US-ALICE Grid Operations review

Marta Bertran Ferrer
PhD Student - CERN

Contents

- Initial analysis on CPU Pinning adoption at ORNL and HPCS_Lr
- Whole node oversubscription opportunities at US sites
- CPU benchmarking for TTL tuning
- Initial TTL tuning attempts in US sites

Initial analysis on CPU Pinning adoption at ORNL and HPCS_Lr

Initial analysis on CPU Pinning adoption at ORNL and HPCS_Lr

Sites were configured with ‘cpuIsolation’ LDAP flag progressively.

- HPCS_Lr (Configured before 21/11 → First day of traces)
- ORNL (Configured on 30/11)

Methodology of analysis: Parsing all job traces from November 21st to January 18th.

Comparison with jobs executed before that day not possible, so evaluation performed with jobs executed on other Grid sites.

- Sites very heterogeneous. Picking 5 sites with ~#jobs/user.

ALICE

Jobs per user in selected sites - `aliprod` Single Core

efficiency	total_jobs	site	numcores
0.94	**22262**	**ALICE::ORNL::ORNL**	**1**
0.97	23050	ALICE::NIPNE::ARC	1
0.92	21461	ALICE::NIHAM::PBS	1
0.96	26597	ALICE::KFKI::LCG	1
0.92	20600	ALICE::Birmingham::LCG	1
0.94	19225	ALICE::Clermont::ARC	1
0.95	**64258**	**ALICE::LBL_HPCS::HPCS_Lr**	**1**

ALICE

Jobs per user in selected sites - `aliprod` Eight-Core

efficiency	total_jobs	site	numcores
0.82	**2447**	**ALICE::LBL_HPCS::HPCS_Lr**	**8**
1.30	1489	ALICE::NIPNE::ARC	8
1.38	974	ALICE::Vienna::LCG	8
1.11	2237	ALICE::CCIN2P3::HTC	8
1.26	12592	ALICE::CERN::CERN-CORONA	8
1.25	7624	ALICE::CERN::CERN-MIRAGE	8
1.21	8398	ALICE::FZK::HTC	8

Jobs per user in selected sites - `alihyperloop`

efficiency	total_jobs	site	numcores
0.54	**1552**	**ALICE::ORNL::ORNL**	**2**
0.44	5243	ALICE::CCIN2P3::HTC	2
0.43	5084	ALICE::CCIN2P3::HTC_2	2
0.27	1551	ALICE::GSI::GSI_8Core	2
0.57	5343	ALICE::KFKI::Wigner_KFKI_AF_8core	2

ALICE

Jobs per user in selected sites - `alitrain`

I/O bound jobs

efficiency	total_jobs	site	numcores
0.52	**200959**	**ALICE::ORNL::ORNL**	**1**
0.51	223209	ALICE::JINR::ARC	1
0.42	213585	ALICE::CCIN2P3::HTC	1
0.39	297586	ALICE::FZK::HTC	1
0.43	216774	ALICE::Prague::LCG	1
0.47	283760	ALICE::CNAF::LCG	1
0.18	**132614**	**ALICE::LBL_HPCS::HPCS_Lr**	**1**
0.32	100787	ALICE::Birmingham::LCG	1

Main observations

- CPU isolation does not have a big (negative nor positive) impact on CPU efficiency of sites
  - Job efficiencies seem to have a **coherent behaviour** with those of similar sites
- CPU isolation is **working fine in different scenarios** - running whole-node scheduling and predefined amount of cores per job slot
  - Checked that it also works ok when sites are running `cgroups` (like UPB)
- CPU isolation proofs to be **constraining jobs to the defined amount of cores**
  - Efficiencies are not surpassing the 100% limit

Whole node oversubscription opportunities at US sites

Study for Grid hosts status for oversubscription

Analysis of the **current usage levels of the Grid worker nodes** during **72 hours**

Goal - to have an accurate picture of the **feasibility of adopting oversubscription policies** for the hosts used for **whole node** submission

All the hosts reporting values to MonaLisa have been included in this survey

- To take into account that they might be running other workloads in parallel from which we do not have any knowledge → ALICE workflows are also heterogeneous

Continuous sampling of the **amount of idle cores** to evaluate the interval and number of unused cores

- Set **grace interval** for deciding **when to start** a new payload (waiting for the amount idle cores to stabilize) and for deciding **when to preempt** a payload (when machine becomes saturated)

Computation of a slot

We define a **slot** as a period of time for which a job will be running without the CPU usage levels going above the set threshold for more than 15 contiguous minutes

Grace interval needs to be awaited for before starting to account for a new idle interval

- This time will not be accounted for in the idle slot. As we set this grace interval to **1 hour,** this idle time **will always be lost.**

