Bottleneck Detection in Parallel File Systems with Trace-Based Performance Monitoring

Julian M. Kunkel Thomas Ludwig

Institute for Computer Science
Parallel and Distributed Systems
Ruprecht-Karls-Universität Heidelberg

Euro-Par 2008

Outline

1. Introduction
2. System overview
3. Performance Metrics and Statistics
4. Results
5. Summary

Outline

Motivation

- A parallel file system should utilize all ressources
  - Existing distributed/parallel file systems distribute data/metadata
  - Load imbalance leads to degraded performance
- Several factors lead to variation in I/O performance
  - Hardware capability
  - Access pattern of clients
  - Degraded RAID-Arrays
  - Efficiency of optimizations could vary
  - Throughput of a component could depend on the order of requests

Questions Regarding Load Imbalance

- How could the system or users detect load imbalance?
- Which hardware/software cause the load imbalance?
- Is the application's access pattern the reason of the load imbalance?
- How will the user figure out the application behavior leading to imbalance?

Long-term aim

Maybe the user can modify the code to increase efficiency of the system:

- Access pattern
- Data layout on the servers
- Give hints to the file system

Maybe the file system can rebalance access

Precondition

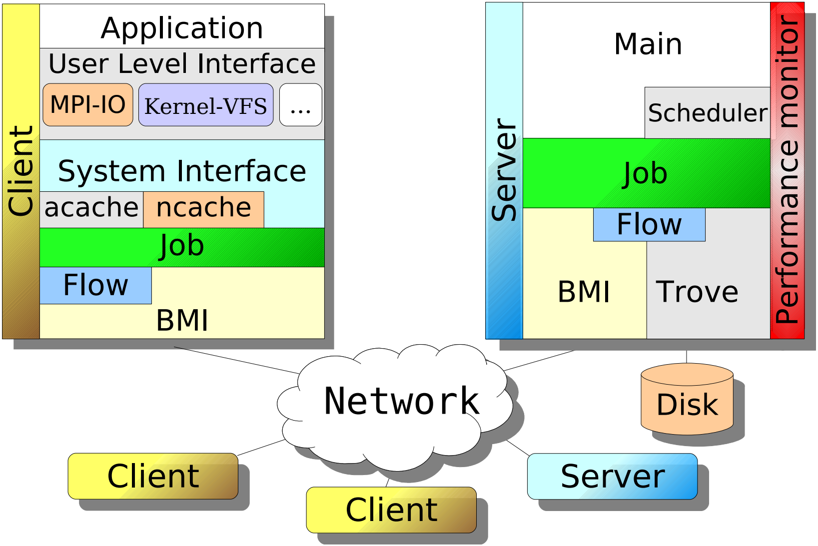
It is important to monitor the systems behavior

Solution

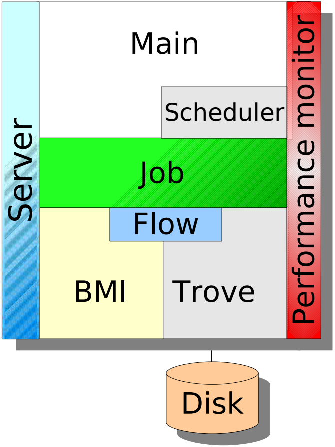
- Integrate meaningful metrics into the parallel file system
- Allow to query these metrics online
- Visualize the metrics for the users and relate the behavior with the application

Outline

Architecture of PVFS2



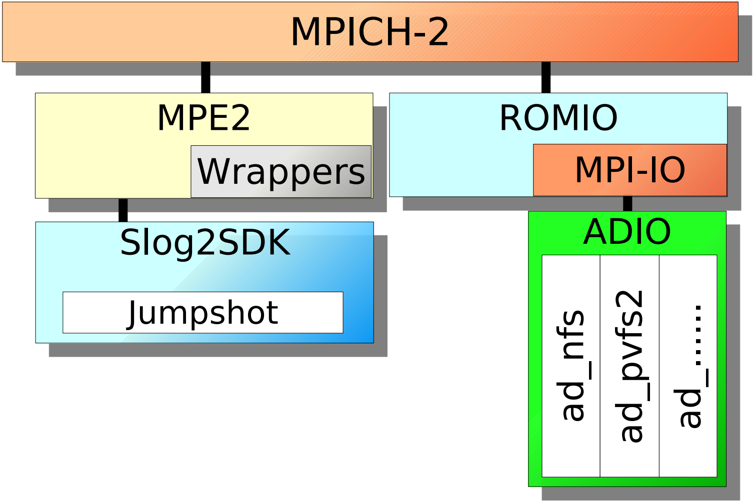
Architecture of the PVFS-Server



Description of the layers/components

- Main
  - Accepts new requests
  - Contains/processes statemachines
  - Scheduler allows concurrent access of non-interfering requests
- Performance monitor
  - Orthogonal layer
  - Stores statistics of internal layers
  - Can be queried remotely
  - Updated in fixed intervals
  - Keeps a history of the statistics

Used Software Environment



PIOviz

- Developed at the university of Heidelberg
- Uses MPE to generate client and server traces
- Add unique ID to PVFS requests in ROMIO
- Provides tools to work on these trace files:
  - Merge
  - Adjust time
  - Correlate client and server activities
- Modifications on Jumpshot:
  - Provide more details on events
  - Allow heights of states proportional to value

Outline

Performance Metrics

A metric is a standard unit of measure together with the way it is measured, in order to assess a process, state, or event ...

