Multicore workflow characterisation methodology for payloads running on the ALICE Grid

Marta Bertran Ferrer
PhD Student - CERN

On behalf of the ALICE Collaboration

The updated Run 3 ALICE Software Stack

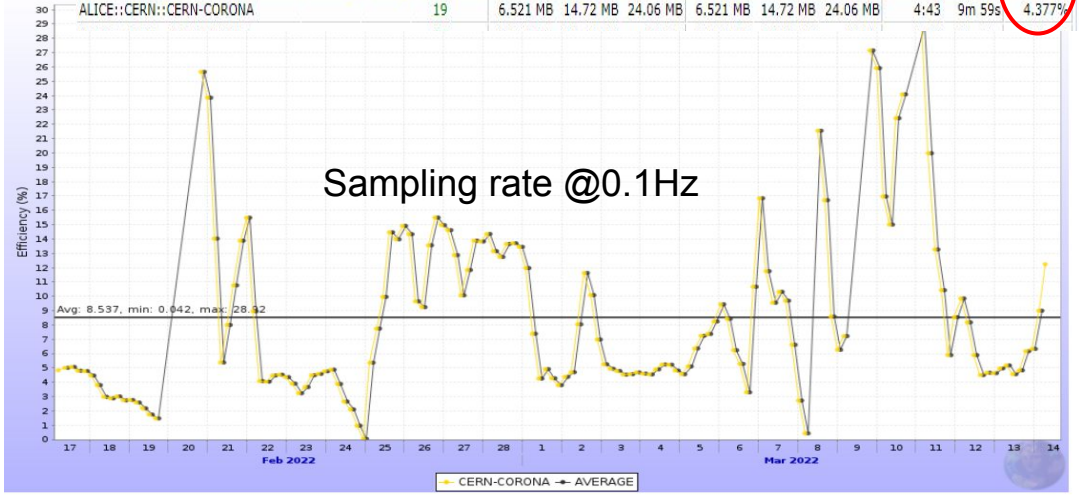
- After upgrade - 10x larger data volume with higher internal complexity
  - Grid single-core jobs with 2GB/core memory limits no longer feasible
  - **Multicore jobs** spawning multiple parallel processes using shared memory

- Run 3 multicore jobs invoke **concurrent short-lived processes at ~10Hz**
  - Grid monitoring framework needs an **update to properly account for the resource usage**
  - Previous methodology monitored single long-lived process
    - Periodic sampling of Linux *ps* output

Efficiency of O2 simulation jobs on eight-core queue

Summaries per site

Site	Number of jobs				RSS			Virtual			Average time		CPU
	Running	Saving	Done	Error	Min	Avg	Max	Min	Avg	Max	Running	Saving	Efficiency
ALICE::CERN::CERN-CORONA			19		6.521 MB	14.72 MB	24.06 MB	6.521 MB	14.72 MB	24.06 MB	4:43	9m 59s	4.377%



- Sampling active processes not sufficient to correctly monitor parameters
- Wrapping job execution with the `time` command - still incomplete picture
  - Child processes detached during execution
- In both cases CPU efficiency takes *user* and *system* components into account

