Performance Monitoring
of
Production Batch Servers

Gyorgy Balazs
gyorgy.balazs@cern.ch

Supervisor:
Dr. Andreas Hirstius
andreas.hirstius@cern.ch

December 2008

Table of Contents

1. Motivations

A large cluster of computing nodes is available at CERN to provide batch type data processing for all users. There is a high demand for examining how efficiently the hardware is utilised under the batch service. The monitoring tools provide information about utilization, and information can be gathered about the running jobs from the scheduler, but the detailed performance figures are not available from the current monitoring infrastructure.

The subject of our investigation is to get very detailed, low level information about the production batch servers to analyse their performance and reveal possible bottlenecks which are hidden from the high level monitoring tools. Another goal of the project is to profile the software of the CERN experiments by examining what type of workload they do generate on the production nodes. This profile helps us setting up a benchmark process which generates similar load to the actual HEP (High Energy Physics) applications, and therefore provides an appropriate basis for both performance and power measurements during the acquisition process for new computing nodes.

2. The batch farm

The CERN Computer Centre is part of the LHC grid but also provides enormous computing facilities to satisfy all demands for computing power at CERN. The production batch farm, which serves the computational needs on site, consists of more than 4500 nodes. A large part of the computing power is dedicated to experiments, but the rest of the computing power is available for everyone at CERN. The main users of the batch farm are the experiments which run numerous types of jobs such as data analysis, simulation and reconstruction.

The batch services offer different queues for scheduling the jobs depending on the required CPU time.

Offered job queues by length:

- 8nm 8 normalised minutes
- 1nh 1 normalised hour
- 8nh 8 normalised hours
- 1nd 1 normalised day
- 1nw 1 normalised week

Queue definitions are empirically defined to match users requirements for turnaround times where a user could expect many short jobs per day and a few long jobs overnight. The normalization changes as machines get faster.
There is a grid queue for each virtual organization (VO) but without any CPU-time granularity. They use mapped team accounts and allow at most 1nw CPU time. Production queues are for low priority work where a specific turnaround is not an issue, they allow a higher number of concurrent jobs. There are also local queues funded by the experiments for fast turnaround of analysis jobs.

The dispatched jobs are scheduled by LSF. Platform LSF (Load Sharing Facility) is a job scheduler, that is used for managing the batch farm at CERN Computer Centre. LSF provides batch execution and balanced workload management in HPC (High Performance Computing) environments. The logs from LSF can be used to get information about the jobs running on the nodes within a specified time frame.

3. Performance monitoring

Monitoring CPU performance allows us to access low level information concerning the actual workload on the system.

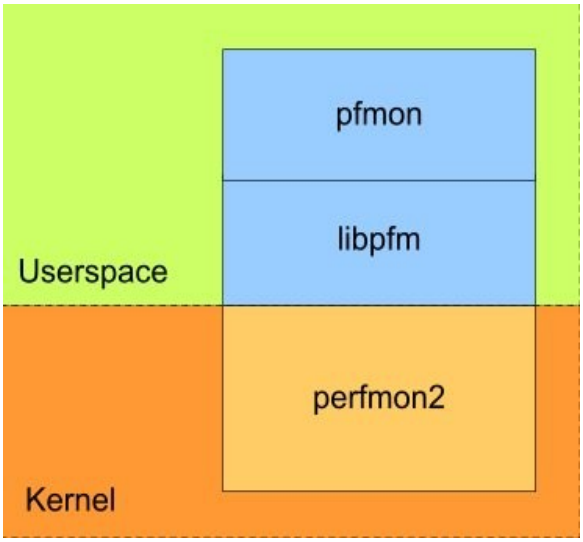
All modern CPUs offer real time statistics on executed instructions via a Performance Monitoring Unit (PMU). The PMU provides both specific and programmable counters to monitor different events inside the CPU. Among many others, information can be extracted about the amount of different type of executed instructions, consumed CPU cycles, ratios of different operations and cache misses.

Perfmon2 – pfmon

Perfmon2 and pfmon provide a robust framework in the Linux/Unix environment to access PMU counters with only a minimal overhead on the monitored system.

Perfmon2 provides a kernel-side interface to access PMU counters on a large variety of processors. Currently it is available as a kernel patch, but there are efforts to integrate it into the main Linux kernel line.

This interface is accessible through the **libpfm** library from a simple command line tool called **pfmon**, but also from other toolkits.



The Perfmon2 -pfmon architecture

The functionality of pfmon is rather broad [2]:

- Counting events
- Sampling in regular intervals
- Flat profile
- Multiplexing
- System wide monitoring mode
- Triggers
- Different data readout “plug-ins” (modules) available

To gain an extensive overview on the performance of the entire system, we need to monitor numerous events throughout all running applications.

Due to the limited number of counters, only a few events can be monitored at the same time, but pfmon allows us to monitor a larger set of events by using time based multiplexing. The program switches between smaller sets of monitored counters on a regular basis, in our case each 12 milliseconds. Multiplexing introduces some uncertainty, but it has only a minor impact in a long term monitoring.

The system wide monitoring mode allows for monitoring all running applications including the operating system, thus it is possible to gain information on overall performance.

4. Hardware setup

The CERN batch farm consists of more than 4500 nodes of various hardware types, mostly of dual and quad-core Intel Xeon based servers. Most of the production batch systems are dedicated for special tasks, but a large amount of computing nodes are available for general purposes.

The performance monitoring was started on 60 nodes of the public batch services.
All monitored nodes are of the same hardware configuration:

- CPU : 2 x Quad-Core Intel(R) Xeon(R) CPU E5410 @ 2.33GHz
- Memory (swap) : 16052 MB (4095 MB)

5. Software setup

All 60 nodes are installed with the standard 64bit Scientific Linux CERN (SLC) release 4.7 that is based on the Red Hat Enterprise 4 distribution and also compatible with most 32bit applications. The original software configuration was not modified, only the performance monitoring program (pfmon) and the patched kernel that includes perfmon2 were installed.

6. Performing the monitoring

Perfmon was executed on the 60 nodes in system wide mode with multiplexing, and the results were sampled each hour. The produced data was collected for 50 days. Unfortunately not all the machines “survived” until the last day, the monitoring was stopped on some of the nodes due to various failures that were not related to the monitoring process. On the last day, 43 nodes were running perfmon without any failure.

6.1 Monitored events

The following performance events were monitored:

- UNHALTED_CORE_CYCLES
- INSTRUCTIONS_RETIRED
- BRANCH_INSTRUCTIONS_RETIRED
- MISPREDICTED_BRANCH_RETIRED
- INST_RETIRED:LOADS
- LAST_LEVEL_CACHE_MISSES
- LAST_LEVEL_CACHE_REFERENCES
- INST_RETIRED:STORES
- X87_OPS_RETIRED:ANY
- RESOURCE_STALLS:ANY
- BUS_TRANS_ANY:ALL_AGENTS
- BUS_DRDY_CLOCKS:ALL_AGENTS
- BUS_BNR_DRV:ALL_AGENTS
- SIMD_COMP_INST_RETIRED:PACKED_SINGLE:SCALAR_SINGLE:PACKED_DOUBLE: SCALAR_DOUBLE
- CPU_CLK_UNHALTED:BUS

For further details, please refer to Intel's reference manual [1]

6.2 Derived information [1],[4]

- **Cycles per Instructions (CPI)**
  A ratio of executed instructions and core cycles that shows the efficiency of the executed code. A greater value indicates more opportunity for code tuning to improve performance. Intel Core2 based CPUs can have CPI values as low as 0.25
  - Calculation:
    UNHALTED_CORE_CYCLES / INSTRUCTIONS_RETIRED

- **Load and store instructions (LDST)**
  The percentage of memory operations compared to all executed instructions
  - Calculation:
    (INST_RETIRED:STORES +INST_RETIRED:LOADS) /
    INSTRUCTIONS_RETIRED * 100
- **Resource stalls (RESST)**
  The percentage of core cycles when the CPU was stalled due to waiting for resources. Resource stalls can occur during memory operations, after mispredicted branches and in several other cases when the CPU has to wait for busy resources.
  - Calculation:
    RESOURCE_STALLS:ANY / UNHALTED_CORE_CYCLES * 100

- **Branch instructions (BRANCH)**
  The percentage of branch instructions among all executed instructions.
  - Calculation:
    BRANCH_INSTRUCTIONS_RETIRED/ INSTRUCTIONS_RETIRED * 100
- **Mispredicted branches (BRMISS)**
  The percentage of branch instructions that were mispredicted. A wrongly predicted branch can result in a very high penalty on execution time.
  - Calculation: MISPREDICTED_BRANCH_RETIRED/
    BRANCH_INSTRUCTIONS_RETIRED*100
- **Last level cache misses (L2)**
  The percentage of cache references when the data was not found in the L2 cache. A cache miss normally results in memory access, which has a large negative impact on performance.
  - Calculation:
    LAST_LEVEL_CACHE_MISSES / LAST_LEVEL_CACHE_REFERENCES * 100
- **Bus utilisation (BUS)**
  Percentage of bus cycles consumed by bus transactions of any type.
  - Calculation (NOT VALIDATED):
    BUS_TRANS_ANY:ALL_AGENTS*2 / CPU_CLK_UNHALTED:BUS* 100
- **Data bus utilisation (DATA)**
  The percentage of bus cycles used for data transfers among all bus agents including the CPU and the memory.
  - Calculation (NOT VALIDATED):
    BUS_DRDY_CLOCKS:ALL_AGENTS / CPU_CLK_UNHALTED:BUS * 100
- **Bus not ready (BNR)**
  The percentage of bus cycles when bus transactions could not be executed, often because of the high load on the bus.
  - Calculation:

    BUS_BNR_DRV:ALL_AGENTS * 2 / CPU_CLK_UNHALTED:BUS * 100

- **Computational SIMD instructions (SIMD)**
  The percentage of SIMD (Single Instruction Multiple Data) instructions among all executed instructions. A good indicator of 64bit programs, since in 64bit mode the compile generates code that uses the SSE (SIMD) instructions almost exclusively for floating point operations.
  - Calculation:
    SIMD_COMP_INST_RETIRED:PACKED_SINGLE:SCALAR_SINGLE:
    PACKED_DOUBLE:SCALAR_DOUBLE / INSTRUCTIONS_RETIRED* 100
- **Computational x87 instructions (x87)**
  The percentage of 'traditional' floating point instructions among all executed instructions. A good indicator of 32bit programs, since in 32bit mode the compiler generates code that uses exclusively x87 instructions for floating point operations.
  - Calculation:
    X87_OPS_RETIRED:ANY /INSTRUCTIONS_RETIRED* 100
- **CPU utilisation (CPU)**
  The load on the CPU indicated by the ratio between the theoretical speed of the CPU and the actually consumed core cycles.

  Calculation:
  UNHALTED_CORE_CYCLES / (CPU_frequency * Number_of_cores * 3600 (seconds)) * 100

7. Transforming the results

7.1 Hourly results from raw data

The monitoring process generated one performance data file for each node for each hour throughout the whole monitoring period. To make this large amount of data processable, the data was extracted and merged together from all files from all machines, thus giving a single file for each machine containing all the data from all monitored hours.

7.2 Merging hourly results with LSF logs

To know what type of workload is investigated, it was necessary to find out what applications were running on the nodes during the monitored time period. The logs from the scheduler (LSF) provide, among many others, the following information about the executed jobs:

- Time of execution
- Time finished
- Queue
- User's Login ID

Using the time information, the jobs for each monitored hour can be extracted. The type of the job can be derived from the queue information, while the Login ID can be used to associate the job to a group or experiment.

The user's Login ID is first translated to the Group ID of the user's main group with the help of the passwd file from the interactive linux logon service (lxPlus). After that, the Group ID is translated to an experiment or department name, using the group overview of CERN's Xwho service, which is accessible on the following link:

http://consult.cern.ch/xwho/help/group_overview

That way, the performance monitoring results can be extended with the information of the running jobs. To each hour of performance data, the following information of all running jobs has been added: username, experiment, queue.

7.3 Generating Reports

The hourly results contain all collected data in a very detailed format, which makes it very complicated to read it. To convert the results to a human-readable format, different reports are generated:

- hourly report for each node
- daily report for each node
- summarized report for all time, all nodes
- summarized report of experiments
- profile report for each dominant group/experiment

7.3.1 Hourly report

The hourly report for each node contains an abstract from the corresponding hourly results file. Only the most relevant information from each hour was kept in an easily readable format, but also new information has been added.

The report contains the above described set of derived values extended with the following additional information:

JOB_COUNT – The number of jobs that were running in the actual time frame.
EXPERIMENT – The list of experiments that were running the jobs.
QUEUE – The list of job queues that the jobs were executed from.

7.3.2 Daily report

The daily report for each node consists of the accumulated results for each monitored day. The contents are the same as in the daily results, only the job information has been changed to counters to keep the file easily readable. A counter has been introduced for each experiment, that indicates how many “jobhours” the given experiment consumed.

The performance figures are collected on an hourly basis, so in each hour, the number of running jobs have been summarized for each of the groups and experiments, indicating their activity. The jobhours counter is generated by summarizing the parallel jobs in each hour run by the actual experiment. If an experiment runs 2 jobs for 3 hours, it is counted as 6 consumed jobhours.

7.3.3 Summary report

The report summarizes all information gained from the whole set of monitored machines. Each line corresponds to the summary of one day's result, keeping a similar structure to the daily reports. The last line contains a total summary of all values.

7.3.4 Experiments report

The experiments report shows the activity of the experiments throughout all days on all monitored nodes. Each column corresponds to one experiment, which are ordered by activity. The last line is a total summary of all values.

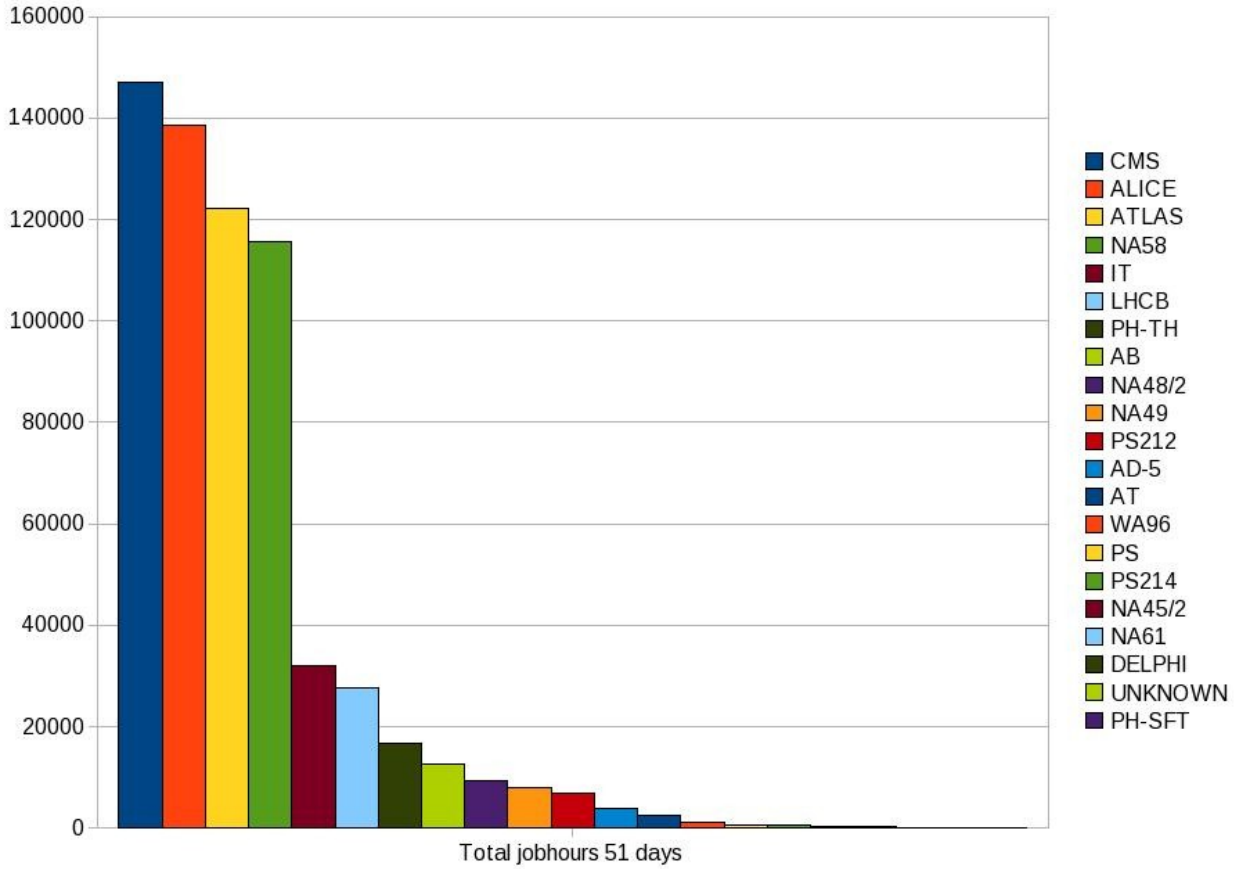
7.3.5 Profile files of the main user groups

A very important goal of the current investigation is to examine the activity of the main users of the batch nodes. Unfortunately this is extremely complicated to do so as it almost never happens that only one single user runs applications on a node at a given time. To get the best assessment, records are collected from the hourly reports separately for each of the dominant experiments and groups. Each data file consists of lines (hours) from the hourly report files when the actual experiment is dominant on the node, meaning that most of the running jobs are run by the actual experiment or group. The files are organised by relevance of the records, starting with the most relevant (more jobs), and the continuous time periods are separated out to make manual processing easier. When enough data is collected, these reports can be used to profile the usual workload that is generated by the different groups and experiments.

8. Analysis of the results

8.1 Distribution of experiments

The following diagram shows the accumulated activity of the user groups (experiments). The values are gained by summing up all jobhours that the actual experiment has consumed during the monitored time period.