- Meaning of a component's metric can be different:
  - Represent current state
    - i.e. value is updated instantly - valid only the moment it is fetched
    - e.g. number of currently pending requests
  - Accumulates value during runtime
    - e.g. number of totally processed requests
    - allows to compute values for an arbitrary interval
  - Updated/computed in fixed intervals (statistic)
    - e.g. average number of processed requests

“Absolute“ metrics

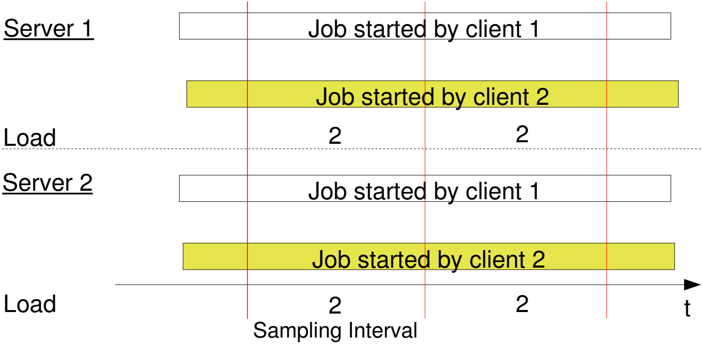
- Measure observable usage or performance
  - e.g. Throughput of network or disk,
  - Processed requests (of a given type),
  - Number of bytes read/written
  - ...
- Is a value of *50 MiB/s* a high value for network throughput?
  - One could relate the value with the maximum network throughput
    - Good value of a server if clients access data only half the time
    - Bad value if the server does not manage to process all requests
  - Is there a congestion in network?
  - Are servers not utilized by client requests?

Relative metrics

- Relate usage/performance with actual demand
  - Idle time of a component in percent within an interval
    - $\gg 0$ % - Wasted processing capability
    - 0 % - Does the component benefit from concurrent operations?
  - Average number of pending jobs within an interval (load-index)
    - The more complex a job the longer it is processed
    - The faster a component operates the shorter the queue
    - e.g. Linux Kernel 60 second system-load
    - e.g. drive queue depth
    - Does the component share its ressources among pending operations?
    - Is only a subset of operations serviced at a given time?
    - What if a set of long running jobs is serviced prior short jobs?
  - ...

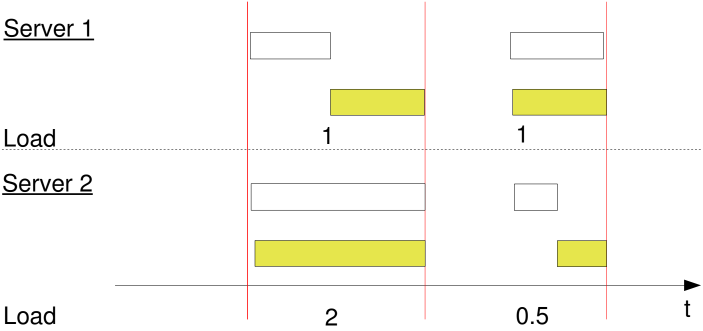
Balanced Hardware - Long-Running Jobs

- Assume requests which utilize both servers equally



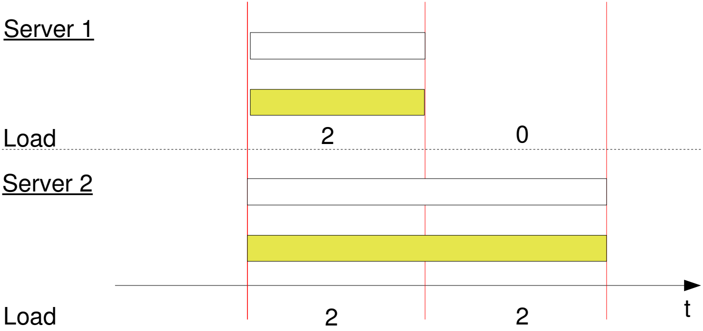
Balanced Hardware - Short-Running Jobs

- Assume a component shares ressources among pending requests
- Requests might arrive at different time