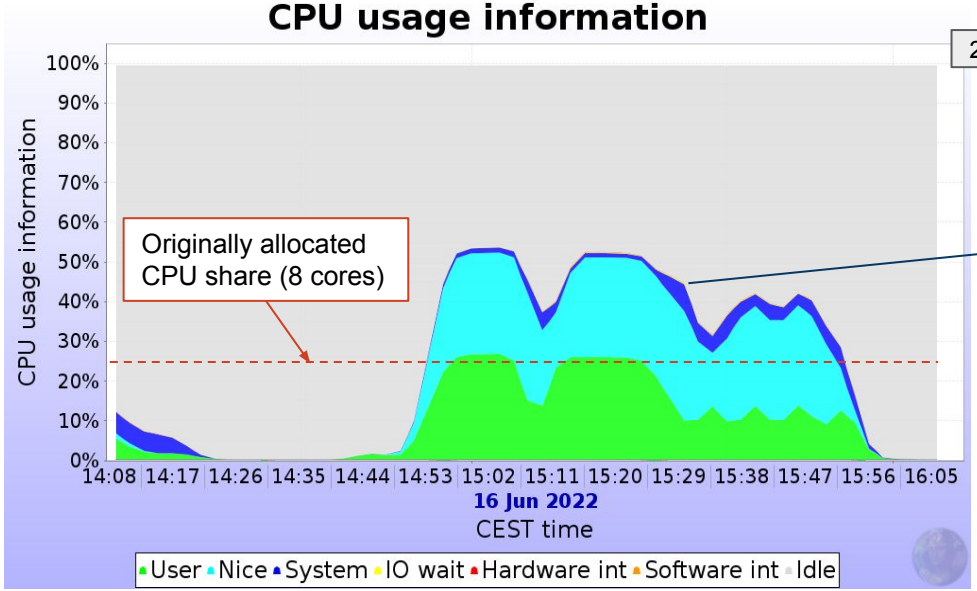
$$\textit{CPU efficiency} = \frac{\textit{User time + System time}}{\textit{Wall time}}$$

New efficiency computation and reporting

- For every job monitor iteration → listing of all the running processes and their children
- For every child - Inspection of the `/proc/PID/stat` file
  - Parsing and summation of increases on field 13 (`utime`) and 14 (`stime`)

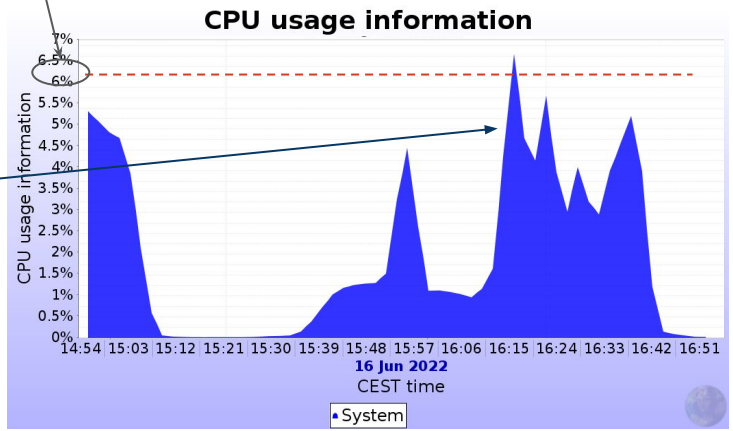
$$\textit{CPU efficiency} = \frac{\textit{Sum (User time + System time)}}{\textit{Elapsed time * numCPUs}}$$