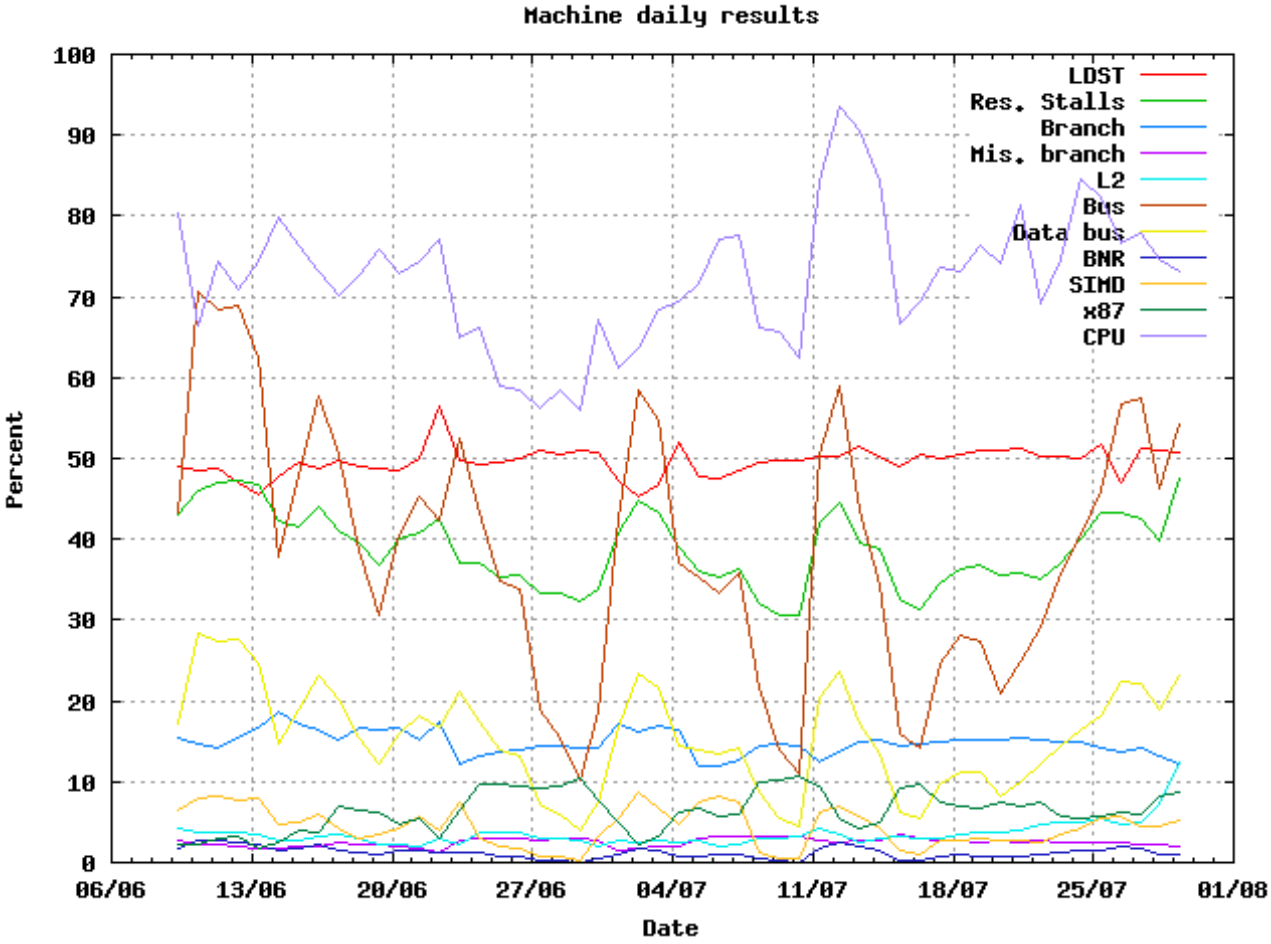
As it can be seen on the plot, the activity of the four most active experiments is much higher than the rest of the groups. CMS, ALICE, ATLAS and NA58 consumed more than 80% of the entire runtime, the other 17 groups were running jobs only in the 20% of the available CPU time, which perfectly matches the Pareto's principle (80/20 rule) [5]. As a consequence, the gained performance figures describe mostly the behaviour of the four most active experiments.

Total consumed jobhours/experiment:

CMS	147156
ALICE	138644
ATLAS	122328
NA58	115586
IT	32094
LHCB	27742
PH-TH	16684
AB	12568
NA48/2	9320
NA49	7836
PS212	6932
AD-5	3773
AT	2421
WA96	1154
PS	487
PS214	460
NA45/2	388
NA61	210
DELPHI	26
Unknown	20
PH-SFT	16

8.2 Overview of the summary

The following automatically generated plot shows the accumulated results for the whole monitored time period from all machines. All data is shown in percentage of the possible highest value.



The most remarkable fact on the plot is the large jumps in Bus utilisation and Data bus utilisation. The changes are only partly related to the CPU utilisation. Results of further investigations to reveal the reasons of this behaviour are presented later in this document.

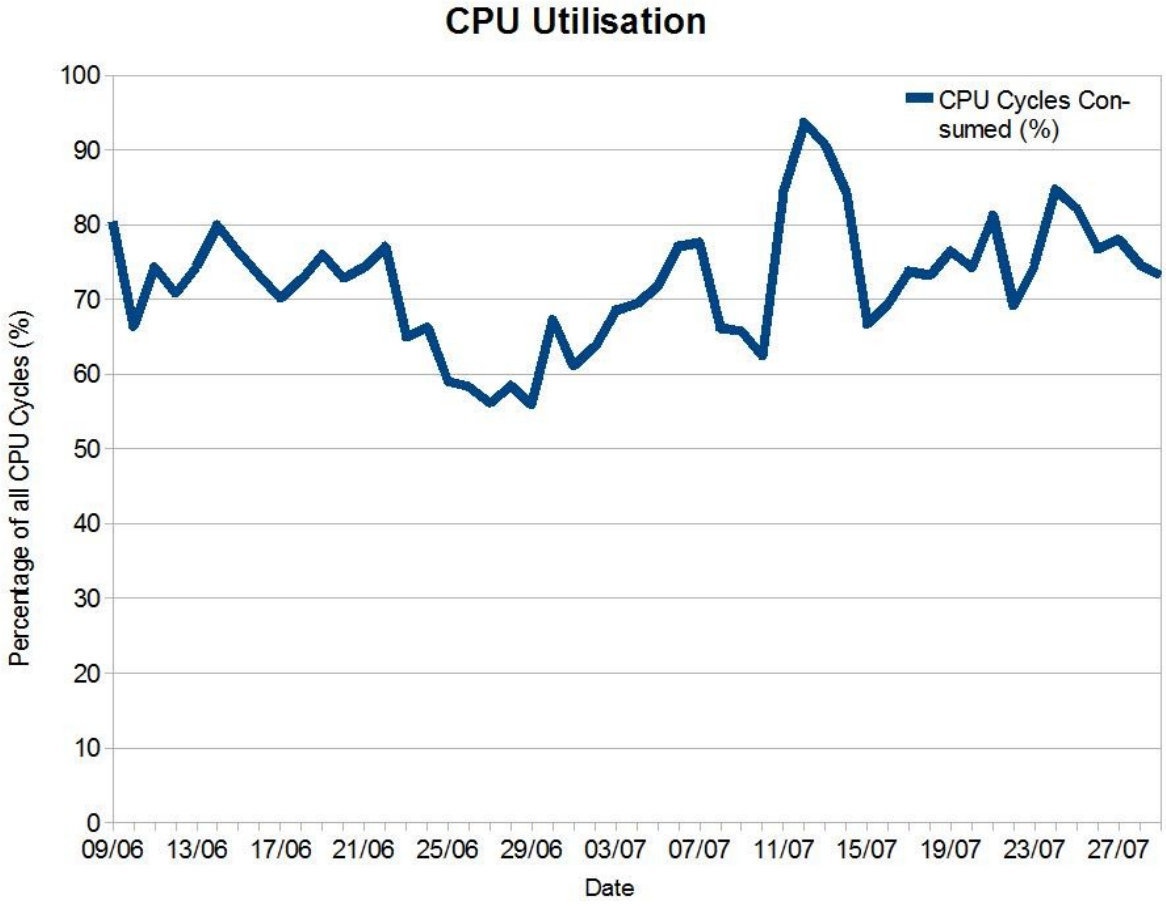
8.3 Average values

The following average values could be calculated by accumulating all results from the performance monitoring. These numbers represent the average performance figures of standard batch nodes.

CPI:	1.14
Load-Stores %:	49.63
Resource stalls %:	38.85
Branch instructions %:	14.85
Branch misses %:	2.51
L2 cache misses %:	3.55
Bus utilisation % (not validated):	38.95
Data bus utilisation % (not validated):	15.51
Bus not ready %:	1.21
SIMD instructions %:	4.39
x87 instructions %:	6.39
CPU utilisation %:	72.34
Average number of jobs/hour:	~9.5
Average runtime/job (hours):	3.27
Average working hosts:	52.3
Users total:	761
Jobs total:	197500
Consumed jobhours total:	645845

8.4 CPU utilisation

The plot shows the accumulated CPU utilisation for each day on all monitored nodes.



The overall CPU utilisation throughout the cluster fits the values gained from LEMON, the standard monitoring tool at CERN. The average 72.34% is lower than expected in an HPC environment, where the optimal value would be 80-90%. The fact that the utilisation drops even below 60% indicates that a better exploitation of the resources could be achieved.