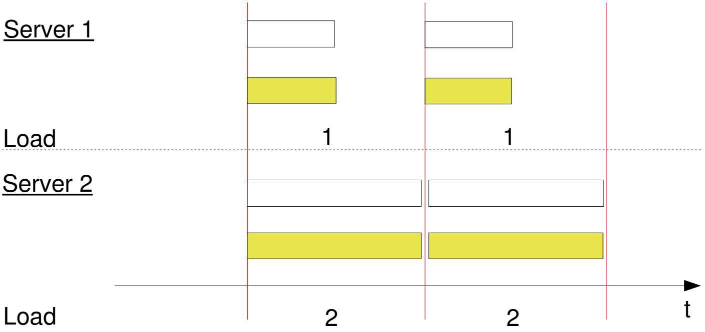


Inhomogeneous Hardware - Long-Running Jobs

- Assume server one is twice as fast as server two



Inhomogeneous Hardware - Short-Running Jobs



Conclusion

Problem

It is not easy to define meaningful metrics!

Expected behavior of a load-index

- Load should be accurate for long-running jobs
- Load average over longer periods should be accurate (even if there are short jobs)

Added Statistics/Metrics (1)

PVFS Statistics

- Request average-load-index
- Flow average-load-index
- I/O subsystem average-load-index
- I/O subsystem idle-time [percent]
- Network average-load-index

Added Statistics/Metrics (2)

Kernel Statistics

- Average kernel load for one minute
- Memory used for I/O caches [Bytes]
- CPU usage [percent]
- Data received from the network [Bytes]
- Data send to the network [Bytes]
- Data read from the I/O subsystem [Bytes]
- Data written by the I/O subsystem [Bytes]

Relation of several statistics could reveal the component causing imbalance

Outline

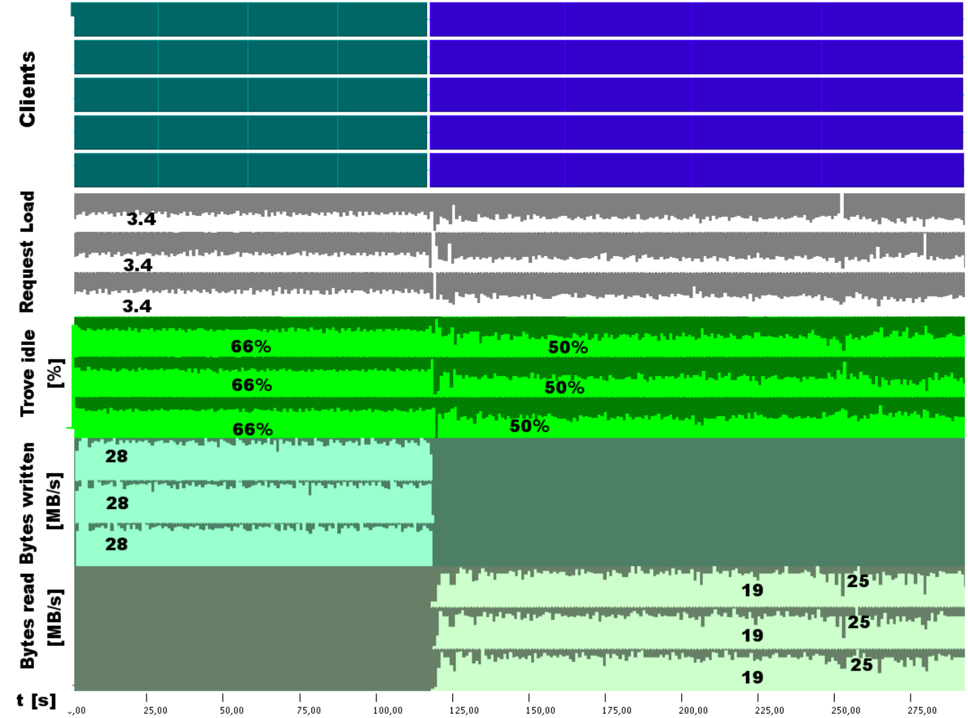
Test Environment

- 10 node cluster each node equipped with:
    - Two Intel Xeon 2GHz CPUs
    - 1000 MiB memory
    - GBit Ethernet (throughput: 117 MiB/sec)
    - IBM Hard disk (sequential throughput: $\approx$ 45 MiB/sec)
    - RAID Controller with two disks ($\approx$ 90 MiB/sec)
- Test program
    - Allows to select a level of access ((non-)contig., (non-)collective)
    - Clients write a fixed amount of data
    - Barrier
    - Clients read (their) data
- Disjoint clients and server partition
- Five clients, one metadata server and three data servers

