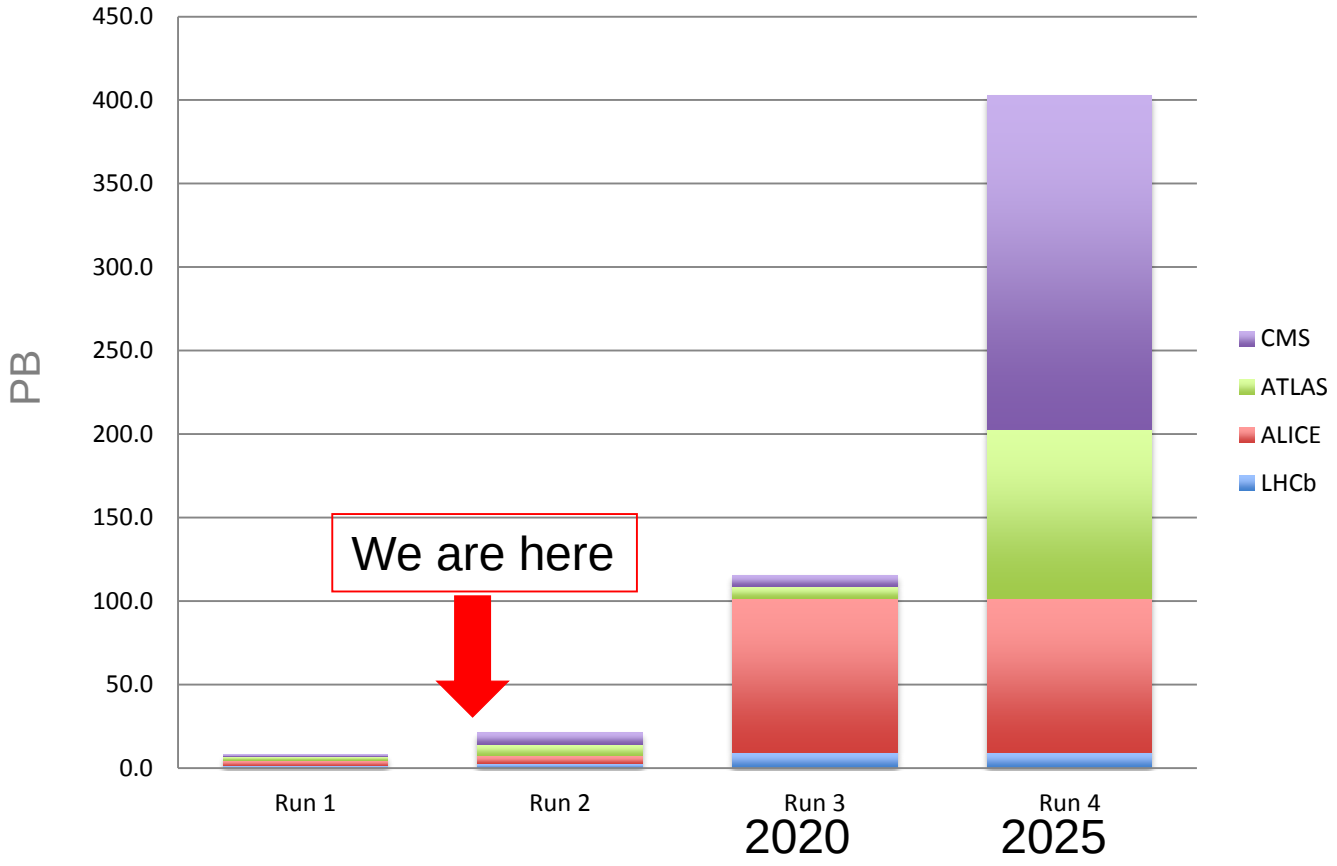
Understanding Performance

Activity in IT

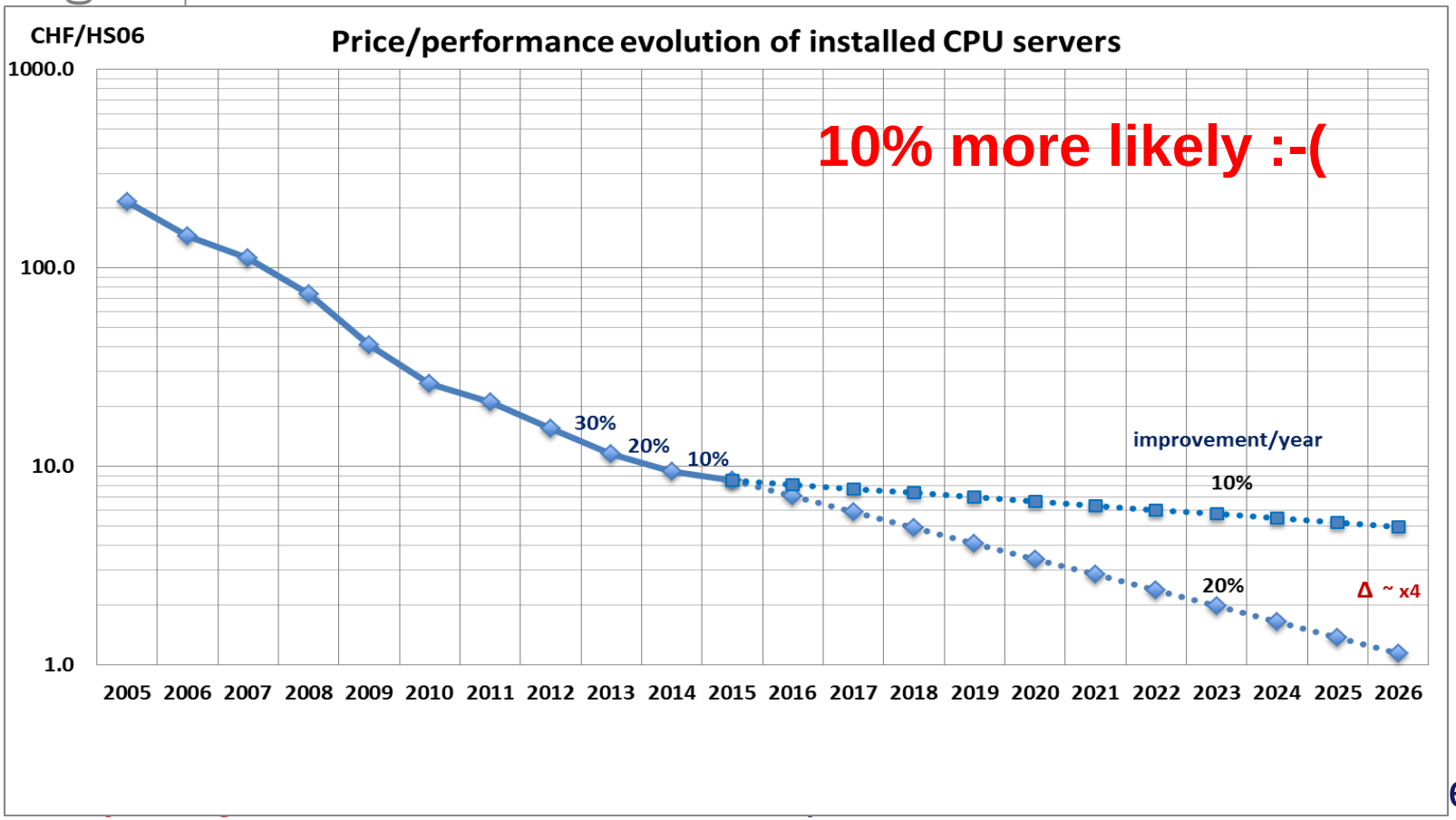
Markus Schulz IT-DI-LCG
markus.schulz@cern.ch

Data: Outlook for HL-LHC



- Very rough estimate of a new RAW data per year of running using a simple extrapolation of current data volume scaled by the output rates.
  - To be added: derived data (ESD, AOD), simulation, user data…

CPU: Online + Offline



- e processing per year of data taking using a simple extrapolation of Run 1 performance scaled by the number of events.

Understanding Performance Team (from 2016)

4FTEs (5 people)

- Software optimisation/performance tools
  - IT contribution to HSF
    - in collaboration with OpenLab and Techlab
  - Support of existing tools
  - Develop and support new tools when appropriate
  - Including improvements to build and test chains to integrate performance tools
  - Provide some resources for testbeds
- Investigations of overall (global) system efficiency
  - Use of analytics on various monitoring and performance metrics
  - Work with existing information and identify missing data
    - close collaboration with experiments and sites
- Development activities for 5-10 year evolution of WLCG
  - Propose & coordinate projects or prototypes to evaluate ideas for the future
- Contribute to the WLCG architecture/planning activities
  - may include stimulating projects together with experiments and sites
  - setting up prototypes
- Modelling of distributed infrastructure and their interaction with workflows
  - Not seen as beneficial by the community
  - At some rudimentary level part of the above

Current Activities

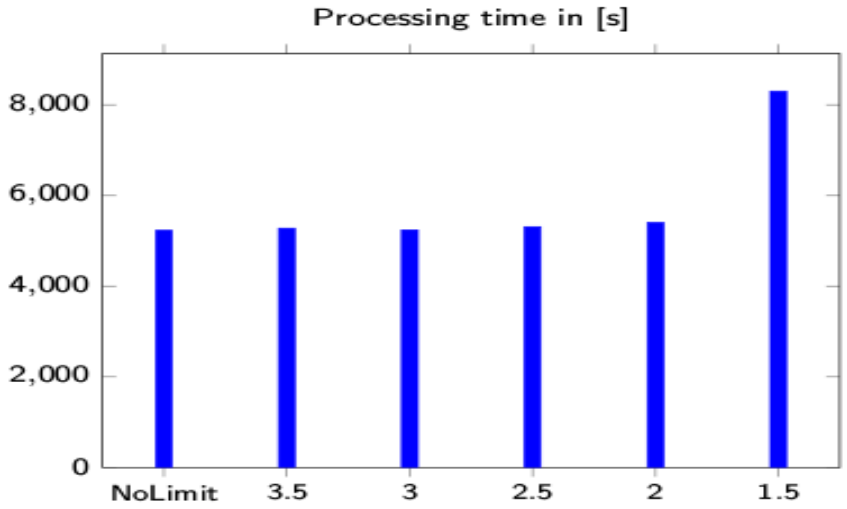
- Developing tools and procedures by working with experiment applications
  - As a first step before generalization
- Trying to make sense of experiment workflow logs
  - Starting with concrete experiment workflows

- Getting an overview of existing activities
  - Within IT
  - Within the community
    - Session at the last HSF workshop
  - Keeping track of them
    - Started to document and summarise

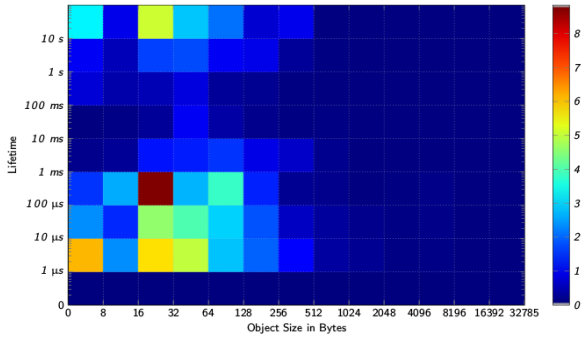
- Helping to link activities
  - Organising the session of performance at the HSF workshop
    - With Vincenzo Innocente and Paolo Calafiura
    - https://indico.cern.ch/event/496146/timetable/
  - Joint meetings on analytics for workflows
  - Atlas Computing Workflow Performance working group
    - https://indico.cern.ch/event/587918/
    - Regular meetings
  - Sessions on performance at the WLCG Workshops
    - https://indico.cern.ch/event/555063/sessions/207262/#20161009

Example Memory:
FOM-tools, x32-ABI, active usage
Nathalie Rauschmayr with Sami Kama

- Understanding how memory is used
  - Tools are now part of the HSF toolset
- Observation:
  - Reducing available memory for ATLAS and LHCb had not the expected impact
  - Most allocated memory is rarely used



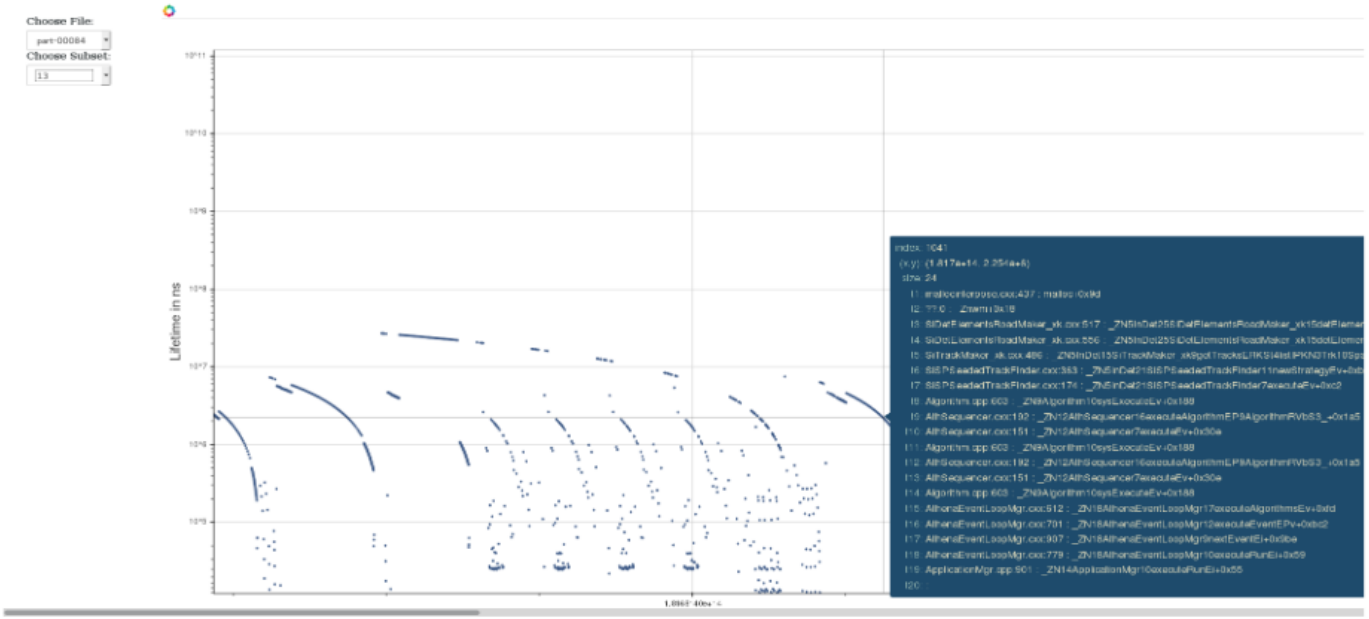
Memory



- Re-evaluation of x32-ABI
  - Combines advantages x86-64 and x86 instruction sets
  - Benchmarks for Alice and Geant4
    - Memory reduction of up to 15%
  - See presentation given at the Concurrency Forum https://indico.cern.ch/event/468210/
- FOM
  - Studies of memory allocation patterns in the time and size domain
  - 60-90% of allocations live less than 100µsec
    - After malloc+free+constructor call not too much time left...
  - 70-95% of allocations are smaller than 64 Bytes
  - Identified several reasons: wrong data structs (std::list / std::vector)
  - Arguments passed by value → copy constructor
  - Fragmentation?

New look at memory and applications …

- Allocation patterns study stimulated a lot of discussions
- Toolset to be used by developers will be made available
  - Simplified functionality added to Valgrind (Summer Student with OpenLab)
- Nice Web based interface to explore the patterns
- Object lifetime gives inside into data structures and algorithm
  - Raises some fundamental questions concerning performance and C++



Some application of FOM-Tools

- Several interesting problems have been spotted (and addressed)
  - ATLAS excessive data read from DB
  - ATLAS analysis jobs / ROOT interaction
    - Lead to improvements in ROOT
  - ATLAS/ROOT issue on memory usage during histogram I/O

Compiler based project

- Self-tuning code with AutoFDO
  - Feedback directed optimisation for gcc from Google
    - Collects performance metrics from running jobs then uses these at compile/build time
  - Geant-4 detector simulation as a proof of concept
    - Indication of 10-15% performance gain
  - See last GDB for details: https://indico.cern.ch/event/394788/contributions/2357347/attachments/1368686/2074705/slides.pdf
- Comparing workloads from different experiments on different CPUs
  - Geant4 ATLAS sees 50% performance increase on Haswell over Ivy Bridge
    - 10% difference indicated by the HS06 benchmark!
    - CMS didn't show the same increase
  - Follow up tests achieved improvement of ~13% for commonly used Intel microarchitectures prior to Haswell (for ATLAS)
    - Initial proof of concept: targeted removal of some of the overhead associated with position independent code by patching the process at runtime
    - practical steps that could be adopted to get gains, two approaches tried which also showed positive results
      - Enable the linker to eliminate overhead (like CMS)
      - Compile geant4 non-position independent, with x64 large code model ~10%
        - Easier but less gain

IPC for some jobs

	Haswell IPC	Ivy Bridge IPC
CMS, Gen	1.59	1.58
CMS, Sim	1.47	1.18
CMS, Digi + HLT	1.66	1.53
CMS, Reco	1.55	1.4
ATLAS, Sim (19.2.4.9)	1.43	0.92
ATLAS Sim (20.7.8.5)	1.42	0.96
ATLAS, HITtoRDO	1.74	
ATLAS, RDOtoRDOTrigger	1.45	
ATLAS, RAWtoESD	1.51	

- Ratio of retired instructions / unhalted clock cycles (most over whole job). Physical machine.
- Atlas simu with single process athena, HT on, affinity fixed to 1 core. No other significant load.
- Haswell was Xeon E5-2683 v3 (~3GHz); Ivy Bridge i7-3770k (~3.8GHz)
- checked Ivy Bridge also on Xeon E5-2695 v2 (~3.1GHz) running ATLAS Sim (19.2) => 0.91 IPC
- checked Atlas simu (19.2) with athenaMP (8) affinity to 4 cores on one socket => 1.58/0.97 IPC
- ATLAS sim was job 2972328065 (19.2.4.9, slc6-gcc47-opt or 20.7.8.5, slc6-gcc49-opt; mc15_13TeV.362059.Sherpa_CT10_Znunu_Pt140_280_CFilterBVecto_fac4)
- Looked up previous HS06 results; usually ~10% higher for Haswell (per job slot/per GHz)

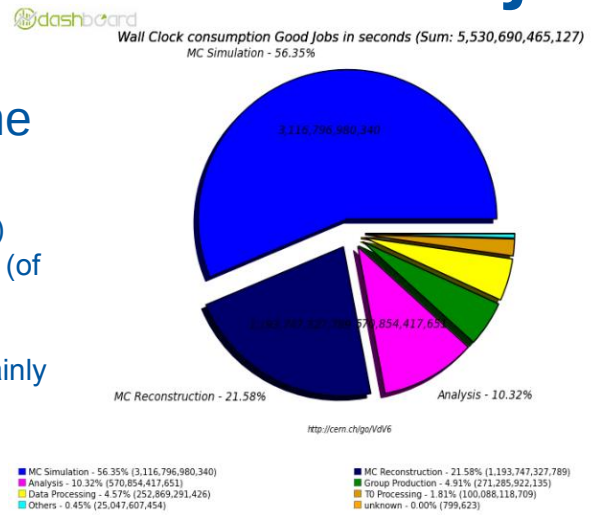
Mining Experiment Logfiles

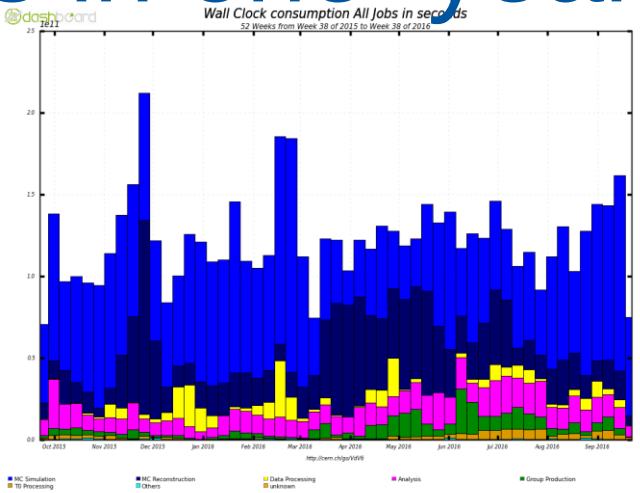
- Analysis of CPU utilization of production jobs
  - Outside the T0, but will be included in the future
- Understand how experiments use their CPU resources
  - Are they “efficient” (i.e. do they waste wall-clock time)?
  - How can jobs be modelled in the context of a simulation of the WLCG computing infrastructure?
  - How many resources are required?
- Using data analytics techniques, understand the behavior of the infrastructure
  - Can we validate commonly used benchmarks using “real” jobs?
  - Can we measure the “speed” of CPUs, or sites, by looking at different types of jobs? Are the results compatible?