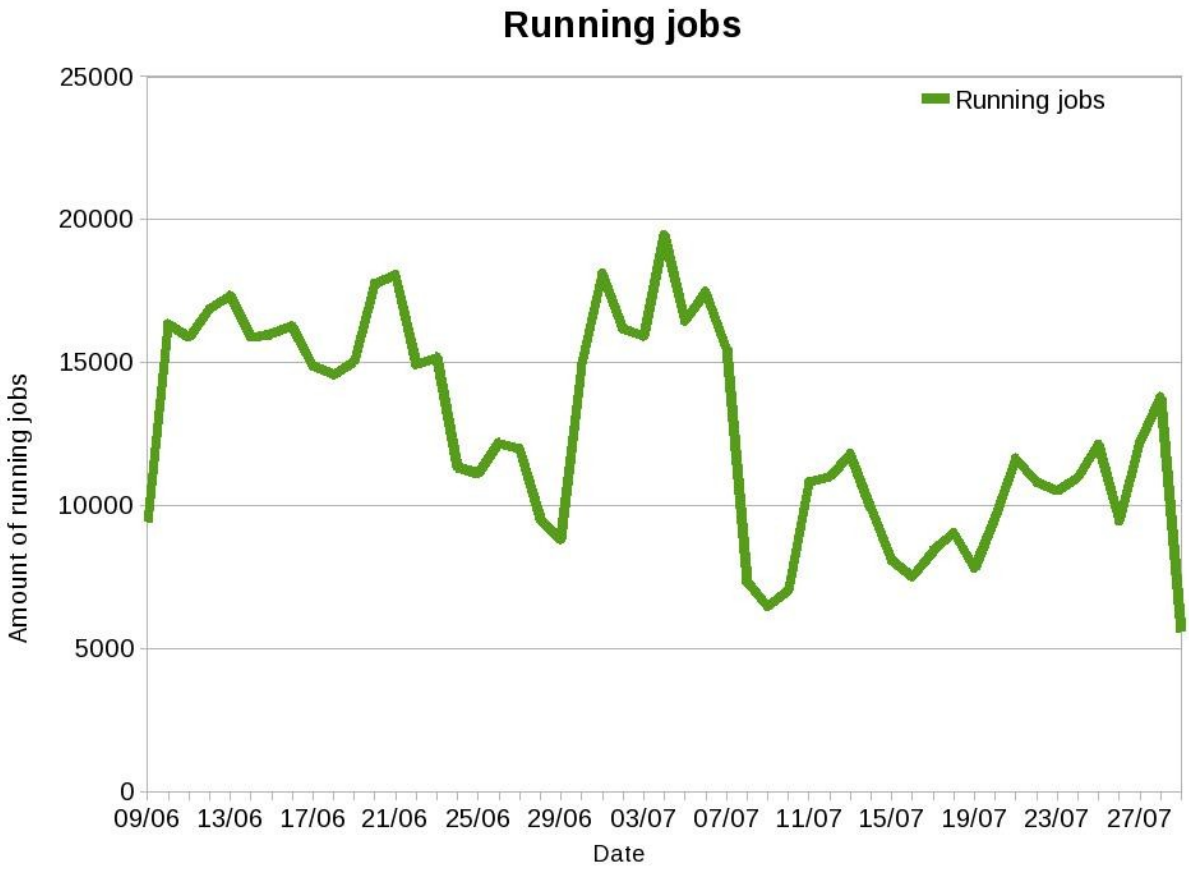
CPU utilisation of the most active experiments and groups:

Experiment	Records	CPU%
NA58	1414	68.86
LHCB	231	86.48
PH-TH	227	71.25
ALICE	183	87.44
IT	171	83.5
ATLAS	114	81.22
CMS	69	62.19
PS212	55	81.51

The summarising extract from the profile files of main user groups is sorted by relevance of the sample from each group. The more records are found when an experiment or group is dominant on a node, the more accurate the results are.
Apparently the most CPU intensive jobs are run by the LHCB and ALICE experiments, while NA58 and CMS are running less CPU intensive applications.

8.5 The number of running jobs

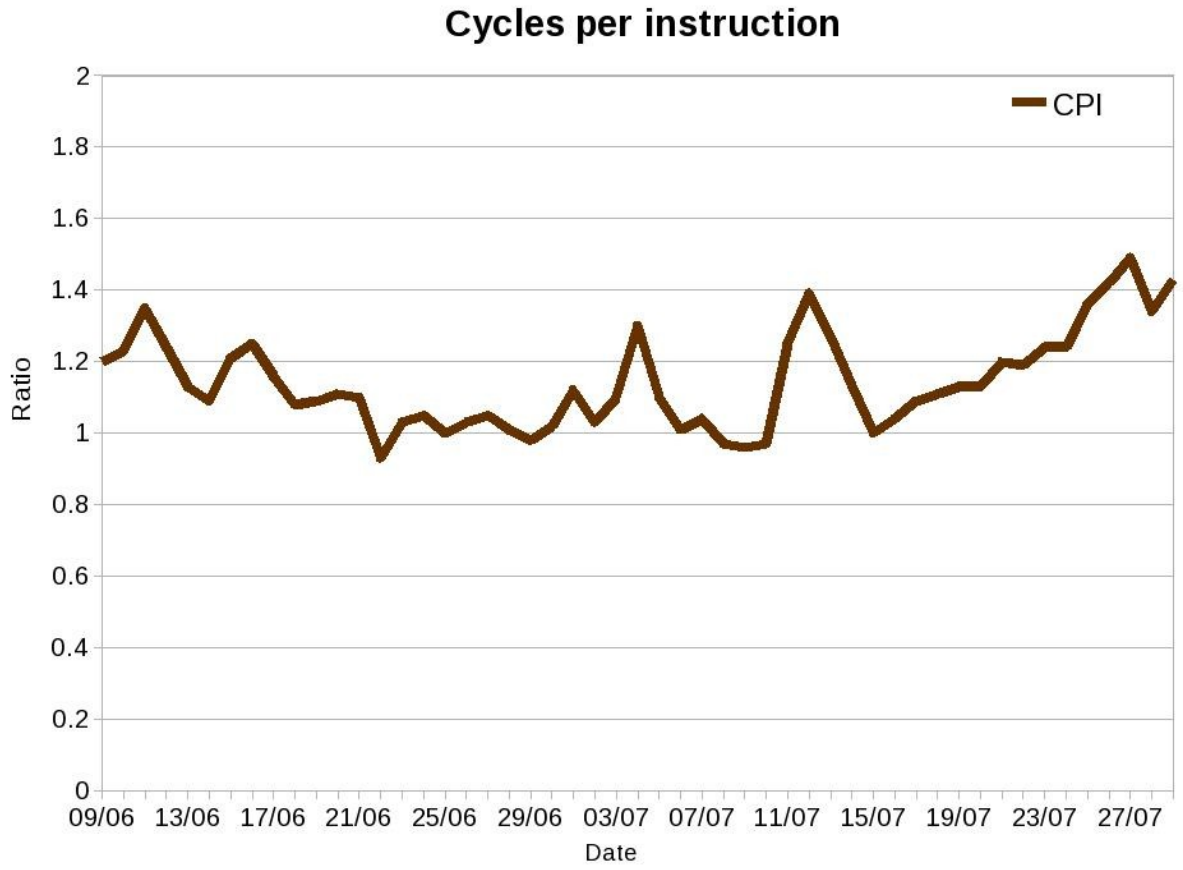
The plot shows for each day the accumulated number of jobs that were running on all monitored nodes. The values were gained by summarising the number of parallel jobs in each hour.



Apparently the CPU utilisation and the number of jobs are only partly related, but in most cases more jobs generate more CPU utilisation. The graph shows quite large changes in the number of running jobs, although the number of execution slots is constant. An explanation could be the changes in execution times, since shorter jobs raise the counters. Another explanation is discussed in the “SIMD and X87 instructions vs. CPU utilisation and the number of jobs” (8.13.3) section of the document.

8.6 Cycles per instruction (CPI)

The plot shows the accumulated CPI values for each day from all monitored nodes.



The values are visibly related to the CPU utilisation figures. The more loaded the CPU is, the more cycles have to be spent on each instruction. The 1.14 average CPI could be reduced by effective optimisation of the source codes. Modern CPUs can reach even 0.25 CPI by running optimised codes.