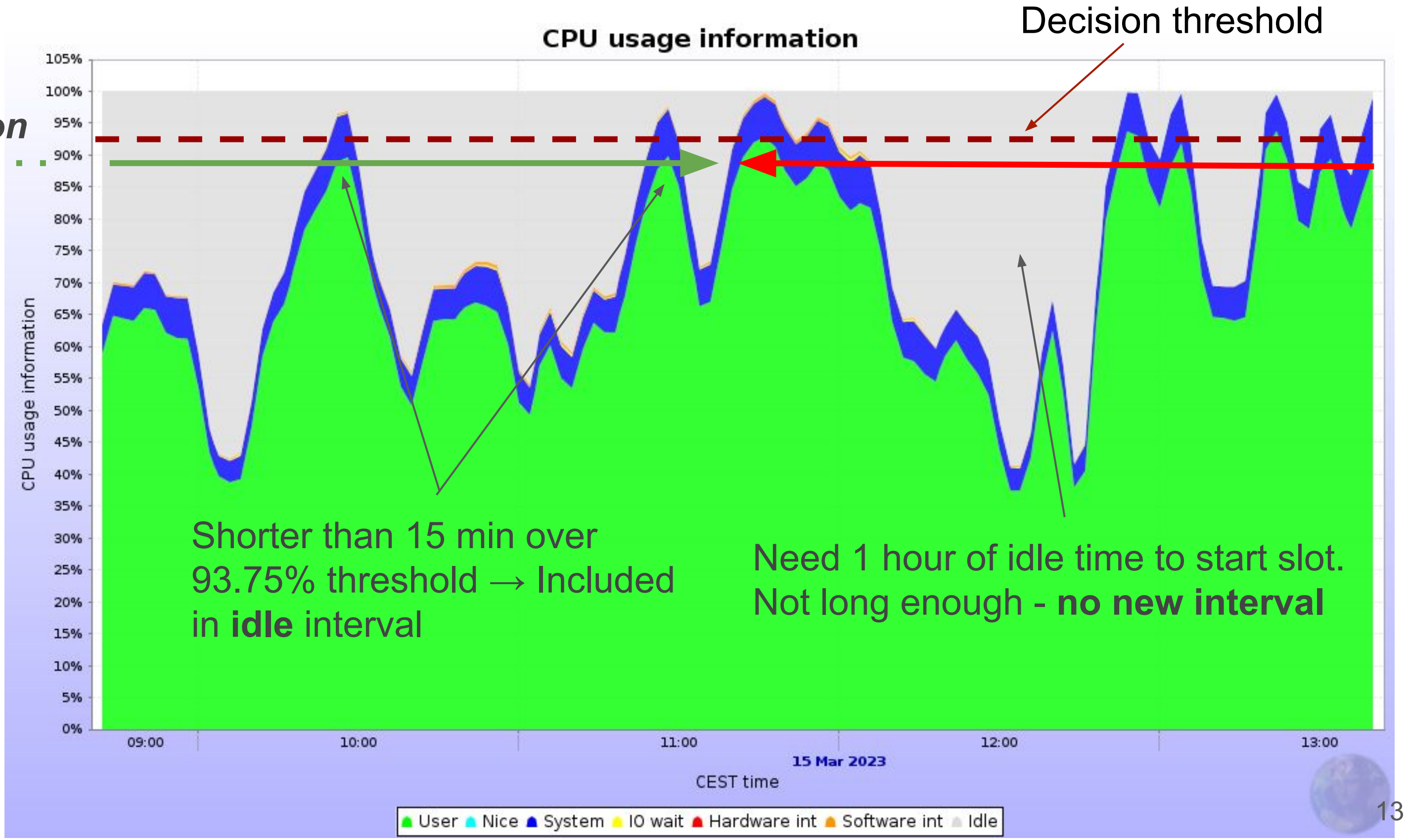
All the extra slots are filled with **MonteCarlo Simulation** payloads (CPU intensive)

ALICE

Computation of a slot

1 hour in idle has passed
*We **start slot computation***

Considering machine with 32 cores.
Set threshold on 1 core:
30/32 = 93.75%



Results from the analysis at US sites - eight-core slots

If we take **slots of 8 hours**:

Eight-core slots that would be started but **preempted** before completing

- **184 slots (in 72h)**

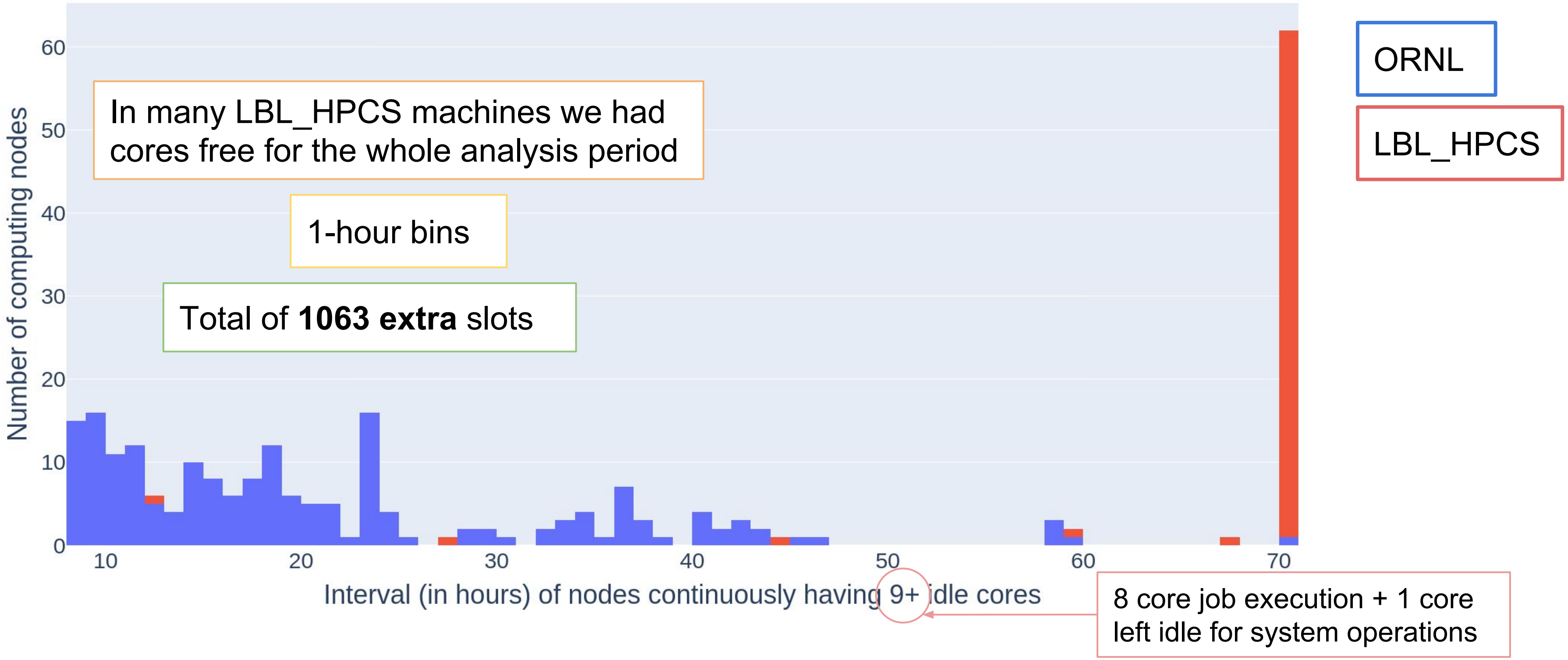
Additional usable **CPU hours** from the idle periods