Experiments

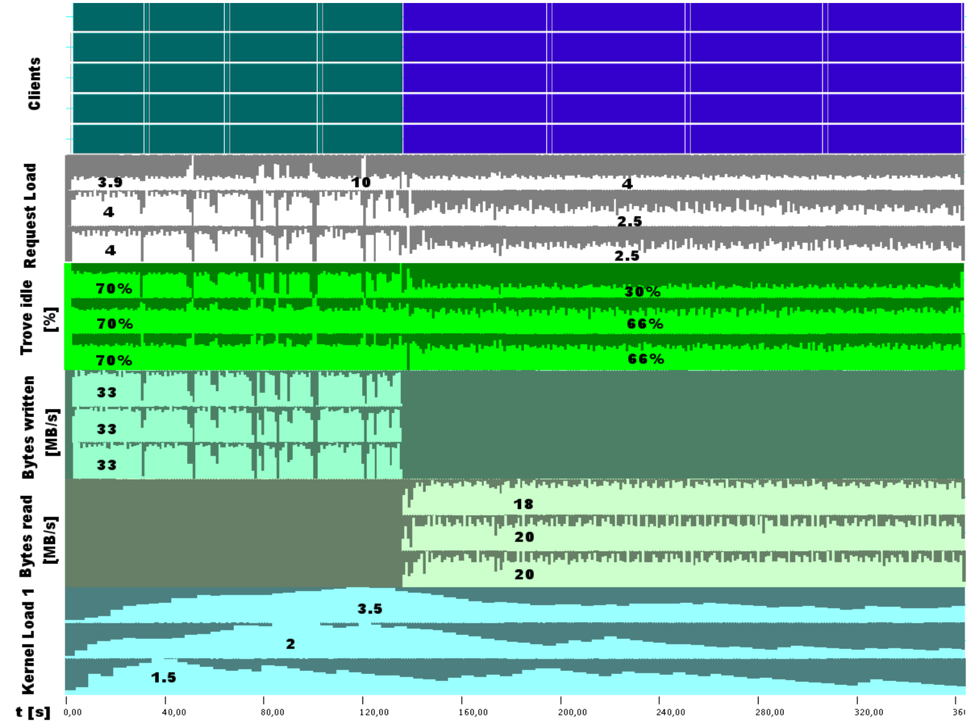
- Independent contiguous I/O (each client accesses 200 x 10 MiB)
- Collective, non-contiguous I/O (each client accesses 4 x 500 MiB)
  - Note: Size of ROMIO's collective I/O buffer is 4 MiB
- Both cases measured also on an inhomogeneous I/O subsystem
  - One server uses its system disk and not the RAID system

Collective, non-contiguous I/O with homogeneous hardware



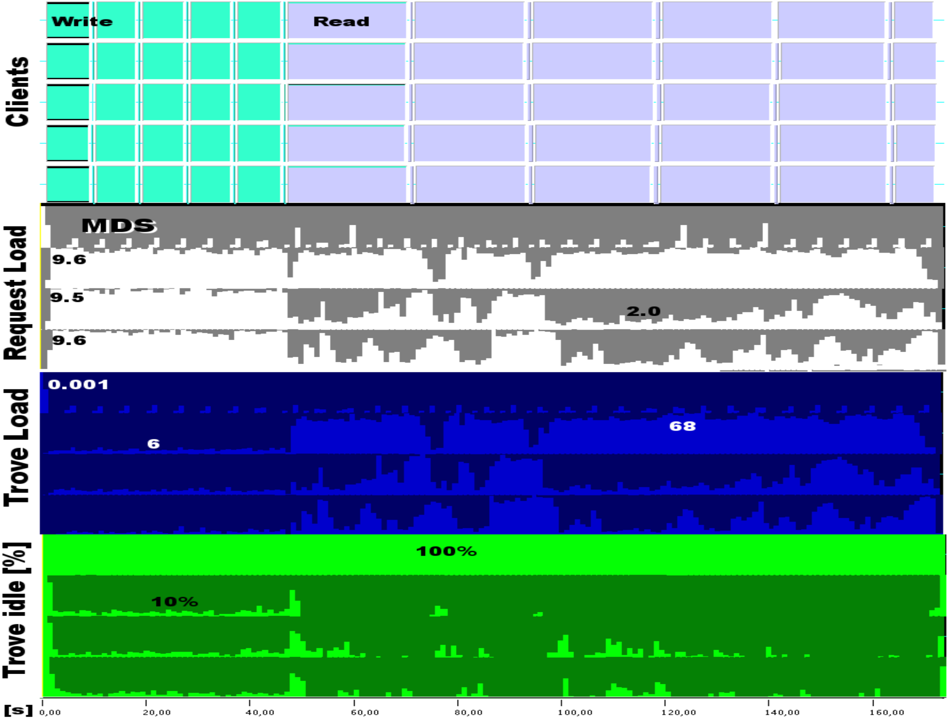
Clients
Request Load
3.4
3.4
3.4
Trove idle [%]
66%
66%
66%
50%
50%
50%
Bytes written [MB/s]
28
28
28
Bytes read [MB/s]
19
19
19
25
25
25
t [s]
0,00
25,00
50,00
75,00
100,00
125,00
150,00
175,00
200,00
225,00
250,00
275,00