Example: ATLAS jobs in one year

By wall-clock time

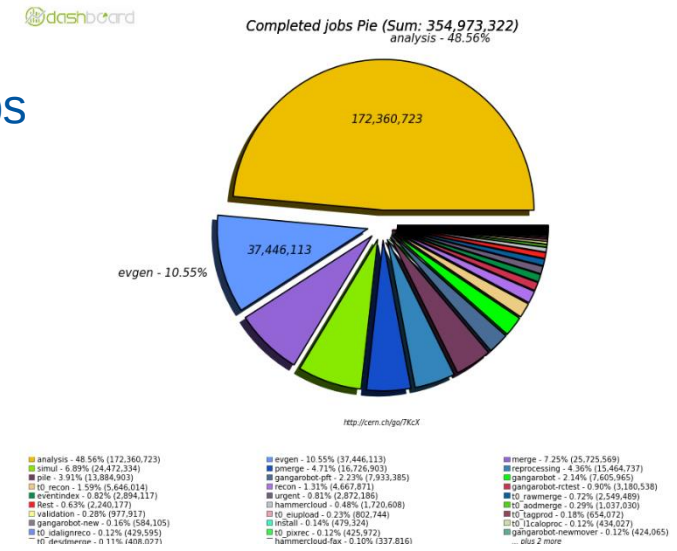
- MC Simulation: ≈ 60% (of which 2/3 simul, 1/3 evgen)
- MC Reconstruction: ≈ 22% (of which 4/5 pile)
- Analysis: ≈ 10%
- Data processing: ≈ 5% (mainly reprocessing)
- Group production: ≈ 5% (mainly merge)

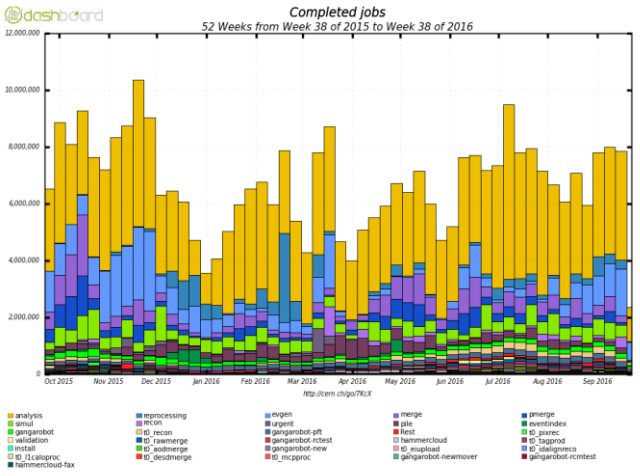




By number of jobs

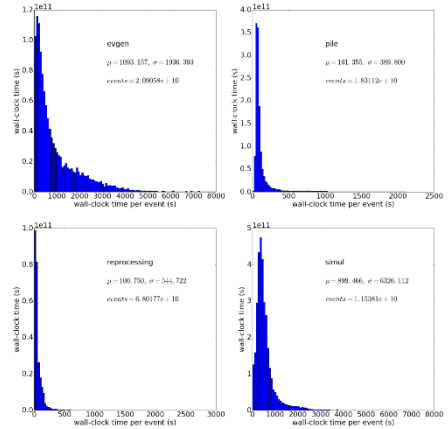
- Analysis: ≈ 50%
- Evgen: ≈ 10%
- Merge: ≈ 7%
- Simul: ≈ 7%
- Pmerge: ≈ 5%
- Reprocessing: ≈ 4%
- Pile: ≈ 4%

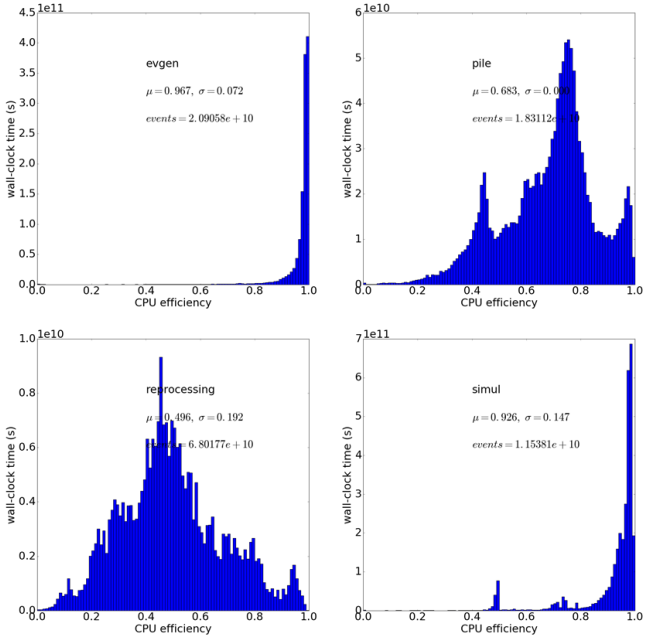




ATLAS wall-clock per event and CPU efficiencies

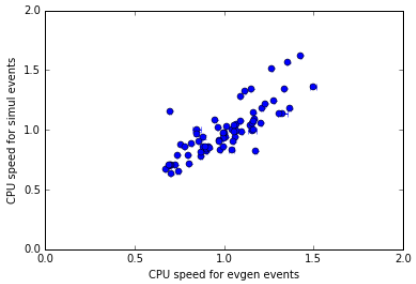
- Event generation and simulation very CPU intensive and efficient
- Reconstruction (pile, reprocessing) faster but inefficient (due to heavy I/O)

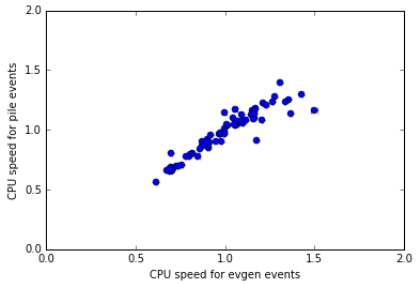


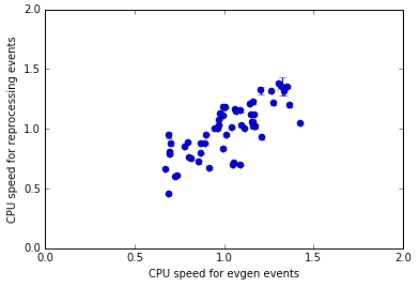


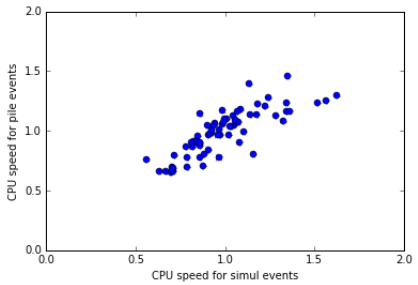
Speed correlations in ATLAS

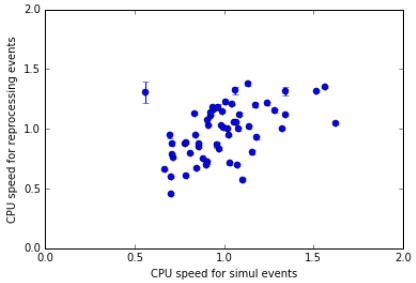
- CPU factors from different job types are tightly correlated
- Agreement is around 4%
- The conclusion is that all jobs are acceptable benchmarks if CPU time is measured

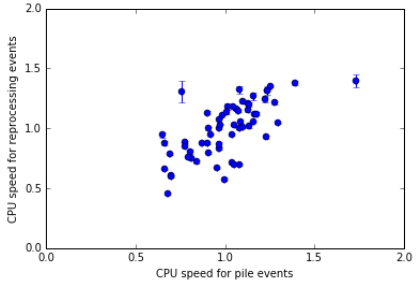






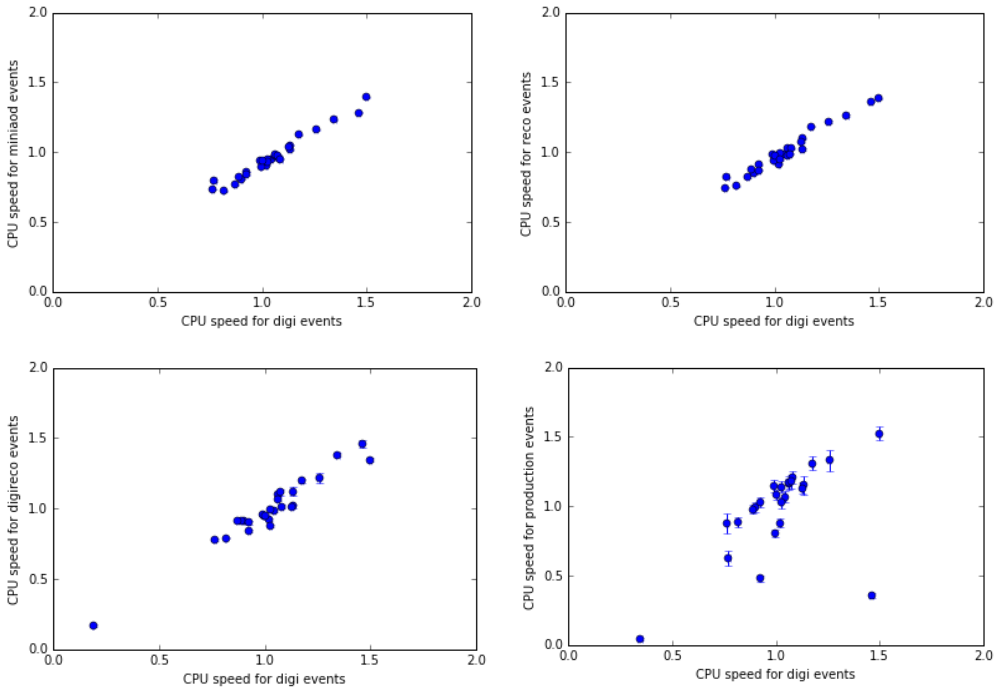






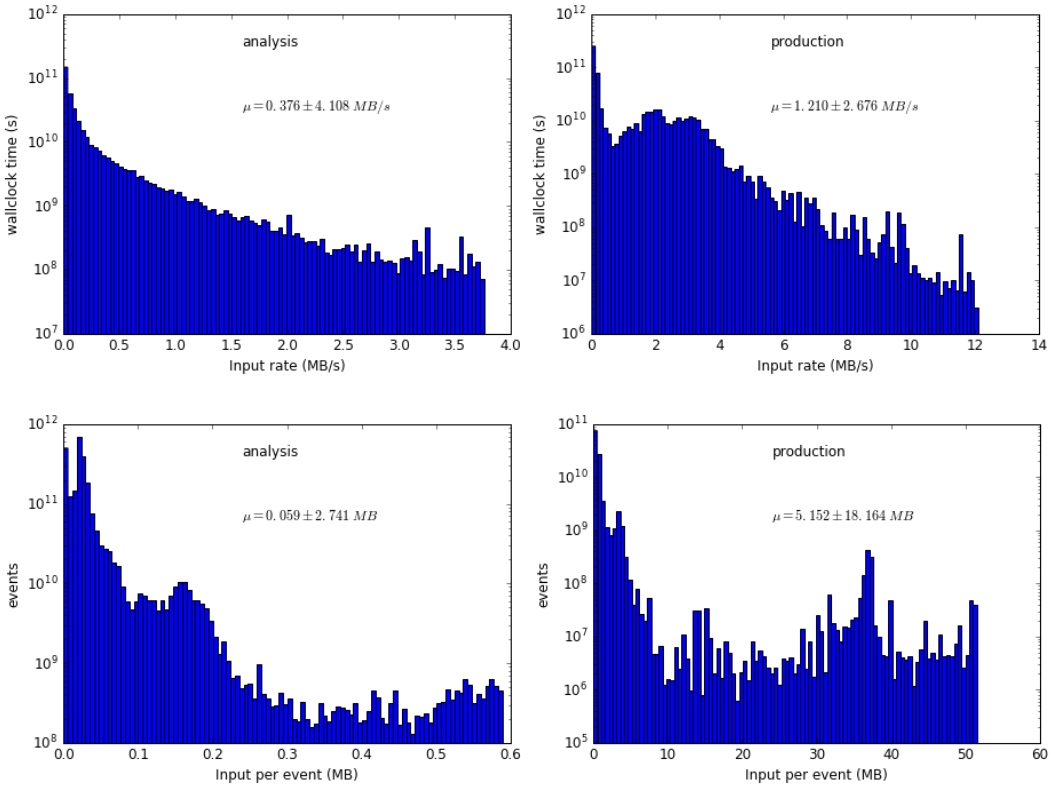
Speed correlations in CMS

- Similar results are obtained on CMS jobs
  - Speed factors agree by ~4% on average
- All jobs can be used as benchmarks



CMS production vs. analysis

- Analysis jobs are relatively I/O-light
  - They usually run on small format events
- Production jobs can be very I/O intensive for certain workflows



ATLAS job inefficiencies (1/2)

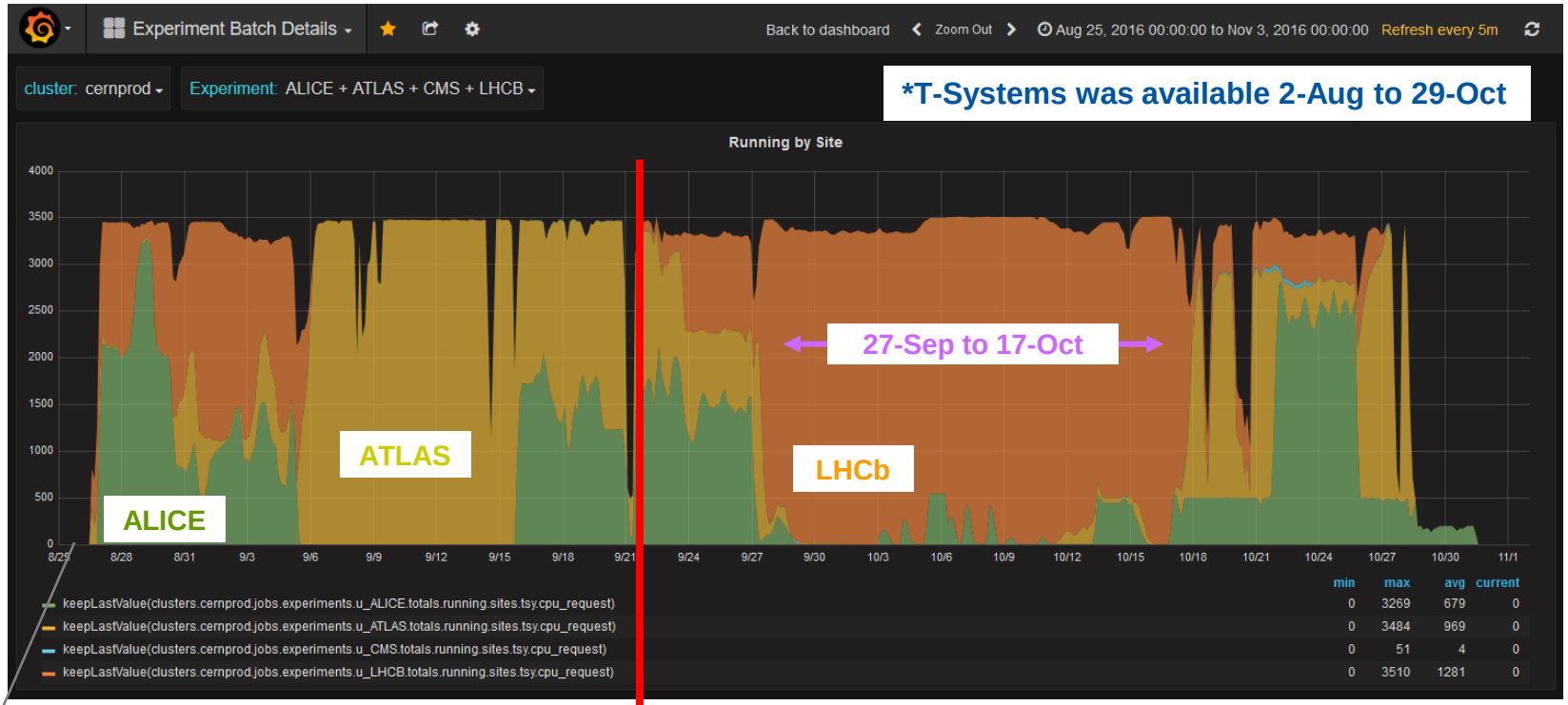
- New analysis, just started
- Goal is to optimize ATLAS workflow submission using as metric:

  (CPU time of good jobs) / (wallclock time of all jobs)
- Examples:
  - Too short jobs can be inefficient due to time spent on initialization phase
  - Too long jobs can waste a lot of wall clock time if they fail

How do our workloads behave in Clouds?

- Local
- Commercial
  - There are many different ones
- Data local/staging/caching/remote???
  - Impact
  - Costs

LHCb and other VOs on T-Systems 2016*



2-Aug to 21-Sep
see my previous talk on 26-Sep

LHCb tests on a "small scale"
Left the floor to ALICE/ATLAS

(Grafana monitoring started only on 28-Aug)

21-Sep to 29-Oct: this talk!

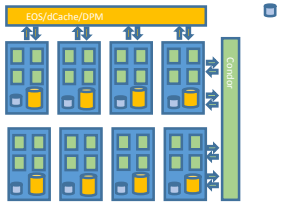
LHCb stress test 27-Sep to 17-Oct on 3k - 3.5k cores
(thanks to ATLAS and ALICE for returning the favor ;-)

Updated (positive and negative) feedback

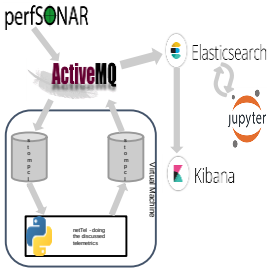
- **CPU resources – very good!**
  - It has been easy to use these CPUs managed by IT via a batch queue
    - VAC/Vcycle remains the preferred cloud model for CPUs managed by LHCb
  - In particular, it was much easier to use these CPUs than those of DBCE
    - Learnt the lesson that CPUs must (at least seem) to come from a single site
    - No major issues from the resources themselves or their operation
  - LHCb needs a dedicated CE for monitoring and configuration
    - Thanks for having provided that! (LHCb does/will not use ClassAd)
  - On future production cloud: would like to have MJF, as on all LHCb sites; should also clarify WNs are responsibility of provider and agree GGUS support model
- ***Network – the bottleneck, but ~OK even for some data-intensive workflows***
  - *Not a problem for MCSimulation (NB: MC is LHCb's main use case on clouds!)*
  - *The latest tests show it was ~OK for DataReconstruction and even DataStripping*
    - *Provided ATLAS and ALICE are not there! academic question, not a realistic use case*
    - *The links to some T1 sites (CNAF) were weaker than others*
  - *Not enough for Merge jobs (not a good idea anyway, another academic question)*
- **Cloud storage – not interested**
- Thanks for setting up this meeting, it is useful to sit together sometimes
  - Also to hear the experience from the other experiments

Building Prototypes and testing Tools

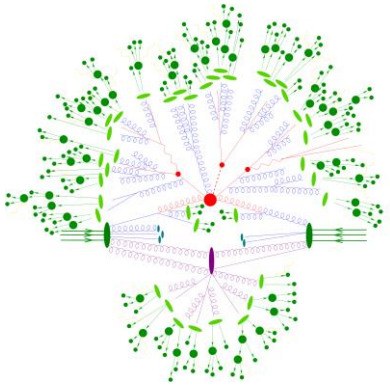
- Opportunistic computing on storage servers (Orthogonal Scheduling)
  - Most WLCG storage servers have low CPU utilisation
  - Prototype has been build
  - First measurements



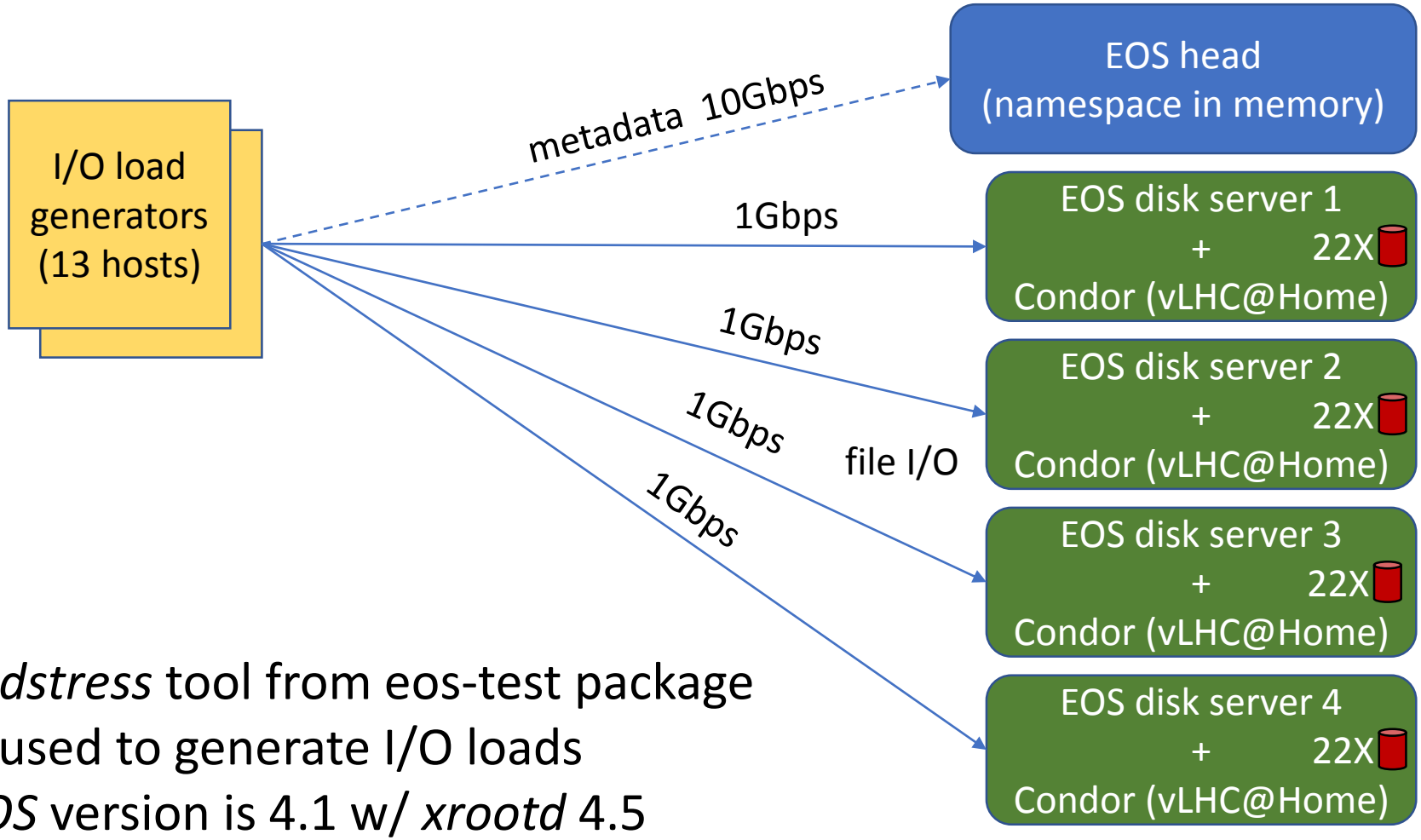
- Prediction of available network bandwidth/congestions:
  - Using packet loss, latency, historical data
  - Analytical model
  - Machine learning based predictor
  - Goal: cost model



- Compiler comparison: Event generator Sherpa
  - Different gcc versions
  - Started to port to Intel C++ compiler
    - With openLab
  - Test case to use code analysis and optimisation tools



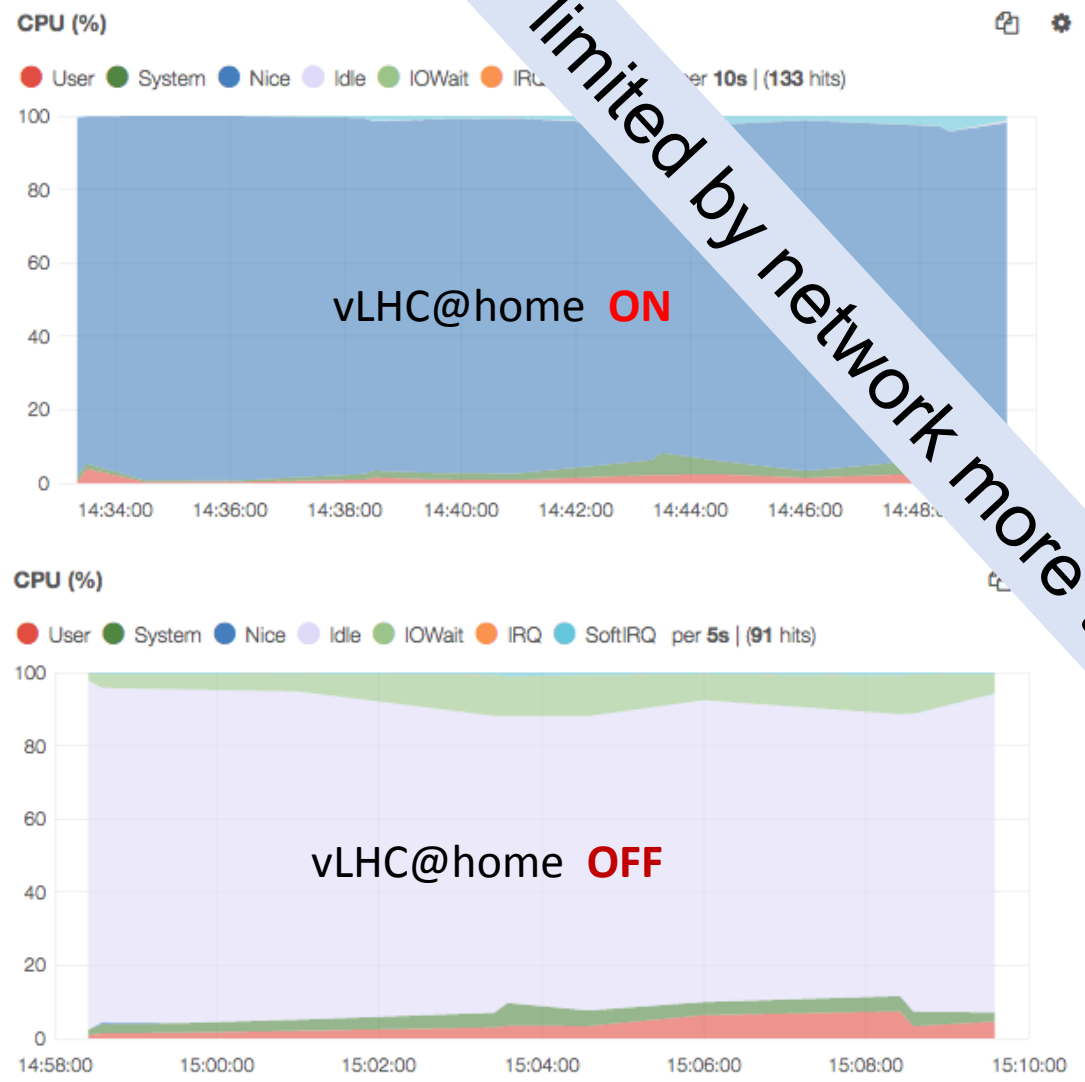
Hybrid testbed



- *Xrdstress* tool from eos-test package is used to generate I/O loads
- *EOS* version is 4.1 w/ *xrootd* 4.5
- *Condor* runs payloads directly, no virtualization involved

EOS performance measurements

Xrdstress [illegible] EOS was running with (top) and without (bottom) vLHC@Home [illegible]esses in the background.



