CERN

B E E R

# Improving site efficiency by integrating storage nodes and batch processing

# **Batch on EOS Extra Resources**

# BEER Contributors

- IT-CM
  - Tim Bell, **Ben Jones**, Domenico Giordano,
  - **Gavin McCance**, **Jaroslava Schovancova, Havard Tollefsen**
- IT-ST
  - Dirk Duellmann, **Massimo Lamanna, Herve Rousseau**
- IT-DI
  - Markus Schulz, Andrea Sciaba, Andrea Valassi, **David Smith**
- Petersburg Nuclear Physics Institute
  - **Andrey Kirianov**

# Background

- EOS is CERN’s large storage management system
  - https://eos.web.cern.ch/
  - Frequent reports at HEPiX ☺
- The hardware architecture follows CERN’s commodity paradigm
  - Disk servers are based on standard servers with shelfs of disks

# Motivation

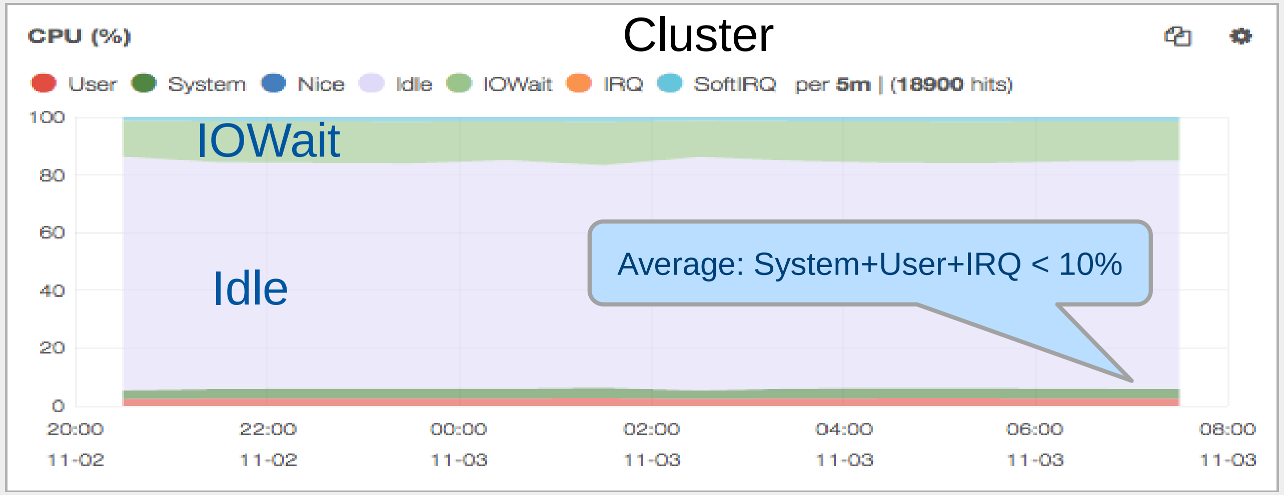
- Observation of relative low ***average*** loads on our storage systems
  - Often I/O bound (internally and network)
    - Most nodes are not CPU saturated even when they are I/O saturated
- Average usage per server node (snapshot):
  - Read: 35MB/sec - 67MB/sec, write: <10MB/sec
  - IOPS: read 290-500 Hz, write: 28Hz

- CERN has a relatively large storage system ***O(1000)*** nodes
  - Several have 32 cores with 2GB/core

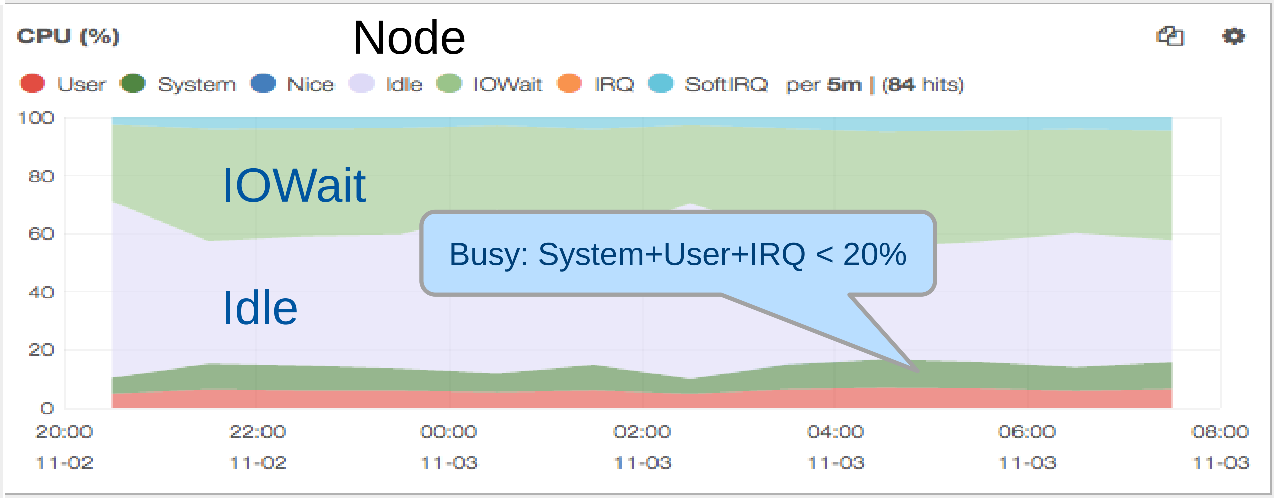
- Questions:
  - Can we make use of some of these cores?
  - What value does this correspond to?
  - Proof of concept tests where done by Andrey using some older nodes.

# Computing loads on storage servers

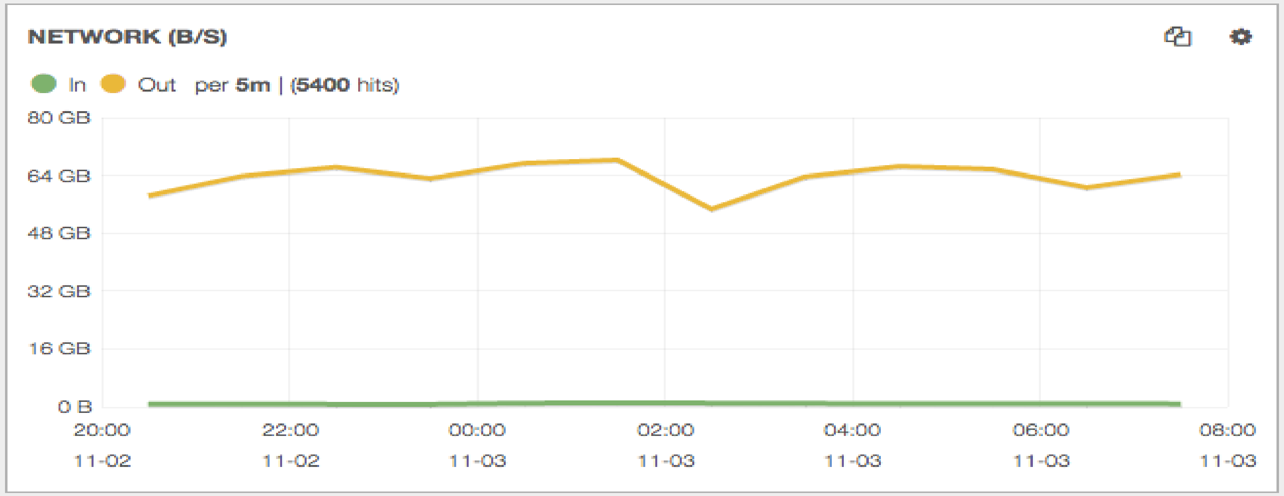
- On the ALICE EOS cluster ***average*** CPU utilization is ~20% – left plot
- ALICE EOS disk server with ***highest*** CPU utilization is ~60% – right plot
  - But: Mostly **IOWait**, which can be used by other processes → Significant Potential