Collective, non-contiguous I/O with inhomogeneous hardware

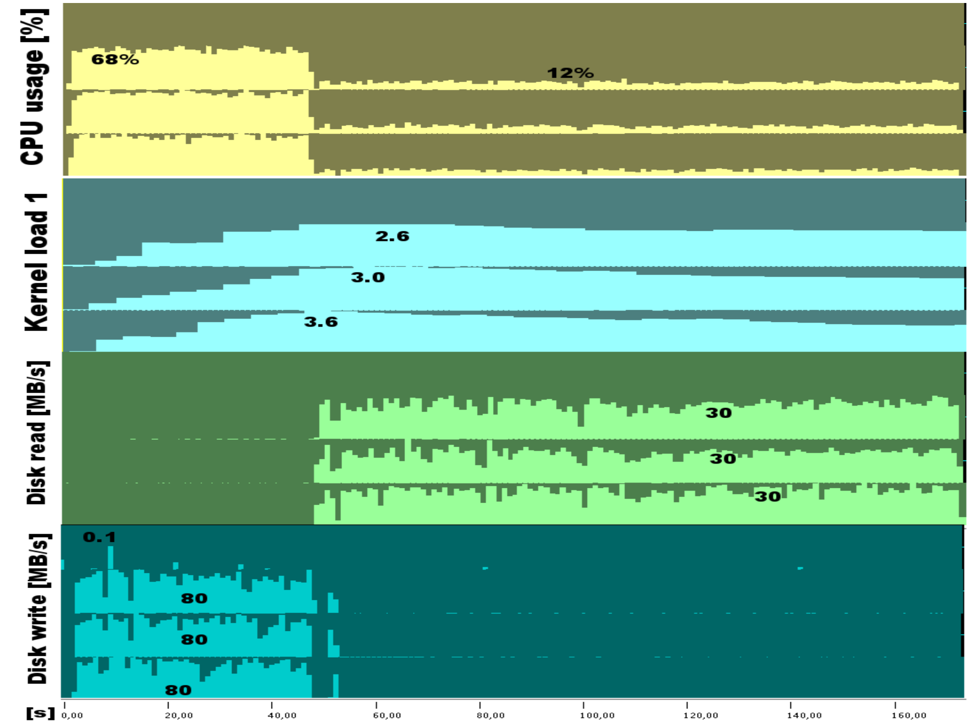


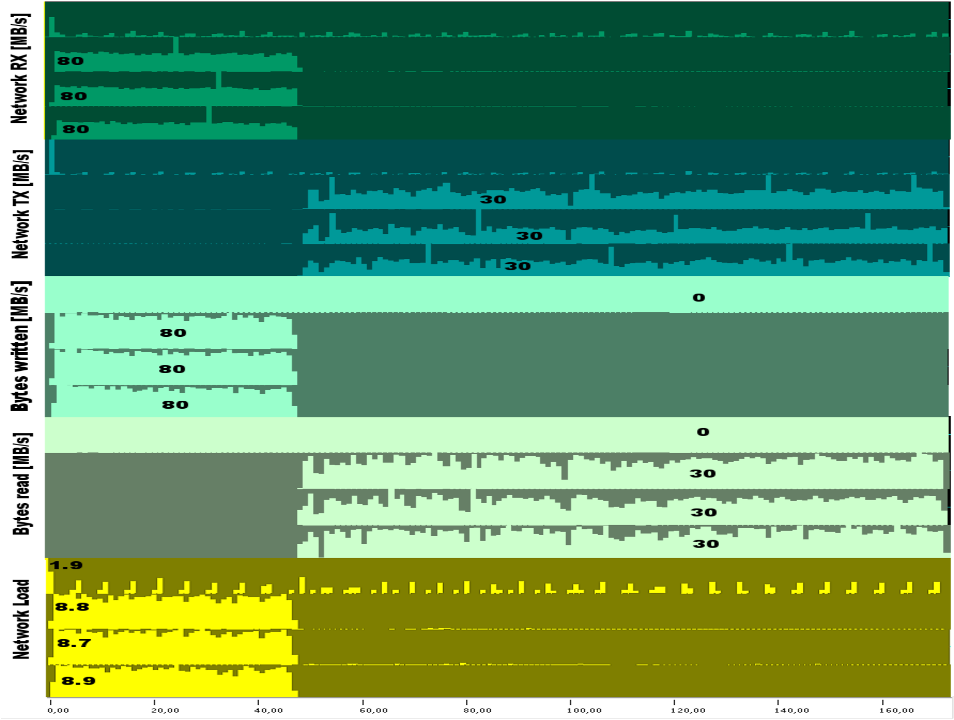
Clients
Request Load
3.9
10
4
4
2.5
4
2.5
Trove idle [%]
70%
30%
70%
66%
70%
66%
Bytes written [MB/s]
33
33
33
Bytes read [MB/s]
18
20
20
Kernel Load 1
3.5
2
1.5
t [s]
0,00
40,00
80,00
120,00
160,00
200,00
240,00
280,00
320,00