But I/O limited by network more tests needed!

No significant difference in I/O numbers:

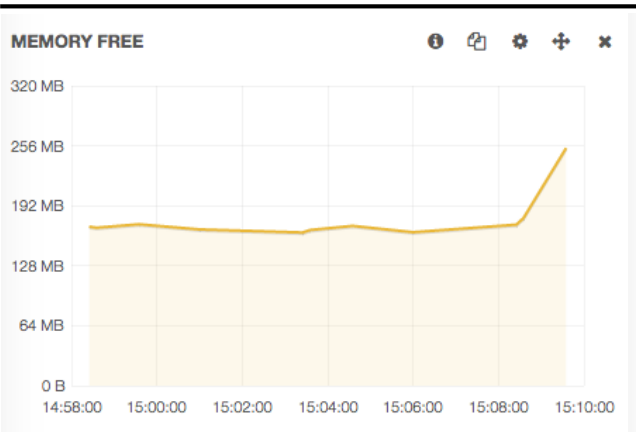
- 357 MB/s read
  91 MB/s write
  in hybrid mode
- 357 MB/s read
  [illegible]6 MB/s write
  [illegible]OS-only mode

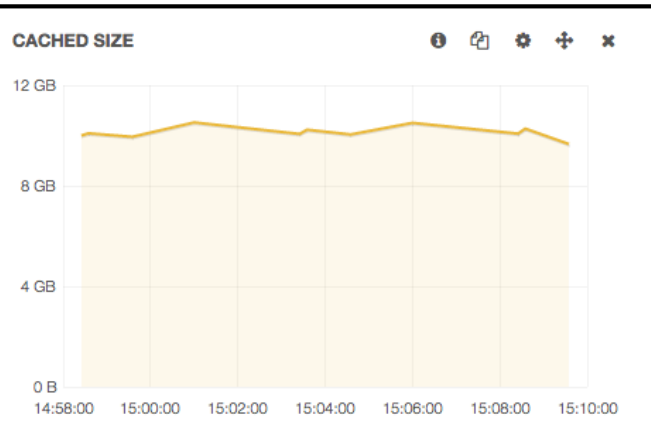
What about memory?

No significant difference, but memory usage is more optimal when vLHC@Home is running (remember: free memory is wasted memory)

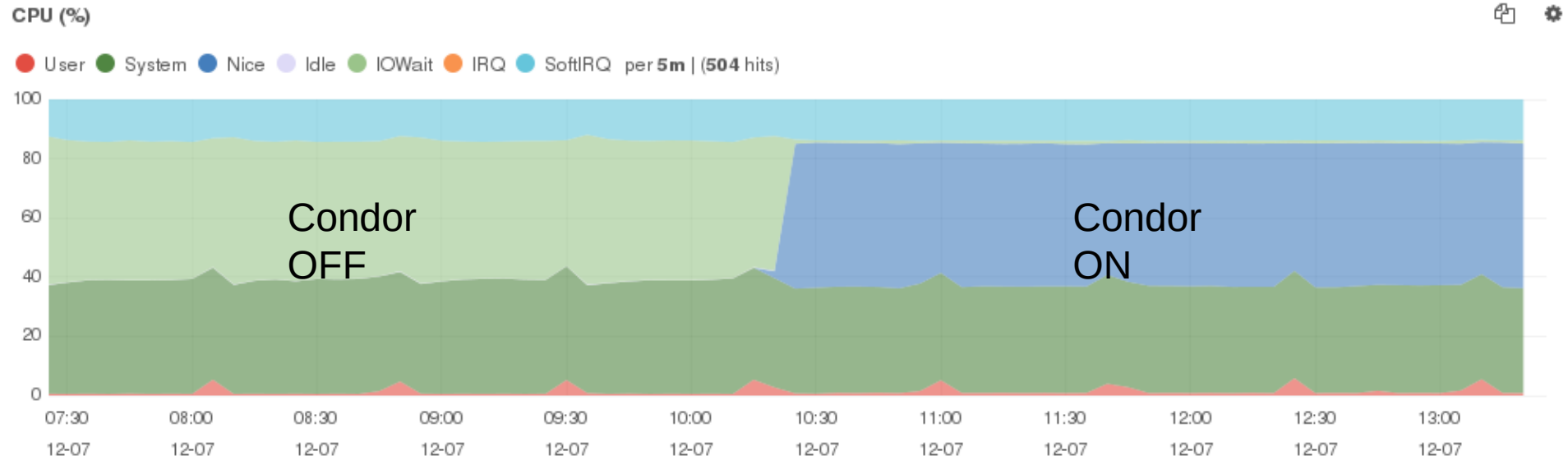
Pure EOS

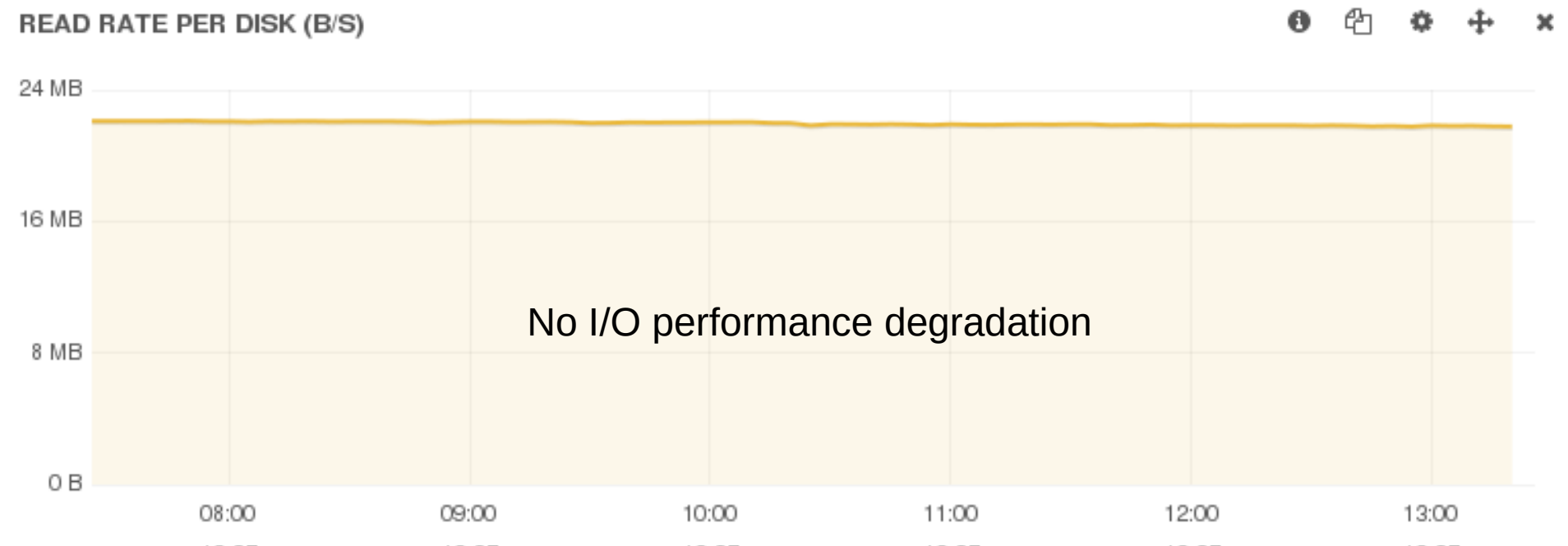
Hybrid mode





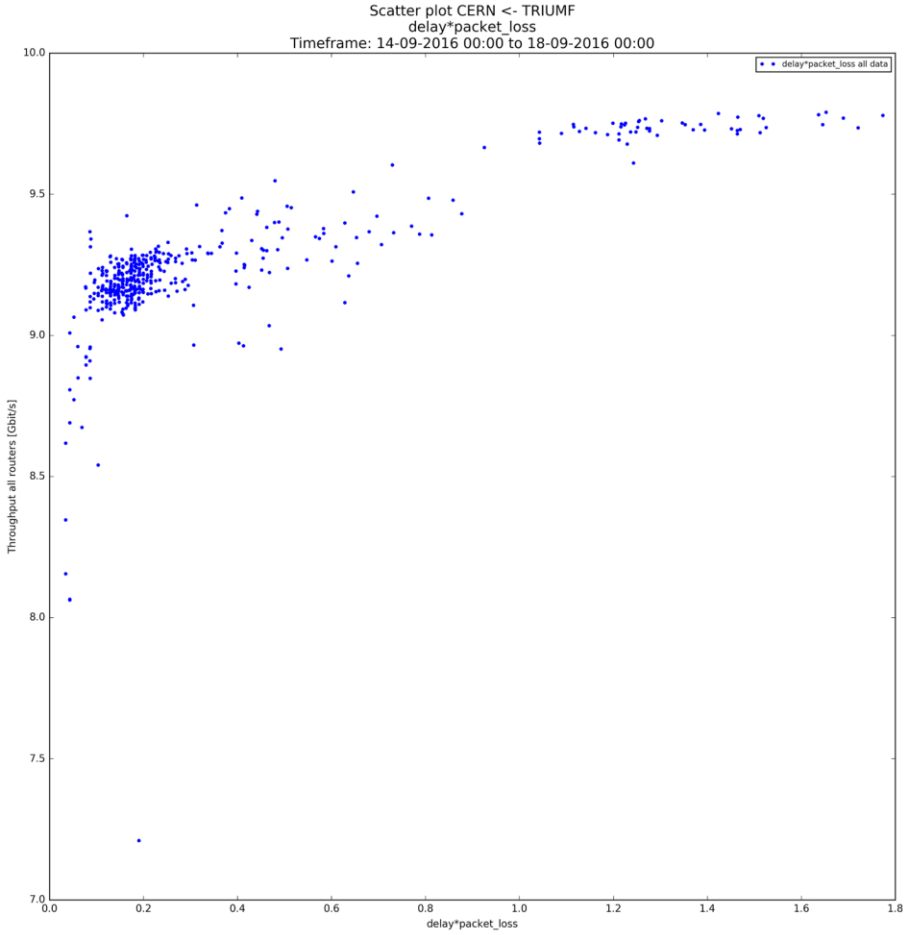
Corner case: 100% busy I/O



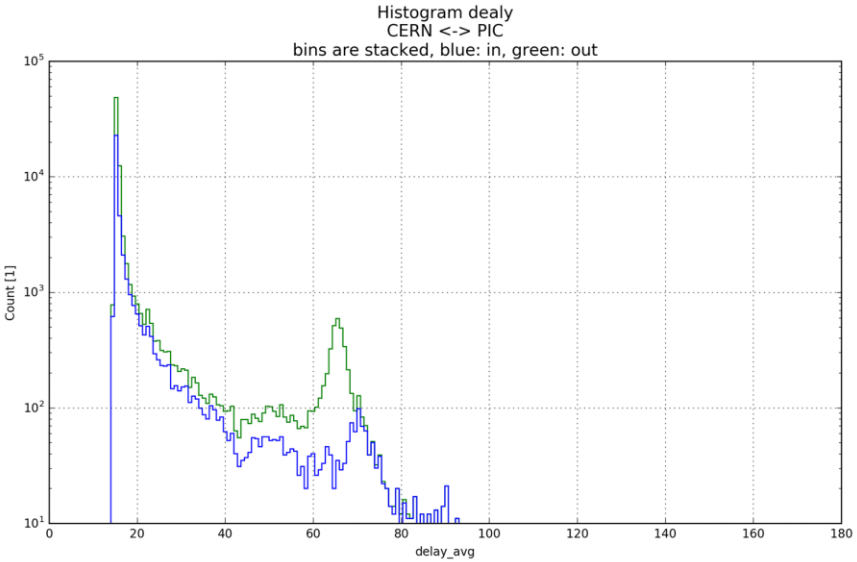


Network Analytics Activities

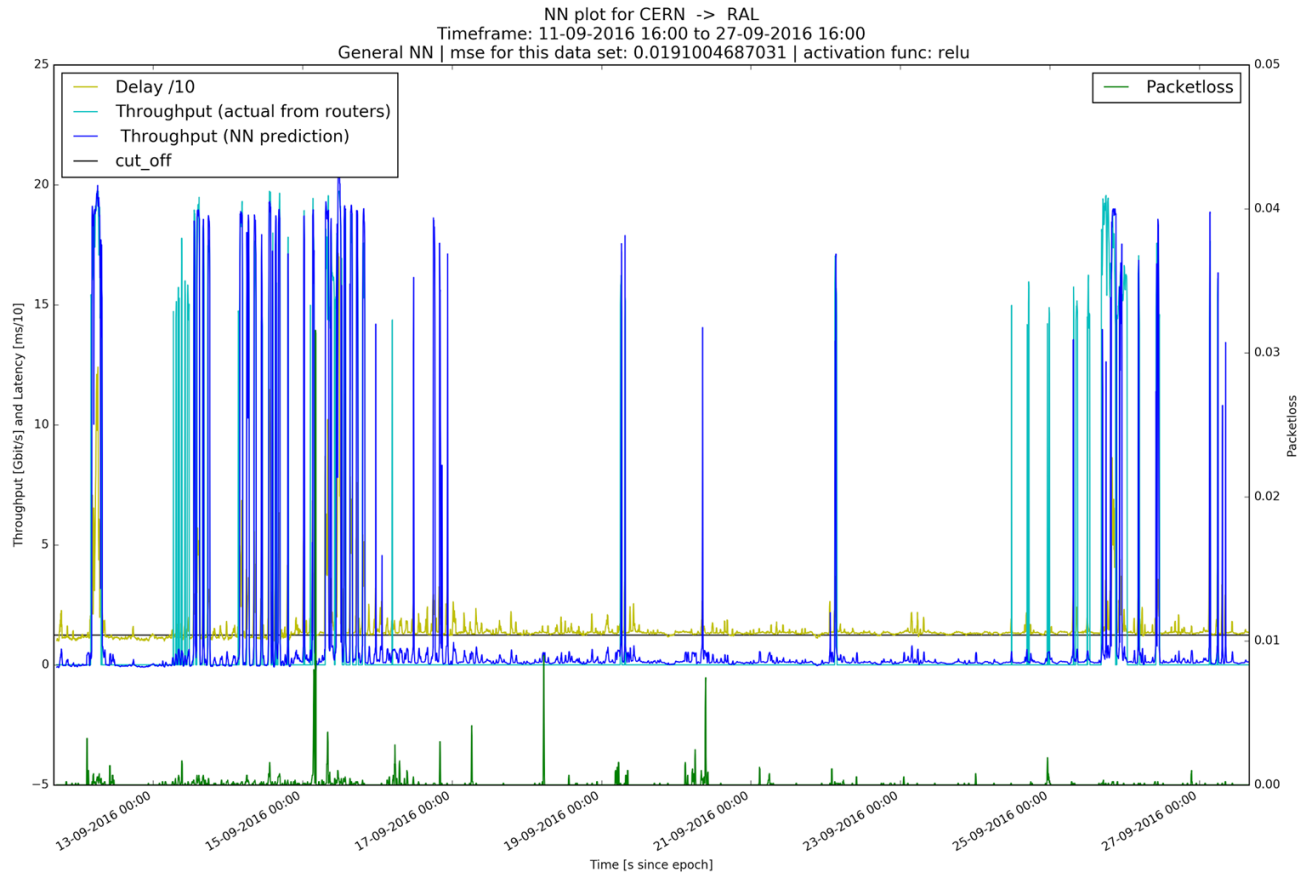
Can it work ?



- Multiplied delay and packet loss shows good correlations for high throughput (above ~80%)
  - Only clearly visible on incoming connections
  - Probably because we only see the routers at CERN
- Distributions of delays have a clear tail towards higher delays



Predicting Congestions with machine learning



Inputs to the Neural Network

- Smoothed delay & packet loss
- 15 measurements from the past (15 min)
- Average, standard deviation and minimum over the whole observation time

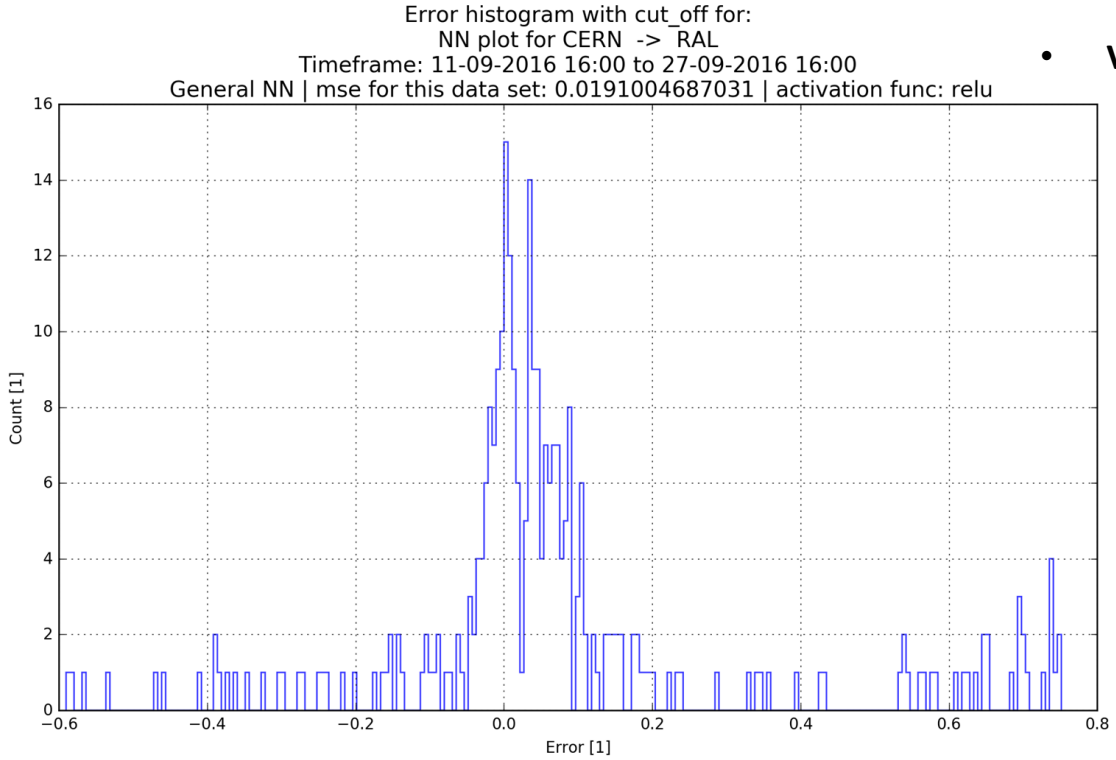
Architecture of the NN

- Three hidden layers
- Decreasing number of neurons
- Activation function: rectified linear unit

Training sets for the NN

- Prediction if a connection is used above 70% of its capacity
- Training on all connections but RAL
- Verification on RAL -> CERN

Predicting Congestions with machine learning



The error is given in a measure of percent,
where 0.8 means the NN is 100% above the real value

- **Verification Results**
  - Mean squared error on verification set: 0.019
  - Seems to be working on a connection that the NN has never seen before
  - Quite good prediction of throughput spikes
  - Distribution of errors shows that the NN is likely not overtrained
  - In the error plot predictions below the cut_off line in the previous plot were excluded

Next Steps

- Memory
  - Continue to work on the tools to understand memory FOM etc.
  - Simplify the use of the tools
  - Identify patterns in the code that leads to many very short lived chunks
- CPU
  - Invest more in the understanding of the use of hardware counters
- Workflows
  - Classification of the “Zoo”
    - To generalise across experiment boundaries
  - Understanding the reasons for differences
    - Goal: Cost functions
- Tools
  - The team is currently building expertise
    - AutoFDO, Intel Compiler, Vtune
- Prototypes
  - Network forecast to a product
  - “Orthogonal Scheduling” testing on production systems


CERN

Additional, not necessarily coherent material

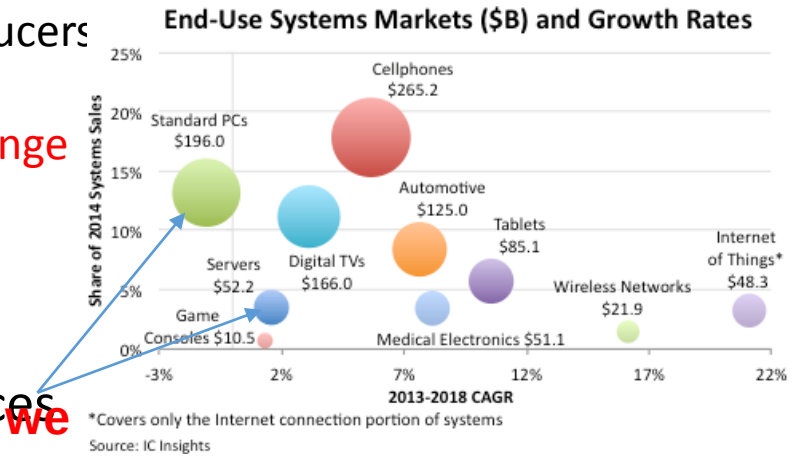
More

What does Moore's Law and friends offer?