- **8510 hours (in 72h)**

If we convert it to 8h extra full slots

- **1063 extra eight-core 8-hour slots (in 72h)**

Distribution of idle length (eight-core) above 8 hours in US sites



Distribution of idle length (eight-core) above 8 hours in US sites

Sum of samples per contiguous CPU-idle interval length:

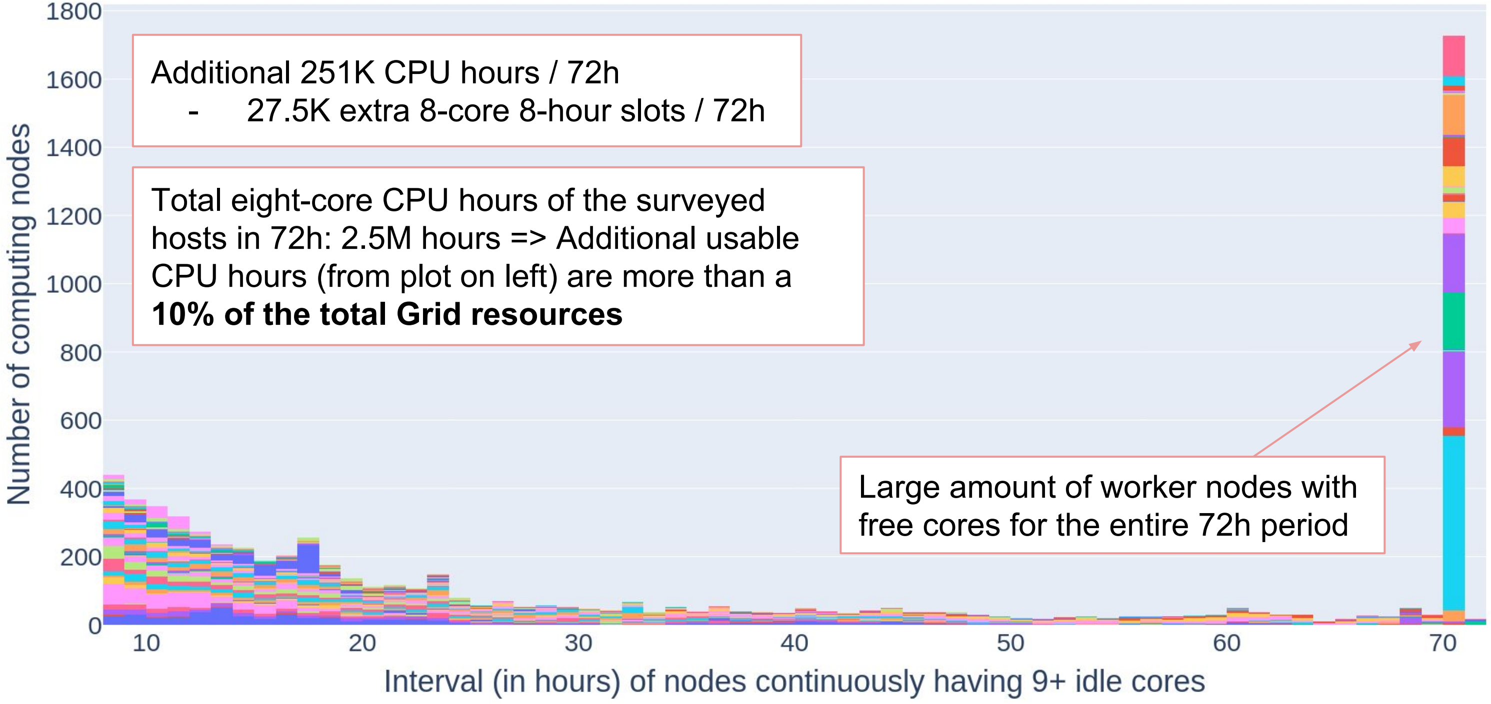
Interval length (hours)	Sum of intervals
8-10	31
10-20	83
20-30	37
30-40	21

Interval length (hours)	Sum of intervals
40-50	14
50-60	5
60-70	1
70-72	62

Total eight-core CPU hours of the surveyed hosts in 72h: 72K hours => Additional usable CPU hours (8.5K) are more than a **11.7% of the total US Grid resources**