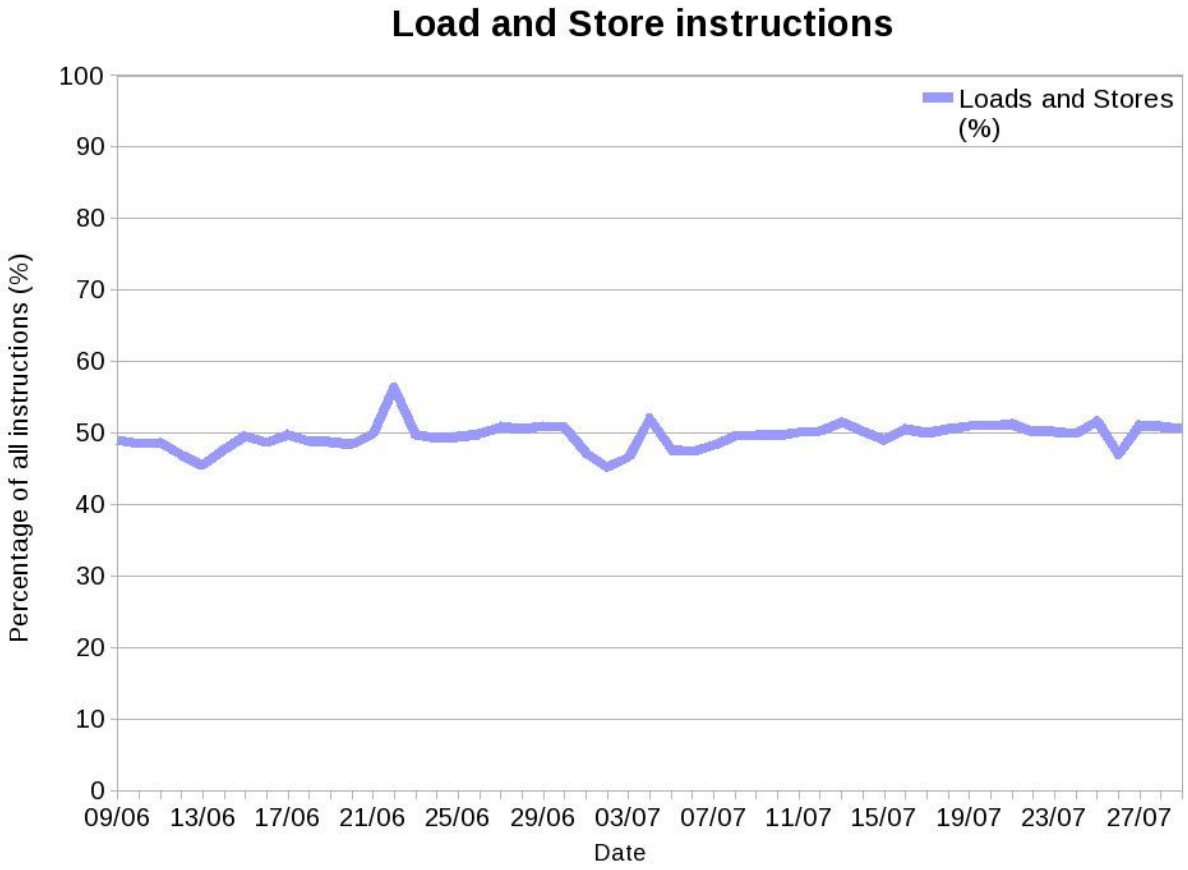
Average CPI values of the most active experiments and groups:

Experiment	Records	CPI
NA58	1414	0.96
LHCB	231	1.16
PH-TH	227	0.87
ALICE	183	1.28
IT	171	1.6
ATLAS	114	1.28
CMS	69	1
PS212	55	1.36

There is more than 40% of difference between the experiments. PH-TH executed the most efficient code (0.87), while the IT group had an unimpressive 1.6 CPI.

8.7 Load and Store instructions

The percentage of memory handling instructions can be seen on the following graph:



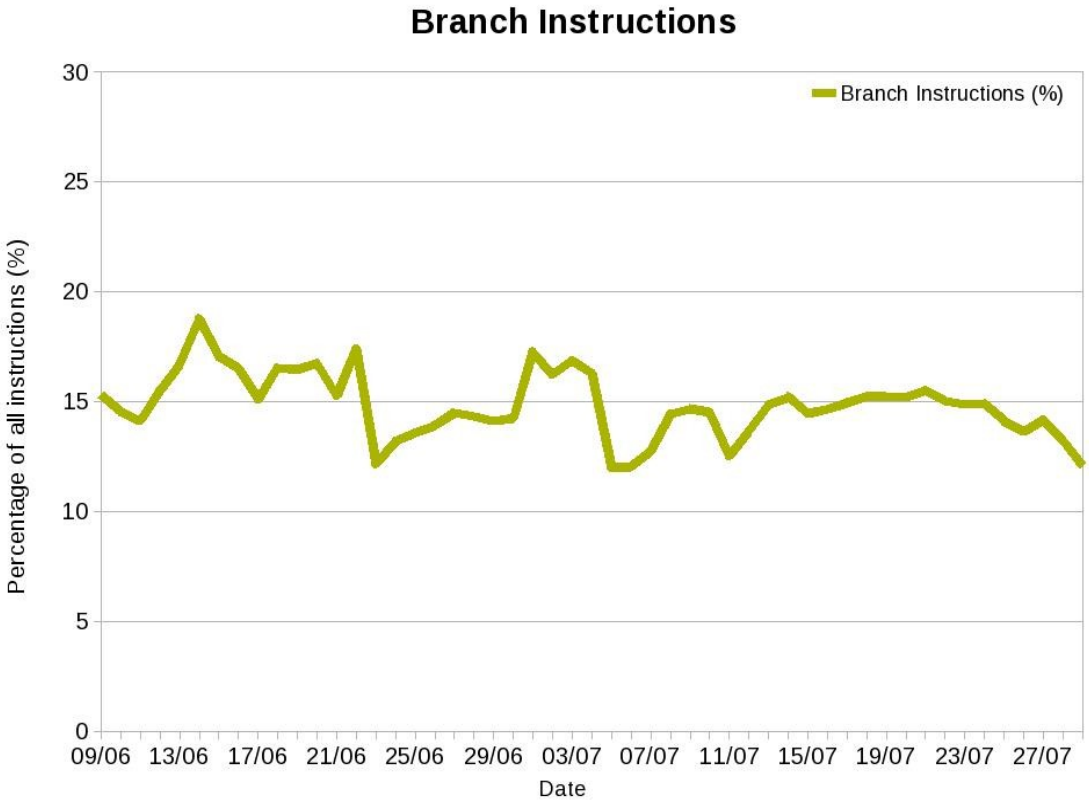
Apparently, the amount of memory load and store instructions is constantly around 50 percent regardless to the type of the running applications.

Experiment	LDST (%)
NA58	50.04
LHCB	51.23
PH-TH	46.42
ALICE	46.78
IT	45.4
ATLAS	52.07
CMS	52.54
PS212	51.05

The experiments that are using 64bit applications (IT, ALICE, PH-TH) are performing less Loads/Stores. About the relation of the experiments to 32 and 64 bit codes, please refer to the x87-SIMD instructions (8.13) session of the document.

8.8 Branch instructions

The following plot shows the percentage of branch instructions among all executed instructions.



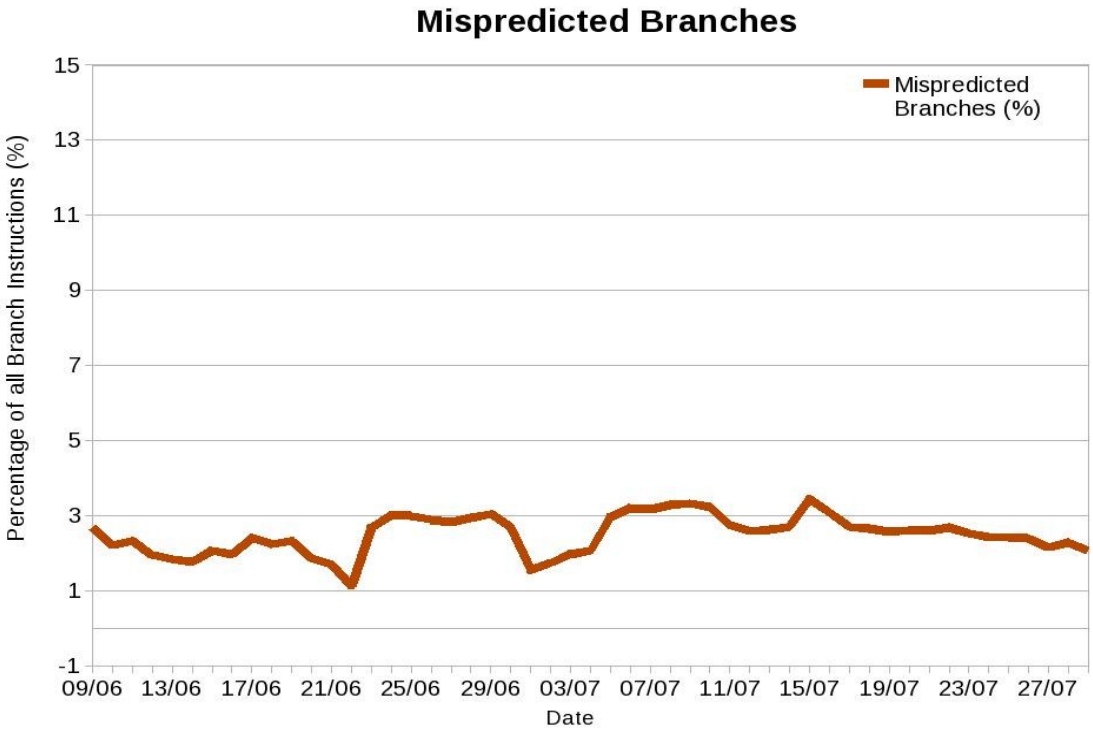
The amount of branch instructions does not have large changes throughout the whole time period, which means that the average ~15% of branches can be applied to most type of jobs running on the batch nodes.