- Moore's 1st law: Number of components/chip double every 24 months
- House's law: Performance doubles every 18 months
  - Smaller struct. → higher frequency
- Kryder's law: disk storage density doubles every 18 months
- Butters' Law of Photonics: data rate of a fibre doubles every 9 months
- Pollack's Rule: Architecture Gain ~ sqrt(#transistors)
  - In addition to frequency etc.
- Proebsting's Law: Compilers double code efficiency every 18 years
- ……..
- All this would give over 10 years a factor of "only" 50- 120
  - Ignoring market, usability of new architectures, …….

And it is unlikely to happen…..

- Moore's 2nd (Rock's) law: cost of semiconductor fabs grow exponentially
  - Current generation: ~ 16B$ per fab
  - 2003: 25 state of the art producers
  - 2015: 4
  - → not much incentive for change
  - → or competition
- Same pattern for disks
  - Just worse…
- Market shifts to mobile devices
- General slowdown
  - INTEL moved away from TickTock
    - Now two architecture changes for one new hardware gen.
  - International Technology Roadmap for Semiconductors (industry oracle)
    - expected to adjust their forecasts



CPU utilization of production jobs

Andrea Sciabà

Motivation

- Understand how experiments use their CPU resources
  - What types of jobs are (primarily) run?
  - How many resources do they require?
  - Are they “efficient” (i.e. do they waste wall-clock time)?
  - How can jobs be modelled in the context of a simulation of the WLCG computing infrastructure?
- Using data analytics techniques, understand the behavior of the infrastructure
  - Can we measure the “speed” of CPUs, or sites, by looking at different types of jobs? Are the results compatible? Can we validate commonly used benchmarks using “real” jobs?

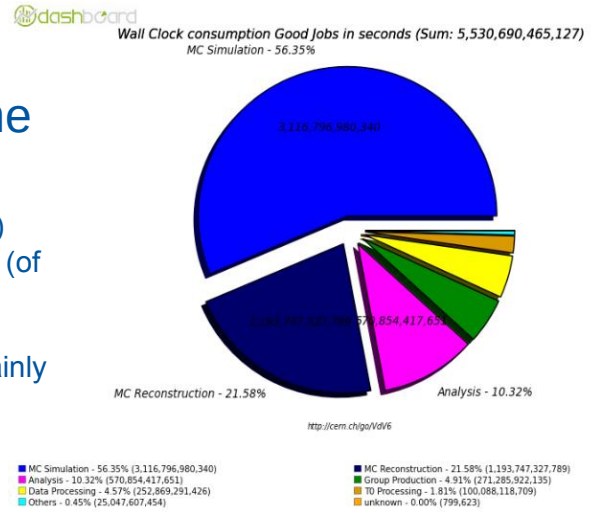
ElasticSearch

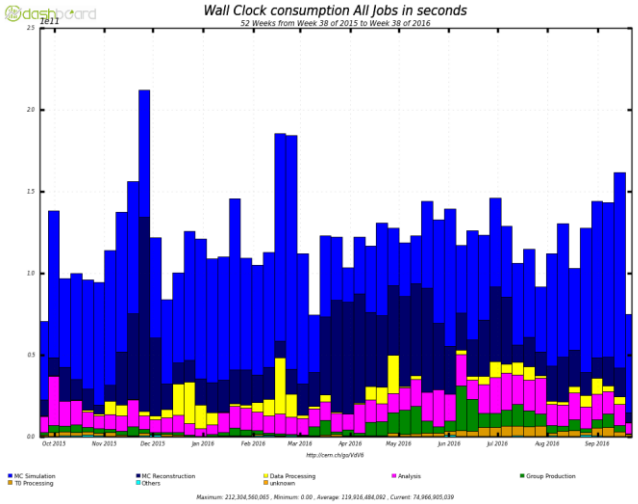
- ES is an incredibly convenient tool for data analysis
  - Both ATLAS and CMS have instances with data from their computing systems
    - Job information, data transfers, etc.
    - In the process of being migrated to the CERN IT ES service
- Interactive data analysis based on SWAN
  - Aggregated data from ES is analysed using notebooks

Example: ATLAS jobs in the last year

By wall-clock time

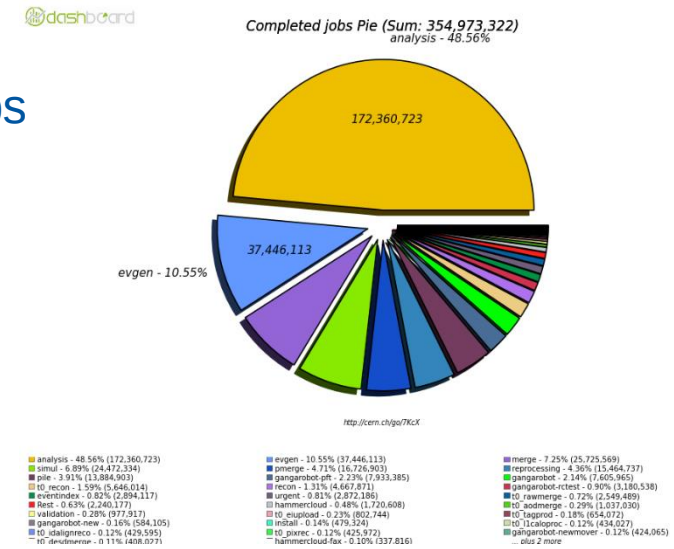
- MC Simulation: ≈ 60% (of which 2/3 simul, 1/3 evgen)
- MC Reconstruction: ≈ 22% (of which 4/5 pile)
- Analysis: ≈ 10%
- Data processing: ≈ 5% (mainly reprocessing)
- Group production: ≈ 5% (mainly merge)

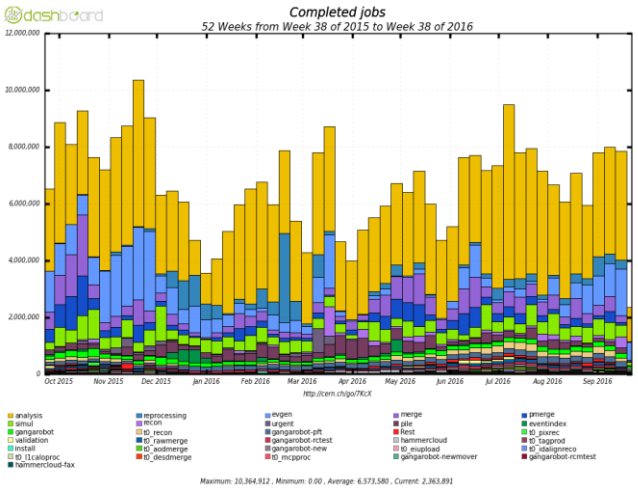




By number of jobs

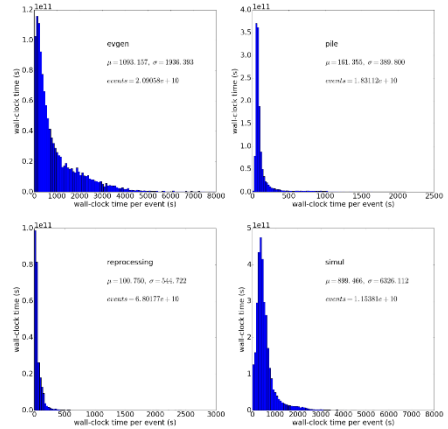
- Analysis: ≈ 50%
- Evgen: ≈ 10%
- Merge: ≈ 7%
- Simul: ≈ 7%
- Pmerge: ≈ 5%
- Reprocessing: ≈ 4%
- Pile: ≈ 4%

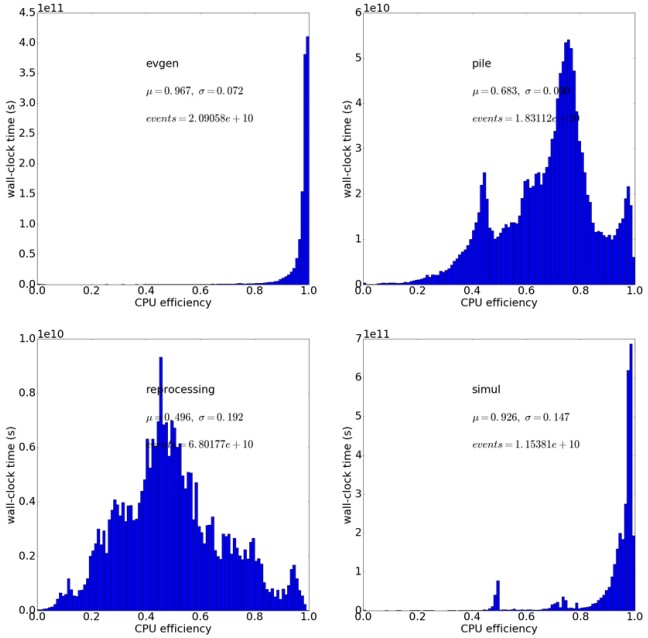




ATLAS wall-clock per event and CPU efficiencies

- Event generation and simulation very CPU intensive and efficient
- Reconstruction (pile, reprocessing) faster but inefficient (due to heavy I/O)



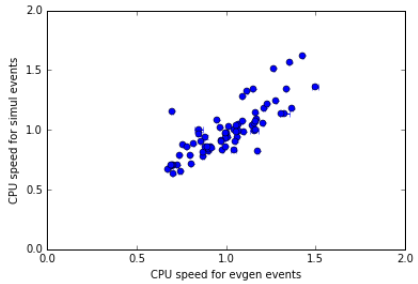


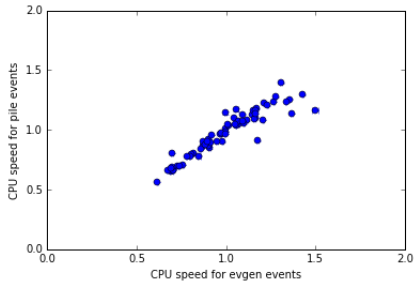
Fitting CPU speeds

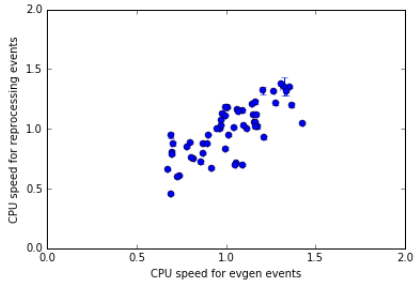
- Assume that CPU “speed” is inversely proportional to CPU time / event
- Compare CPU time /event for similar jobs (from same “task”) on different CPUs to extract relative speed factors
- Repeat for different types of jobs and compare
- The goal is to “benchmark” CPUs with real jobs

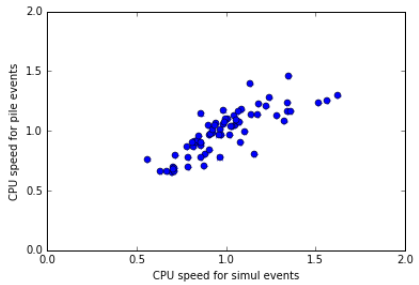
Speed correlations in ATLAS

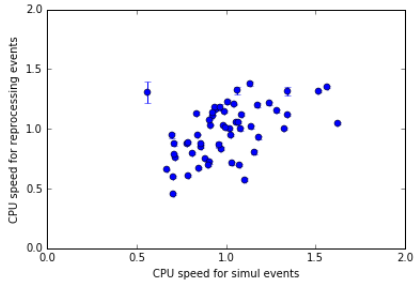
- CPU factors from different job types are tightly correlated
- Agreement is around 4%
- The conclusion is that all jobs are acceptable benchmarks if CPU time is measured

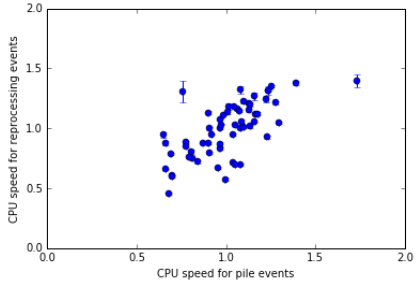






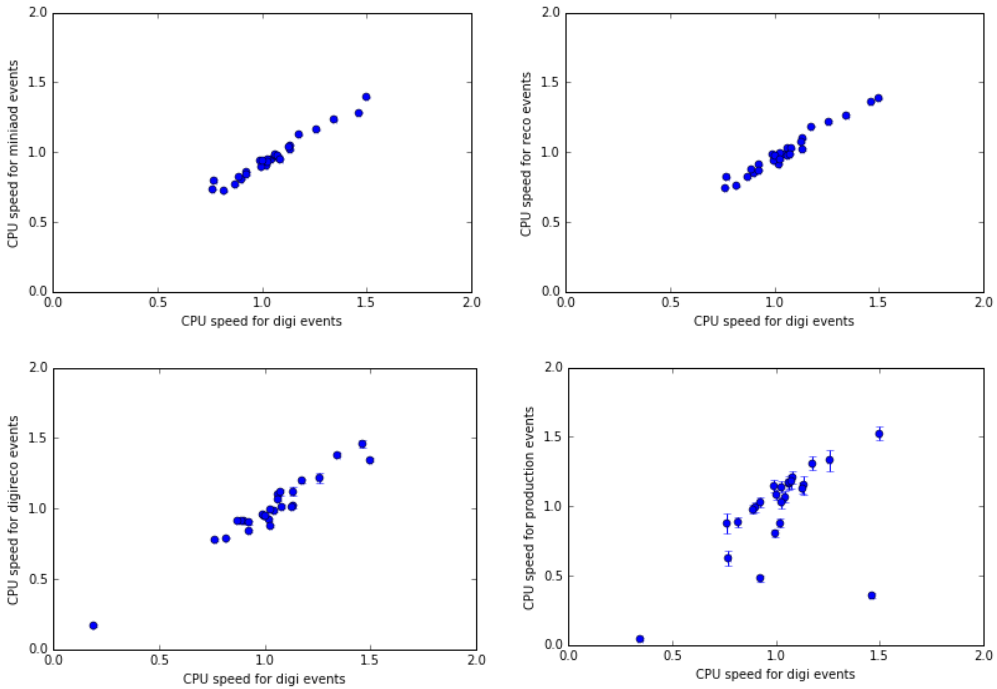






Speed correlations in CMS

- Similar results are obtained on CMS jobs
  - Speed factors agree by ~4% on average
- All jobs can be used as benchmarks



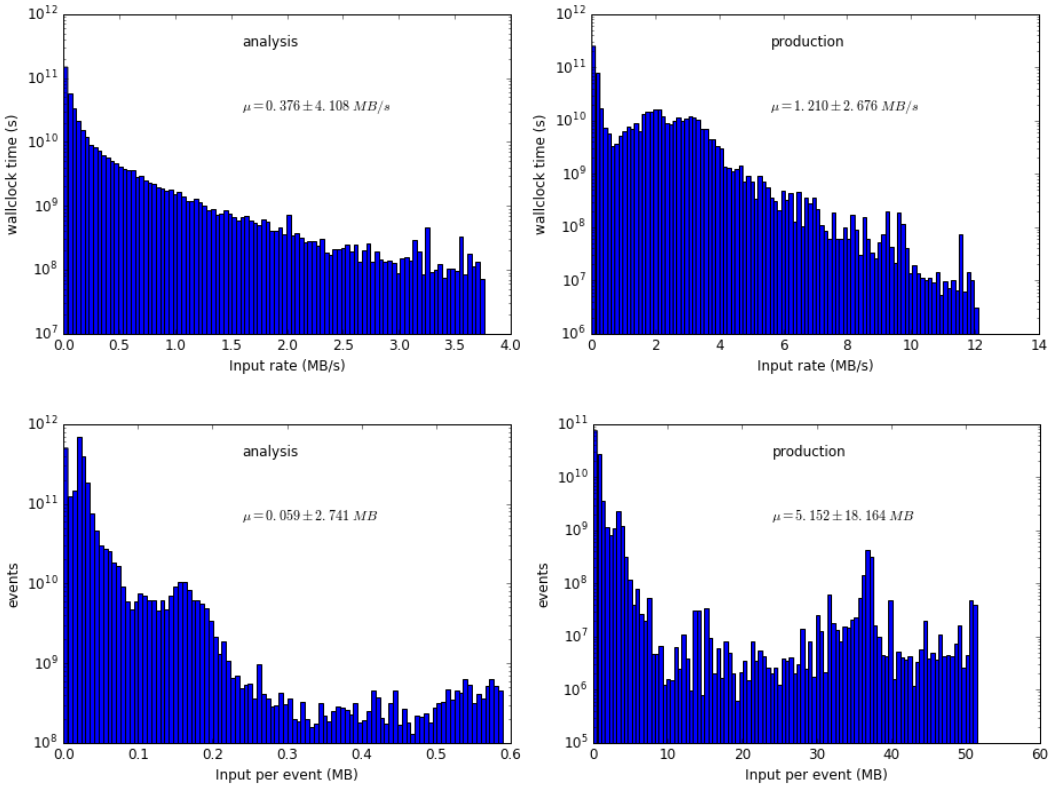
I/O of CMS jobs

IO metrics

- Using ElasticSearch it is possible to also study I/O metrics for different types of jobs
  - Input/output IO rates
  - Input/output data per event and per job
- IO patterns are highly relevant for sites
  - Network capacity, storage scalability, etc.
- And for WLCG
  - Resource utilization, modelling

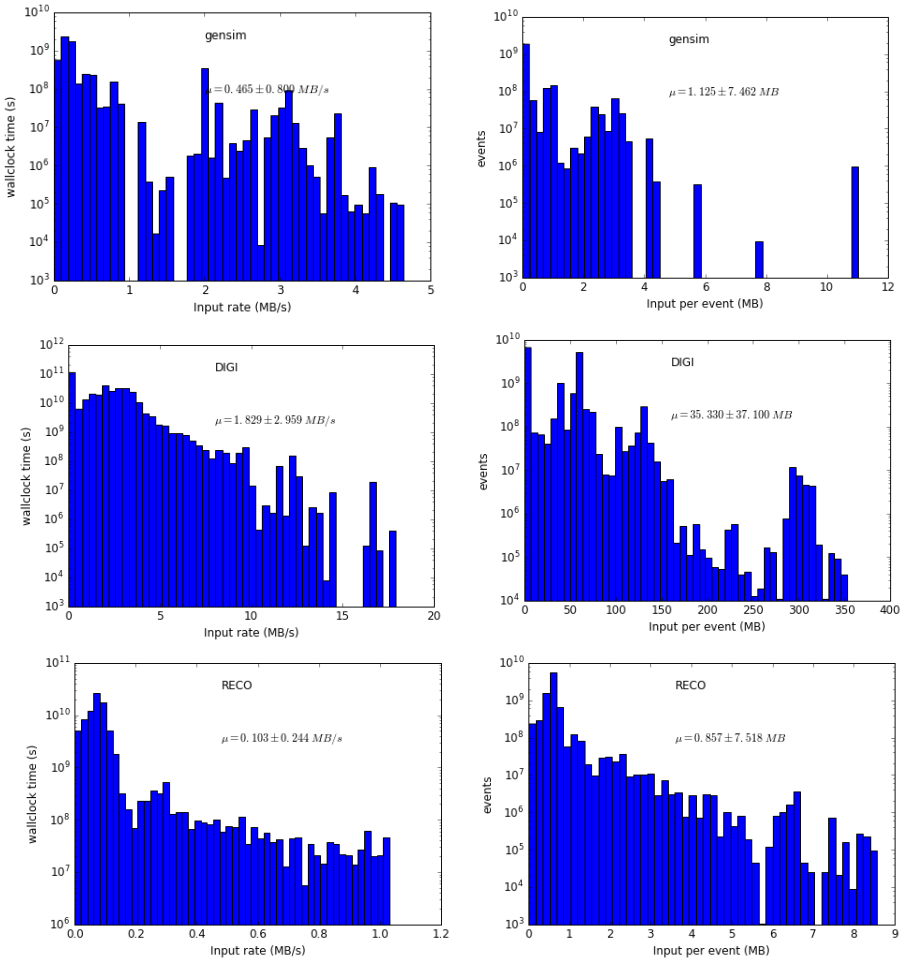
CMS production vs. analysis

- Analysis jobs are relatively I/O-light
  - They usually run on small format events
- Production jobs can be very I/O intensive for certain workflows



Comparing production job types

- Different workflows have hugely different IO rates
- DIGI jobs are the heaviest due to reading multiple pile-up events



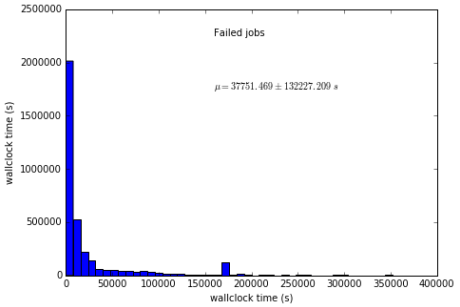
ATLAS job inefficiencies (1/2)

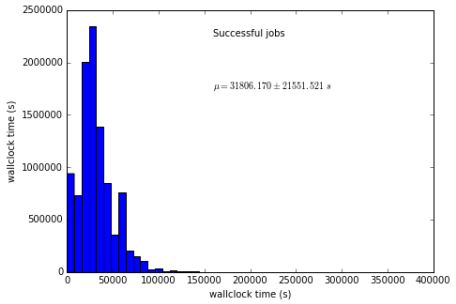
- New analysis, just started
- Goal is to optimize ATLAS workflow submission using as metric

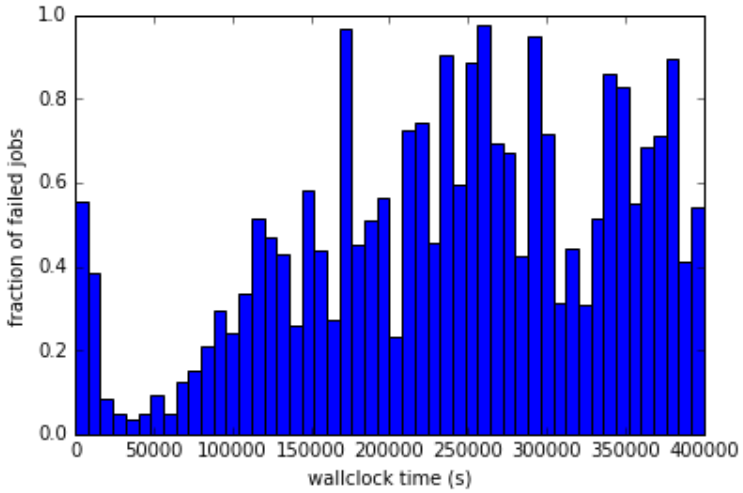
  (CPU time of good jobs) / (wallclock time of all jobs)
- Examples:
  - Too short jobs can be inefficient due to time spent on initialization phase
  - Too long jobs can waste a lot of wallclock time if they fail

ATLAS job inefficiencies (2/2)

- Measure job failure probability as a function of its duration
  - By job type, by failure mode
- Estimate ideal site performance from data







Fraction of failed jobs as a function of their duration

Instruction rate with Ivy Bridge vs Haswell for some common jobs

David Smith on behalf of IT-DI-LCG, UP team.
20 Oct 2016, ATLAS computing workflow performance meeting

Introduction

- Get some insight about how the job's code is interacting with the CPU while running by looking at Instructions per Cycle (IPC)
- This is not our usual performance measure, but I hope this may let one more easily see how the cpu pipeline is handling the code, and to some extent compare microarchitectures

IPC for some jobs

	Haswell IPC	Ivy Bridge IPC
CMS, Gen	1.59	1.58
CMS, Sim	1.47	1.18
CMS, Digi + HLT	1.66	1.53
CMS, Reco	1.55	1.4
ATLAS, Sim(19.2.4.9)	1.43	0.92
ATLAS Sim (20.7.8.5)	1.42	0.96
ATLAS, HITtoRDO	1.74	
ATLAS, RDOtoRDOTrigger	1.45	
ATLAS, RAWtoESD	1.51	

- Ratio of retired instructions / unhalted clock cycles (most over whole job). Physical machine.
- Atlas simu with single process athena, HT on, affinity fixed to 1 core. No other significant load.
- Haswell was Xeon E5-2683 v3 (~3GHz); Ivy Bridge i7-3770k (~3.8GHz)
- checked Ivy Bridge also on Xeon E5-2695 v2 (~3.1GHz) running ATLAS Sim (19.2) => 0.91 IPC
- checked Atlas simu (19.2) with athenaMP (8) affinity to 4 cores on one socket => 1.58/0.97 IPC
- ATLAS sim was job 2972328065 (19.2.4.9, slc6-gcc47-opt or 20.7.8.5, slc6-gcc49-opt; mc15_13TeV.362059.Sherpa_CT10_Znunu_Pt140_280_CFilterBVecto_fac4)
- Looked up previous HS06 results; usually ~10% higher for Haswell (per job slot/per GHz)

Which microarchitectures are used?