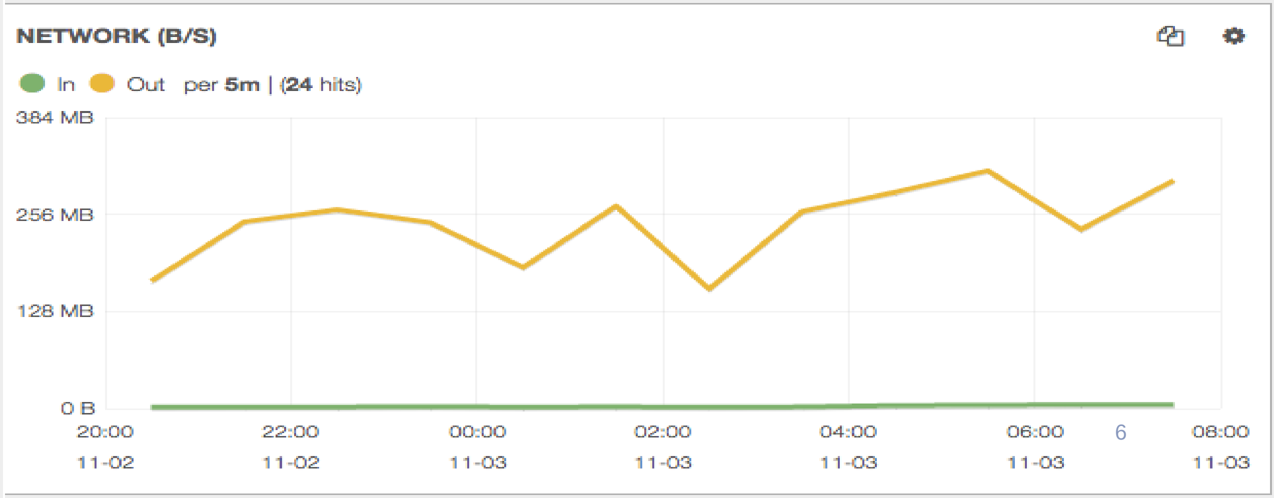


Average: System+User+IRQ < 10%



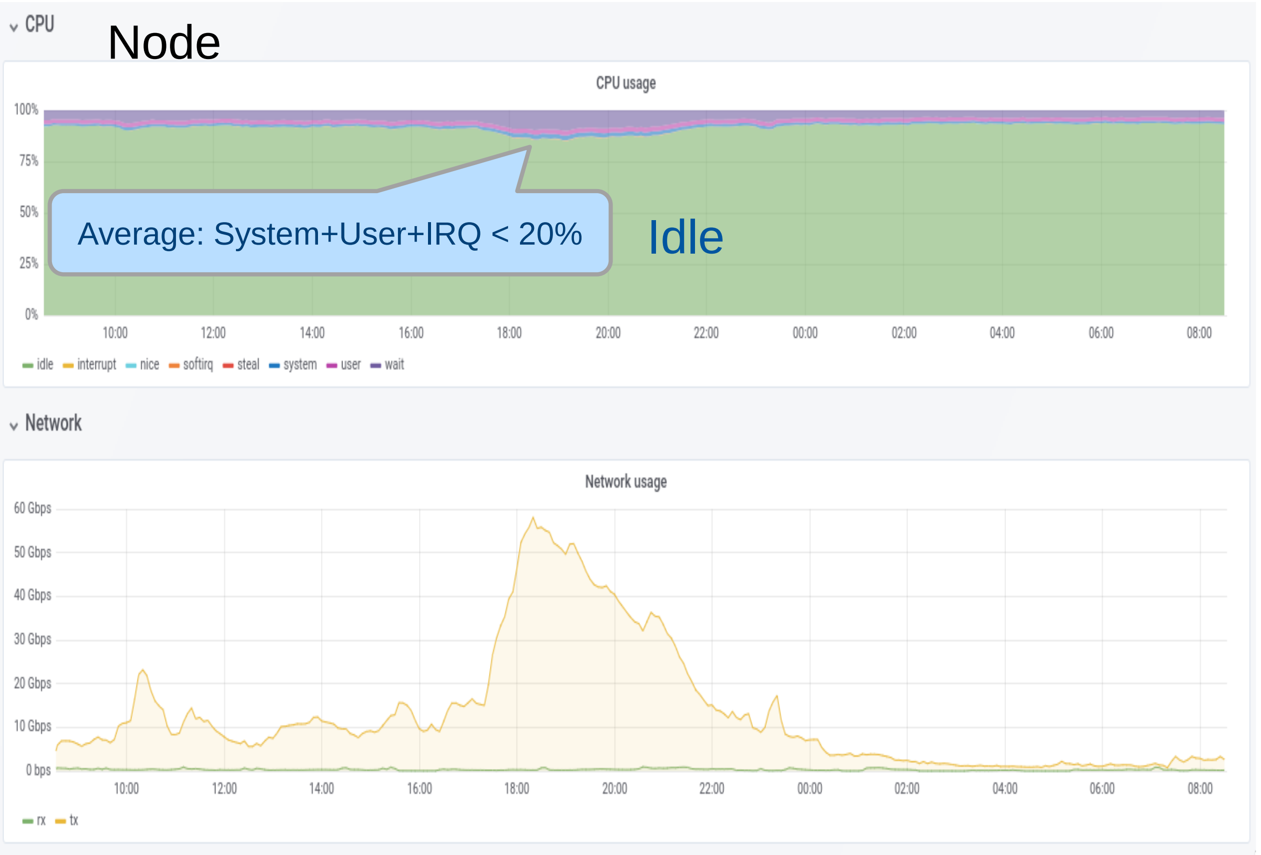
Busy: System+User+IRQ < 20%





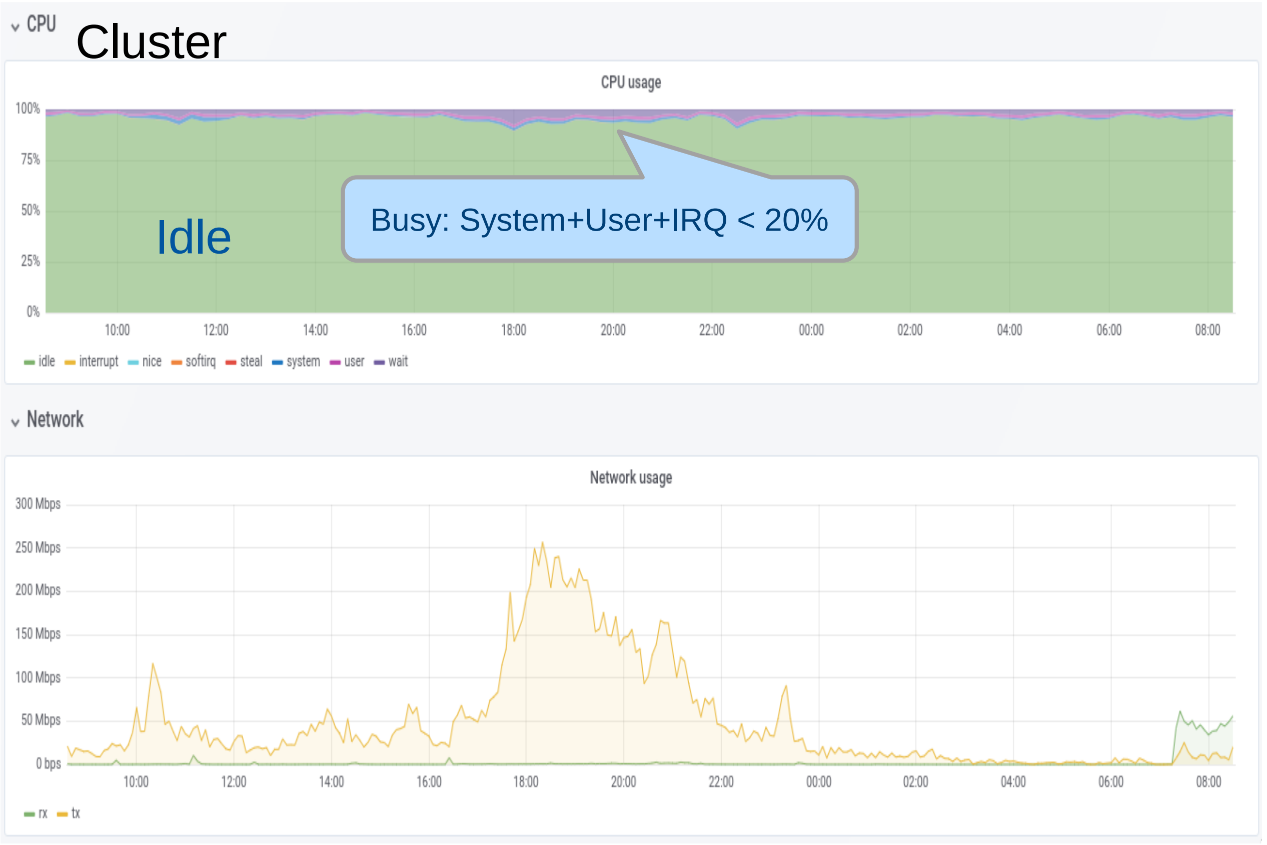
# Current Loads

- ALICE EOS cluster ***average*** CPU utilization is <=20% – left plot
- ALICE EOS disk server CPU utilization is <=20% – right plot
  - Mostly **IOWait**, which can be used by other processes → Significant Potential



Average: System+User+IRQ < 20%

Idle



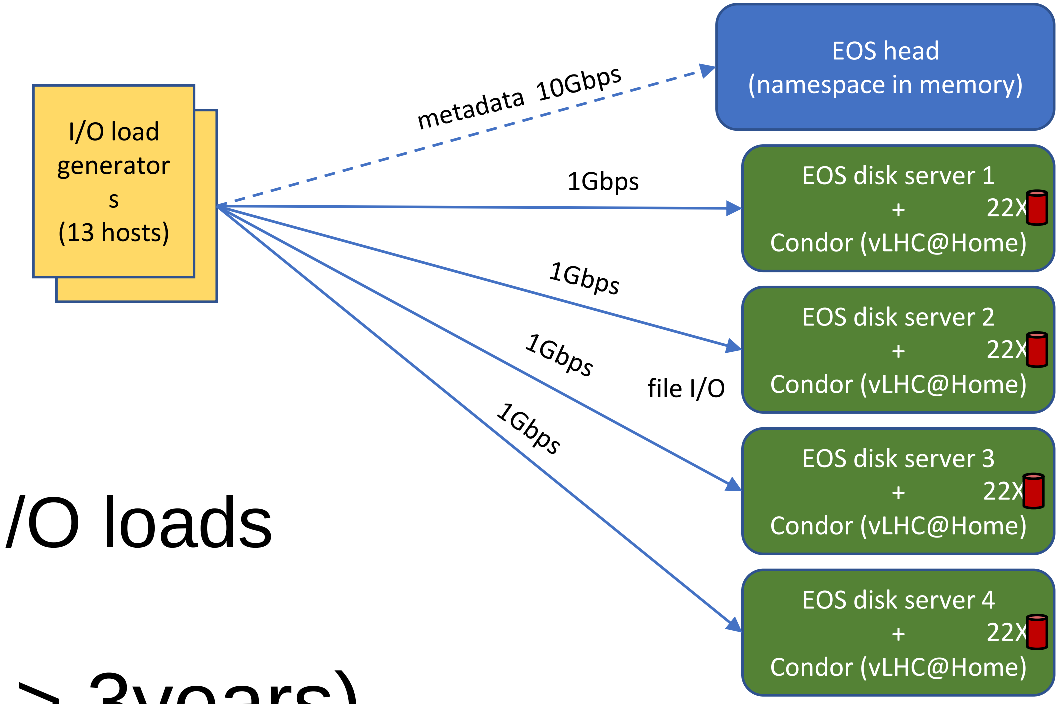
Idle

Busy: System+User+IRQ < 20%

# First Steps

- Proof of concept tests by Andrey Kiryanov
  - Testbed (puppetized)
    - Small EOS system(s)
    - Client/load-generator cluster
    - Condor based test with computational loads
      - Using lhc@home
      - Boinc based system
      - Using *nice* to limit impact on EOS