Distribution of idle length (eight-core) above 8 hours in **ALL** Grid sites

ALICE

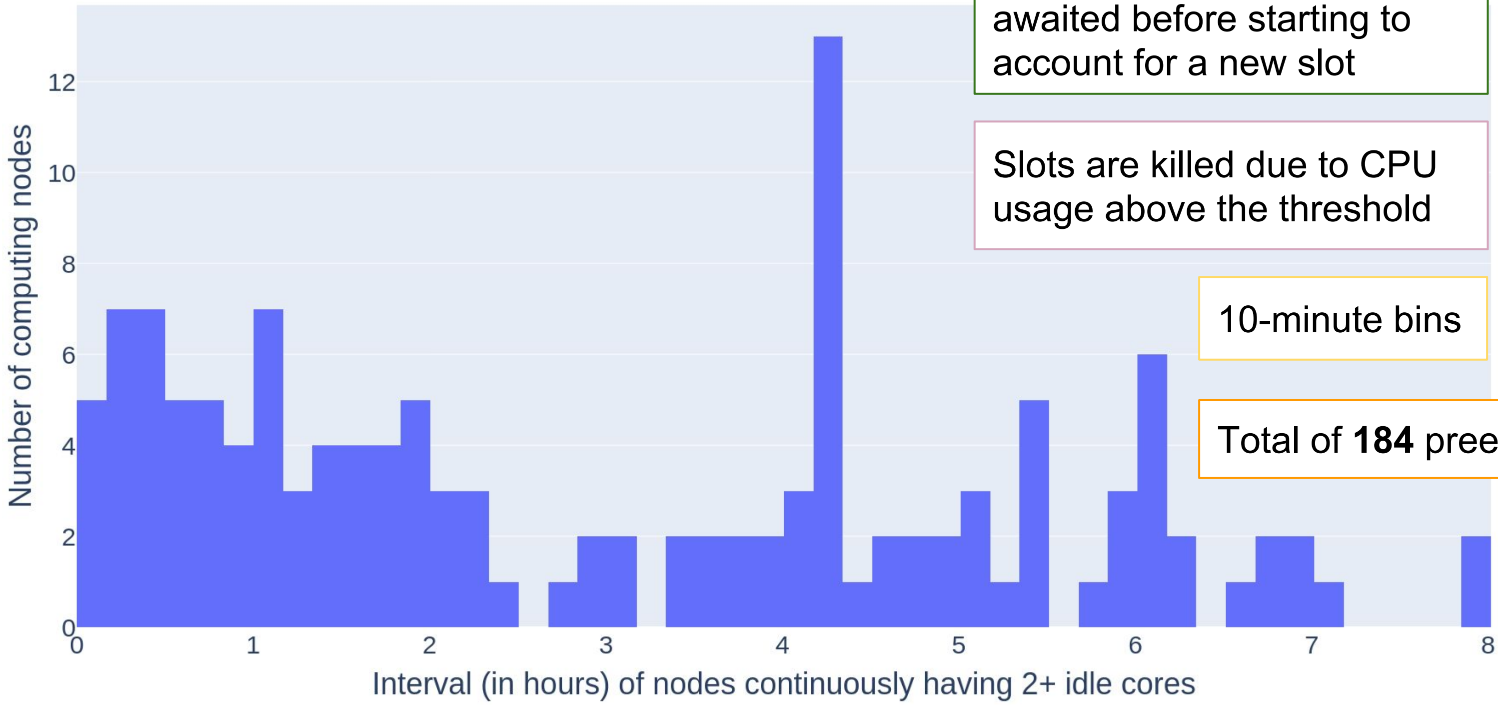


Additional 251K CPU hours / 72h
- 27.5K extra 8-core 8-hour slots / 72h

Total eight-core CPU hours of the surveyed hosts in 72h: 2.5M hours => Additional usable CPU hours (from plot on left) are more than a **10% of the total Grid resources**

Large amount of worker nodes with free cores for the entire 72h period

Caveat: Potential eight-core jobs preemption in US sites



The 1-hour grace interval is awaited before starting to account for a new slot

Slots are killed due to CPU usage above the threshold

10-minute bins

Total of **184** preempted slots

ORNL

LBL_HPCS

Caveat: Potential eight-core jobs preemption in US sites

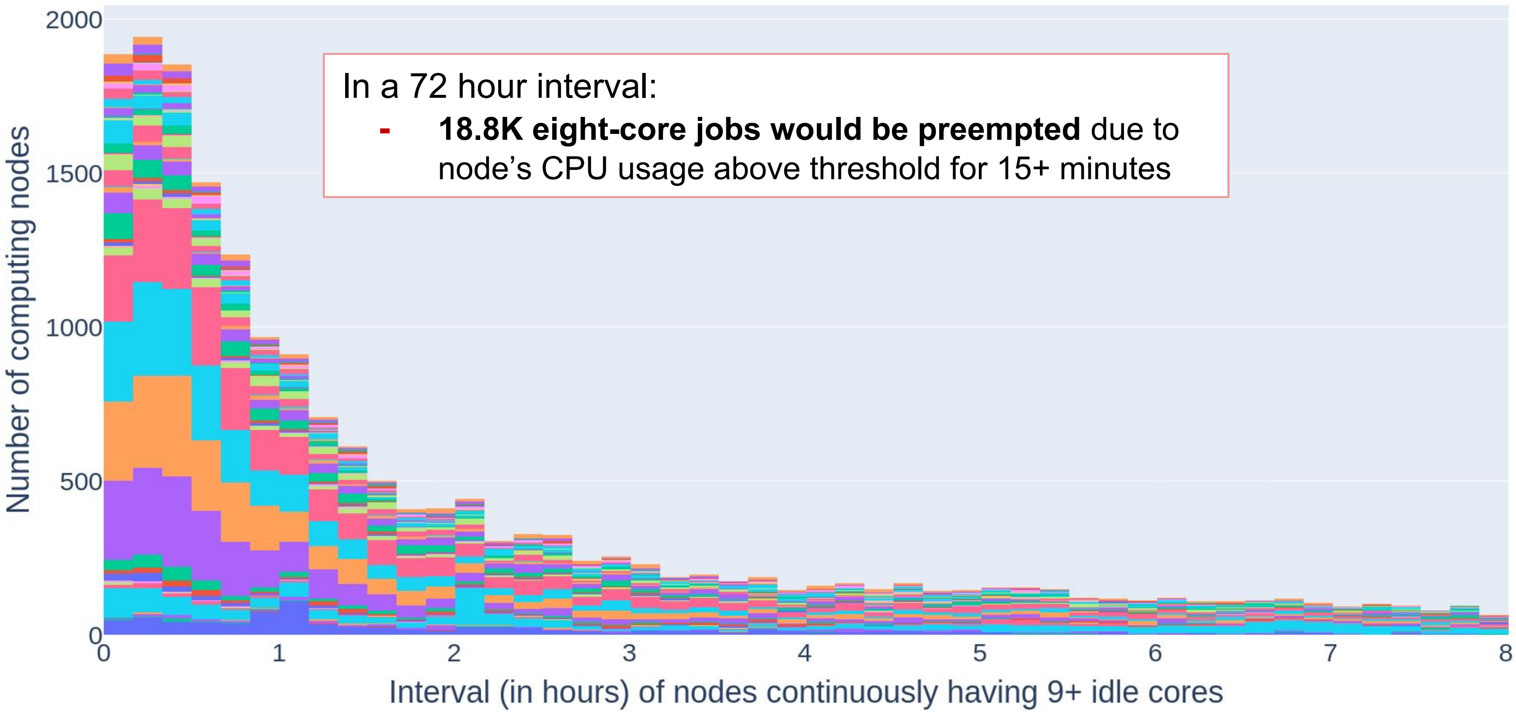
ALICE

Sum of samples per contiguous CPU-idle interval length:

Interval length (hours)	Killed slots
1	40
2	44
3	12
4	15

Interval length (hours)	Killed slots
5	36
6	18
7	10
8	9

Distribution of preempted eight-core slots duration in **ALL** Grid sites



CPU benchmarking for TTL tuning

Situation overview

- Large variation of job running time due to Grid resources heterogeneity
  - To compensate, the requested TTL for the running jobs is often set to large values
- Jobs in the queue with higher TTL are **more difficult to match** with the sites advertised resources
  - Especially when the slot does not have much time left to expiration (max 24 hours)
  - Leads to allocated resources idling for a long time, in particular if the slot is occupied by a single-core job
  - The remaining time could match a job with a properly adjusted TTL
  - Problem increases with slot size due to **slot fragmentation** (current main caveat of whole-node scheduling)
    - With TTLs close to 24h, fresh job slot is require to match
    - 8-core job slots are slower to get - we should exploit the lifetime of the already allocated
- To increase Grid resource usage efficiency, we aim to **dynamically adjust the job TTL based on executing nodes CPU power**

Methodology and profile of analyzed Grid jobs

- Parsing of traces of **all** large-scale productions executed on the Grid during 4 months
  - From 21/11/2022 to 22/03/2023
- For our study, considered jobs:
  - Final status: `DONE`
  - Users: aliprod, alidaq, alihyperloop, alitrain
    - This presentation will contain results for only **aliprod** & **alidaq**
- Considered parameters for classifying running jobs:
  - Production ID
  - Executing site
  - Worker node CPU model
  - CPU hyperthreading enabled

Methodology and profile of analyzed Grid jobs

- Summary table of the jobs that are included in this presentation

User	Production internal physics	Production IDs
`aliprod`	Pb-Pb	- *LHC22i1*
`aliprod`	p-p	- *LHC22f5b* - *LHC22f5a* - *LHC22b1b* - *LHC22b1a*
`alidaq`	asynchronous reconstruction	- *LHC22o_apass3* - *LHC22q_apass3* - *LHC22m_apass3* - *LHC22o-test_apass3* - *LHC22m_apass3*

All analyzed jobs are O2, 8-core jobs

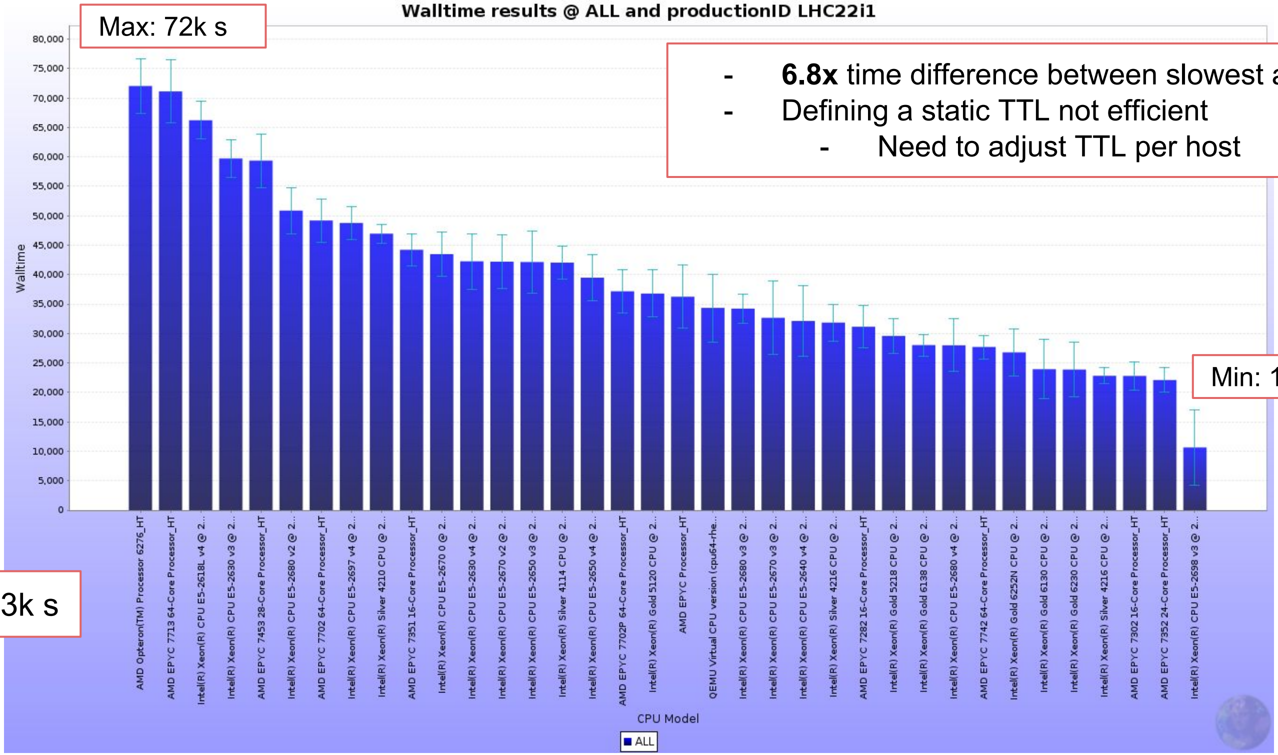
Walltime for jobs of Pb-Pb production within CPU models

ALICE

For CPU models of **ALL** Grid sites

Max: 72k s

- **6.8x** time difference between slowest and fastest
- Defining a static TTL not efficient
    - Need to adjust TTL per host