Resource usage of the payloads in a 32-core idle machine



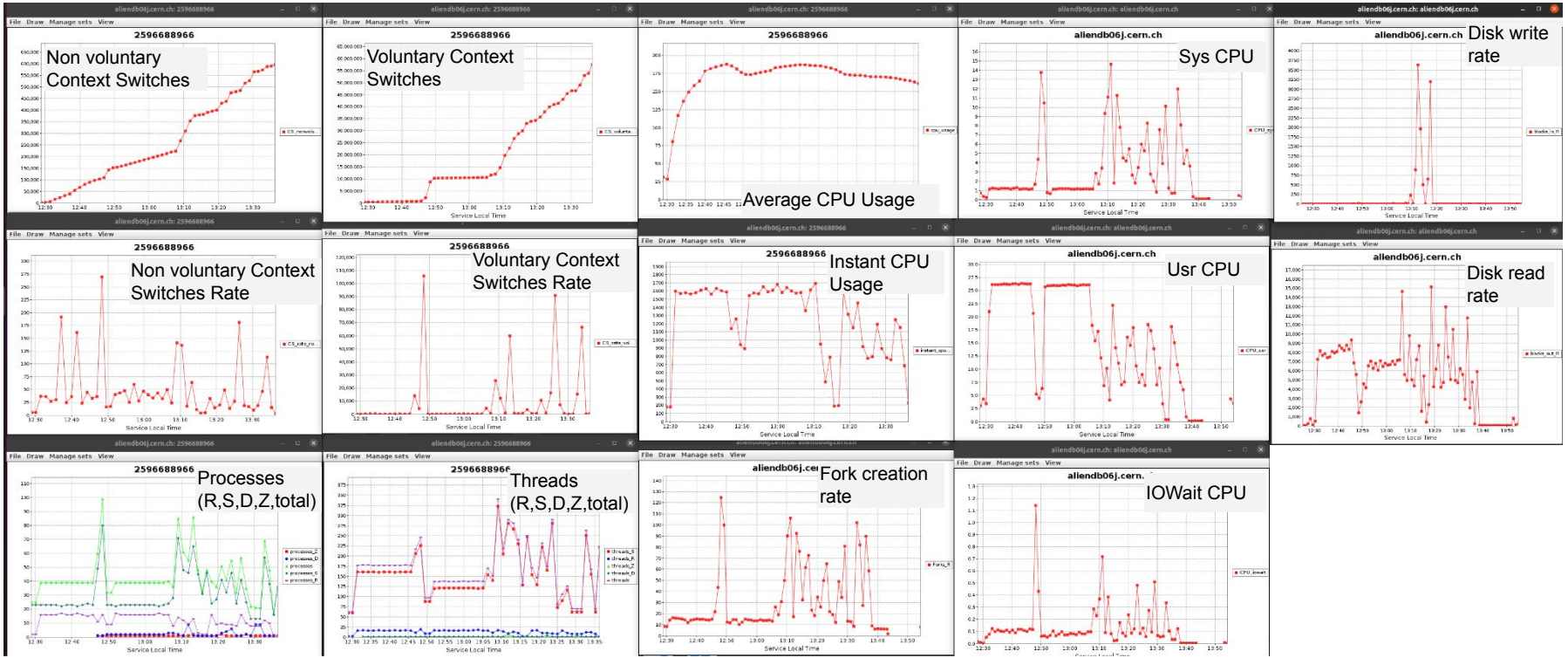
2 CPU cores

Elevated *System* CPU component

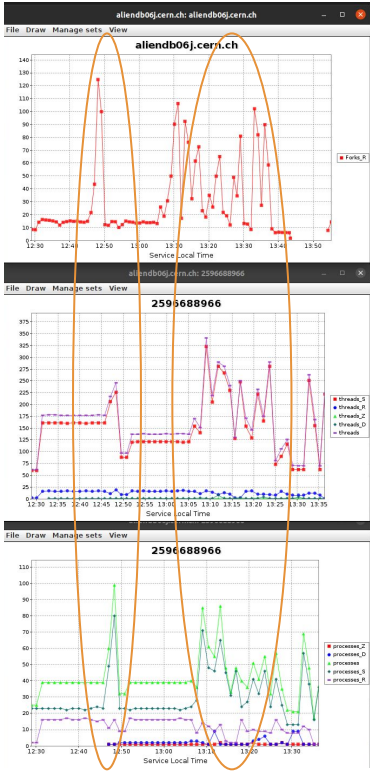


CPU efficiency →**160.25%**

Analysed parameters of the system



Deeper look into fork deployment rate



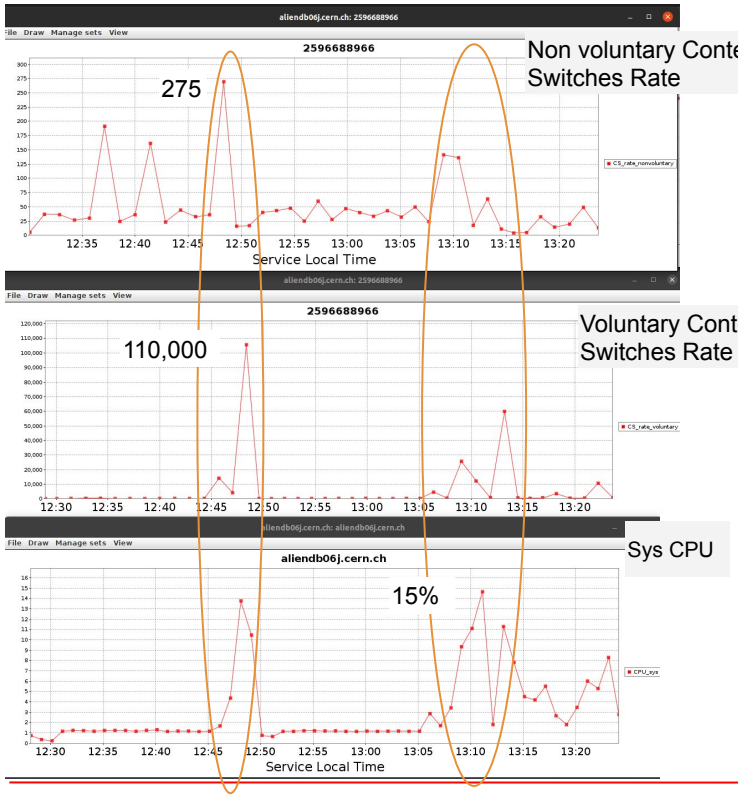
Fork creation rate

Threads (R,S,D,total)

Processes (R,S,D,Z,total)

- Fork creation rate correlated with peak in number of concurrently running processes and threads
- This behaviour is associated with the deployment of **short-lived processes**
  - Most of the time in sleeping state
- Detected large **overhead** in process creation
  - Deeper analysis on underlying causes - detailed monitoring to search for areas to be optimized

Deeper look into context switching



- The *system* CPU is greatly impacted by the context switching rate