# First test system



I/O load generators (13 hosts)

metadata 10Gbps

EOS head (namespace in memory)

1Gbps

1Gbps

1Gbps

1Gbps

file I/O

EOS disk server 1 + 22X Condor (vLHC@Home)

EOS disk server 2 + 22X Condor (vLHC@Home)

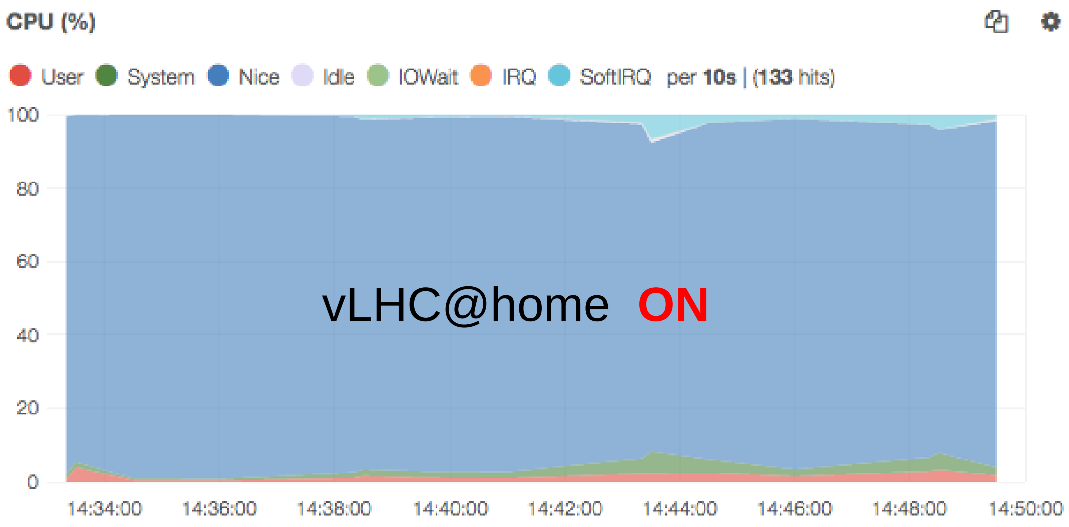
EOS disk server 3 + 22X Condor (vLHC@Home)

EOS disk server 4 + 22X Condor (vLHC@Home)

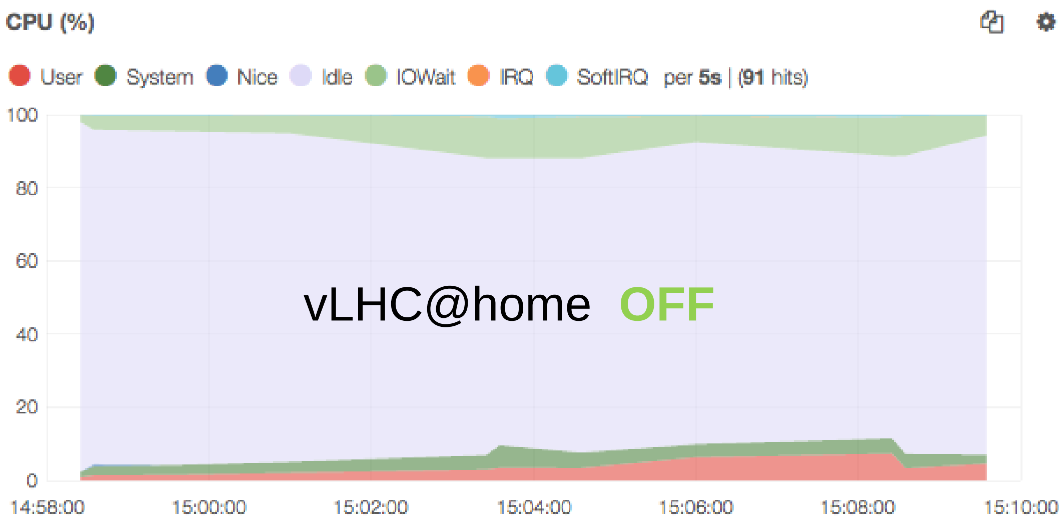
- ❑ 13 hosts as load generators
  - ▪ ***EOS Xrdstress*** tool generates I/O loads
  - ▪ Up to full saturation
- 4 EOS storage nodes (old nodes > 3years)
  - 22 disks 1 Gbits network, EOS 4.1, xrootd 4.5, 6.54 HEP-SPEC06/core, 4 cores
- 1 EOS head node
  - 10Gbits
- ***Condor*** runs payloads directly, no virtualization involved
  - ***psacct*** has been used for accounting

# Test system performance measurements

Xrdstress  on and the vLHC@Home processes in the background.



vLHC@home **ON**



vLHC@home **OFF**

No significant difference in I/O:

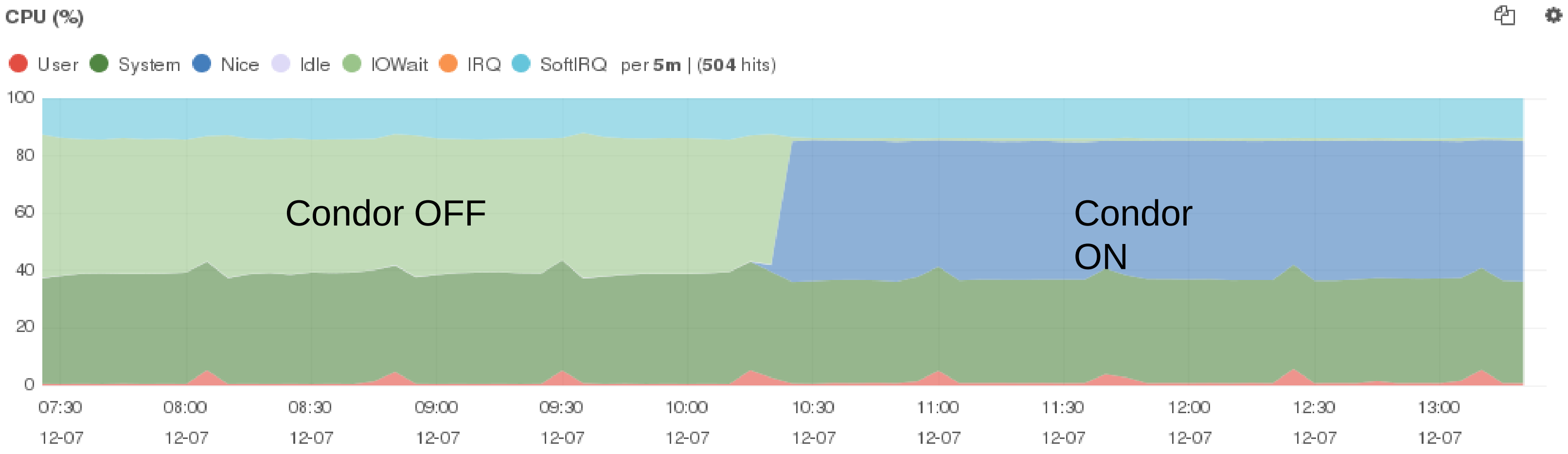
- 357 MB/s read
  91 MB/s write
  in hybrid mode

- 357 MB/s read
  96 MB/s write
  in EOS-only mode

- Servers limited by **network**.
  - Load/node  comparable with production
- ~90% CPU used by vLHC@home