The averages from the experiments are seemingly confirm this too:

Experiment	BRANCHES (%)
NA58	14.45
LHCB	16.49
PH-TH	11.96
ALICE	17.47
IT	12.51
ATLAS	15.22
CMS	15.95
PS212	14.07

8.9 Mispredicted branches

The percentage of wrongly predicted branch instructions can be seen on the following plot:

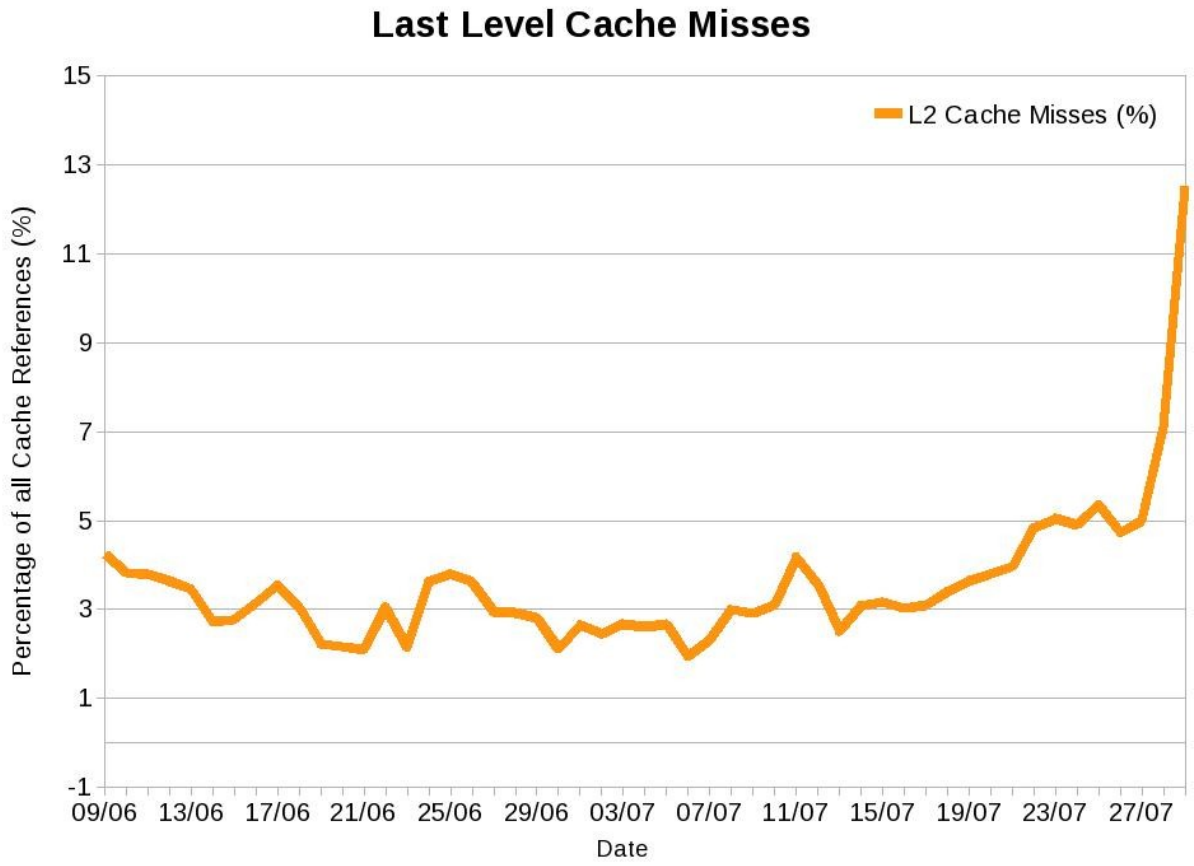


The amount of mispredicted branches is quite constant over time, the average is 2.51%, which is higher than it would be optimal.

Experiment	BRMISS (%)
NA58	3.22
LHCB	2.31
PH-TH	2.98
ALICE	1.93
IT	2.27
ATLAS	2.55
CMS	2.57
PS212	2.63

The high branchmiss ratio (3.22 % at NA58) reveals a possibility for performance tuning in the source codes. Each mispredicted branch instruction may result in a very large performance penalty during execution.

8.10 L2 cache misses



As shown on the plot, the ratio of the L2 cache misses increases suddenly from 28/07, the average of L2 misses for all machines reaches 12.5% on 29/07. The reason for such a large jump was investigated, and the following was found:

The cache miss ratio on many machines exceeded for shorter times even 30% in the last monitored days, while no change could be noticed on some machines compared to the preceding period. By examining the activity of the experiments, it was also found that the activity of the PS212 (DIRAC) experiment became multiple times higher than before. The examination of the detailed hourly results only strengthened the assumption that the jobs of this particular experiment can be related to the high amount of last level cache misses.

This can be proved by looking at the average L2 cache miss ratios of the experiments:

Experiment	L2 (%)
NA58	3.08
LHCB	2.04
PH-TH	1.72
ALICE	4.29
IT	6.25
ATLAS	2.49
CMS	4.67
PS212	10.86

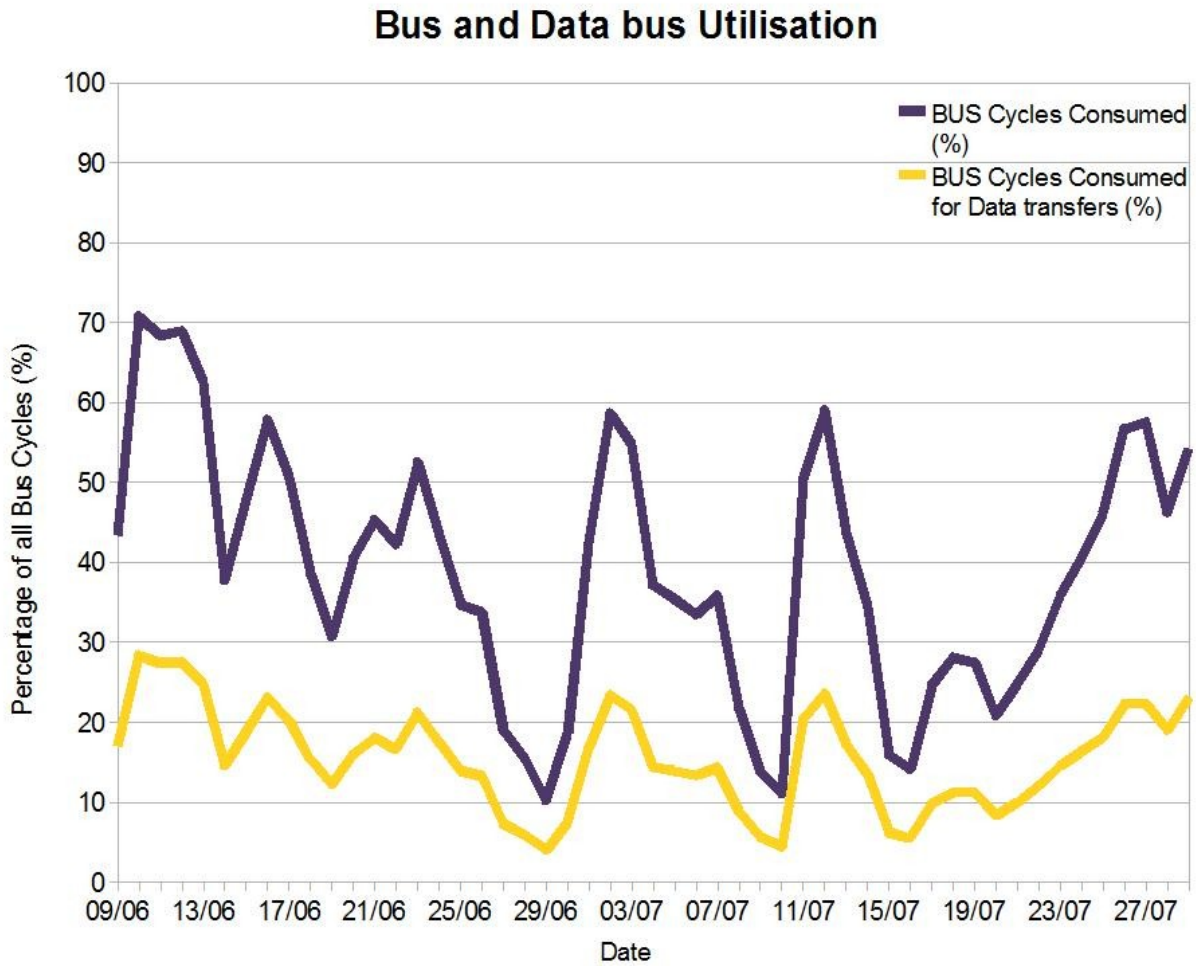
Besides the 10.86% of PH-TH, which is a very high rate that causes serious impact on performance, the 6.25 % cache miss ratio of the IT department is remarkable.

Reducing the last level cache ratio may result in noticeable performance improvement, since almost all occurrence of a cache miss results in memory access, which means longer execution time.

8.11 Bus and Data bus utilisation

8.11.1 Bus and Data bus utilisation figures (not validated)

The following diagram shows the overall Bus utilisation and the amount of Data transfers performed on the Front Side Bus.



The diversity of the accumulated bus utilisation is extremely high, and even larger jumps can be seen on some particular nodes. The amount of data transactions follows the overall bus utilisation figures, and apparently makes up a constant 40% of the consumed bus cycles.

8.11.2 Extreme bus utilisation values

To find out the reason for such a large diversity, the bus utilisation was examined in the detailed hourly reports.

The hourly reports on some of the machines showed extremely high bus utilisation that exceeds even 100% for shorter times. Values can be seen up to 300% for bus utilisation and 110% for data bus utilisation at the high peaks.