Microarchitecture	Est % of avail cycles	Introduced from
Haswell (HW)	20%	2013
Sandy Bridge (SB)	18%	2011
Nehalem	27%	2008
Intel Core	11%	2006
AMD (not split)	23%	

- The above are usually classed as the intel microarchitectures: e.g. Ivy Bridge is the die shrink version of SB, and is classed as SB microarch.
- This is the last 90 days of ATLAS jobs, raw data from elastic search (thanks Andrea)
- Used wall clock time per cpu type, with classification based on type string, weighted by quoted cpu freq, and a rough weighting of x1.5 for Intel Core, as that microarch. has no hyper threading

ATLAS simulation

- Both CMS and ATLAS simulation showed larger difference in IPC between Ivy Bridge & Haswell than other types of job
  - Looking over related work, e.g. ‘MJF vs simulation in LHCb’ (Philippe C.) may have also been seen results consistent with this (but not yet discussed with them)
    - An LHCb benchmark was found to show large SB vs HW difference but it was concluded that was attributable to reduced branch misprediction rate
    - Believe it’s not the case here

Activity

- Did some measurements of the ATLAS simulation using PMU counters and Last Branch Record facility
  - e.g. intel's top-down classification
  - Appears frontend latency bound
  - Branch misprediction is lower on haswell, but assuming standard 20 cycles per miss this is far from accounting for IPC difference
  - Estimated basic block counts & code footprint using LBR
  - But won't go into these details more now (don't want to get lost in details)
- Noticed that CMS's simulation showed smaller difference
  - CMS's pluginSimulation.so contains the geant4 routines and was linked with g4 static archive libs -Wl,--exclude-libs,ALL
    - One effect of this is non-virtual method calls in G4 avoid the overhead of going via the PLT => avoids one indirect jmp per function call

Activity

- Haven’t tried recompiling ATLAS SW and G4 in that way but
  - modified ld-linux-x86-64.so.2, the dynamic linker from glibc, to patch up some of the PLT call sites (where this could be inferred from return address on stack) in the code during runtime binding to call the function directly
  - Sampling IPC after the patch-up (after 1st event of job) this apparently improved the IPC on Ivy Bridge to 1.09 (about 13-18% increase depending on ref version). However little change on HW.

Possibilities for further study

- Appears 13% gain is certainly possible for simu on Ivy Bridge (maybe more if a similar approach to CMS is feasible). However probably only on SB microarch. (and maybe earlier). Plan:
  - Decide if it's worth it (check impact on Nehalem?) in consultation with atlas simu group. Probably would want to go route of changing the build rather than patching!
  - Reasonable to query cost of doing so (change lib packaging maybe; re-validate after changes): little impact with HW (and the fraction of worker nodes with HW microarchitecture or later will grow)
- Could try to identify the cause of difference in simulation IPC between SB and HW in more detail
  - Interesting, but not certain if there will be concrete gains beyond possible plt changes for atlas simu.
  - May involve parts of the pipeline where I think exact operation is not disclosed. (e.g. the BPU, frontend steering or instruction prefetch).
- Investigate behavior of IPC as other cores on the CPU are loaded?

Some general i/o measurements and some interpretation

David Smith on behalf of IT-DI-LCG, UP team.
25 May 2016, ROOT I/O Workshop meeting

Scope

- This is ongoing work
  - What is shown is not supposed to be a recommendation or conclusion
  - Some unanswered questions

- Not specifically a ROOT I/O study

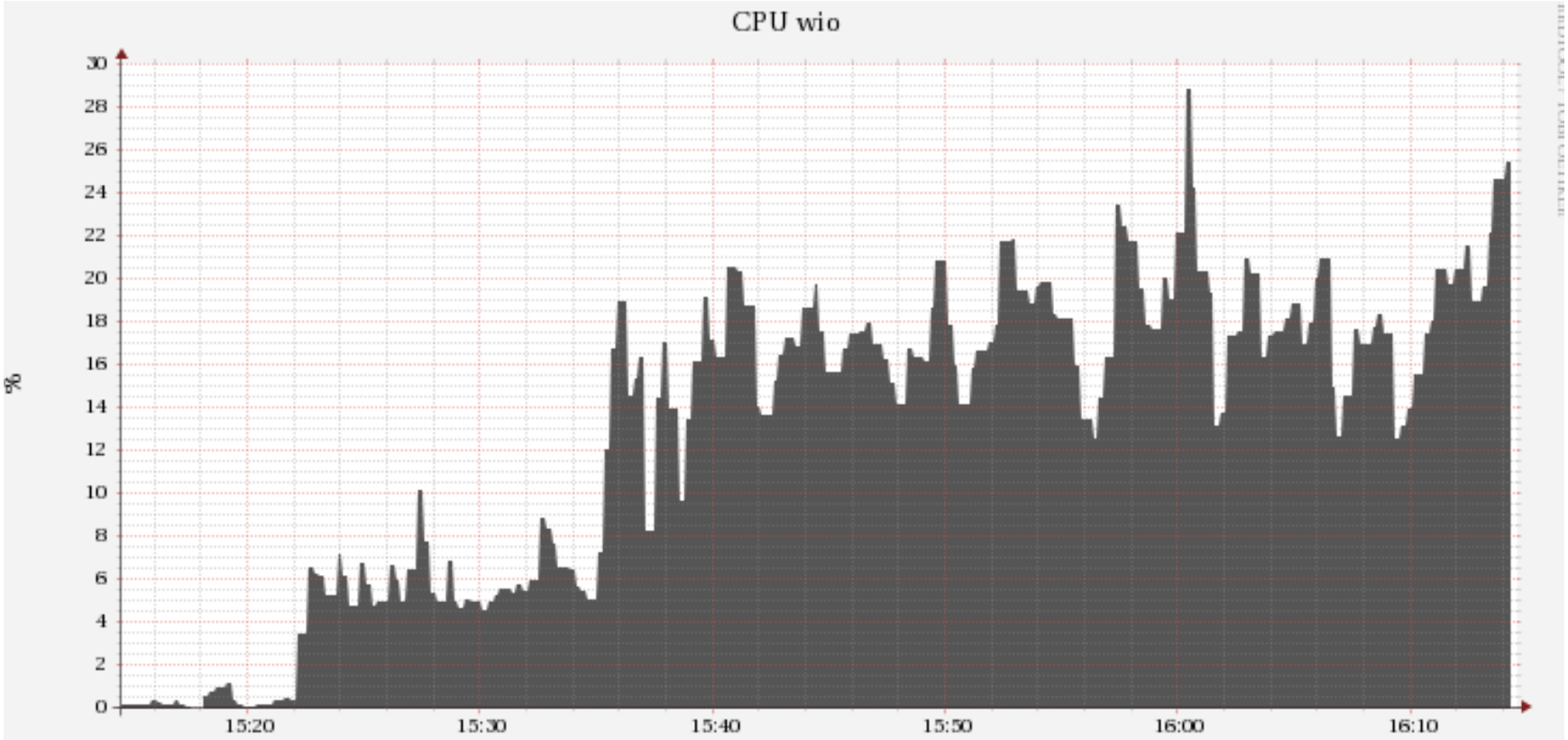
Overview

- Background, rational
- Type of workload (job)
- Description of hardware and VM environment
- Examples of measured quantities
- Features of comparative runs
- Examples of selected plots from the comparison runs
- Interpretation and open questions

CERN-P1_MCORE (atlas site)

- Initially got involved when there was a problem on CERN-P1_MCORE a cluster run as an atlas site on some of the HLT nodes at point 1
  - Failures had been noticed when starting to accept pile jobs to the site, in addition to previous simu ones
  - An problem was identified which was causing filesystem corruption in the VMs.
- After workaround, some issues still noticed
  - Dashboard reported lower efficiency compared to CERN-PROD_MCORE for some job types
  - Few percent of jobs failing with lost heartbeat
  - Sometimes the HLT fabric monitoring goes to alarm because of high io wait

Example monitoring plot



(plot kindly provided by my ATLAS colleague, shown for illustration only)

Job performance

- The pile jobs were thought involved:
  - The simu jobs show good performance
  - The pile jobs known to have to need read more data during their run; more concerned with i/o effects
- Decided to collect some machine level i/o measurements to gain some insight into what has happening for the job
  - And as a side effect get experience of how these measurements could be treated (e.g. if measurements are routinely collected)

Monitor machine during a job

- Idea was to repeatedly rerun an instance of a pile job (digi, trigger, reco): AthenaMP (8 processes)
  - Data is read or written locally (stage in or out not done, for these tests data already staged)
- HLT node has these features
  - 7200 rpm, sata, 250GB disk (WD2502ABYS)
  - 24 GiB ram
  - 2 x Xeon E5540 base freq 2.53GHz (8 cores, x 2 with SMT)
- When used as CERN-P1_MCORE
  - Runs virtual machine (qemu)
    - 16 vcpu; CERN VM 3.5.1 (now 3.6.x); disk image is QCOW2
    - About 22 GiB available memory; swap configured
    - Two condor job slots per machine

Collecting information

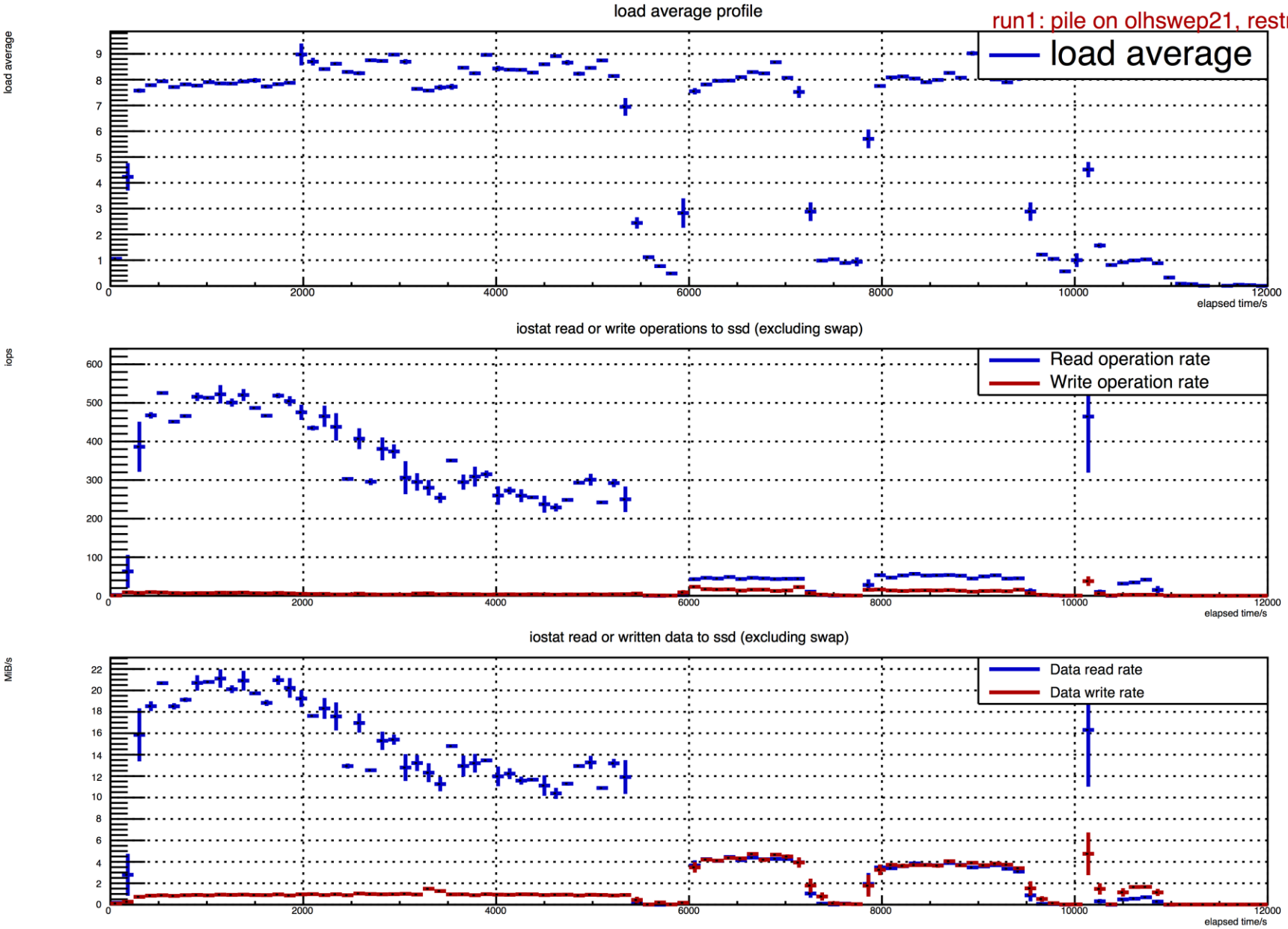
- Mostly from vmstat, iostat and uptime
  - In some cases also collected from the host as well as the guest
- Also took some strace samples from one AthenaMP process for extra information
- Number of “runs” of the job to compare, varying
  - 1 or two concurrent instances
  - Running 1 instance either with limited memory (~10GiB) or less limited (~22Gb)
  - block device (containing the data) readahead
  - One set from a test machine with an SSD, the rest from one of the ATLAS HLT VMs.

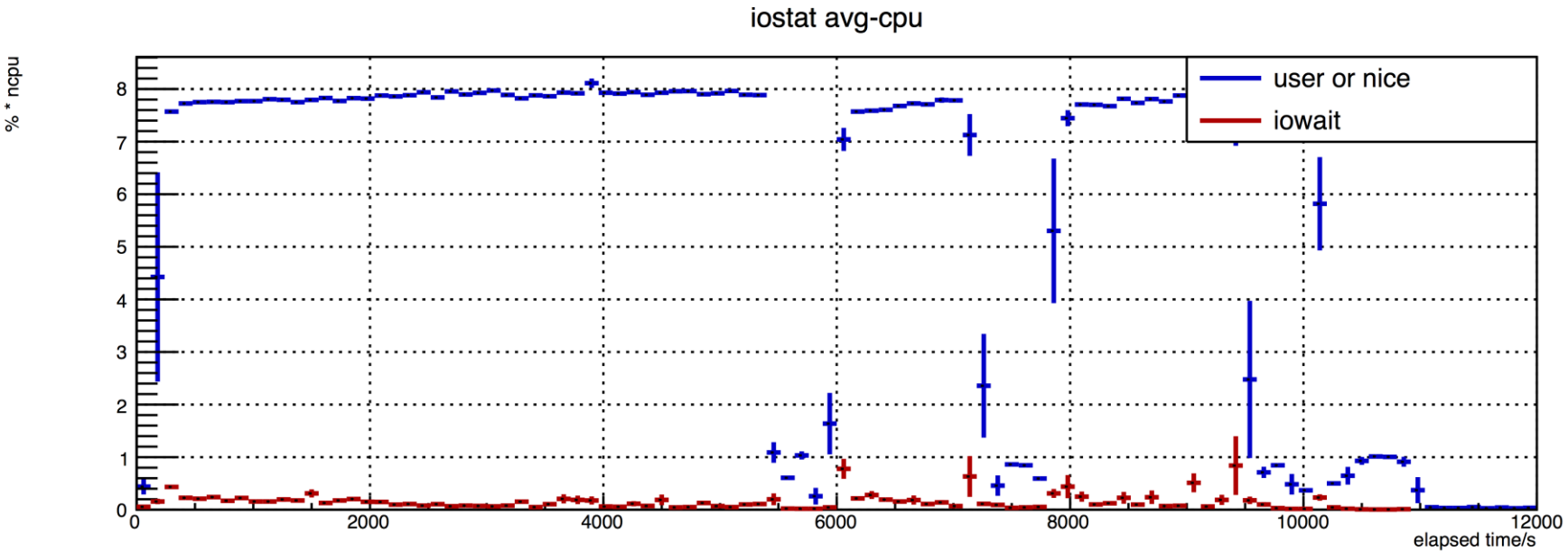
Basic features of the job

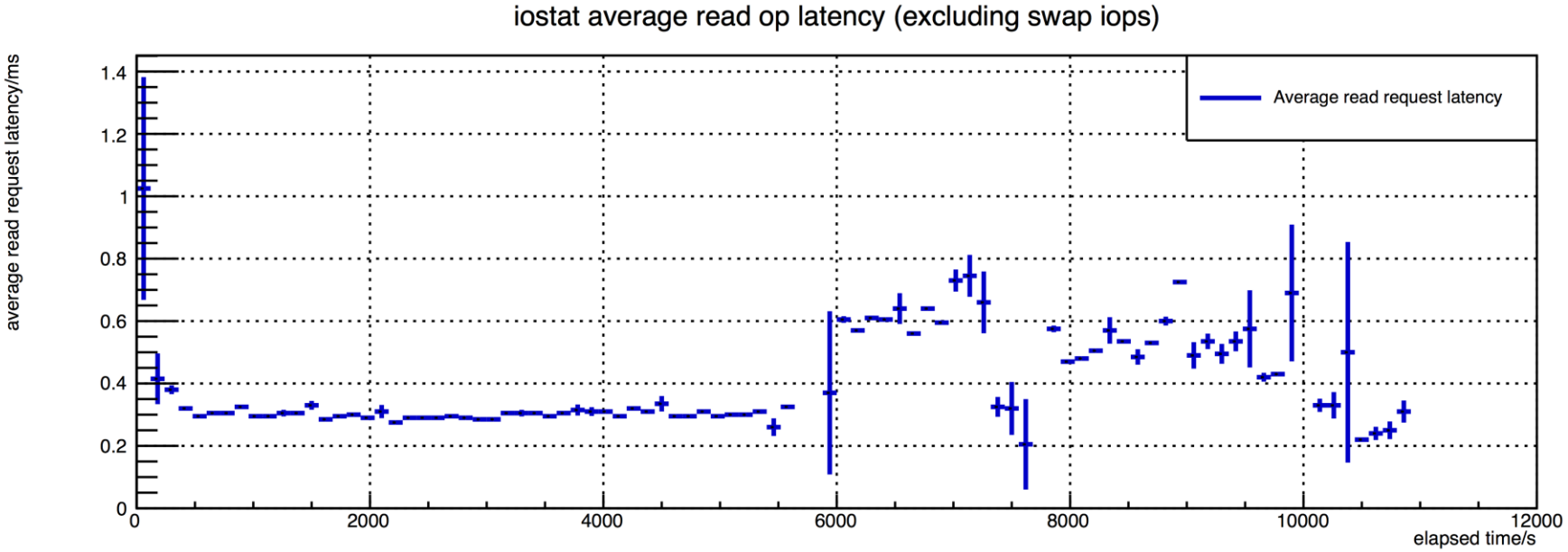
- Job consists of several steps, 3 of which are AthenaMP jobs (running with 8 processes)
  - With some steps in between, some are single process steps

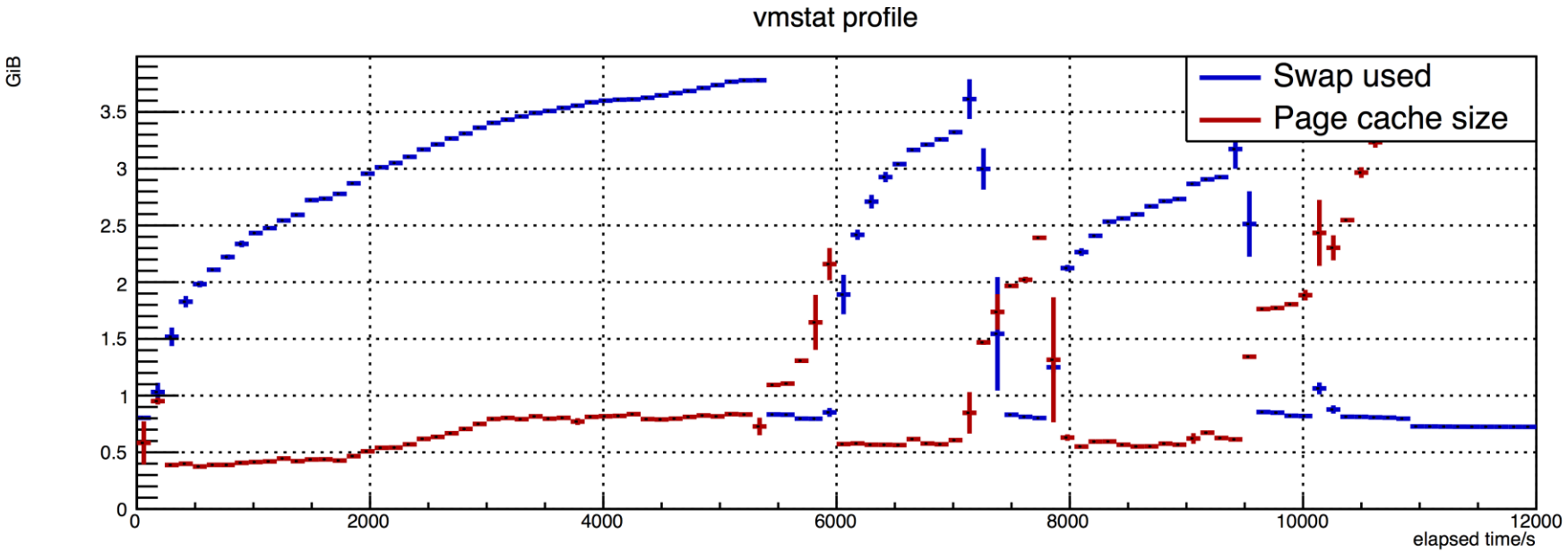
- Will now run through some of the typical distributions for an initial trial job the job
  - On an physical machine, with SSD. Different type of processor from the HLT; aim was to see the features and introduce the measured values