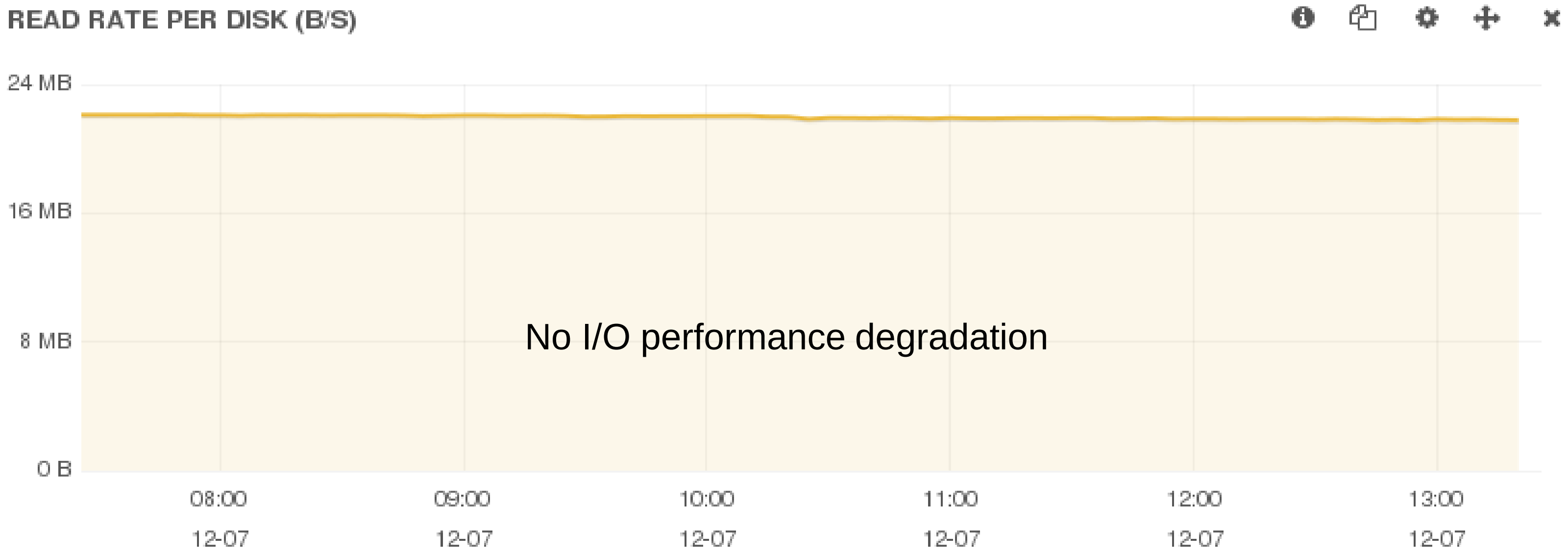
# Corner case: 100% I/O saturation

- ❑ Used Hybrid Testbed v1
- ❑ Generated ***local*** load on disks
- ❑ Almost no impact on I/O or memory
- ❑ ~40% CPU used by vLHC@home



Condor OFF

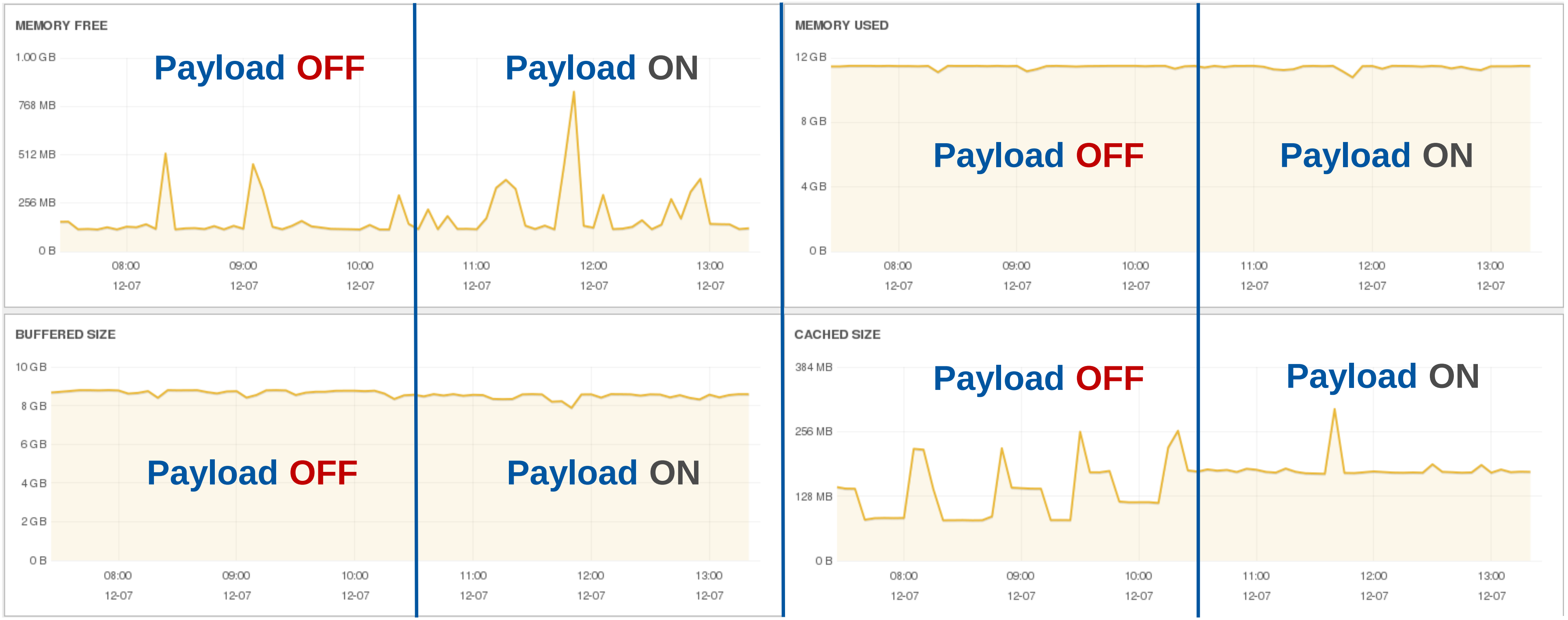
Condor ON



No I/O performance degradation

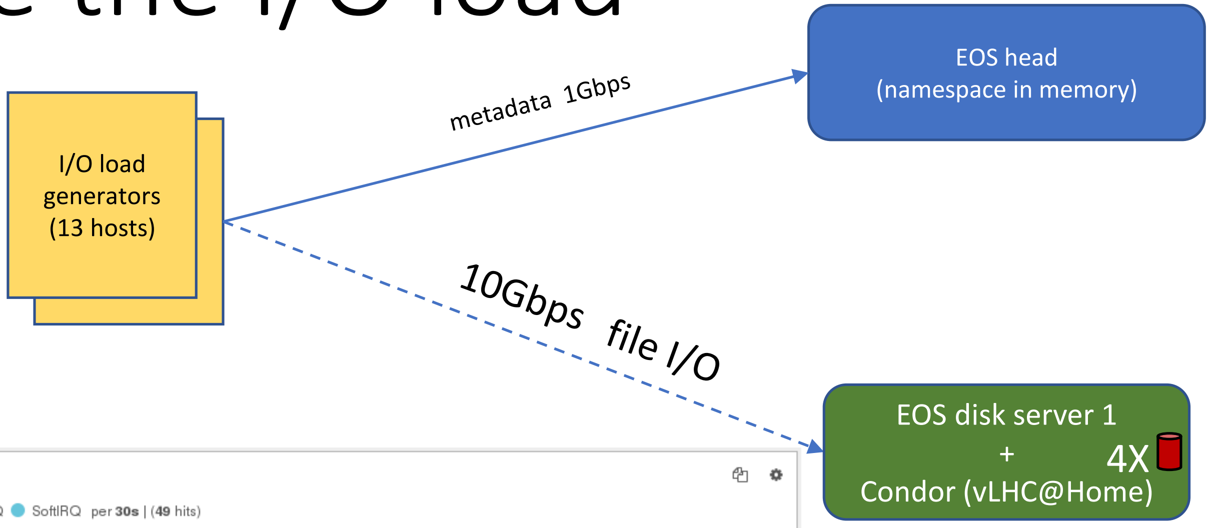
# Corner Case

There’s almost no visible impact from extra compute payload: neither disk performance nor memory utilization suffer from background low-priority tasks.

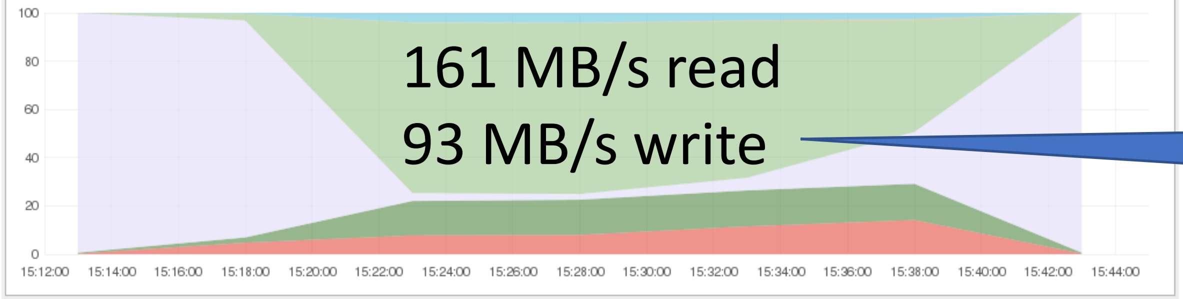


# Testbed 2 to increase the I/O load

- Hybrid Test Bed vs 2
- 10 Gbps link for data access
  - But only 4 disks


I/O load generators (13 hosts)
metadata 1Gbps
EOS head (namespace in memory)
10Gbps file I/O
EOS disk server 1 + Condor (vLHC@Home)
4X