- Specifically, peaks on ***voluntary*** context switching are clearly correlated to peaks on *system* CPU

Process deployment analysis

Observed high overhead / **large *System* CPU usage**

Deep analysis of **job internal behaviour**

- Origin of system calls might not be observable at first glance

Job execution wrapped with **`strace`** command:

```
strace -e trace=process -ttt -f -s 10000 -o jobId-execution.strace
```

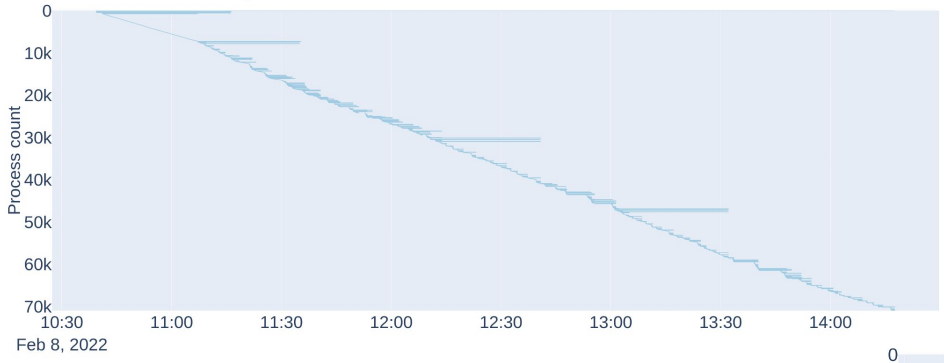
- Detailed study of the deployed processes and threads, their amount, execution frequency, time distribution and resource usage

Process deployment analysis

Observations from the reported metrics revealed some **potential areas for improvement**

- High cost of system calls execve in the *alienv* context
  - Improved by correctly loading the dependent libraries
- System calls accounting and callee identification
  - O2 process merging and decrease processes initialisations