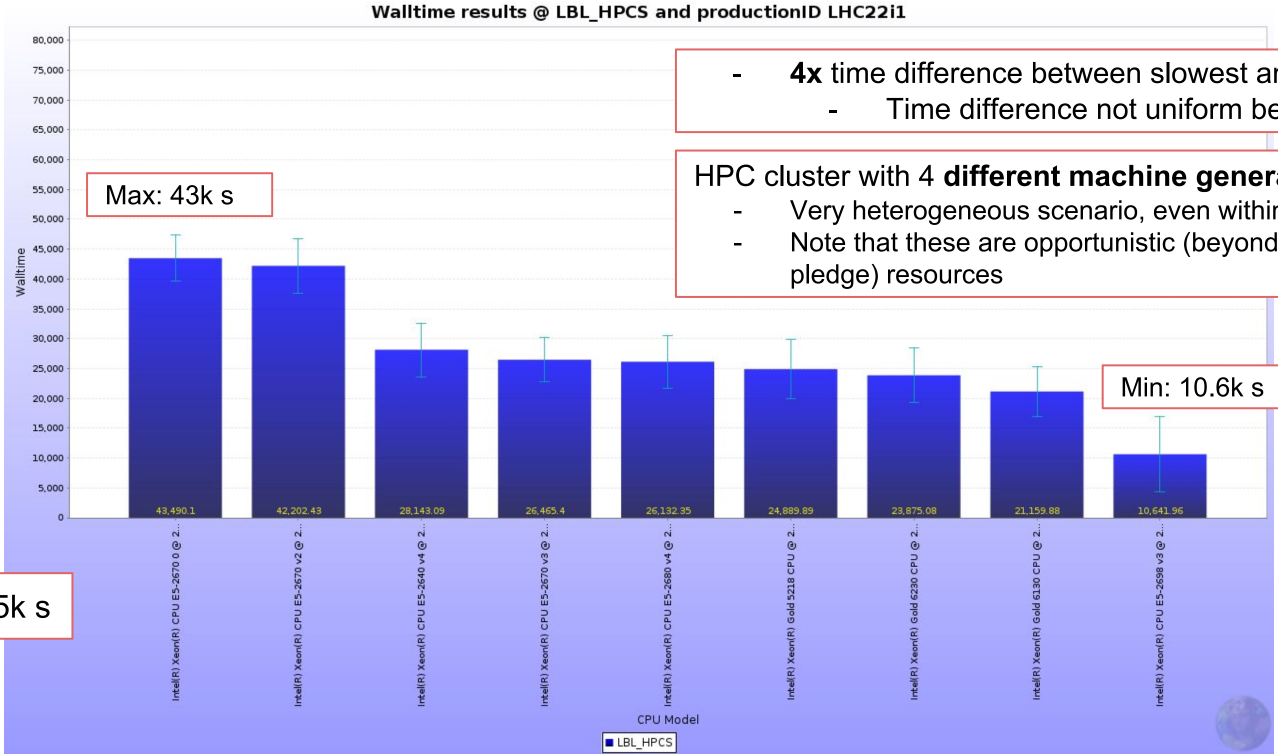


Min: 10.6k s

Avg runtime: 29.3k s

Walltime for jobs of Pb-Pb production within CPU models

For CPU models of **LBL_HPCS** site



Max: 43k s

Min: 10.6k s

Avg runtime: 27.5k s

- **4x** time difference between slowest and fastest
  - Time difference not uniform between sites

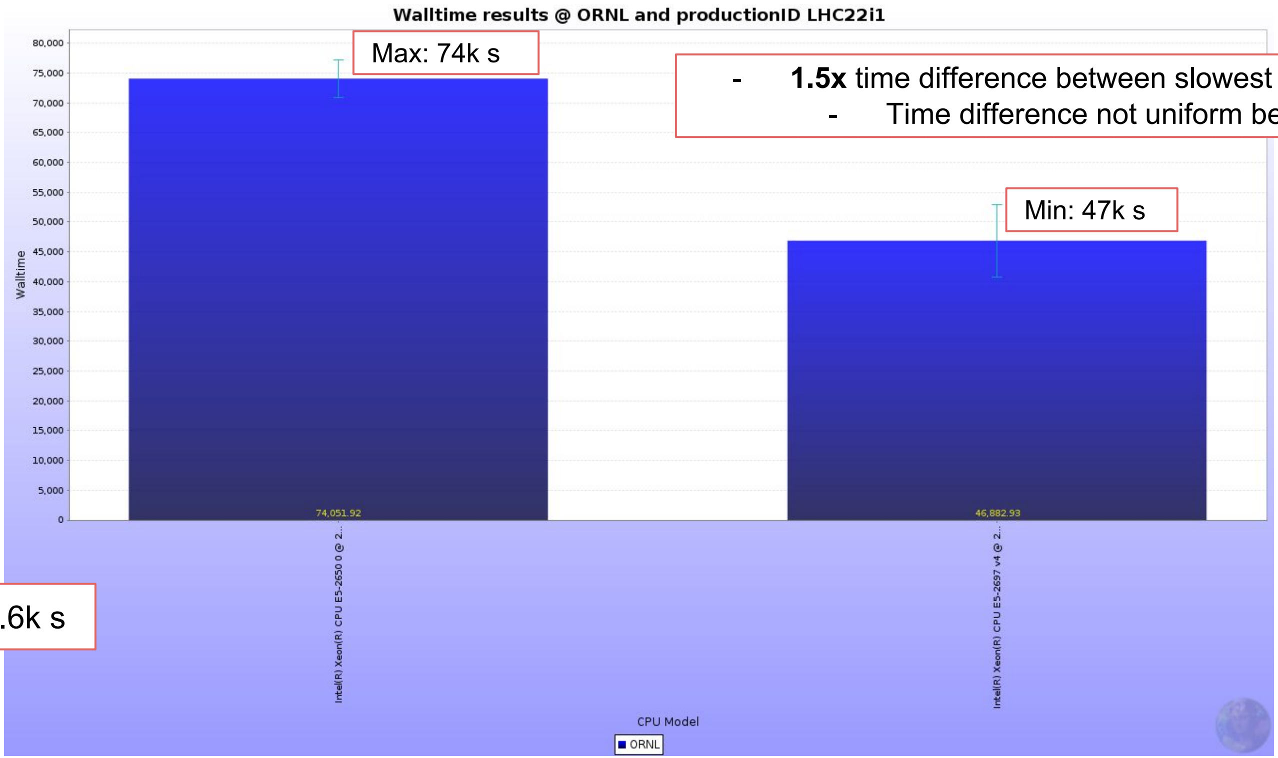
HPC cluster with 4 **different machine generations**