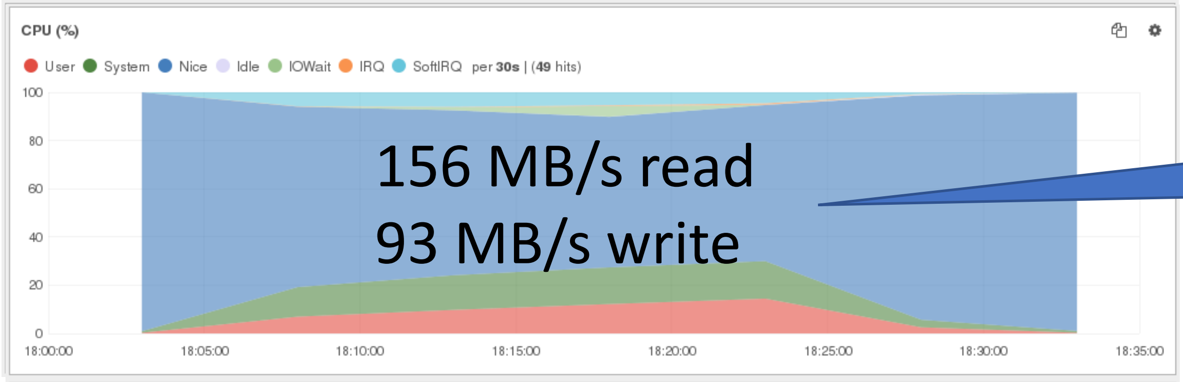
- Very little impact on I/O
  - Limited by disks ....
- **> 60% CPU for vLHC@home**



161 MB/s read

93 MB/s write

EOS only

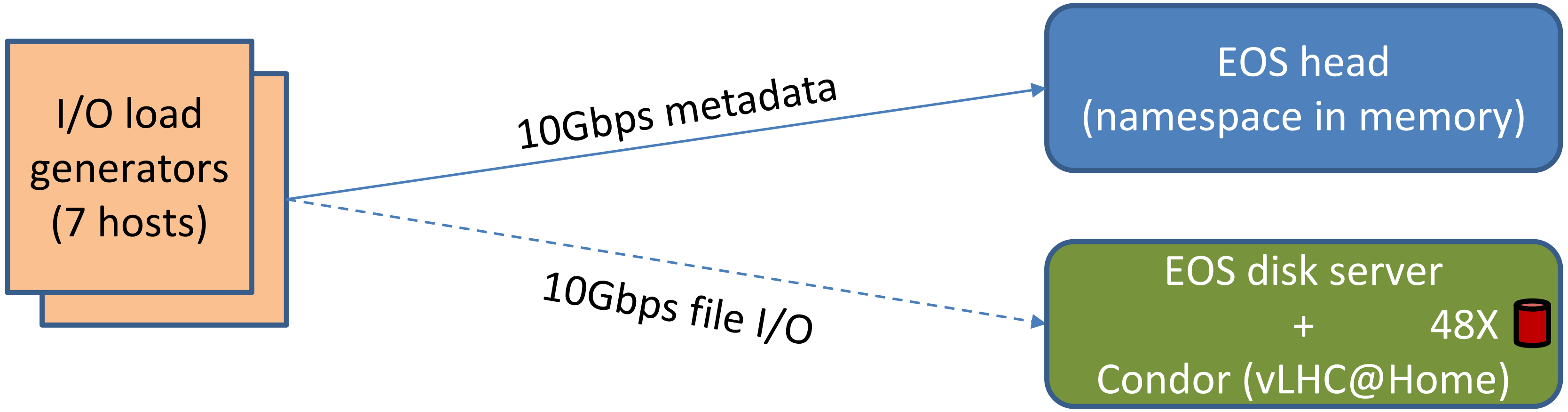


156 MB/s read

93 MB/s write

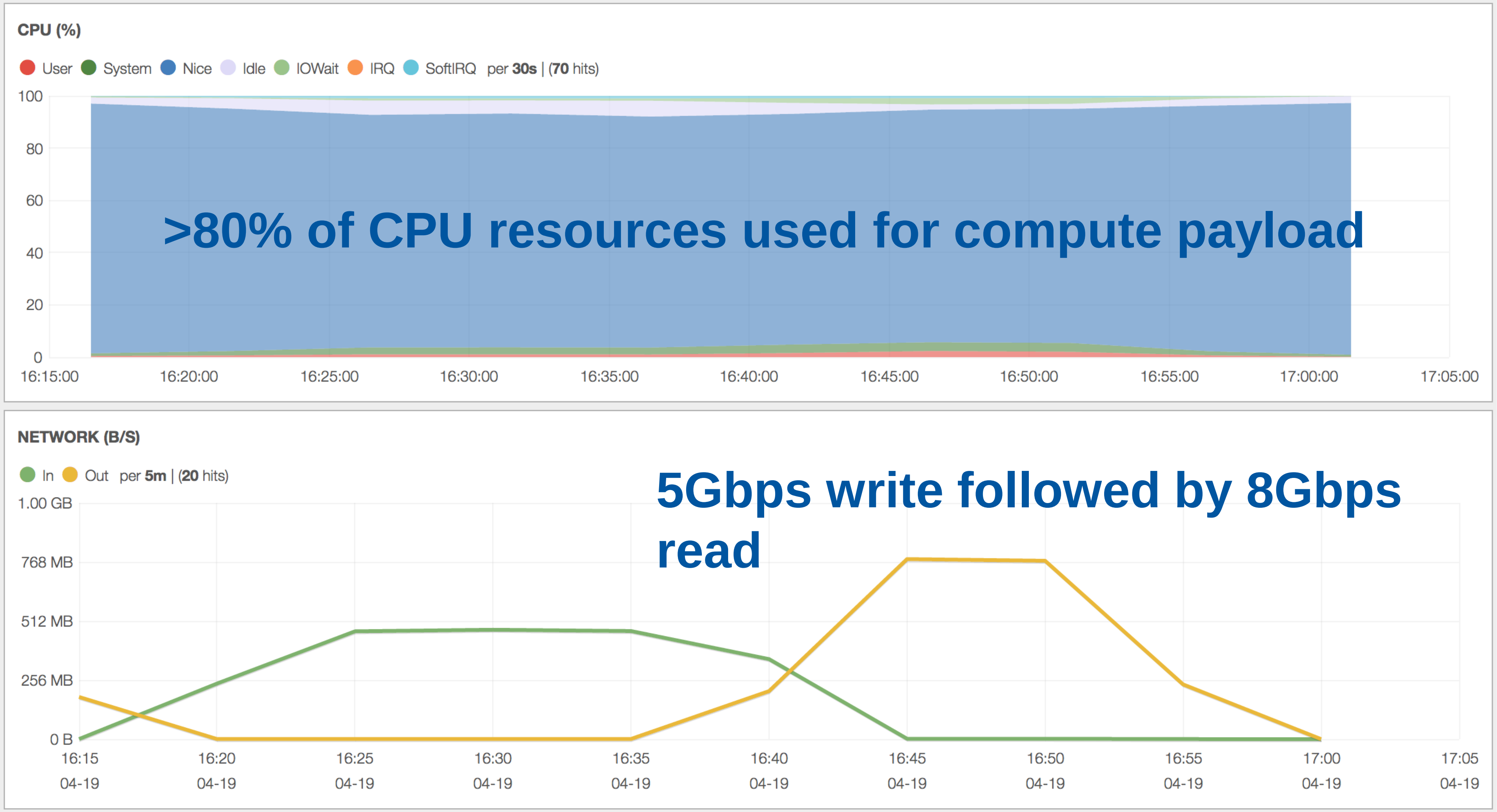
EOS + vLHC@home

# Tests with more recent hardware


I/O load generators (7 hosts)
10Gbps metadata
10Gbps file I/O
EOS head (namespace in memory)
EOS disk server + Condor (vLHC@Home)
48X

- New disk server with decent hardware
  - Dual E5-2630 v3 @ 2.40GHz (32 vcores with HT)
  - 64 GB RAM
  - 48 spinning disks 6TB each + 2 SSD (OS + swap)
  - 10 Gbps network