Independent, contiguous I/O



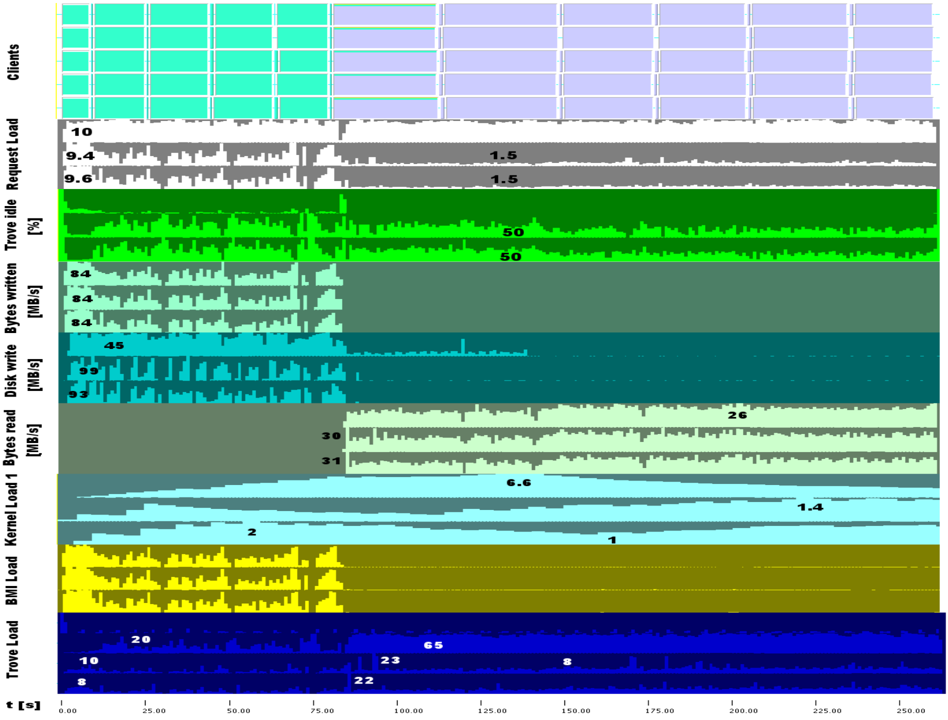
Clients
Write
Read
Request Load
MDS
9.6
9.5
2.0
9.6
Trove Load
0.001
6
68
Trove idle [%]
100%
10%
[s]
0,00
20,00
40,00
60,00
80,00
100,00
120,00
140,00
160,00


CPU usage [%]
68%
12%
Kernel load 1
2.6
3.0
3.6
Disk read [MB/s]
30
30
30
Disk write [MB/s]
0.1
80
80
80
[s]
0,00
20,00
40,00
60,00
80,00
100,00
120,00
140,00
160,00



Network RX [MB/s]
80
80
80
Network TX [MB/s]
30
30
30
Bytes written [MB/s]
0
80
80
80
Bytes read [MB/s]
0
30
30
30
Network Load
1.9
8.8
8.7
8.9
0,00
20,00
40,00
60,00
80,00
100,00
120,00
140,00
160,00

Independent, contiguous I/O with inhomogeneous hardware


Clients
Request Load
10
9.4
1.5
9.6
1.5
Trove idle [%]
50
50
Bytes written [MB/s]
84
84
84
Disk write [MB/s]
45
99
93
Bytes read [MB/s]
26
30
31
Kernel Load 1
6.6
1.4
2
1
BMI Load
Trove Load
20
65
10
23
8
8
22
t [s]
0.00
25.00
50.00
75.00
100.00
125.00
150.00
175.00
200.00
225.00
250.00