Profiled execution analysis and improvements

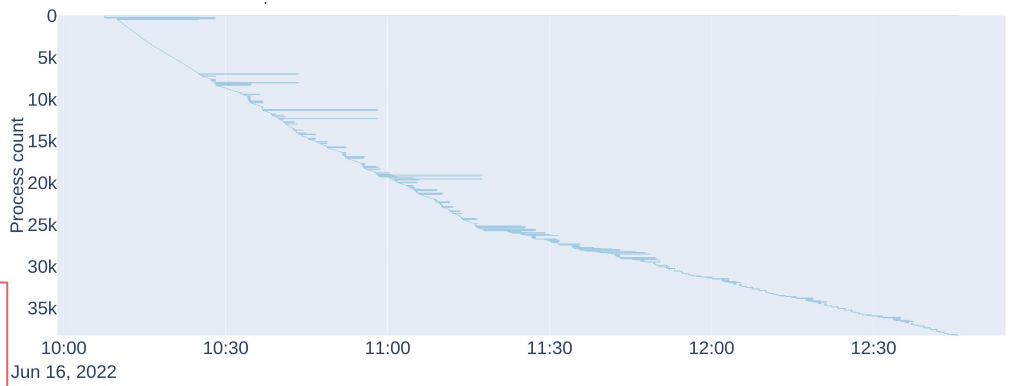


Process durations Gantt plot

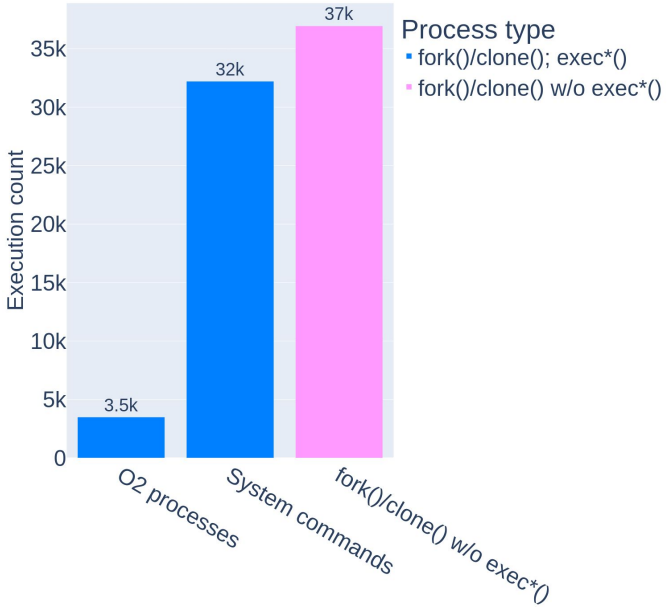
Originally:
Total process+thread count - **72.5K**