- Very heterogeneous scenario, even within a site
- Note that these are opportunistic (beyond pledge) resources

ALICE

Walltime for jobs of Pb-Pb production within CPU models

For CPU models of **ORNL** site



- **1.5x** time difference between slowest and fastest
  - Time difference not uniform between sites

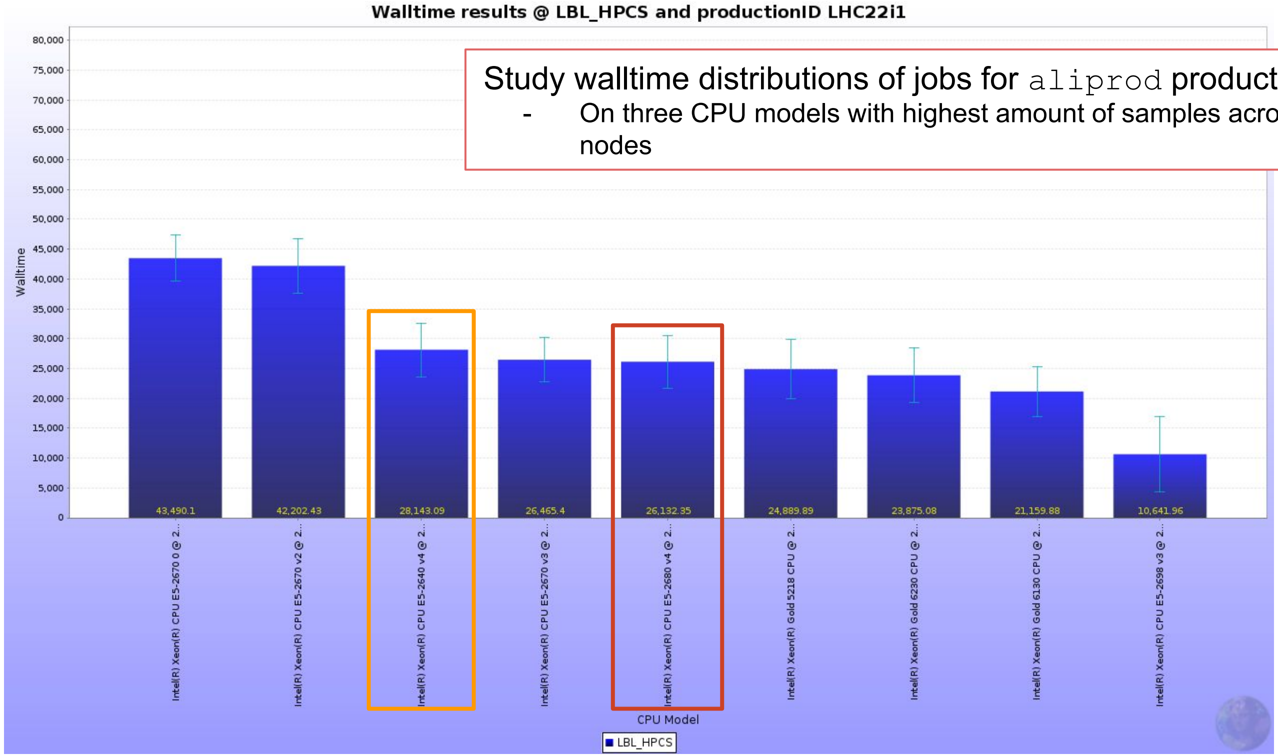
Avg runtime: 48.6k s

ALICE

Closer look to executions of particular CPU models

For CPU models of **LBL_HPCS**

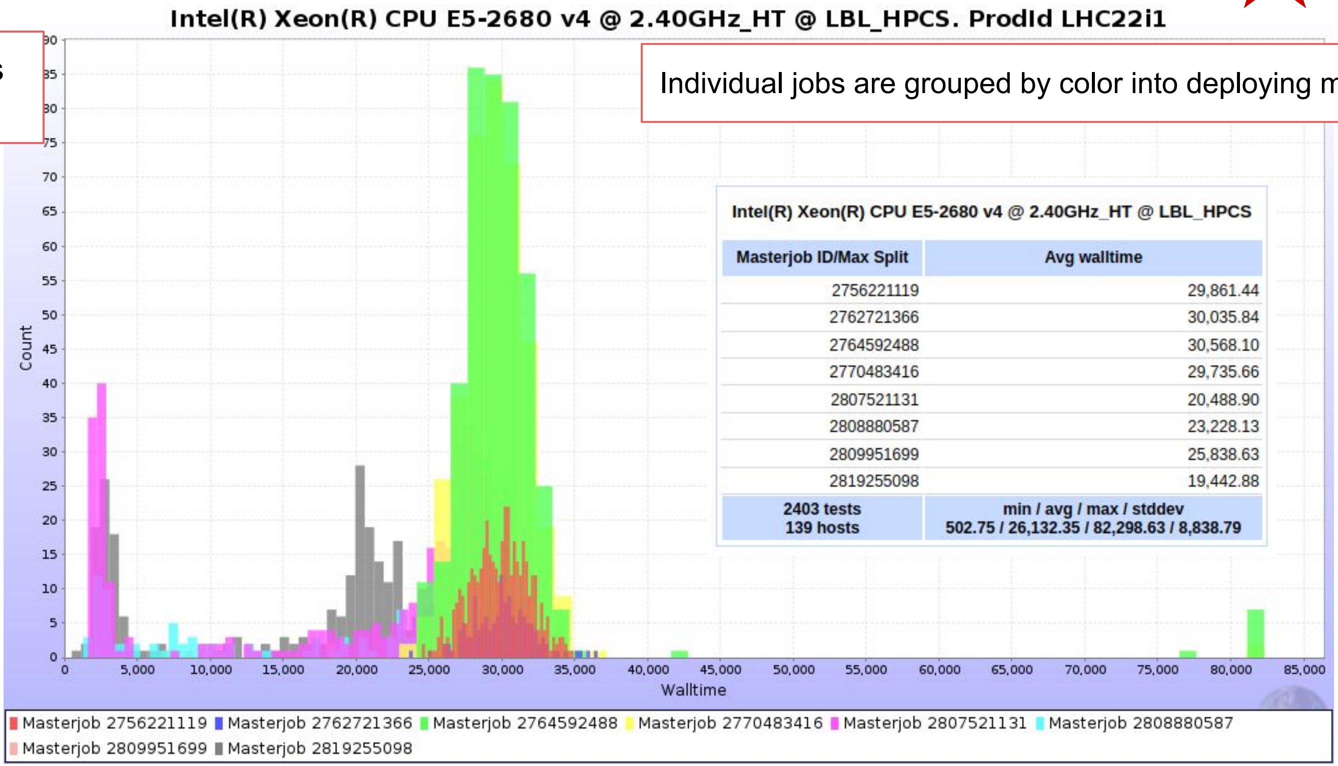
Study walltime distributions of jobs for `aliprod` production (Pb-Pb)
- On three CPU models with highest amount of samples across ALL Grid nodes