Since 300% is not an expected value, the raw perfmon output files were also examined. Using the equation to calculate the utilisation, the calculated values were the same as in the report files. The definition for calculating bus utilisation comes from the Intel 64 and IA-32 Architectures Manual [1]. The manual gives three definitions for calculating bus utilisation:

1. Bus utilisation percentage calculated using ratios and FSB:
FSB DATA READY * Bus Ratio * 100 / Non-sleep clock ticks

2. Bus utilisation percentage:
BUS transaction any performed by all agents * 2 / Cpu clk unhalted bus*100

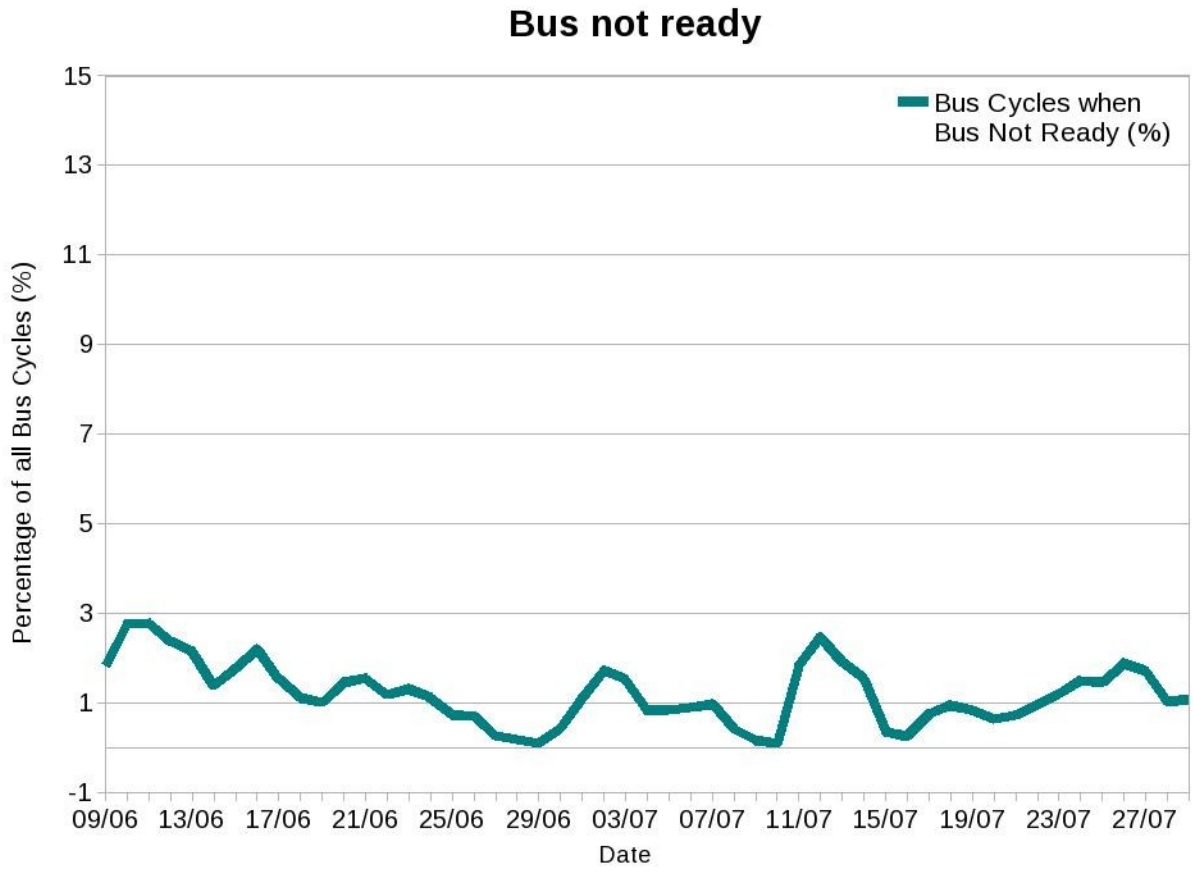
3. Data bus utilisation percentage:
BUS data ready clocks by all agents / cpu clk unhalted bus*100

The first definition is seemingly equal to definition 3, indicating data bus utilisation instead of bus utilisation (a discussion is already open on the topic with Intel). The actual results were calculated using definitions 2 and 3. The reason for such extreme values is unidentified, and a further investigation is necessary to understand and validate the calculation methods for the bus and data bus utilisation figures. The currently calculated values may be appropriate on a different scale, but it is also possible, that they are not even related to the real utilisation figures.

The extreme bus utilisation values could be reproduced artificially on a dual-socket Woodcrest based server, by running a benchmark program (lapack), that heavily stresses the memory subsystem. The test program had to be started as more processes than the number of CPU cores in the system in order to gain bus utilisation values exceeding 100%. Another starting point for a further investigation could be, that the bus and data bus utilisation figures were apparently related to the amount of executed SIMD instructions. At the high peaks of bus utilisation, most of the executed floating point instructions were SIMD instructions, indicating 64bit mode. An inquiry has already been sent to Intel regarding the issue.

8.12 Bus not ready

The percentage of bus cycles when bus transactions could not be executed:



The amount of unsuccessful bus transactions varies over time, and increases when the load is most likely higher on the bus. The average is 1.21%.

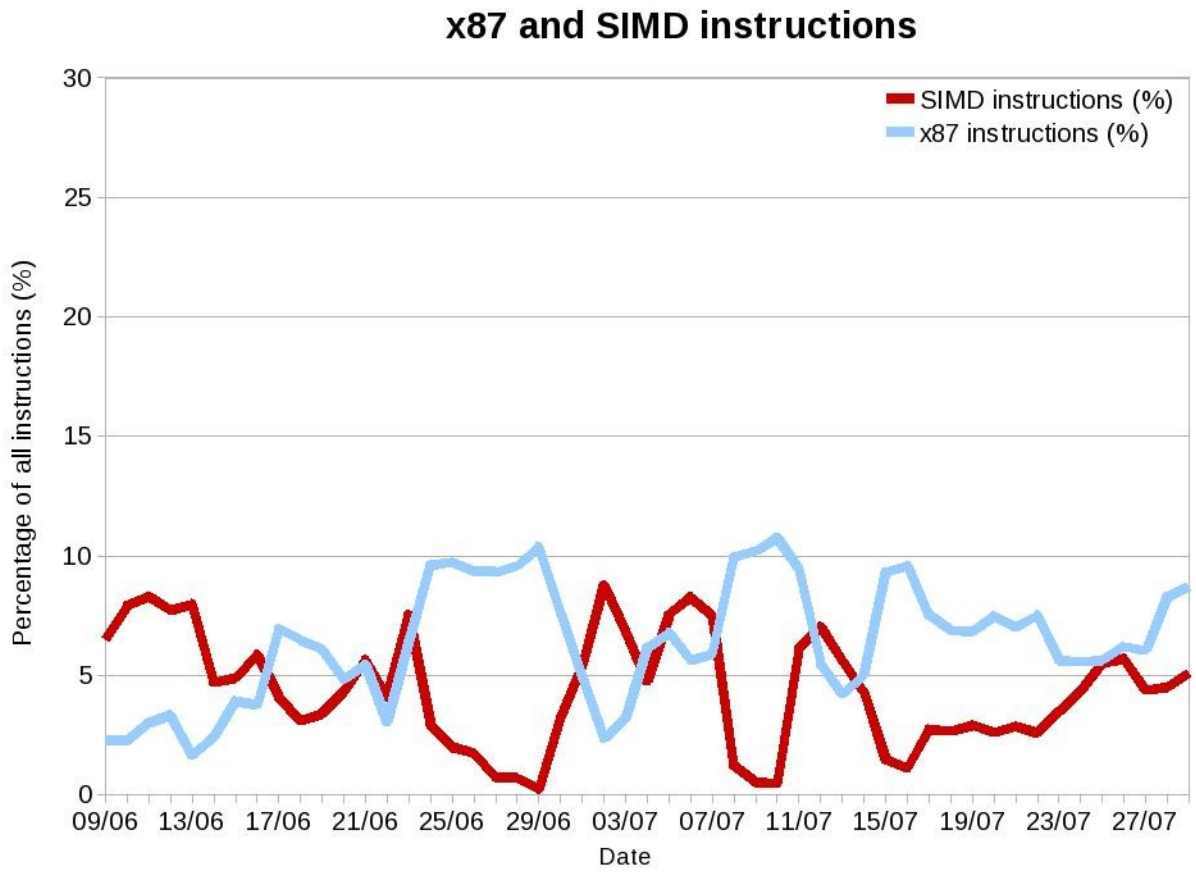
Bus not ready average at the experiments:

Experiment	BNR (%)
NA58	0.14
LHCB	1.31
PH-TH	0.7
ALICE	2.65
IT	4
ATLAS	1.05
CMS	0.13
PS212	0.84

The experiments with higher bus utilisation suffered from more unsuccessful bus transactions.

8.13 x87 and SIMD instructions

The following chart represents the executed x87 instructions as the amount of traditional SISD (Single Instruction Single Data) floating point instructions, and also shows the executed SIMD (Single Instruction Multiple Data) operations.



The amounts of executed x87 and SIMD instructions are good indicators to examine the composition of 32 and 64 bit applications. Today's compilers are using exclusively the traditional x87 instructions when compiling in 32 bit mode to maximise compatibility, while 64 bit applications are compiled to exploit the SIMD capabilities of the SSE instruction set. Thus the amount of x87 instruction represents 32 bit applications, while the amount of SIMD instruction represents the share of 64 bit applications.

The percentage of all floating point instructions can be calculated as a sum of the two averages (x87+SIMD), which is 10.78%. The graphs of the two instructions are apparently mirrored to each other, and the sum of the two values makes up almost constantly 10% of all instructions. This indicates that the amount of floating point instructions (X87+SIMD) is more or less constant when sampled over a large number of applications and a longer period of time.