	Request load			BMI load			Trove load			Trove idleness [%]		
level 0	7.8	5.1	5.5	2.1	2.1	2.1	36.7	17.9	20.6	5.4	9.7	9.8
level 1	5.1	5	5.1	1.7	1.7	1.7	9.2	9	9.2	26.9	27.6	27.3
level 2	8.7	8.8	8.2	2.2	2.3	2.2	54.4	54.9	49.9	3.4	2.1	4.4
level 3	2.9	3	2.9	1.2	1.2	1.2	4.4	4.8	4.6	56.7	54	56.3
level 0 - 1 datafile	3.2	3.1	3.9	1.2	1.2	1.2	14.8	14.5	20.2	15.5	16.9	9.5
level 1 - 1 datafile	2.7	2.5	2.6	1.1	1.1	1.1	8.5	7.6	7.9	30.3	34.9	33.2
level 2 - 1 datafile	4	4.3	4.1	1.2	1.4	1.2	25.7	28.6	26.6	10.1	5.3	8.8
level 3 - 1 datafile	1.5	1.5	1.5	0.9	0.9	0.9	4.1	4.1	4.1	60.8	60.4	60.9
level 0-inh. I/O	8.4	2.3	2.2	1.2	1.3	1.3	44.1	5	4.6	2.6	40.6	42.1
level 1-inh. I/O	5.8	3.5	3.6	1.2	1.3	1.3	11.9	6	6.2	10.1	49.5	49.2
level 2-inh. I/O	9.2	4.1	4.1	2.6	1.1	1.1	65.2	25.6	25	1.6	39.1	39.8
level 3-inh. I/O	3.5	2.3	2.3	0.9	1	0.9	7.2	3.5	3.5	40.4	63.7	64.3

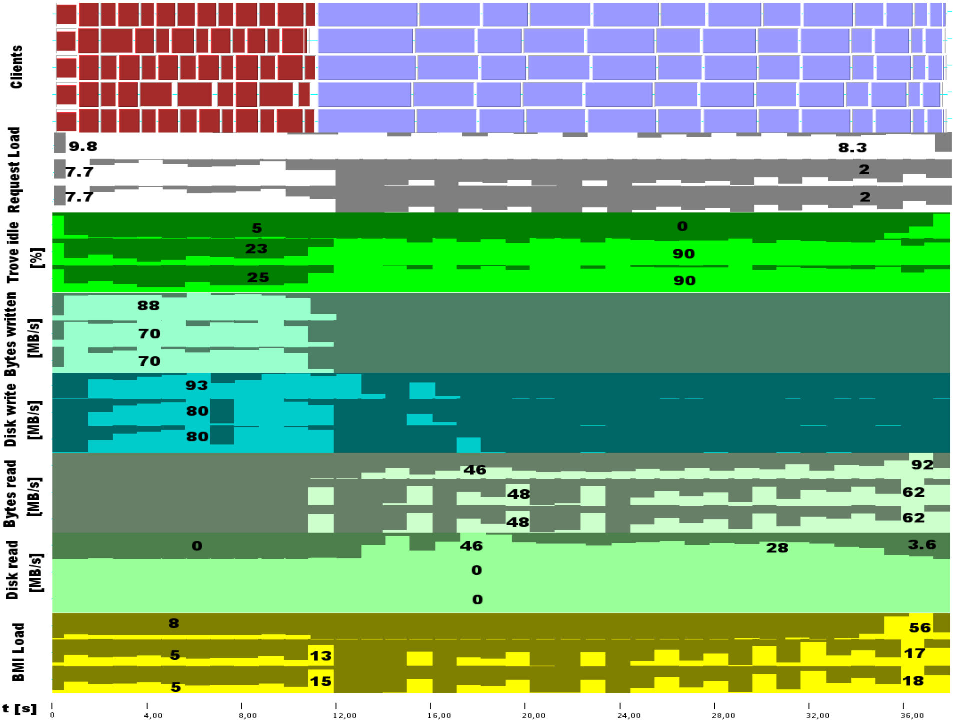
Outline

Summary

- It is important to monitor components performance
  - Relative metrics allow to assess usage with delivered performance
- Introduced an environment which allows to relate server statistics with (MPI) client activities
- Trace could assist the user to detect inefficient MPI-I/O calls and potential reasons
- Relation of several metrics allows to localize the component causing the load imbalance
- More work is needed to assess observed behavior