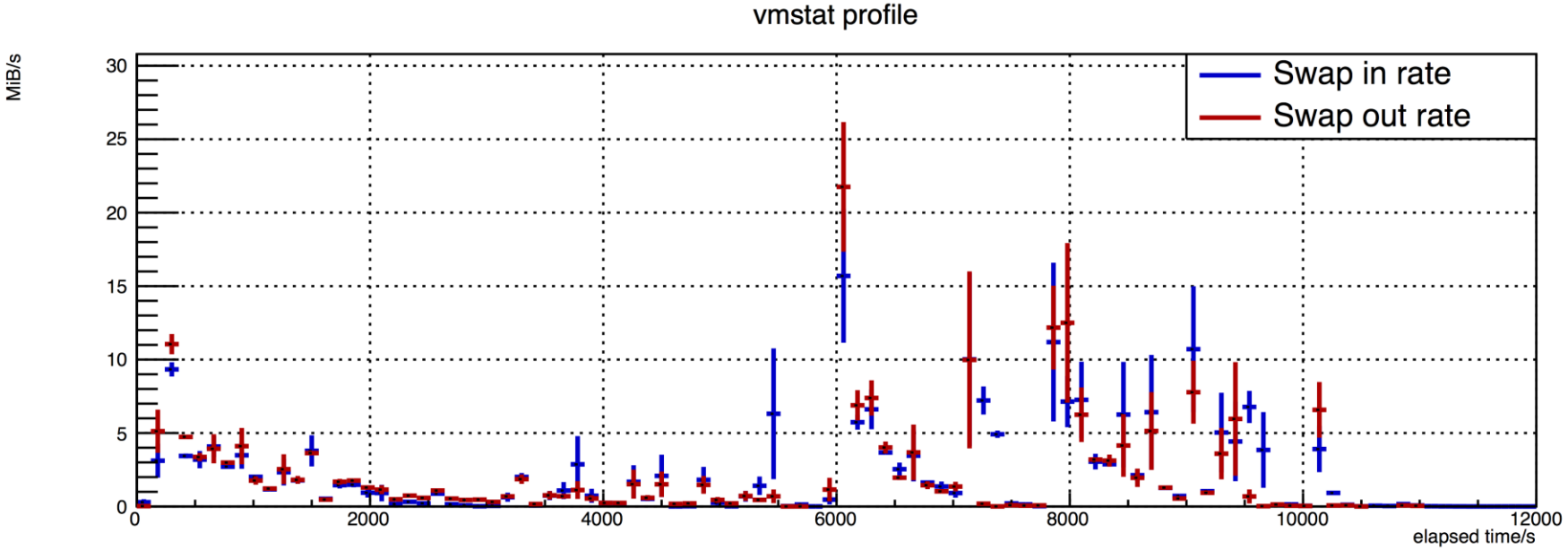
run1: pile on olhswep21, restricted mem

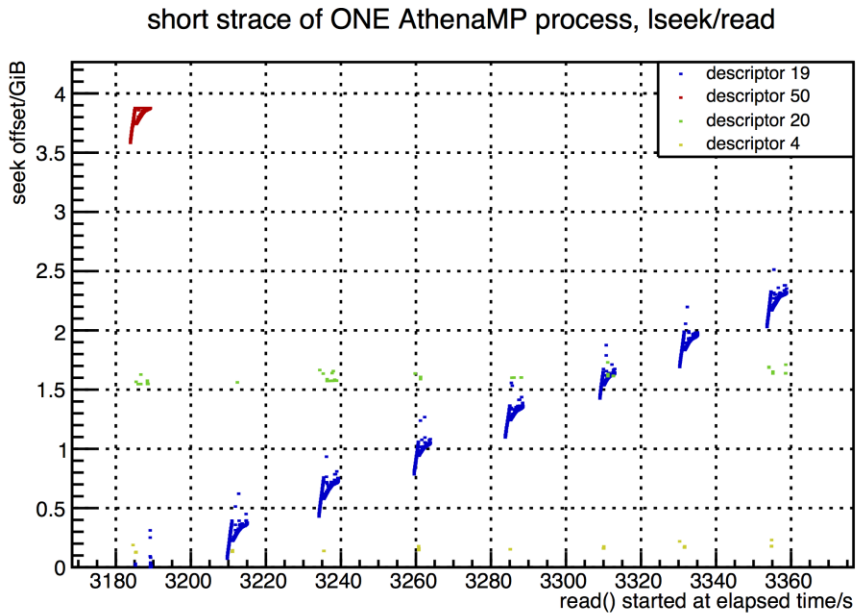


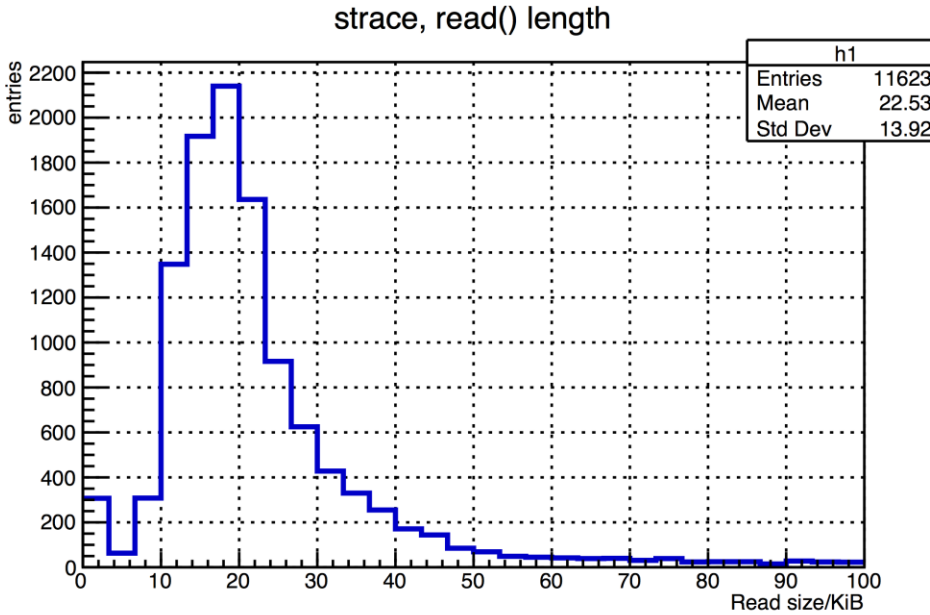

iostat avg-cpu
% * ncpu
user or nice
iowait
0
1
2
3
4
5
6
7
8
0
2000
4000
6000
8000
10000
12000
elapsed time/s

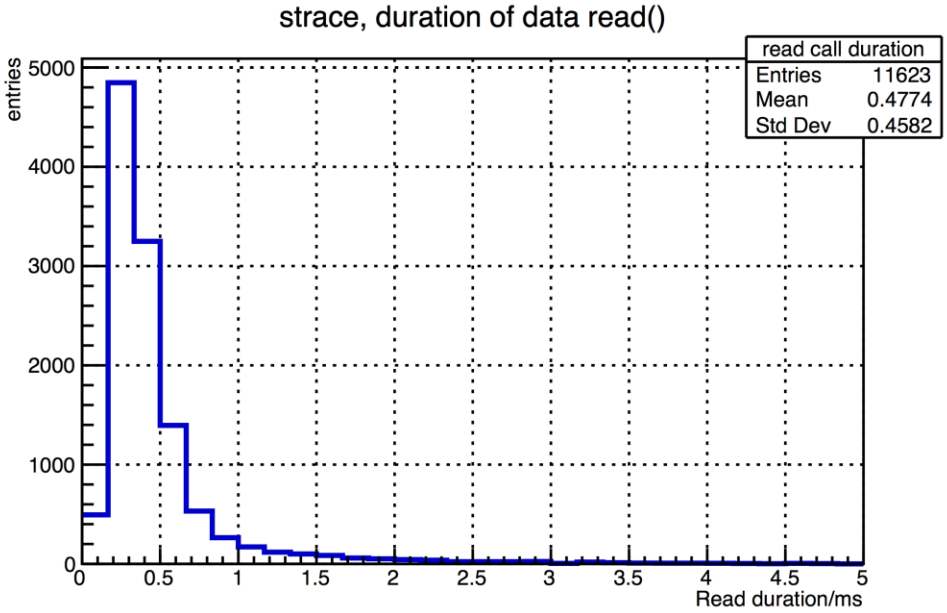

iostat average read op latency (excluding swap iops)
average read request latency/ms
Average read request latency
0
0.2
0.4
0.6
0.8
1
1.2
1.4
0
2000
4000
6000
8000
10000
12000
elapsed time/s

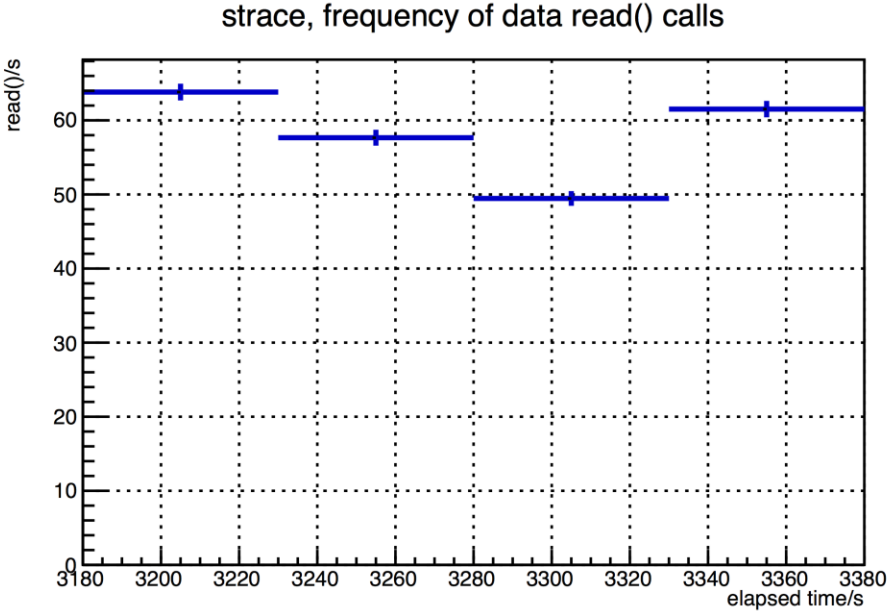

vmstat profile
GiB
Swap used
Page cache size
0
0.5
1
1.5
2
2.5
3
3.5
0
2000
4000
6000
8000
10000
12000
elapsed time/s


vmstat profile
MiB/s
Swap in rate
Swap out rate
0
5
10
15
20
25
30
0
2000
4000
6000
8000
10000
12000
elapsed time/s


short strace of ONE AthenaMP process, lseek/read
seek offset/GiB
read() started at elapsed time/s
descriptor 19
descriptor 50
descriptor 20
descriptor 4


strace, read() length
entries
Read size/KiB
h1
Entries 11623
Mean 22.53
Std Dev 13.92


strace, duration of data read()
entries
Read duration/ms
read call duration
Entries 11623
Mean 0.4774
Std Dev 0.4582


strace, frequency of data read() calls
read()/s
elapsed time/s

Features of the trial runs

Title	events	readahead/sect	mem restr	total time/s	digi time/s	trig time/s	reco time/s	Est. read digi/GiB	Digi step rate ev/s	Event throughput ,
run2	2000	256	no	17000	7750	1750	2500	35	0.3	0.1
run3	2000	256	yes	41000	12000	12000	11000	90	0.2	0.05
run7	2000	0	no	16500	7750	2000	2750	30	0.3	0.1
run5	4000	256	no	63000	24000	22000	9000	200	0.2	0.06
run8	4000	32768	no	54000	22000	17000	9000	350	0.2	0.07
run6	4000	0	no	87000	54000	24000	10000	100	0.07	0.05

- Block device readahead may have benefit, but increases amount of data read
- Restricting memory decreases throughput
- Running the job twice concurrently does not greatly increase throughput compared to memory limited single instance
  - Possibilities for this may include: i/o limit to disk, additional i/o load from swap and reduced page-cache, other non-disk i/o condition reducing throughput of job

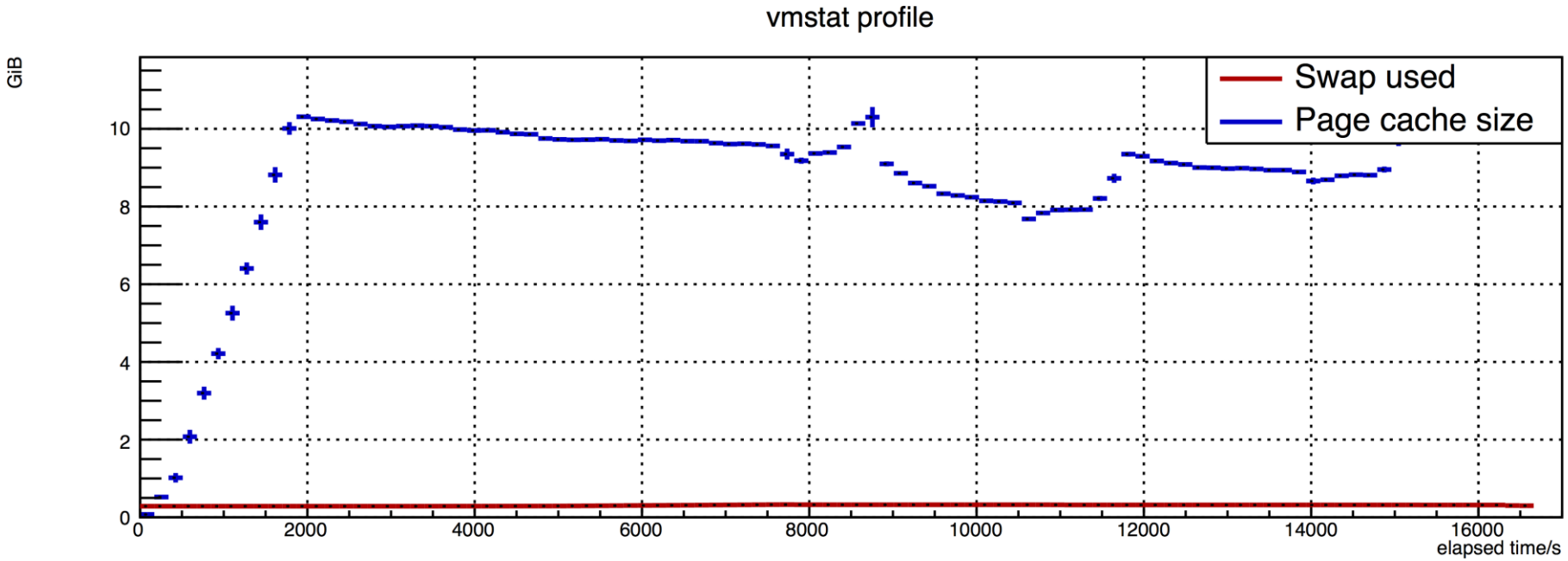
Features of the trial runs

- Considering: run 2, 3 & 5 as nominal single job, restricted memory single job and two concurrent jobs. Sequential changes in elapsed time between these runs, for certain stages:

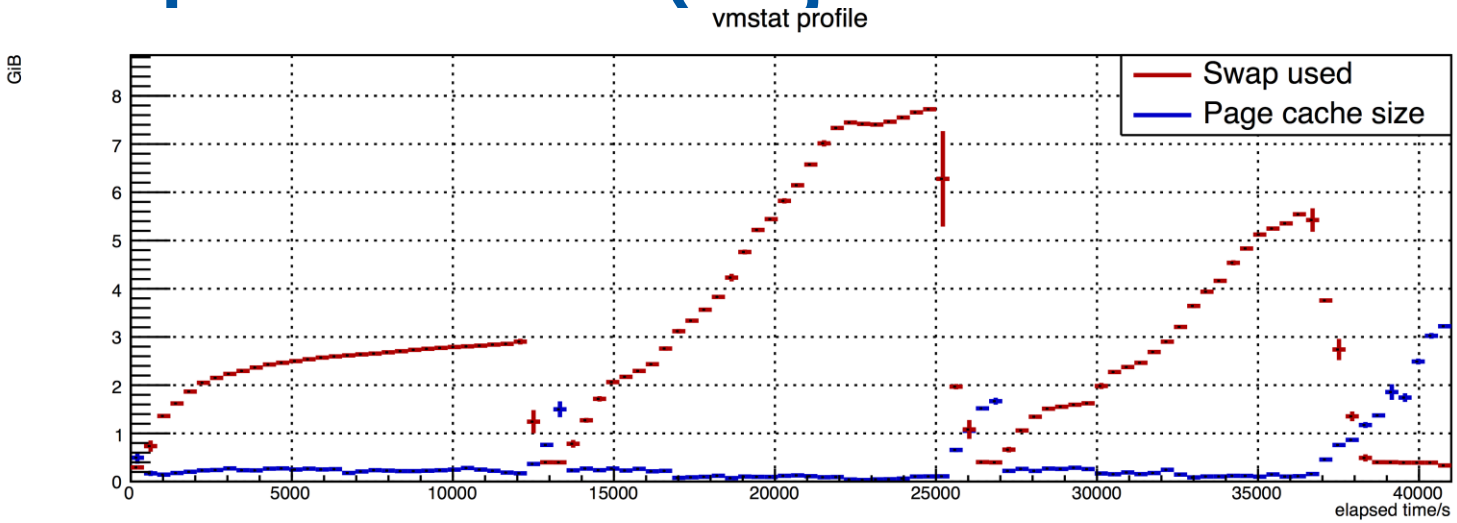
	change digi	change trig	change reco	total
restr. mem	1.5	6.9	4.4	2.4
2 concurrent	2	1.8	0.8	1.5

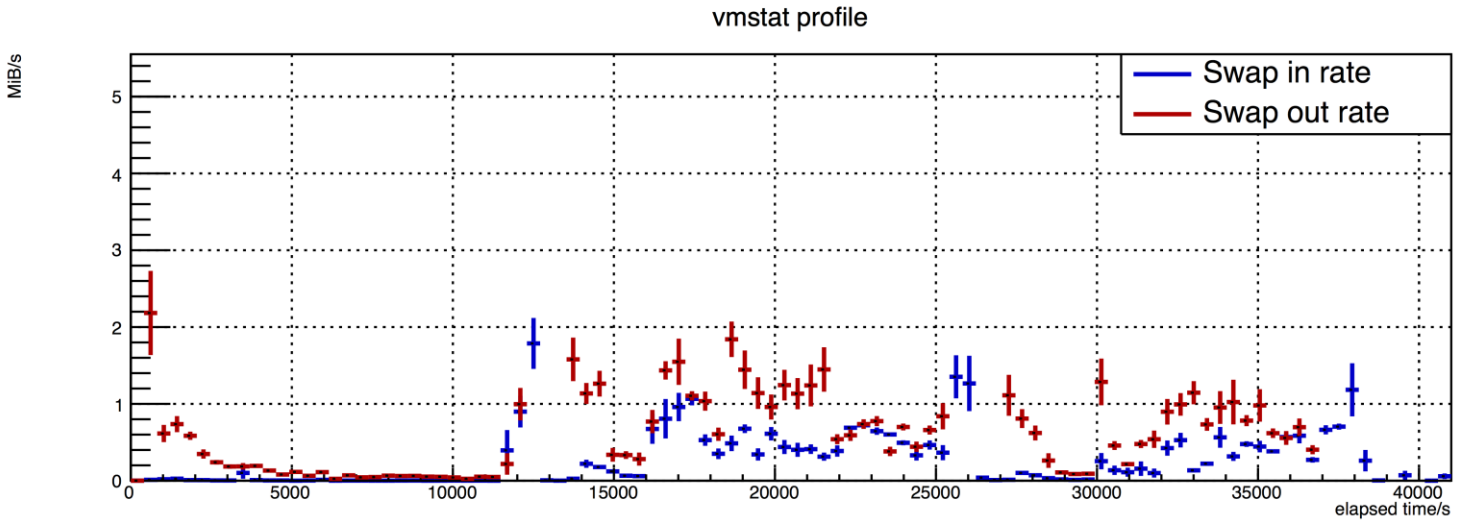
- (2 concurrent means double number of events: identical jobs, started at the same time, stages are considered to overlap)

Swap/page-cache: run 2 (1 job)

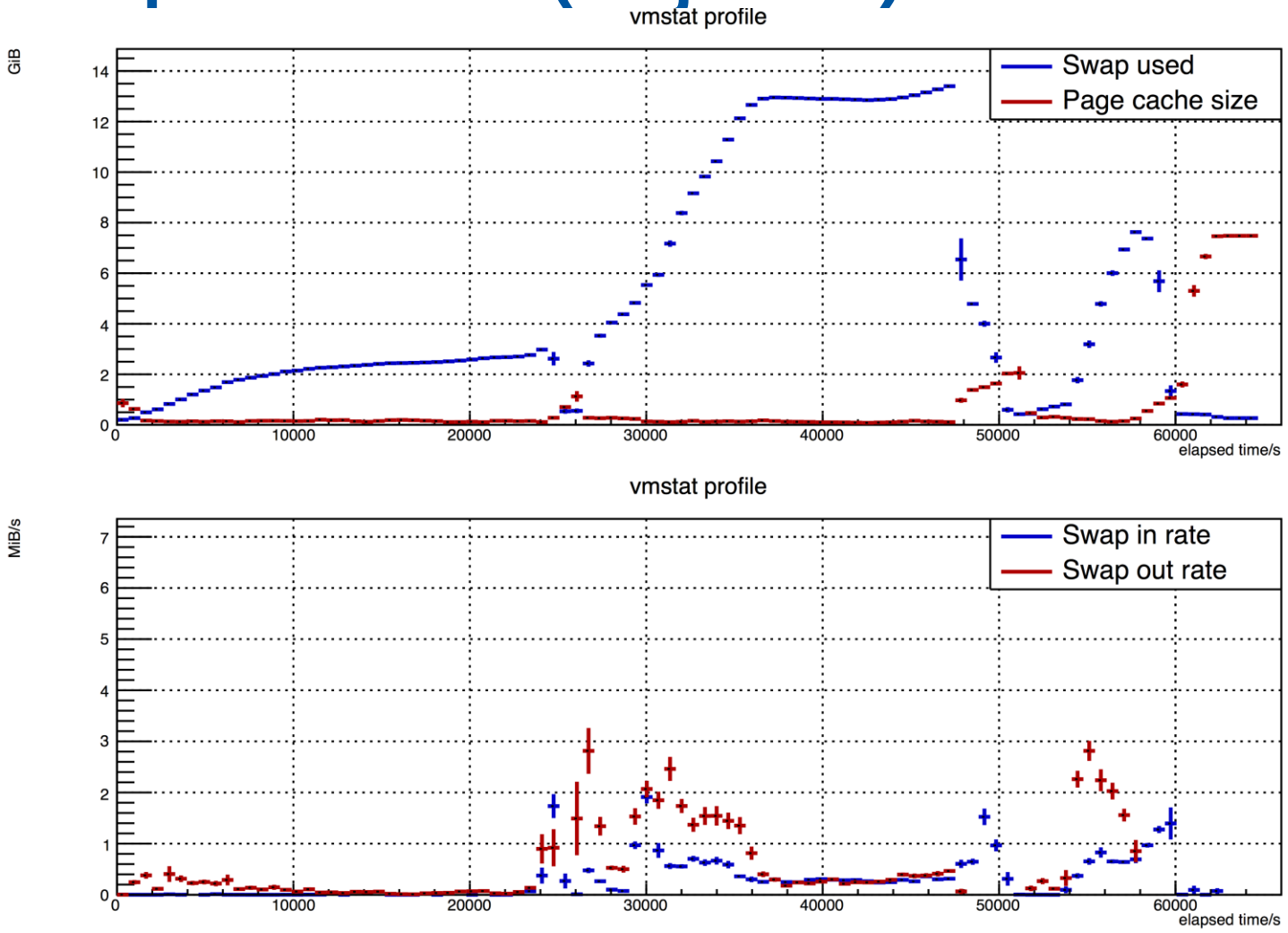


Swap: Run 3 (1 job restr. mem)