After improvements:
Total process+thread count - **38.3K**

Count decreased by **47%**

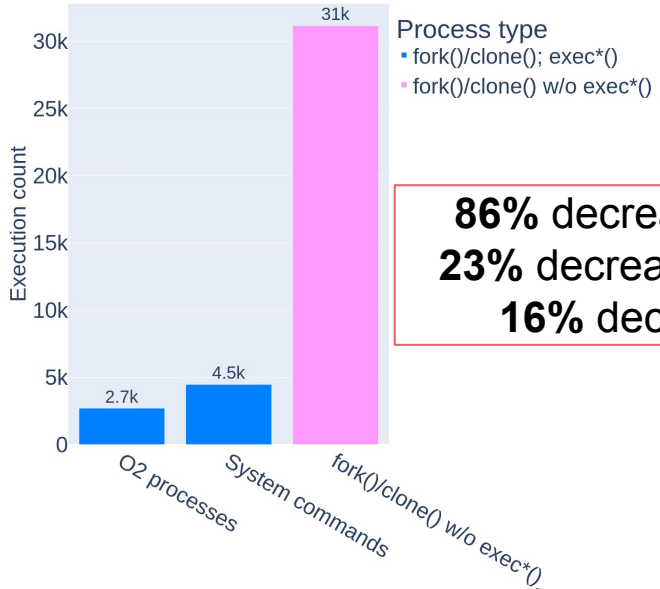


Profiled execution analysis and improvements



32k system calls
3.5k O2 processes
37k threads

72.5k total

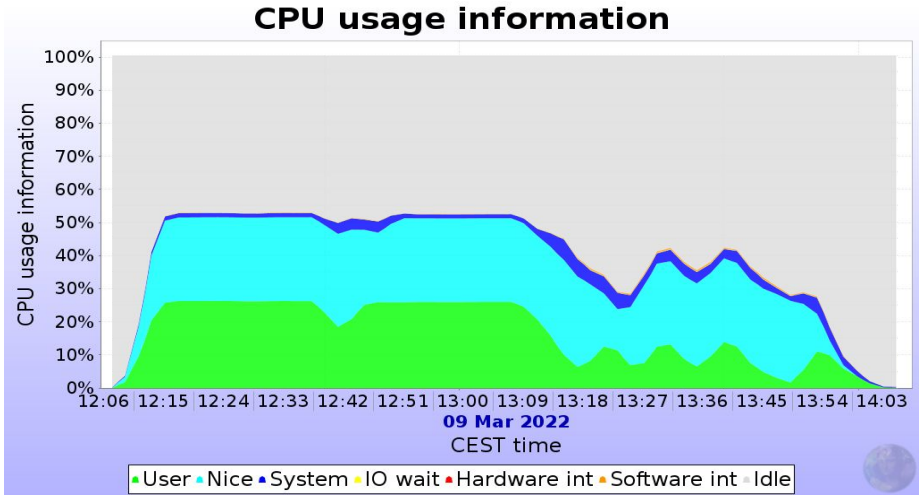


4.5k system calls
2.7k O2 processes
31k threads

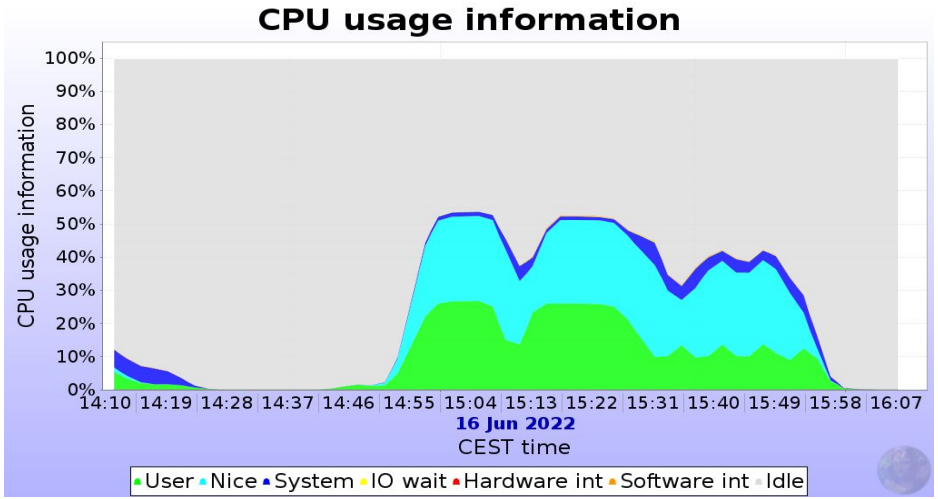
38.2k total

86% decrease on system calls
23% decrease on O2 processes
16% decrease on threads

Profiled execution analysis and improvements



1:46h



1:09h

Execution time decreased by **~35%**

Outlook

- Implemented **improved accounting of the used resources**
- The new efficiency accounting led to **real-time detailed monitoring of jobs**
  - Detailed monitoring as source for **spotting payload optimization areas**
- Execution wrapped with `strace` for process deployment and execution analysis
- Enhancements introduced in framework leading to improvements in several areas
  - Understanding of code behaviour and process count
  - CPU utilization
  - Payload execution time

Outlook

- We aim to continue to study payloads and introduce further optimizations with the implemented **profiling methodology**