# Running with compute payload

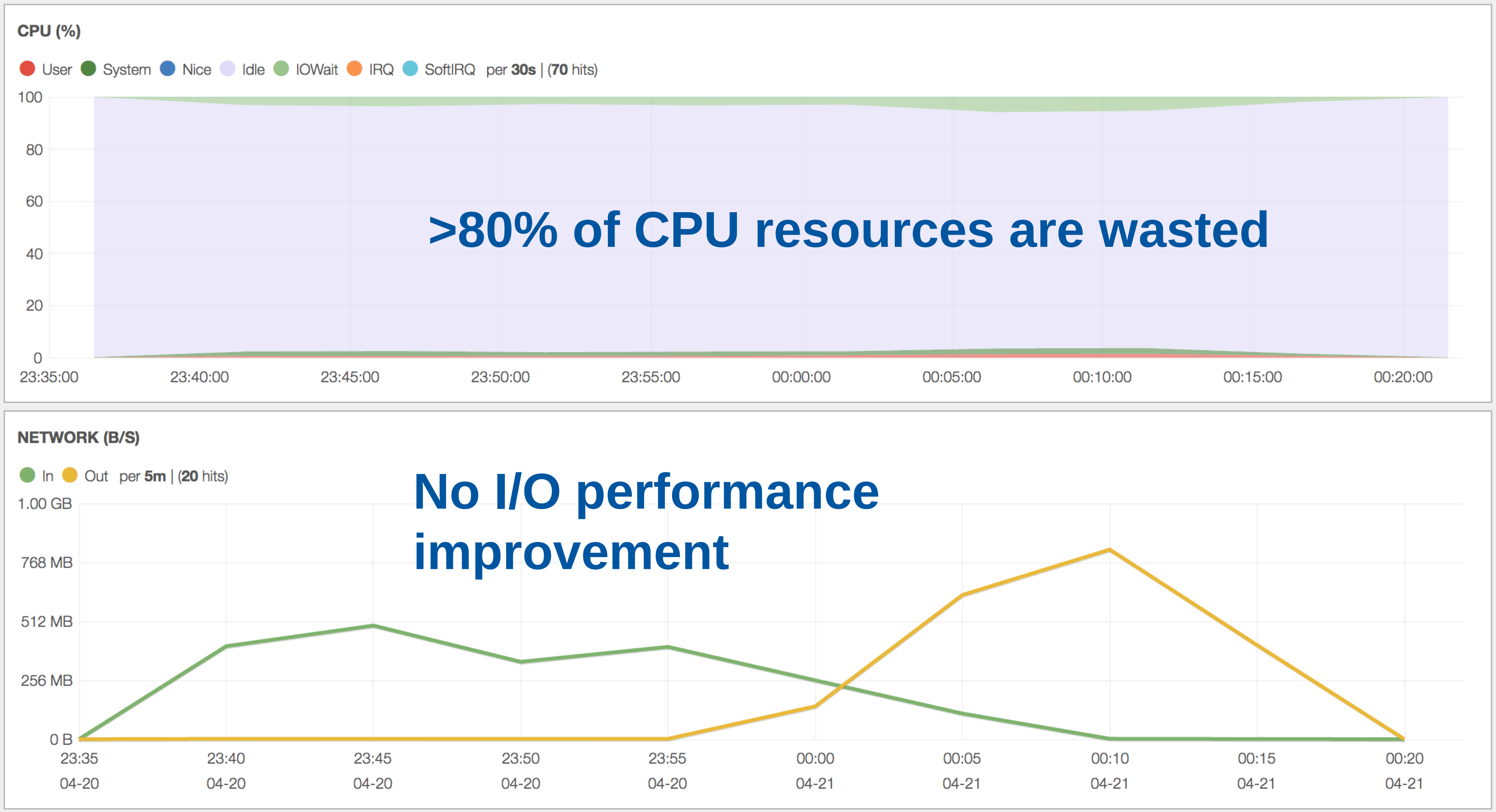


**>80% of CPU resources used for compute payload**

**5Gbps write followed by 8Gbps read**

Almost identical results

# Running with no payload

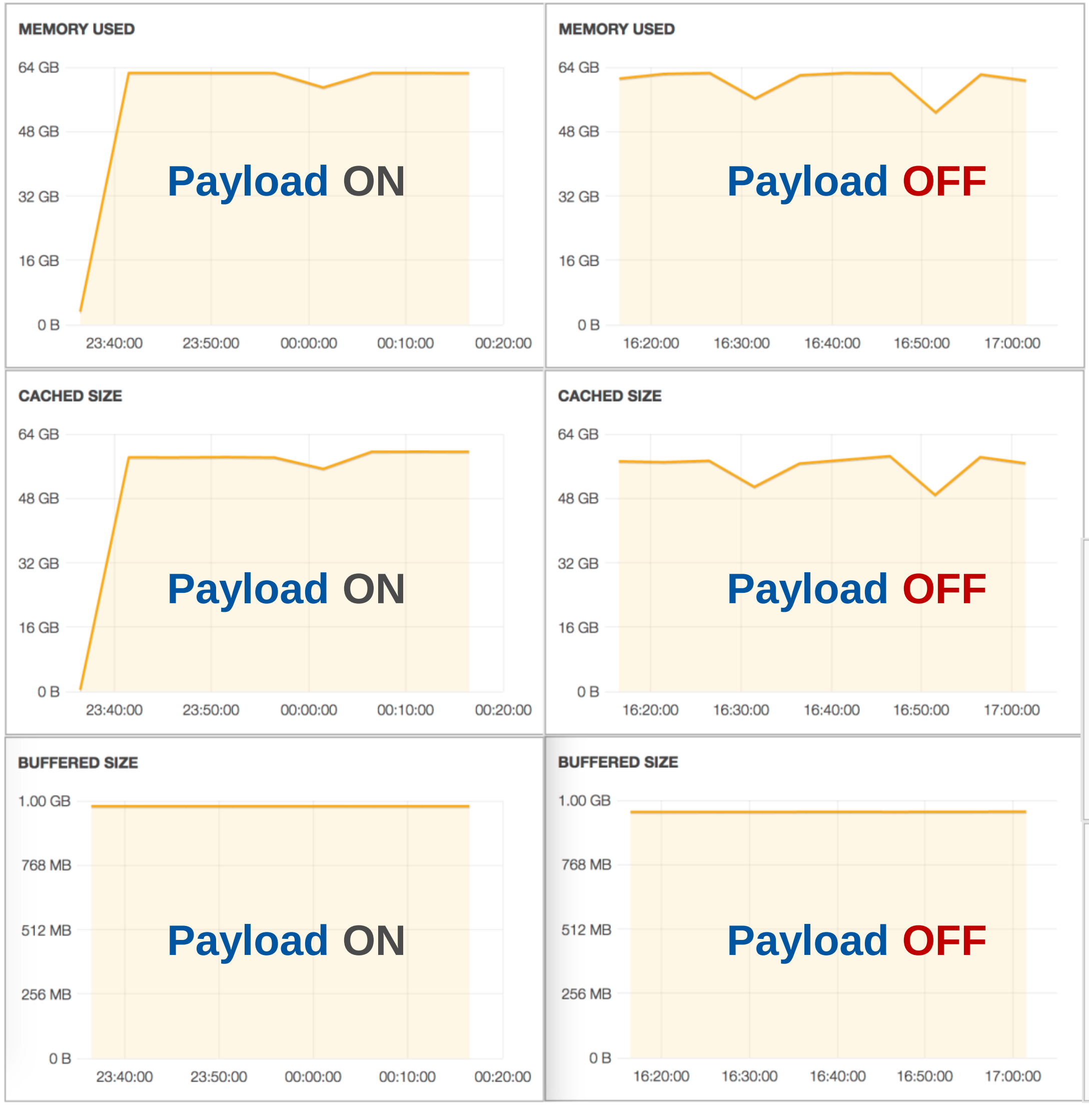


**>80% of CPU resources are wasted**

**No I/O performance improvement**

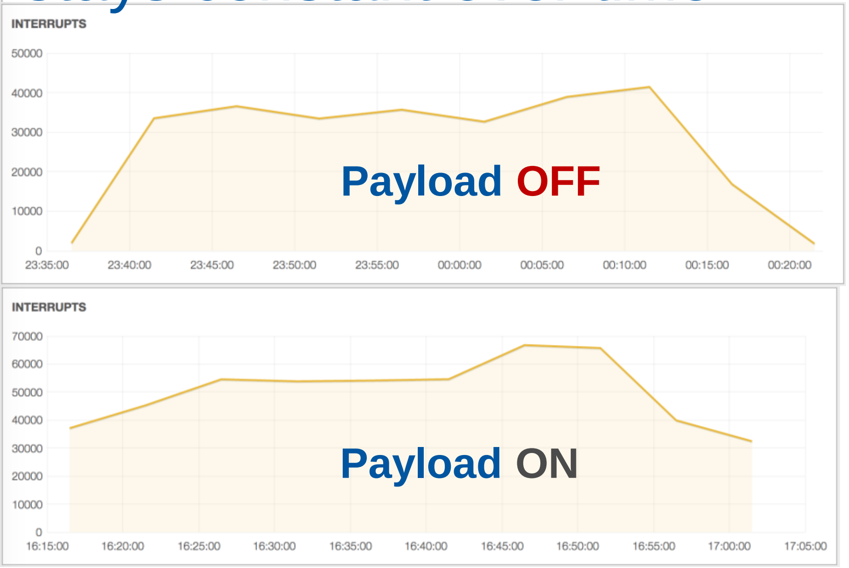
# Other Resources

Memory



Compute payload doubles the interrupt rate, but modern CPUs cope easily with it.

The number of used sockets is also doubled, but stays constant over time.



Interrupts

# What did we learn?

- With a simple setup **Storage** and **Computing** can be run without much interference
  - HTCondor + ***nice***
- For typical I/O loads in production we can expect to use >80% of the CPU for non storage tasks
  - Worst case would be about 50%
- Interesting....... But....

CERN

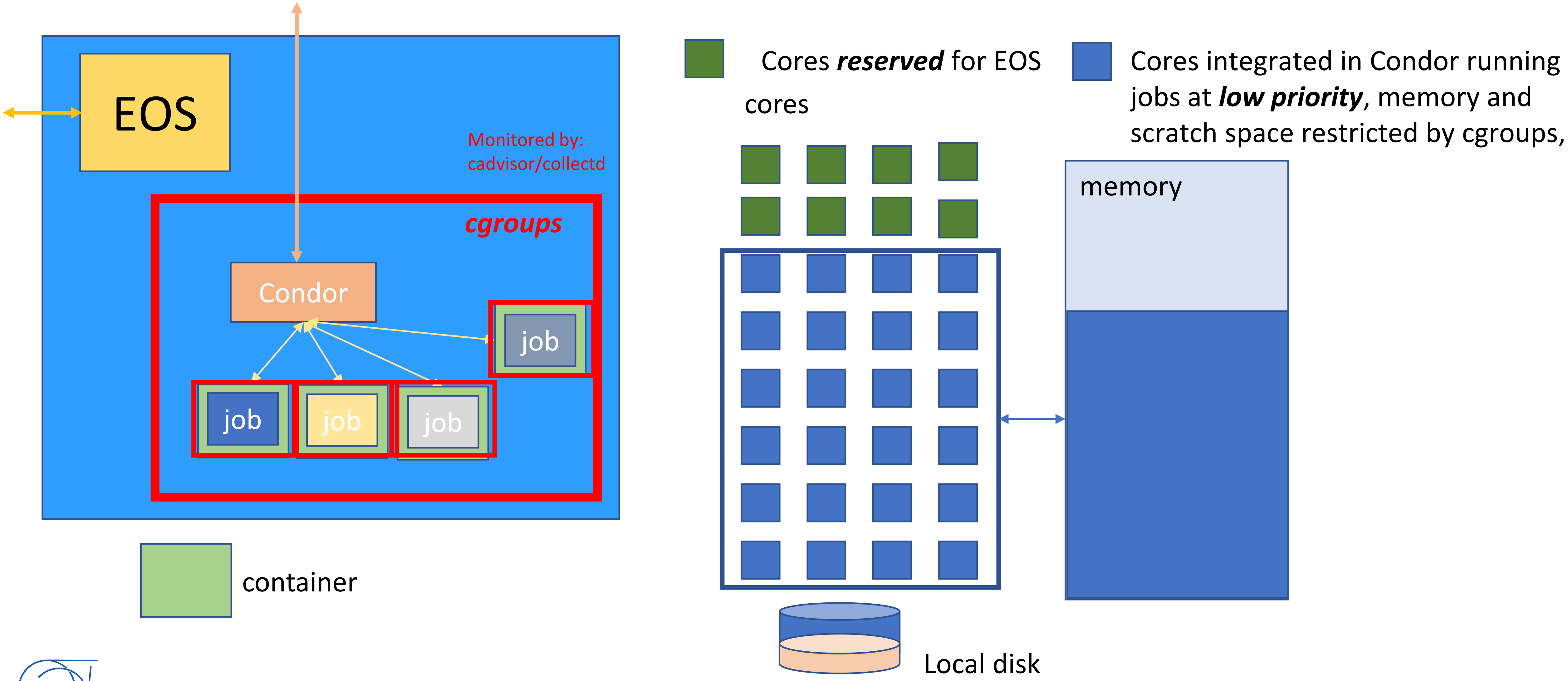
# How to turn this into production?