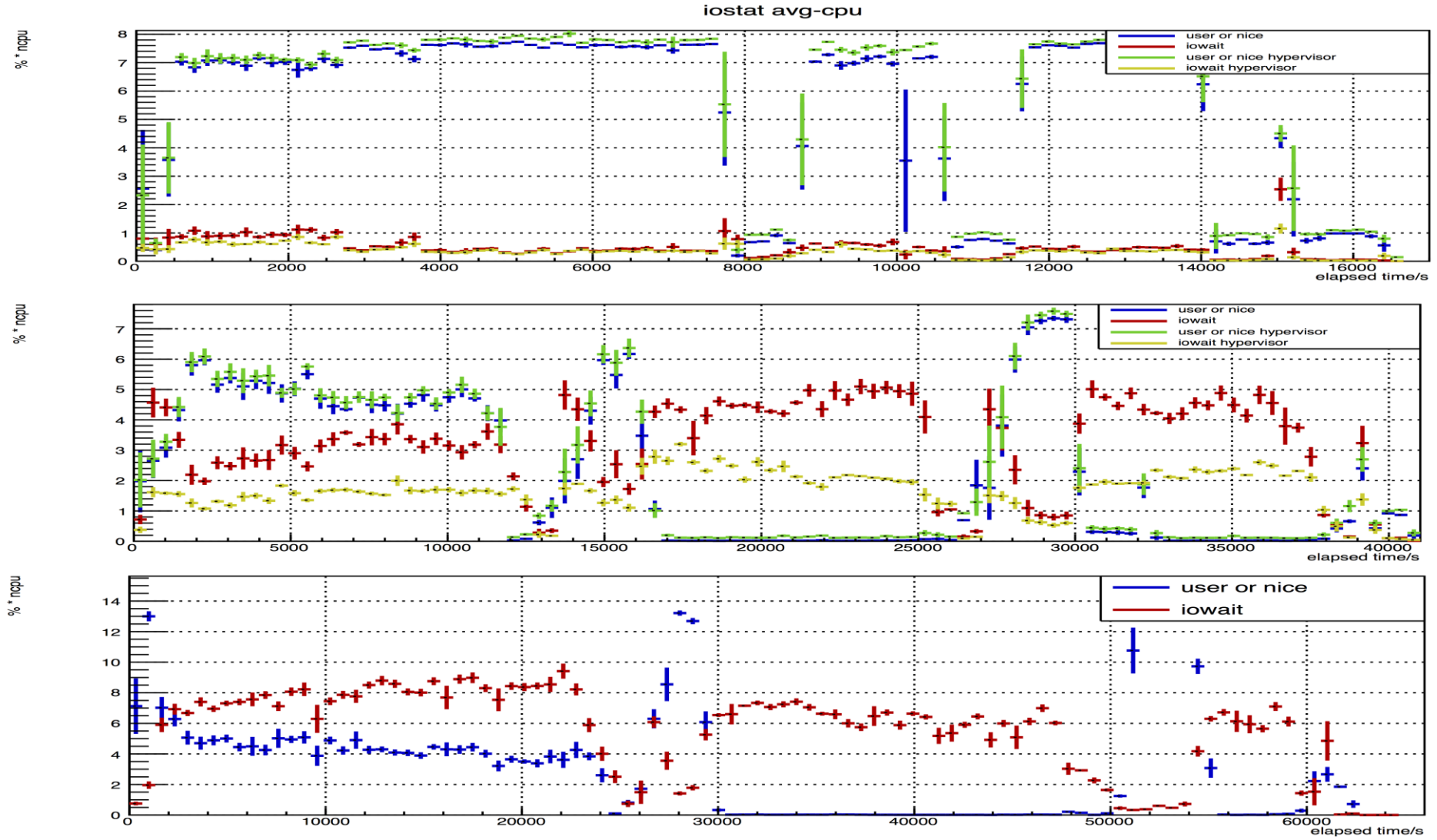




Swap: run 5 (2 jobs)



CPU statistics: run 2,3,5



observations and open questions

- The larger fractional change in walltime appears when memory is restricted for trig, reco stage
- Digi stage appears to closer to the i/o limit
  - Two concurrent jobs have approx. the same overall throughput for this stage as a single, memory limited one
- Increase in estimated data read during digi stage with restricted memory not understood
  - The I/O does also includes swap data (but swapping is limited during this stage)
  - The available page-cache is smaller, but have not yet made agreement between application level reads and the larger estimated I/O (c.f. amount of data for run 6 + run 7)

observations and open questions

- Have been mostly using general measurements (i.e. vmstat and iostat) measurements
  - But have been using knowledge of the payload to characterise it
  - May see what could be concluded from the generic measurements only, without supposing knowledge of the job. (But will probably do this after I believe I have understood relevant features)

Network Analytics

Hendrik Borras, Marian Babik
IT-CM-MM

perfSONAR Infrastructure

- perfSONAR has been widely deployed in WLCG
  - 249 active instances, deployed at 120 sites including major network hubs at ESNet, GEANT
  - Measuring many different network metrics on all existing LHCOPN/LHCONE links
  - Deployment and support coordinated by the WLCG Network Throughput WG
  - In addition there are more than 1600 perfSONAR deployed in public
- The core motivations for this deployment was
  - To ensure sites and experiments can better understand and fix networking issues
  - Measure end-to-end network performance and use the measurements to single out on complex data transfer issues
  - Improve overall transfer efficiency and help us determine the current status of our networks

Network Measurement Platform

- In collaboration with OSG, we have developed an extensive network measurement platform using perfSONAR
  - Tests can be centrally configured and are continuously gathered by the OSG collectors
  - Service and metric-level infrastructure monitoring
  - All measurements are available for subscriptions via ActiveMQ netmon broker at CERN
- What's missing ?
  - We want to run real-time analytics to detect "obvious" issues with the network as they arise
  - **We want to have the ability to detect which network paths perform better in case there is a choice (network cost-matrix)**
    - Valuable for many different systems to decide on placement of jobs, executing transfers, etc.
  - We want to automate debugging of the network issues and help find root causes in real-time

Network Analytics Activities

- Ilija Vukotic (Univ. of Chicago)
- has developed ELK/jupyter stack for ATLAS Analytics
- worked with Xinran Wang on anomaly detection and advanced alerting/notifications for network problems
- Also looked at detection of the anomalies based on machine learning models
- Jerrod Dixon and Brian Bockelman (UNL) exploring network analytics in CMS
- Shawn McKee (Univ. of Michigan) working on real-time root cause analysis (PuNDIT) in collaboration with ESNet
- Henryk Giemza (NCBJ), Federico Stagni integrating perfSONAR in DIRAC for LHCb
- Hendrik and Marian working on developing models for network cost-matrix - determine performance of network paths

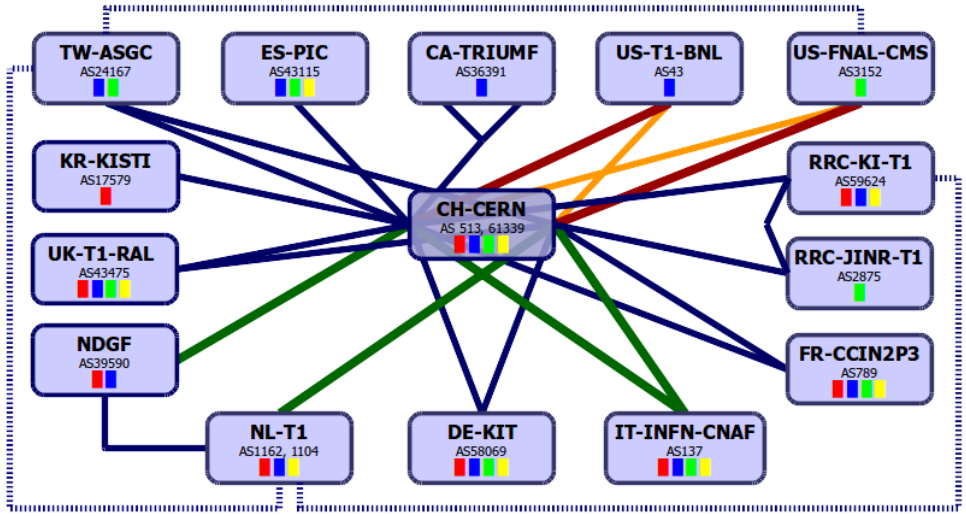
Network Cost Matrix

- We want to have the ability to understand which network paths perform better when choices are available.
  - So how do we determine network path performance ?
- Network throughput measurements
  - Challenging to do right in a distributed environment - iperf3, nuttcp with perfSONAR likely the best, but difficult to run full mesh tests at high frequencies
  - Throughputs already reported by various different data management systems, but it’s very challenging to determine what fraction of it is real network performance
- Router utilizations/Flow
  - Not end-to-end - many issues can be hidden in the passive equipment (switches) which are not accounted, also it depends on the actual implementation in the router (bugs)
  - Very challenging in a federated environment - too many virtualization layers, each NRENs has its own approach

Approach

- Can we determine link utilization from network metrics that can be measured at high frequency ?
  - We measure one-way packet loss and latency @10Hz with perfSONAR
- Sufficiently precise latency/loss measurements should show network equipment “under stress” as it will need to hold packets for a little longer in its buffers
  - Thus causing spikes in latencies or packet loss if out of available buffers
- How can we validate that perfSONAR measurements are sensitive enough to detect latency and loss due to network congestion ?
  - We can compare perfSONAR data with router utilizations on networks, which have simple topology, e.g. **LHC private optical network (LHCOPN)**
- Would it be possible to build models that show near real-time network usage aka network telemetry ?

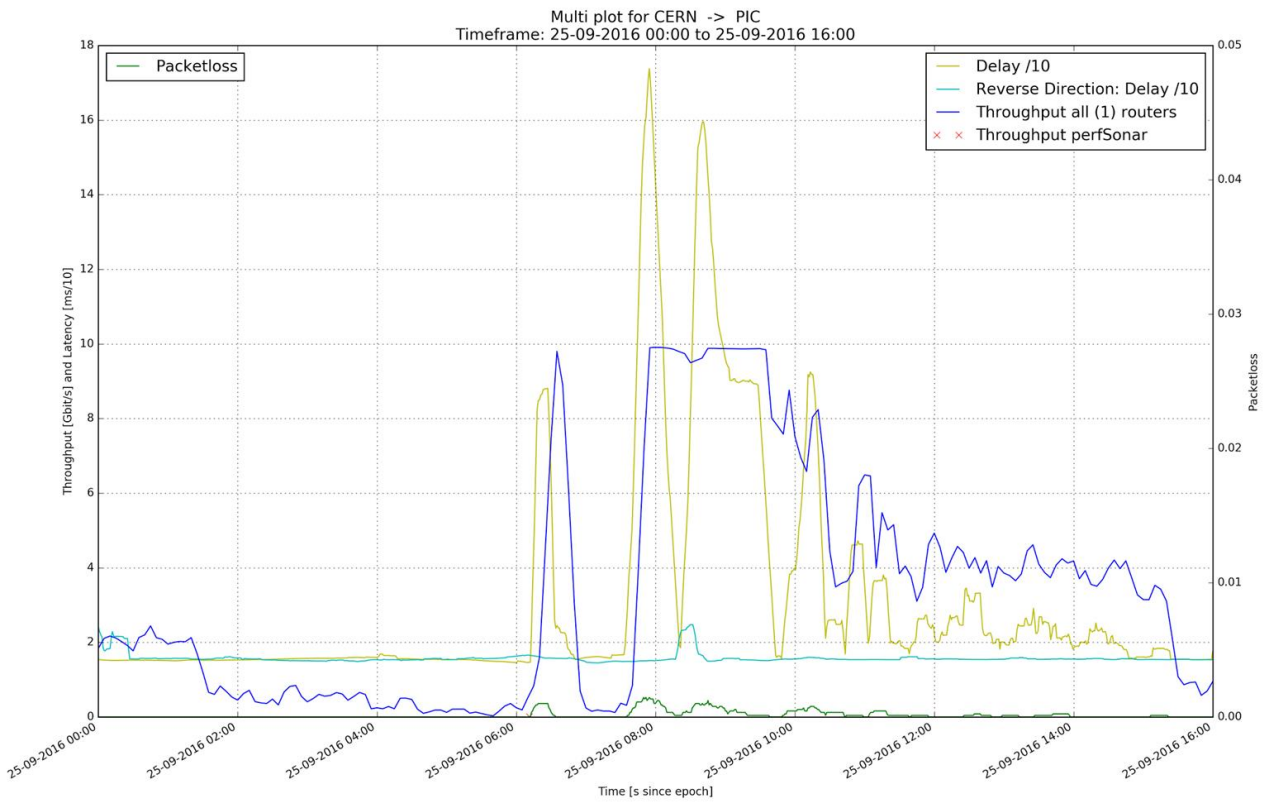
LHCOPN



Reasons for choosing LHCOPN:

- Simple one hop connections
- Accurate assumptions about the available bandwidth
- Routing is controlled and monitored by CERN
  - Router utilization data provided by IT-CS

Approach in practice



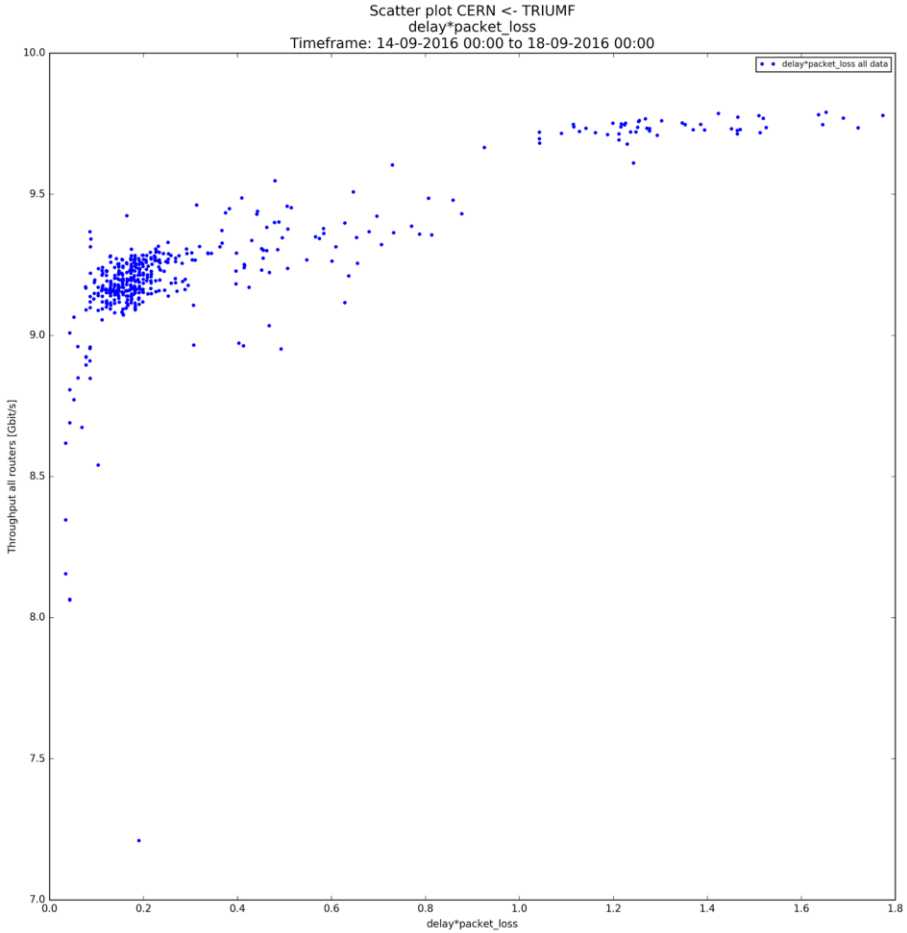
As you can see both latency and packet loss (yellow/green) are quick to react once the traffic (blue) approaches the total link capacity (10Gbps in this case).

Similar pattern visible on all LHCOPN connections.

Nearly no warning for starting congestions, but good follow up once the congestion kicks in.

This shows perfSONAR measurements are sensitive enough, so we have explored ways how to couple delay and packet loss to produce an assumption of the link utilization.

Does it work ?

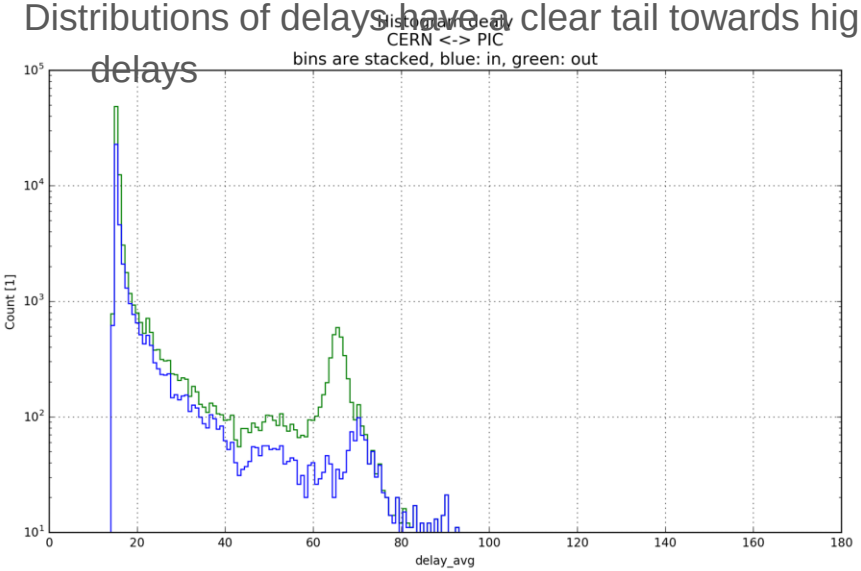


Multiplied delay and packet loss shows good correlations for high throughput (above ~80%)

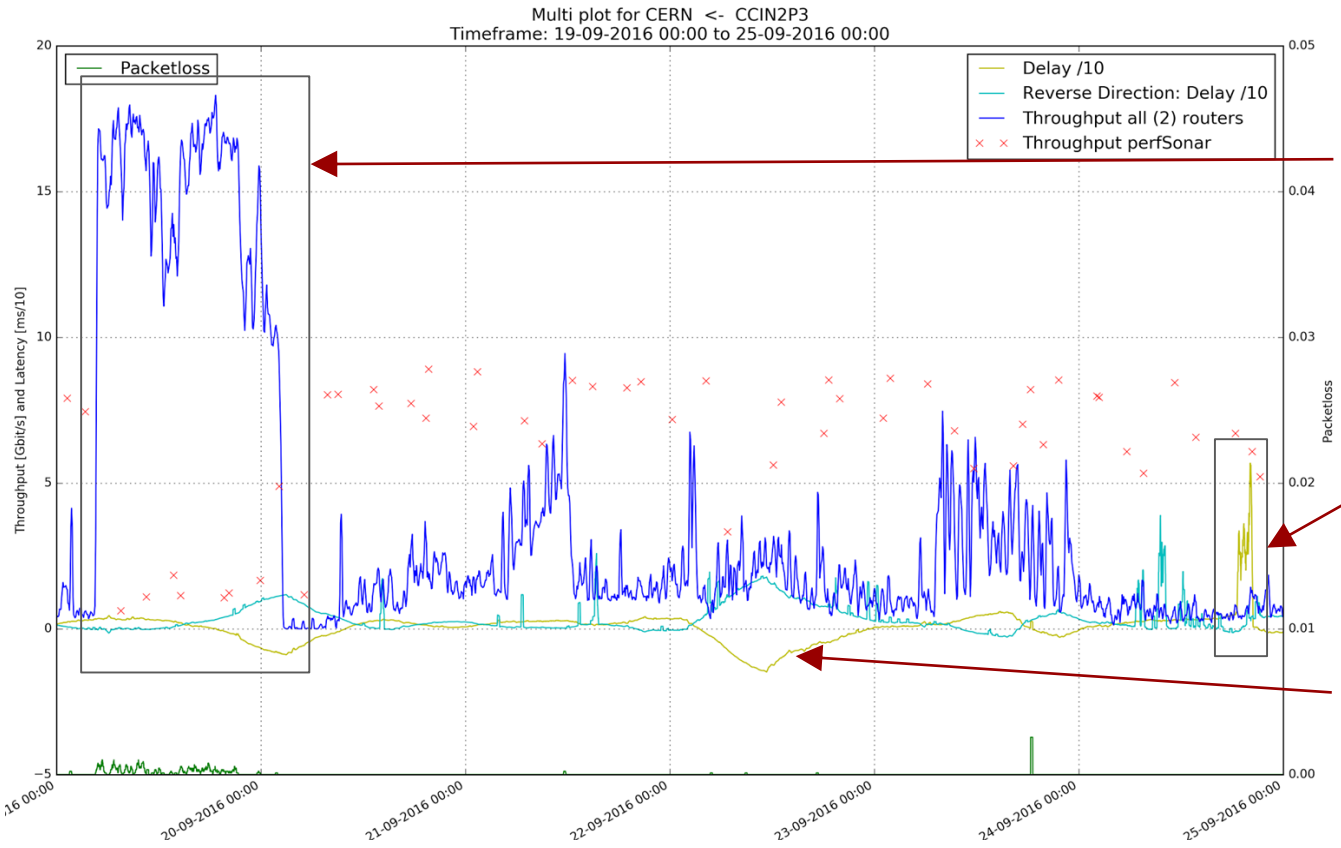
Only clearly visible on incoming connections

