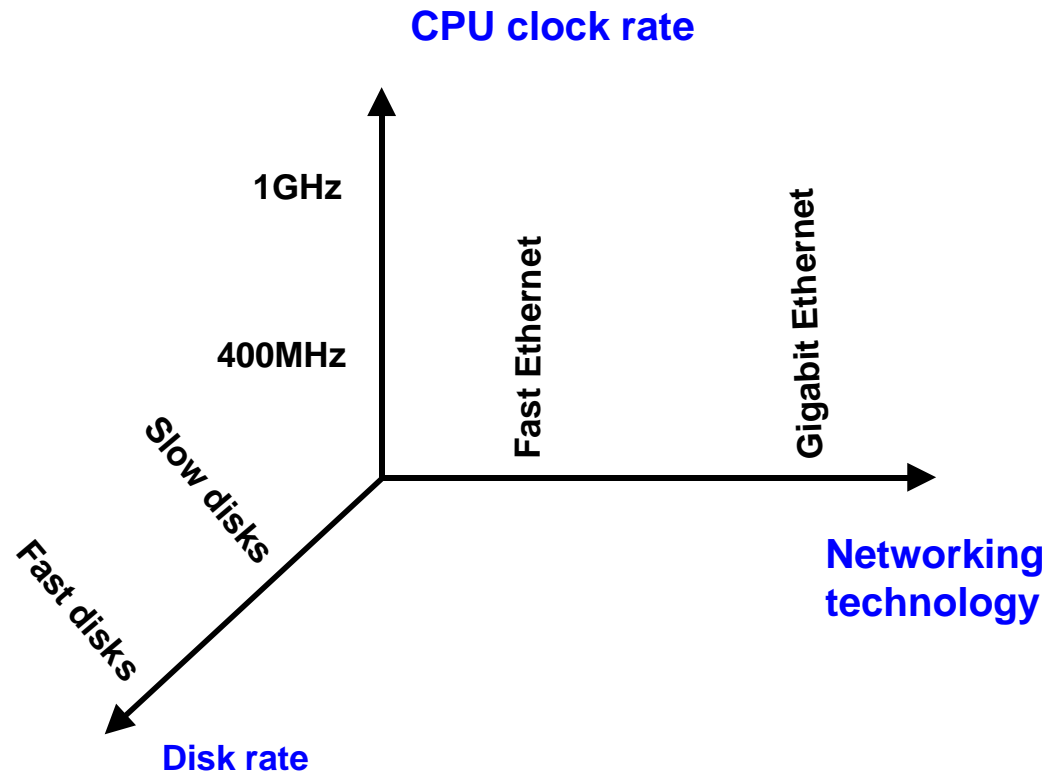
- CPU time
- communication time
- disk time (no seq access)
- disk time (seq access)

Most time is spent accessing data to the disks in a non-sequential way

Target of Performance Studies: Study of Different Platforms

Focus on machine resources:

- **CPU clock rate factor:**
 - Slow rate (400 MHz)
 - High rate (1 GHz)
- **Disk rate factor:**
 - Slow disks (7.3 ms/22 MB/s)
 - Fast disks (6.8 ms/30 MB/s)
- **Network technology factor:**
 - Fast Ethernet (100 Mb/s)
 - Gigabit Ethernet (1000 Mb/s)
- **Cluster size:** 1, 3, 6 nodes



Conclusions

Performance monitoring with middleware in distributed systems such as **clusters and desktop grids** remains difficult. Our attempt to build such a tool **more systematically** and **more scientifically** lead to:

- A new view of the performance debugging tool as **“inverted middleware”** leads to a **modular and layered** design of tools.
- A simple, but effective **performance model** directly connects **execution time** to the performance critical machine resources (CPU, disks, network) during different phases of an application.
- A technique to **keep intrusions low** uses **unreliable communication (UDP/IP)** to collect samples and **a notion of global virtual time** to reconstruct data dropped in network congestion.

Much more work is needed in **performance monitoring** for clusters. We are far from tools that are as **clear in functionality** and as **straight forward to use** as e.g. gcc, gdb or gprof.