- Needs to be deployable
  - Configuration Management challenge
    - Two services on the same node
- Protecting the primary task (EOS)
  - Resources (CPUs, memory .....)
  - Halting the computational tasks on demand
- Monitoring

# A Model emerged

- “Partition” the resources
  - To guarantee that storage performance is not crippled
  - To provide accountable resources (not like lhc@home)
  - Control groups (***Cgroups***)
  - Run Condor jobs in Containers
    - Using Cgroups to limit resource usage
  - One puppet configuration for the node
- Integrate resources in CERN’s Condor batch system
  - Queues for suitable workloads
- BEER Pilot to explore this approach
  - Participation from storage and batch team
  - JIRA for ticketing

# Condor + Containers



🟩 Cores *reserved* for EOS cores

🟦 Cores integrated in Condor running jobs at ***low priority***, memory and scratch space restricted by cgroups,

# Testbed #3

- Three disk servers: each
  - 48 6TB HDD (1 HDD apparently failed on one server)
  - 2 x E5-2630 v3 (haswell; 2 x 8 physical cores => 32 w/ SMT)
  - 2 x 800GB SSD (Intel DC S3510 series; 0.3 DWPD for 5 yrs)
  - 10Gbit network

- Centos7; EOS 0.3.240 (Aquamarine)
  - Using puppet, hostgroup based on eos hostgroup and using modified eos module and cerncondor module (beer branch)
- Local Disc Setup:
  - 1 ssd set aside for the batch work (including addition of 96GB swap)
  - CVMFS installed
  - cerncondor module used to install and run condor
- Changes in the batch environment (possibly amongst others):
  - set memory.memsw.limit_in_bytes and add condor to the cpuset cgroup controller: cpuset.cpus and cpuset.mems
  - (systemd already set mem limit, but no support for memsw or cpuset)

# Limits

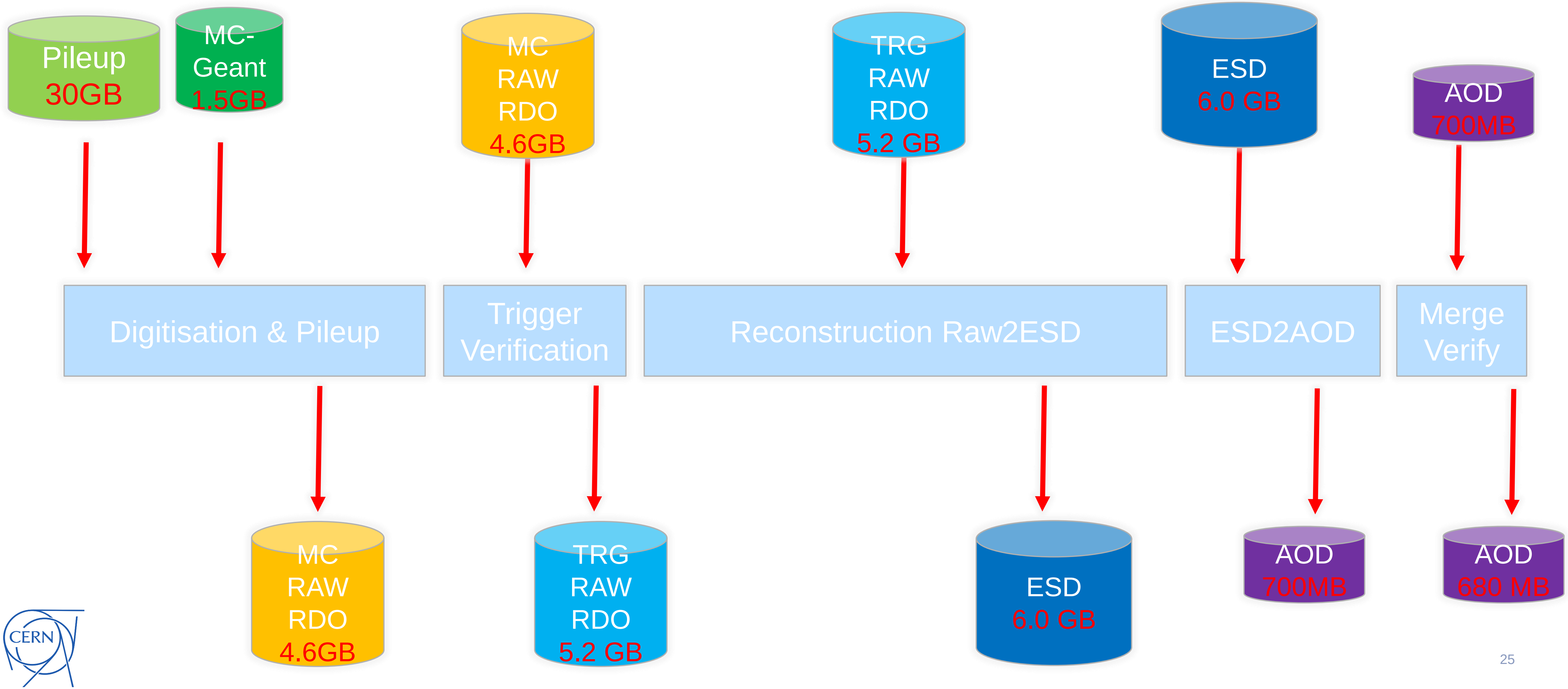
- Memory limit
  - is set as a parameter in /etc/systemd/system/condor.service and systemd uses the setting when setting up the ***cgroup*** for the condor service: 48GB
  - But does not limit swap usage
  - Modified condor.service to also set ***memsw*** limit (ram + swap) to 96GB
- Add condor to another ***cgroup*** using ***cpuset***
  - cpus 2-7,10-15,18-23,26-31
    - leaves 4 physical cores entirely excluded covering both sockets
  - Later only number of cpus has been limited.
- Condor configured to offer 24 job slots and 96GB ram
- Number of processes has been limited to 8000
- ***blkio*** is used to control the I/O scheduling
  - Blkio.weight == 50
- Network traffic limits can be set via iptables on the docker level, but are currently not used.
- See detailed setup description at the end