Probably because we only see the routers at CERN

Distributions of delays have a clear tail towards higher delays



Observations

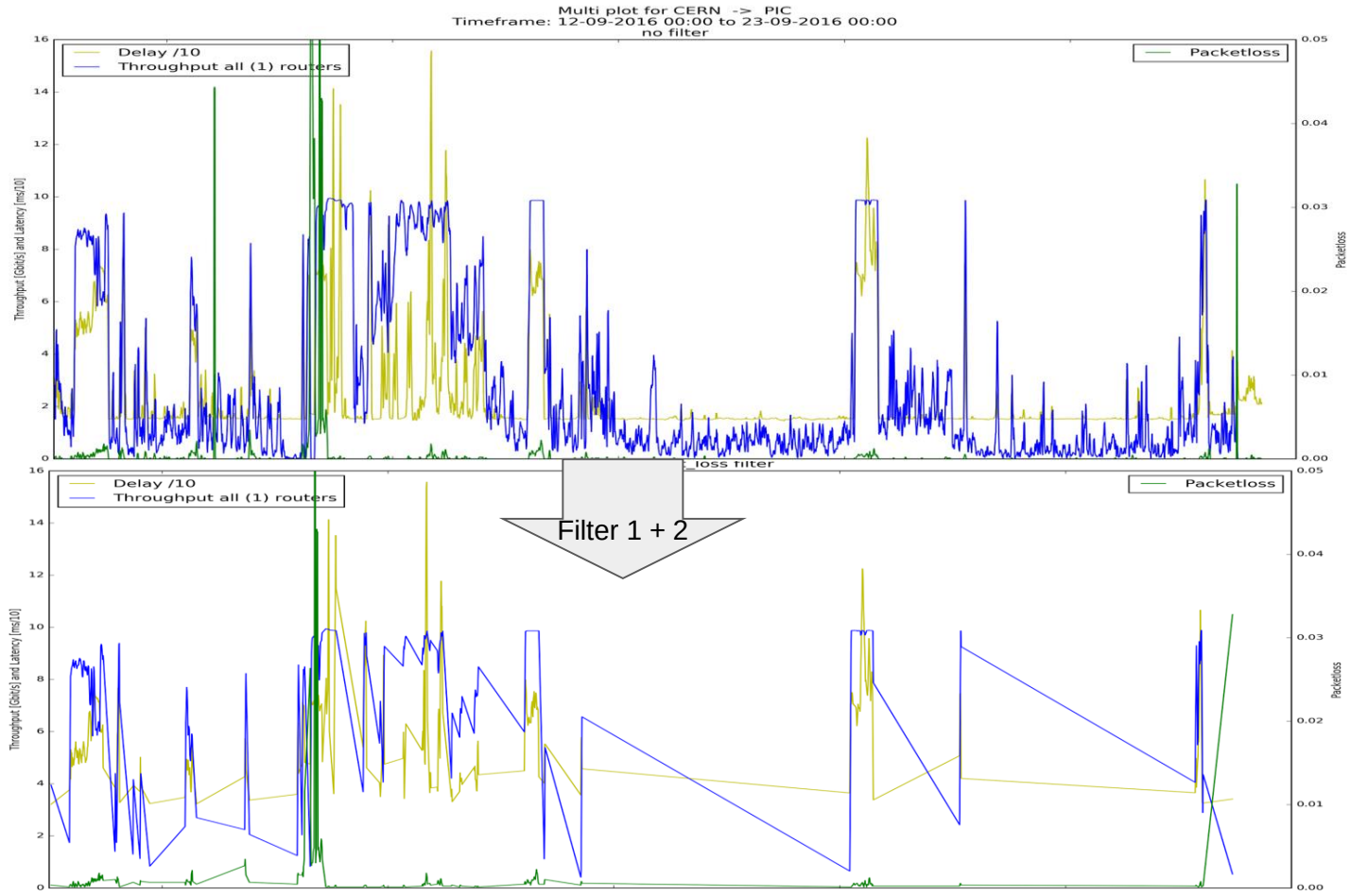


perfSonar throughput measurements - despite low granularity shows it's sensitive to high link usage

Interesting "ghost" latency spike seen on all incoming connections (every weekend) - maintenance ?

Surprisingly large drifts in delays caused likely by non synchronous clocks (ntp) - limits modeling for sites that are "too close" wrt. latency (e.g. on campus)

Predicting Congestions empirically



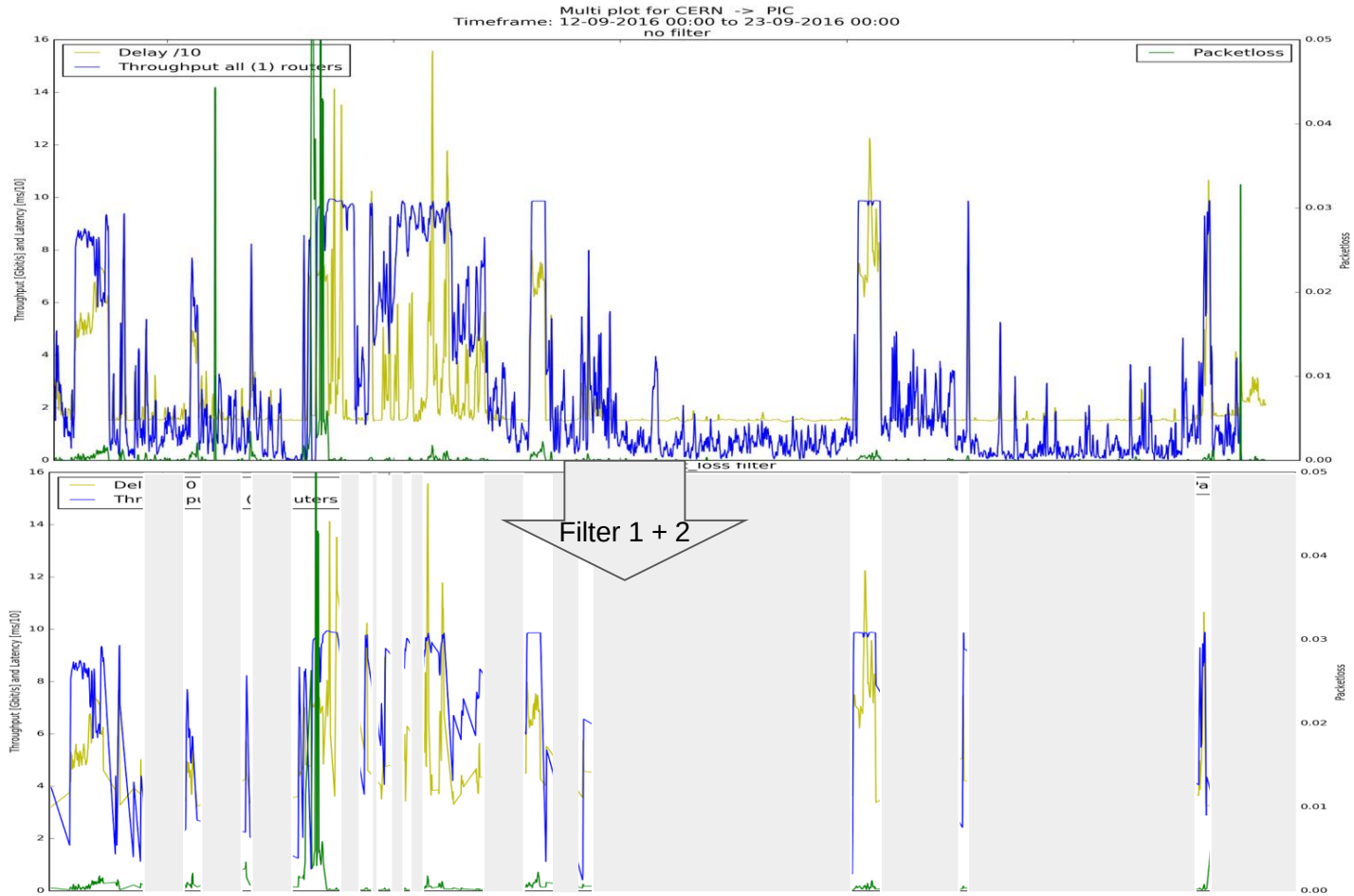
The upper Plot shows an example of used data. It was smoothed via a moving average.

Everything that got through Filter 1 and 2 (logical &&) is shown in the lower plot.

Filter 1: Packet loss below 0.0001

Filter 2: Delay varying less than one standard deviation

Predicting Congestions empirically

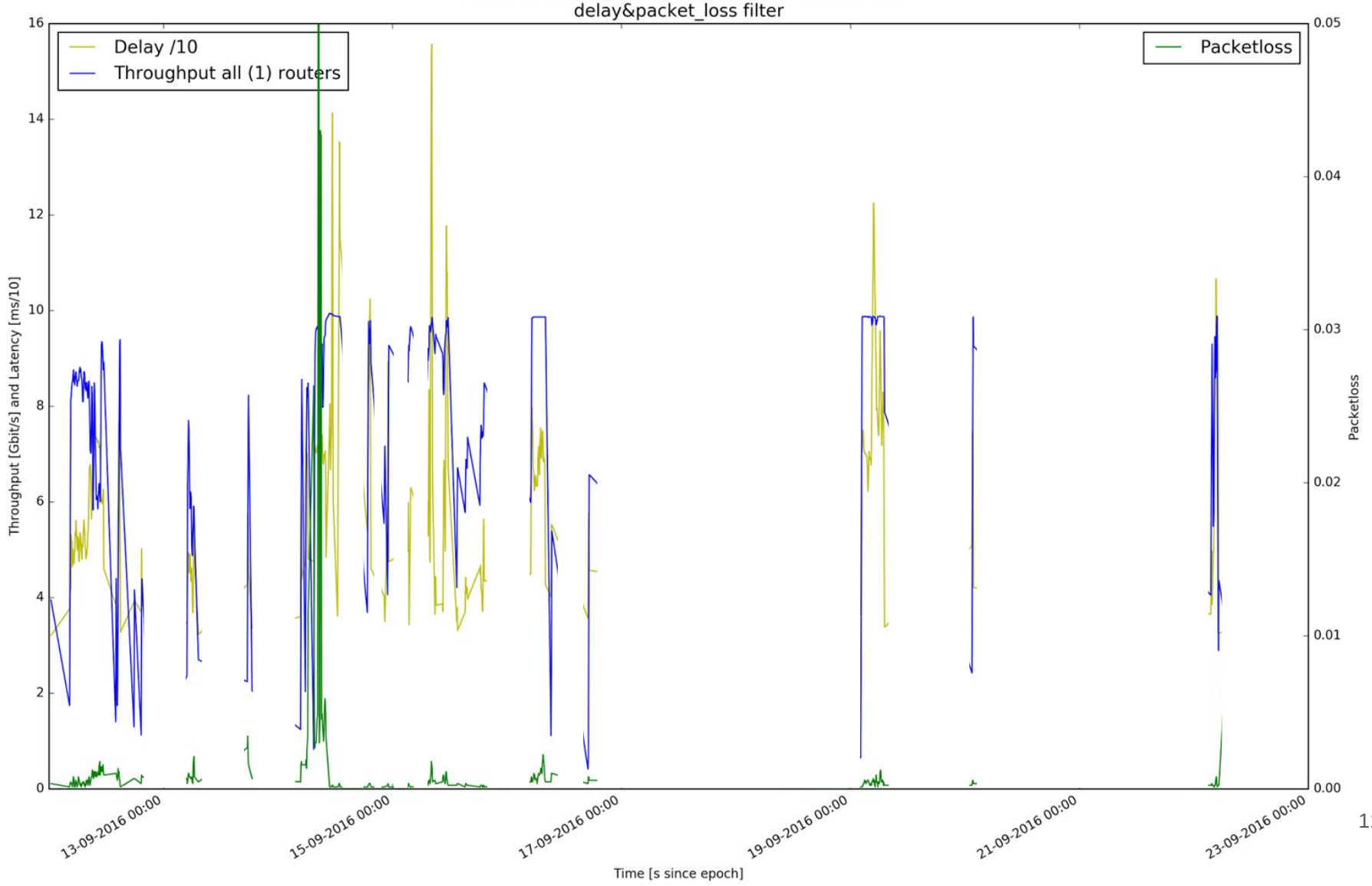


The upper Plot shows an example of used data. It was smoothed via a moving average.

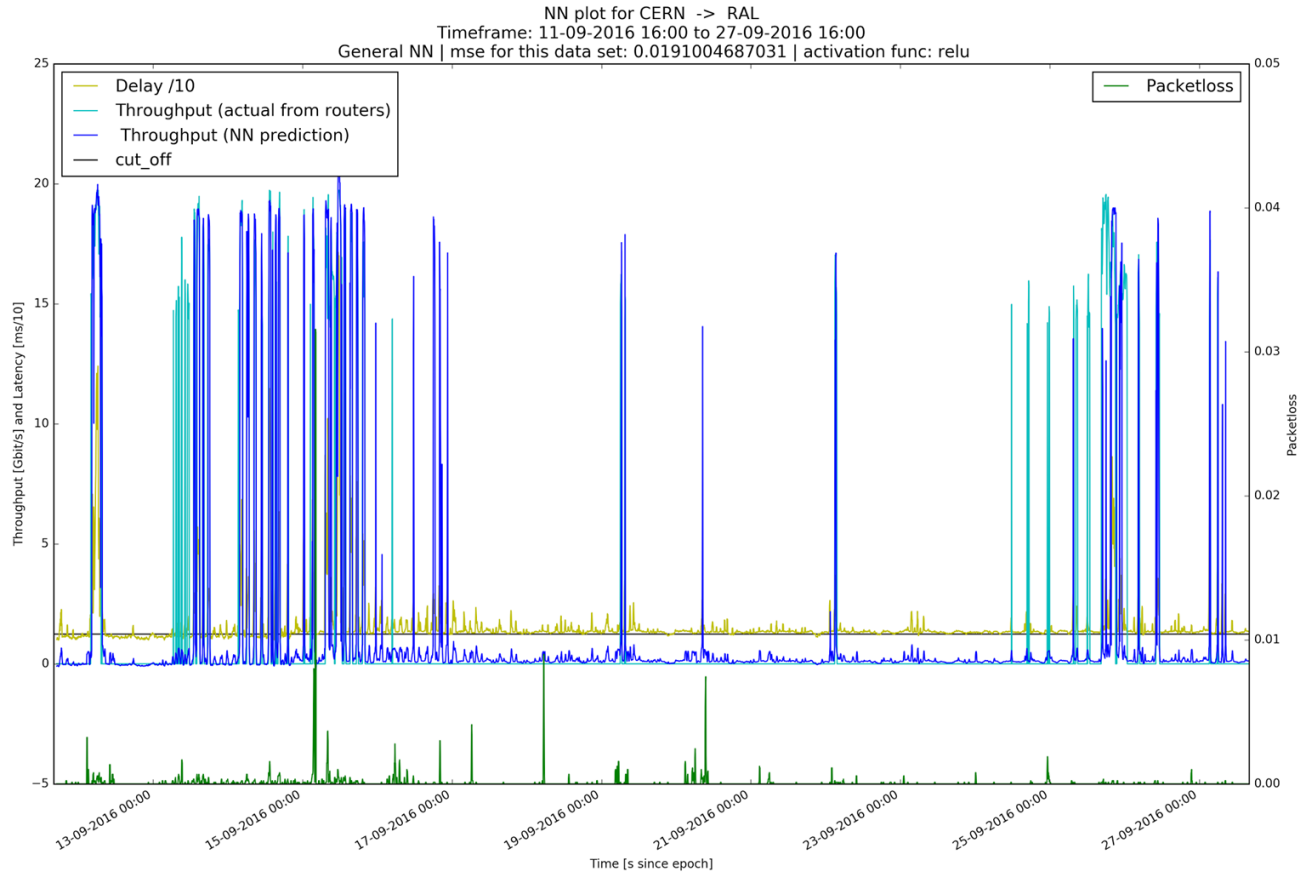
Everything that was not filtered out by Filter 1 or 2 (logical or) is shown in the lower plot.

Filter 1: Packet loss below 0.0001

Filter 2: Delay varying less than one standard deviation


Multi plot for CERN -> PIC
Timeframe: 12-09-2016 00:00 to 23-09-2016 00:00
delay&packet_loss filter
Delay /10
Throughput all (1) routers
Packetloss
Throughput [Gbit/s] and Latency [ms/10]
Packetloss
Time [s since epoch]
13-09-2016 00:00
15-09-2016 00:00
17-09-2016 00:00
19-09-2016 00:00
21-09-2016 00:00
23-09-2016 00:00

Predicting Congestions with machine learning



puts to the Neural Network

Smoothed delay & packet loss

15 measurements from the past (15 min)

Average, standard deviation and minimum over the whole observation time

rchitecture of the NN

Three hidden layers

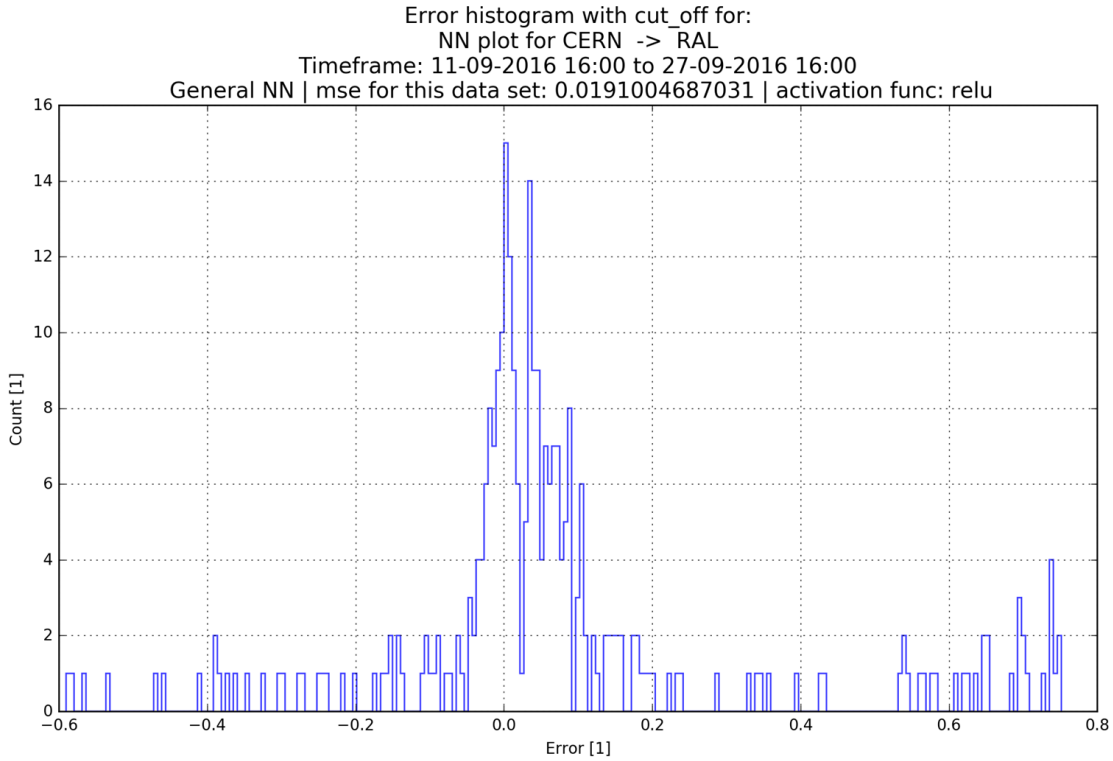
Decreasing number of neurons

Activation function: rectified linear unit

Training sets for the NN

Prediction if a connection is used above 70% of its capacity

Predicting Congestions with machine learning



The error is given in a measure of percent,
where 0.8 means the NN is 100% above the real value

Verification Results

Mean squared error on verification set: 0.019

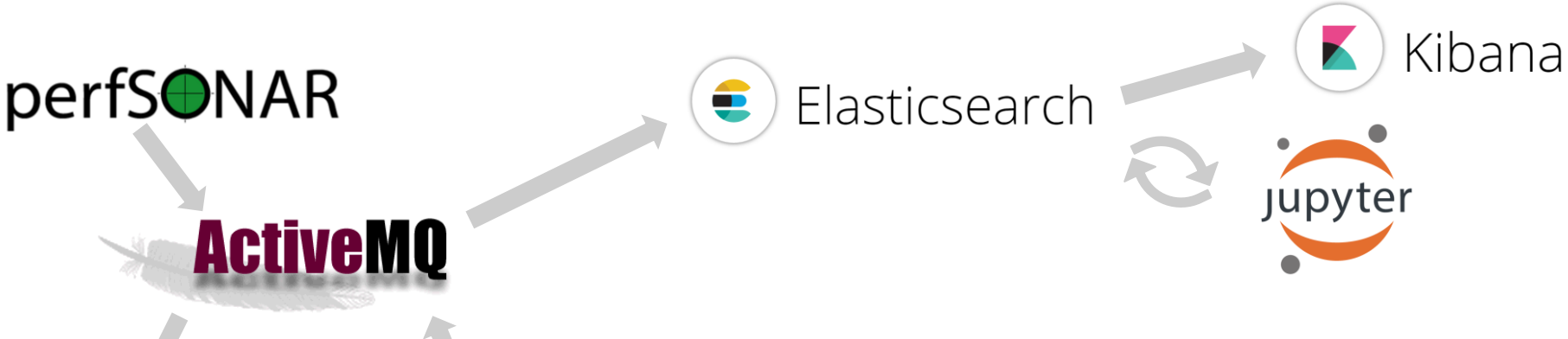
Seems to be working on a connection that the NN has never seen before

Quite good prediction of throughput spikes

Distribution of errors shows that the NN is likely not overtrained

In the error plot predictions below the cut_off line in the previous plot were excluded

Cost matrix for LHCONE (Architecture)



netTel

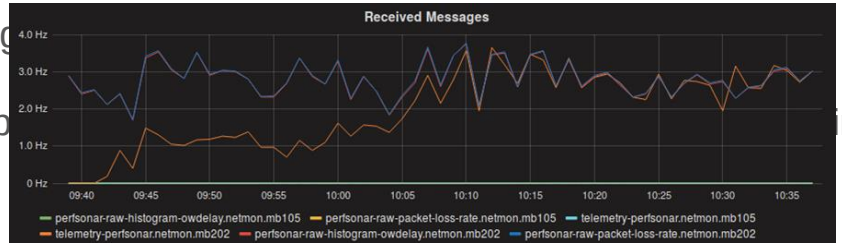
Implements both empirical and machine learning approach

Buffers up to 32 minutes of raw data

Uses: Keras, Theano, scikt-learn, pandas and numpy

Sing

Pub in



What’s next

- Network telemetry based on perfSONAR now published to production netmon brokers:
  - Models are easy to scale to near real-time for all existing links
  - Empirical model could be used as a simple binary filter for detection of anomalies
- More extensive validation needed for LHCONE
  - ESNet provides router traffic for some LHCONE links that could be used for this
    - We would need to find congested links in US and compare
- Exploring ways how we could use Hadoop/SPARK/DataTorrent stack for this
- Follow up work will be done within WLCG Network Throughput WG

Cost matrix for LHCONE (Architecture)