Closer look to executions of a particular CPU model

2403 job samples
From **139** hosts

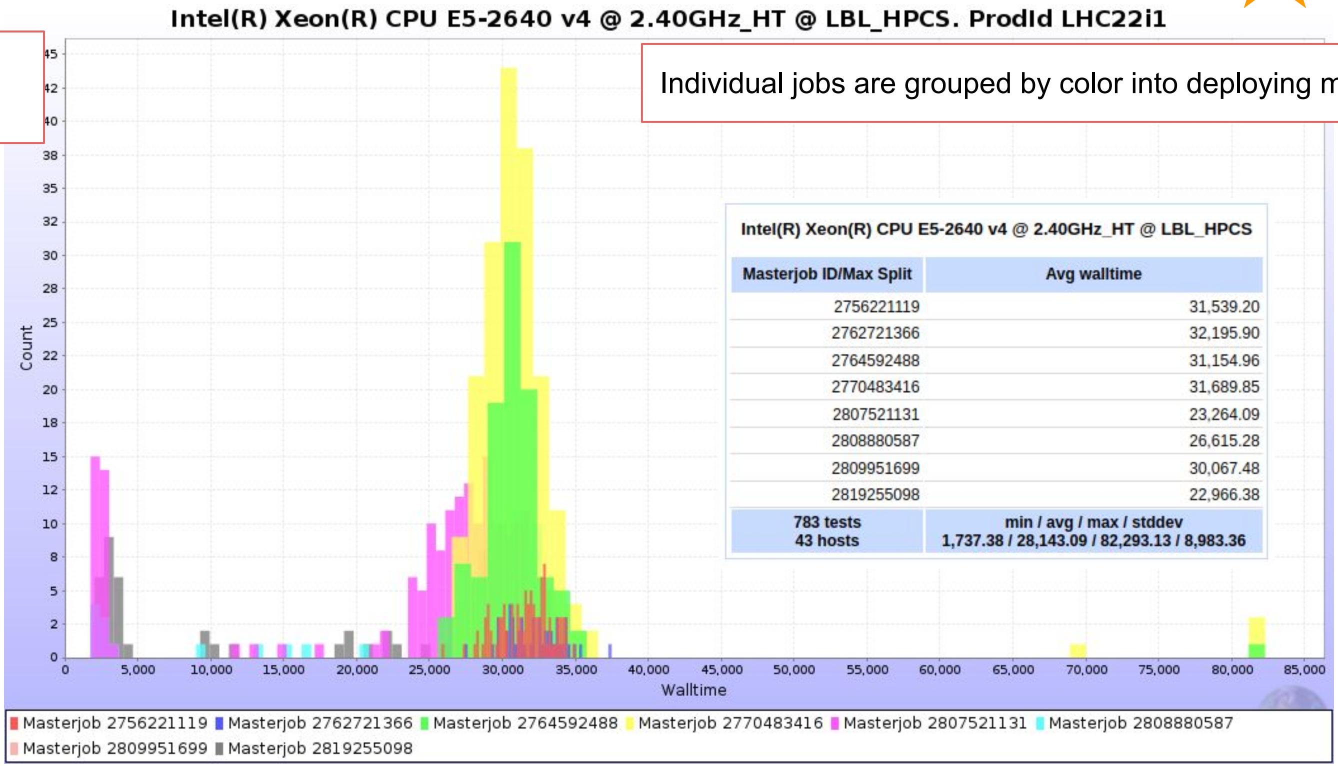
Individual jobs are grouped by color into deploying masterjob



Closer look to executions of a particular CPU model

783 job samples
From **43** hosts

Individual jobs are grouped by color into deploying masterjob



Masterjob ID/Max Split	Avg walltime
2756221119	31,539.20
2762721366	32,195.90
2764592488	31,154.96
2770483416	31,689.85
2807521131	23,264.09
2808880587	26,615.28
2809951699	30,067.48
2819255098	22,966.38
783 tests 43 hosts	min / avg / max / stddev 1,737.38 / 28,143.09 / 82,293.13 / 8,983.36

Initial TTL tuning attempts in US sites

Initial approach - Scaling factor computation

Main idea: Computing a **scaling factor per key** (site/CPU_model/hyperthreading) to adjust TTL

- We have observed considering only CPU model might not be enough

Options for computing benchmark to which to apply the scaling factor:

a) Using a **benchmark machine**
   i) Execution of *X* job samples to have a confident-enough benchmark
   ii) Might consider applying multiplicative factor with amount of events (proportional increase of TTL)
b) Using **Initial sample of Grid jobs** for that production
   i) We do have a good correlation within sites

Establish time margin to add to computed TTL

- Might decrease as we get more confident of production/executing host behaviour

Ongoing research: Predict job duration based on production history

ALICE

Main idea: Computing an **estimated TTL** based on the jobs' history.

- We observed that jobs might have a normal distribution when split based on a specific key (e.g., production/site/CPU_model/hyperthreading)

Estimate TTL based on:

- required TTL (from the JDL)
- stddev
- maximum historical value in the dataset

Formula:

- $estimatedTTL = w_1 * requiredTTL + w_2 * (stddev + n * maxTime)$
- w_1 and w_2 depends on the number of jobs (when the number of jobs increases, we have a better confidence in the prediction)