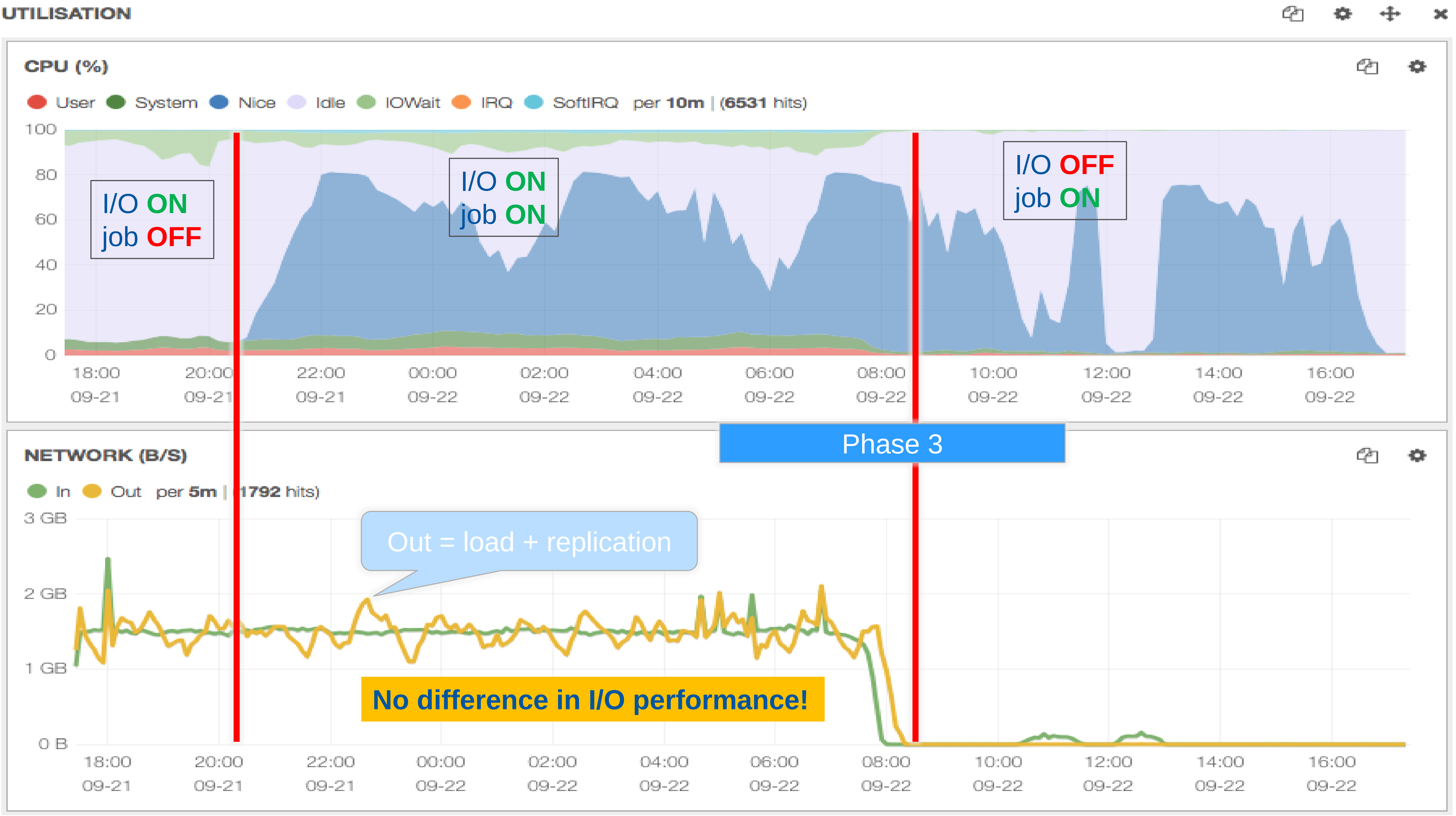
# Load generation

- For load run 10 instances of xrdstress
  - 4 jobs; 20 files; rw; size 1GB +- 256MB on 4 hosts
  - No difference between 3 and 4 hosts → saturation
- Run 3 ATLAS Pile job (8 cores per job)
  - Staging pileup data on node
  - Signal + min bias mixing;
  - Digitisation
  - Trigger simulation
  - Reconstruction
  - Convert ESD to AOD
  - Start jobs with short delay
  - Similar, shorter, job has been added to HammerCloud
    - For generating steady stream of jobs

# Test Job:

## 2000 MC Events, 8 cores





I/O **ON**
job **OFF**

I/O **ON**
job **ON**

I/O **OFF**
job **ON**

Phase 3

Out = load + replication

**No difference in I/O performance!**

# Job Performance

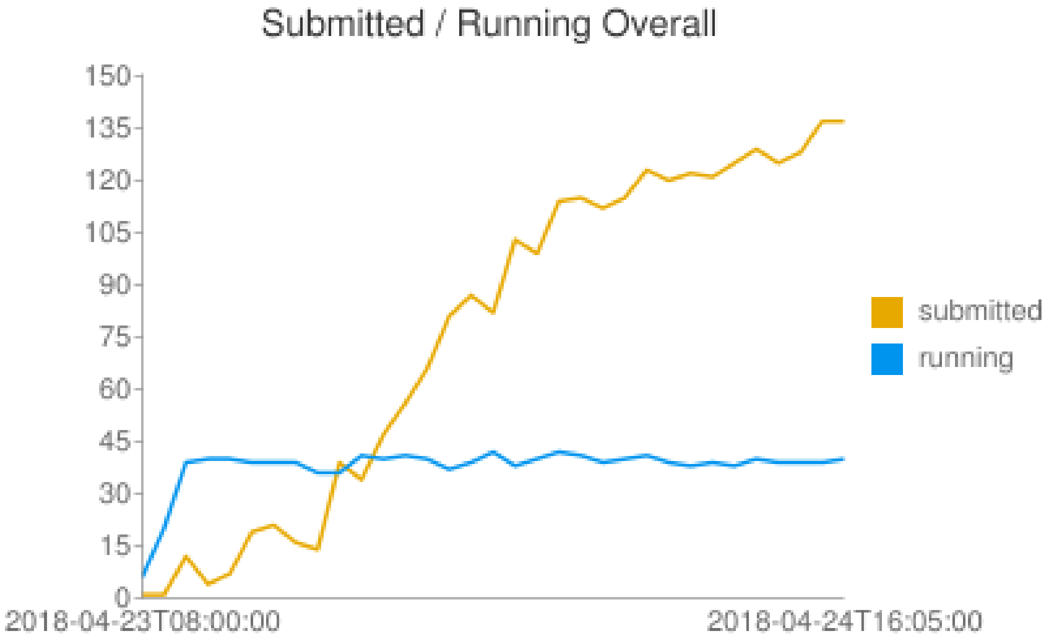
CPU and WALL time are not affected by EOS I/O

- Phase 1: **I/O ON** 9 jobs considered
  - CPU time (not including for stage in): **97901s** spread 2003s
  - WALL time (not including stage in): **15172s**: spread 308s
  - WALL time (stage in): 792s: spread 250s
- Phase 2: ***I/O OFF*** 8 jobs considered
  - CPU time: **95709s**: spread 5048s
  - WALL time (not stage in): **14320s**: Spread 625s
  - WALL time (stage in): 2719s: Spread 64s
- Phase 3: **I/O ON and OFF**,9 jobs
  - CPU time **98600s**: spread 2034s
  - WALL time (not stage): **14651s**: spread 403s
  - WALL time (stage in): 1063s: spread 348s

**WALL for stage in isn't understood**

# Preproduction

- 3 test servers
- 4 nodes from EOS pre-production cluster
- HammerCloud (ATLAS Pile job)
  - Hammercloud submits to Condor, Condor schedules and starts the job →like a standard Grid job
  - ATHENA MP (i.e. each job starts 4 processes)
    - About 30 mins runtime (choose a small number of events / job)
- Currently 70 more nodes are prepared
- Opening for use by experiments use soon after
  - Not as a production service



# Still to be addressed

- Currently we only have the standard node monitoring (monit)
  - We need, at least for the first “real” jobs more detailed monitoring
- Running tests with production I/O loads
  - With and without jobs
  - Running stress tests on production services is not advisable □

# What have we learned?

- ❑ There is always room for another

# Details on the configuration

- EOS standard configuration has been extended by using another module called “cerncondor”:

  https://gitlab.cern.ch/ai/it-puppet-module-cerncondor/tree/beerlite/code

  On disk servers where htcondor is wanted, the cerncondor module is listed in the disk server’s hostgroup along with some related parameters.
  e.g. here is a commit adding it to a selected number of PPS nodes:

  https://gitlab.cern.ch/ai/it-puppet-hostgroup-eos/commit/2a237b3806eaad608ff8bc6616b114ffb57b6273

  The cerncondor module installs htcondor & cvmfs. The parameters set tell puppet how to configure htcondor (number of job slots and memory etc.) The core count parameter is written in terms of a number of cores to reserve, and htcondor is configured by puppet using the total number of cores available in the machine minus the number reserved.

  Docker is now also configured explicitly in the hostgroup, The puppet docker configuration lists a number of images that puppet ensures are installed locally: Images are hosted at a repository, and if the image changes on the remote repository puppet will update the local image. The image name used by the jobs is set on the htcondor servers and not the EOS/beer nodes themselves. Jobs do not modify the image, changes a job makes are kept separate and discarded at the end of the job.

  Jobs are started by htcondor, as a docker job. HTCondor itself has code to invoke the docker-run command to start the job. The user can specify the memory and number of cores requirements inside the htcondor job submission file. HTCondor will make sure the memory limit is passed to docker using the appropriate option, which in turn docker uses to set a cgroup limit. (docker sets the RAM limit as supplied, and the RAM+swap limit to be twice that). HTCondor also instructs docker to set some other cgroup parameters such as cpu shares, which limit the priority of the job tasks with respect to other processes on the system. These cpu shares can not be set by the user directly, but are based on the number of job slots (i.e. cores) that the user declares are needed by the job. HTCondor also accounts for the total number of such slots it has running, and it will stop starting new jobs once all available slots are occupied.

  In addition to the cgroup setup that htcondor instructs docker to do, the cerncondor configuration was also adapted to put all the docker jobs within a sub-cgroup, with some master settings on the upper level cgroup. This was a way to limit things separately of what htcondor requests, or what docker would set. The additional limits added are: the total memory limits, in terms of RAM and also RAM+swap. Also the number of processes which can se started is limited, (“pids.max” and is 8000). The cgroup which can control per block device IO scheduling is also set, so that priority for the jobs is less than other tasks on the system. (blkio.weight is set to 50).

  The cerncondor module was also adapted to support setting cpusets via cgroups, and also network traffic limits with iptables on the docker virtual devices. However these two possibilities are not currently used.

# What could be gained for CERN?

*precision at best 10 – 20 %, but based on conservative assumptions*

- Worst case:
  - 40% can be used:
    - Our measurements show that this is far less than what can be done when the node is fully saturated
  - 780 * 32 * 0.4 = 9984 cores ~ 99.840 HEP-SPEC06
  - **312 Computing boxes**
- Based on average load:
  - >80% can be used: 780 * 32 * 0.8 =19968 cores ~ 199.680 HEP-SPEC06
  - **624 computing boxes**