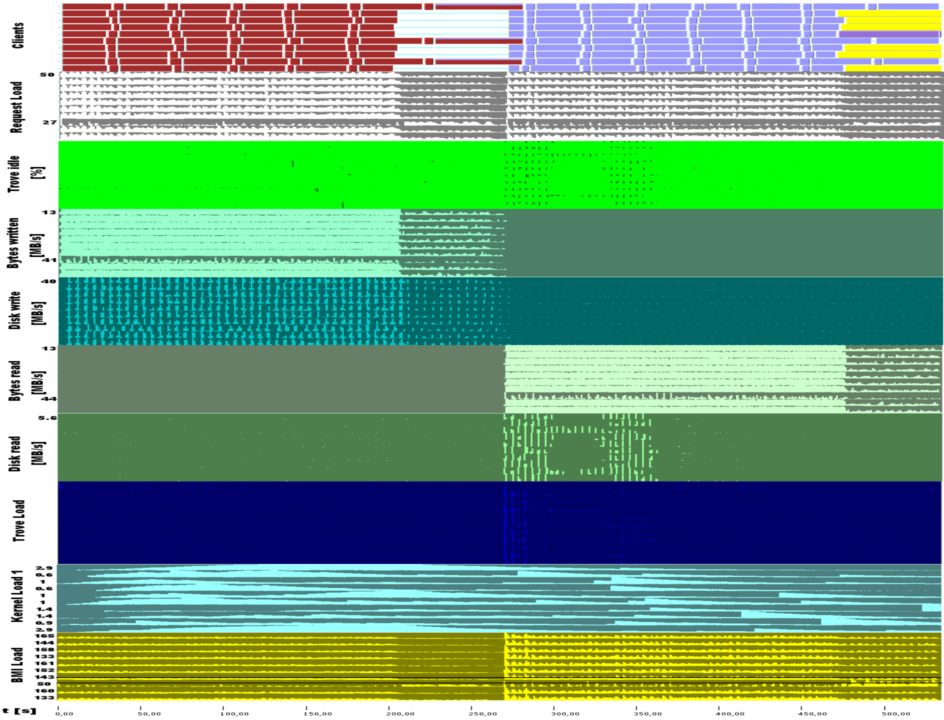
Optional slides

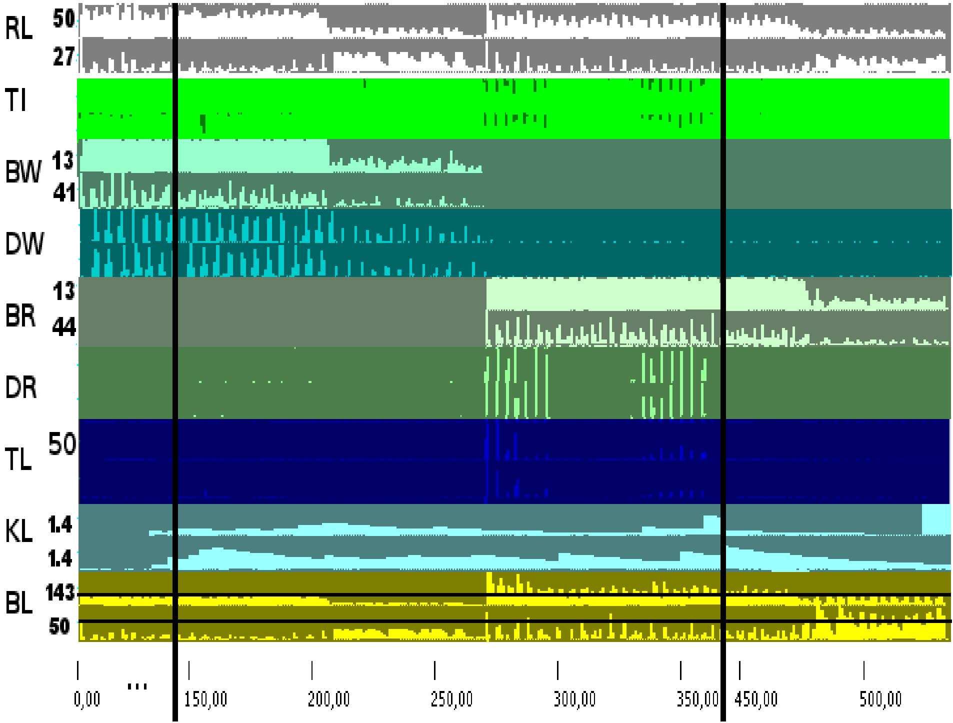
Unbalanced client access pattern



Clients
Request Load
9.8
8.3
7.7
2
7.7
2
Trove idle [%]
5
0
23
90
25
90
Bytes written [MB/s]
88
70
70
Disk write [MB/s]
93
80
80
Bytes read [MB/s]
46
92
48
62
48
62
Disk read [MB/s]
0
46
28
3.6
0
0
BMI Load
8
56
5
13
17
5
15
18
t [s]
0
4,00
8,00
12,00
16,00
20,00
24,00
28,00
32,00
36,00

10 Clients and 10 Serves in PC-Pool


Clients
Request Load
50
27
Trove idle
[%]
Bytes written
[MB/s]
13
41
Disk write
[MB/s]
40
Bytes read
[MB/s]
13
44
Disk read
[MB/s]
5.6
Trove Load
Kernel Load 1
2.9
0.6
1
0.6
1
1
1.4
1.4
0.9
2.9
BMI Load
165
144
158
133
161
152
143
50
160
133
t [s]
0,00
50,00
100,00
150,00
200,00
250,00
300,00
350,00
400,00
450,00
500,00


RL
50
27
TI
BW
13
41
DW
BR
13
44
DR
TL
50
KL
1.4
1.4
BL
143
50
0,00
...
150,00
200,00
250,00
300,00
350,00
450,00
500,00