About 60% of all applications were run in 32 bit mode, while the 64 bit applications made up the remaining 40%.

8.13.1 32bit and 64bit applications at the experiments

One major goal of this investigation was to identify which experiments are using 32bit and which are running 64bit applications. This could be derived from the x87-SIMD usage of the experiments:

Experiment	SIMD (%)	X87 (%)
NA58	0.4	10.43
LHCB	3.32	4.66
PH-TH	9.27	4.45
ALICE	5.74	0.86
IT	11.4	2.98
ATLAS	3.46	5.94
CMS	0.34	9.05
PS212	2.18	8.6

It is already clear from the averages, that NA58 and CMS are running mostly 32bit code, while ALICE runs mainly 64bit applications. To conclude what type of code the other experiments are running, it was necessary to analyse the jobmix in the collected data from each experiment manually, using the profile report of each user group. For example, if jobs from a given experiment run together with jobs from another experiment which is already identified, it is easily decidable whether the experiment uses 32bit or 64bit code.

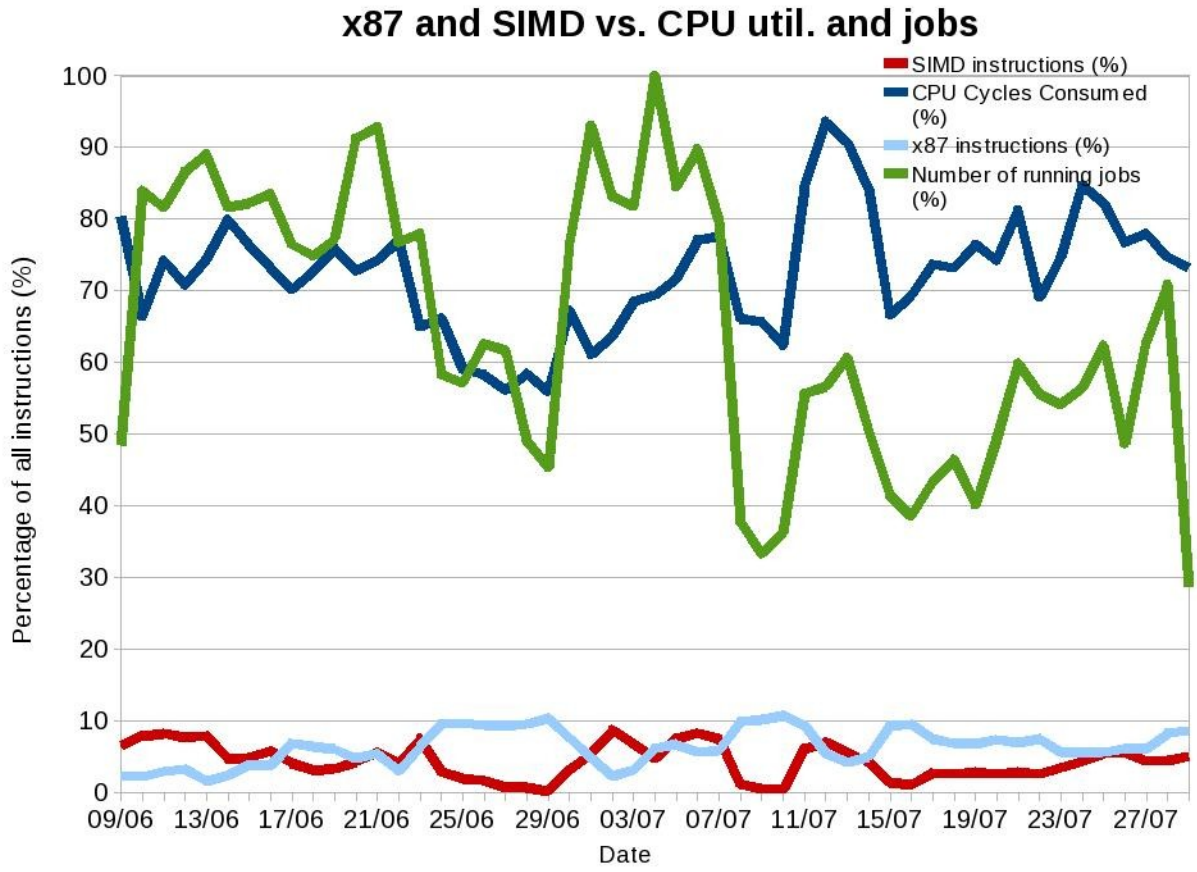
The experiment profile reports contain records when the jobs from the actual experiment make up the majority of the running jobs over a time period, or at most when there is only one other experiment running at the same time. After examining the collected records for each experiment, it was clear that, at least in the recorded time periods, all experiments could be bound to either 32 or 64 bit usage by using almost exclusively x87 or SIMD instructions.

32bit vs. 64bit usage of the experiments:

	Experiment	Principal instr.	Mode
Dominant users	ATLAS	x87	32bit
	ALICE	SIMD	64bit
	CMS	x87	32bit
	NA58	x87	32bit
Minor users	AT	x87	32bit
	IT-GEANT4	SIMD	64bit
	IT-GEAR	SIMD	64bit
	IT-DTEAM	x87	32bit
	LHCB	x87	32bit
	NA45/2	x87	32bit
	PH-TH	SIMD	64bit
	PS212	x87	32bit

8.13.2 SIMD and X87 instructions vs. CPU utilisation and the number of jobs

The following plot shows the percentage of SIMD, x87 instructions, CPU utilisation and the running jobs which is shown as a percentage of the highest number of jobs. The graph allows to compare differences in load caused by 32 or 64 bit applications.



The most interesting fact on the graph is that when mostly 32bit applications are running (x87 is high), both the CPU utilisation and the number of jobs drops down.

The detailed hourly report files showed, that during the peaks of x87 utilisation (32bit jobs), the maximum number of parallel jobs was reduced to 5 or 6 from the average 8-9 on many nodes. Almost all of the remaining jobs were run by the NA58 (COMPASS) experiment, which mainly uses 32bit applications.

As it was explained, the jobs of the NA58 experiment that caused the reduction in the number of parallel jobs and lowered the CPU utilisation were using up all the swapspace on the nodes from time to time. This behaviour inhibited the scheduler from starting new processes, so the number of parallel jobs dropped down, and the CPU utilisation also remarkably decreased.

9. Conclusion

The performance of a large cluster in CERN batch farm was monitored for several weeks in order to get low level, detailed information about the HEP batch computing environment. A low level performance monitoring tool (perfmon) was collecting data for 52 days from 60 standard production batch nodes in the CERN Computer Centre. The main subject of the investigation was to analyse the performance of production batch nodes and reveal possible performance bottlenecks in the infrastructure.

The second goal of the investigation was to analyse the workload that is generated by the different experiments and work groups at CERN. The jobs that were running on the nodes were successfully coupled with the corresponding experiments and groups using the group identifier of the users. The detailed results were then summarised in several different human readable reports to facilitate the analysis.

The monitoring was successful, although some of the nodes were shut down during the data collection due to different errors that were not related to the performance monitoring.

However one issue was found during the data analysis regarding to the bus and data bus utilisation figures. Large jumps, and suspiciously high utilisation values (exceeding 300%) were found. At the moment, the origin of such events is not identified yet. A further investigation is needed to examine and validate the way the bus and data bus utilisation can be accurately calculated.

The overall CPU utilisation of the machines does not reach the optimum 80-90%. An average of 72% utilisation was found, which is coherent with the data from LEMON, CERN's standard monitoring tool. The under-utilisation was partially caused by an issue, when the number of parallel jobs was several times remarkably diminished, thus causing degraded performance. The problem was related to jobs from the NA58 (COMPASS) experiment, that were using up all available swapspace preventing the scheduler from starting new jobs.

Enough data was collected to profile efficiently the workload generated by the most active experiments. However it was very challenging to clearly see the profile of each experiment, since normally the jobs are executed by different user groups. The data filtering was done by a script that selected the most adequate time periods for each experiment. These time periods showed surprisingly consistent workload from most of the user groups, which allowed for profiling.

The profile files allowed to compare the performance of applications run by the different user groups. The distribution of 32bit and 64 bit applications, the CPU or data intensive applications could all be examined.

The information about the usual workload on batch nodes also helps us profiling a benchmarking method that measures the performance under similar load conditions that the computing nodes in production are dealing with, helping the procurement process for new servers.

The current investigation proves that monitoring performance can be a considerable tool for everyday use, to detect performance bottlenecks and support the development of hardware efficient software. For example, if the exceptionally high amount of L2 cache miss ratio at the PS212 experiment, causing a noticeable impact on the whole cluster's performance, was reported back to the developers, not only their applications, but the performance of the entire batch production service would increase.

References:

[1] Intel® 64 and IA-32 Architectures Optimisation Reference Manual (2007, Intel)

[2] Andrzej Nowak – Understanding performance tuning (2008, CERN openlab)

[3] Andreas Hirstius – Results of Low Level Profiling of the SPEC2000 and SPEC2006 (2008, CERN openlab, unpublished)

[4] Paola Cecuk – CERN openlab summer student report: Benchmark (2008, CERN openlab, unpublished)

[5] Understanding the Pareto Principle, (December 2008)
http://betterexplained.com/articles/understanding-the-pareto-principle-the-8020-rule/