Adding multi-core support to the ALICE Grid Middleware

Sergiu Weisz [1] and Marta Bertran Ferrer [2]

University POLITEHNICA of Bucharest[1], CERN[2]

E-mail: `sergiu.weisz@upb.ro`[1], `marta.bertran.ferrer@cern.ch`[2]

Abstract.
The major upgrade of the ALICE experiment for the LHC Run3 poses unique challenges and opportunities for new software development. In particular, the entirely new data taking and processing software of ALICE relies on process parallelism and large amounts of shared objects in memory. Thus from a single-core single thread workload in the past, the new workloads are exclusively multithreaded. This requires a profound change in the ALICE Grid midleware job handling, from scheduling to execution, and thus the entire middleware has been rewritten during the past 3 years to support the new multithreaded reality.

This paper presents the ALICE middlewre development for multi-core job management and the tools used to achieve an efficient and secure environment. In particular, it covers job isolation and scheduling and how they can be implemented in different site configurations, such as sites shared with other experiments or High Performance Computing resources.

1 Introduction

The ALICE Grid pools together the computational, storage and network resources of multitude of distributed computing centres around the world. This allows for the massive amounts of collected data to be processed and analyzed by the physicits in relatively short amount of time and assures the necessary growth of the combined resources as more data is collected by the experiment in the ongoing LHC operation.

For LHC Run3 the ALICE experiment has upgraded its detector and changed the data acquisition model from a triggered to a streaming mode with sharp increases in bandwidth and storage requirements. The paradigm shift from processing events to looking at 10ms long continuous data frames required a complete rewrite of the experiment software, from simulation and reconstruction to the analysis framework. For an efficient processing of the new data type the framework requires larger physical memory allocations per job, in the order of 10 to 20 GB. Swapping them out is not an option as the entire data file content is accessed and thus the CPU efficiency would be dramatically impacted.

The new experiment software is multi-threaded and allows for efficient use of multiple core slots on the sites while maintaining the existing 2 GB per core ratio demand from the resource providers. This paper addresses the modifications that have been made to the experiment Grid middleware JAliEn for brokering and running multi-core jobs.

Another requirement of the Grid is for an overall increase in resource allocations from all sites. This paper will also present the modifications that have been made to JAliEn to enable it running efficiently on supercomputers, overcoming issues that other experiments have faced in their HPC resource deployments.

2 State of the Art

All the LHC experiments (ATLAS [1], CMS [2], LHCb[3]) have implemented support for running multi-core jobs on their Grid. Their focus was on running parallel software that performs memory sharing in order to reduce the memory cost of running on the Grid, and to increase the amount of computation that can be done in parallel.

The integration of HPC resources has already happened on all the other LHC experiments. Only ATLAS [4] and CMS [5] support running their software on supercomputers that have limited external connectivity. In the case of ATLAS, this is accomplished by using MPI to synchronize jobs with the outside through a shared file system, and in the case of CMS by having pre-placed data on the shared file system once the jobs start running. LHCb [3] is for the moment limited to using supercomputers for running computing heavy simulation jobs, unlike ATLAS and CMS. It can be noticed from the literature that all the supercomputer integrations for the different experiments need large amounts of ad-hoc modifications and collaboration with the local teams, as each supercomputer has its own peculiarities to which the software must be adapted. Another important aspect of the HPC implementations is the fact that HPC resources have limited connection to the outside, which means that data intensive payloads need to stage data locally, on the super computer, in order to access it, as opposed to regular workloads, which stream data from a storage element that might be inside the site or outside.

3 The JAliEn software architecture

The Java ALICE environment (JAliEn) is the Grid middleware used by the ALICE experiment to store, process and analyse data obtained from the physical phenomena [6]. It is an evolution of the legacy AliEn [7] middleware and has been developed taking into account the evolving needs of the upcoming LHC Run 3. The Worldwide LHC Computing Grid (WLCG) infrastructure [8], which is composed of more than 200 computing centres in 39 countries around the world, is used for its operation.

The JAliEn framework has two types of services, some of which run on each of the Grid sites and others which are run centrally on servers hosted at CERN. Figure 1 shows how the main components of the framework are organised and how they relate to each other.

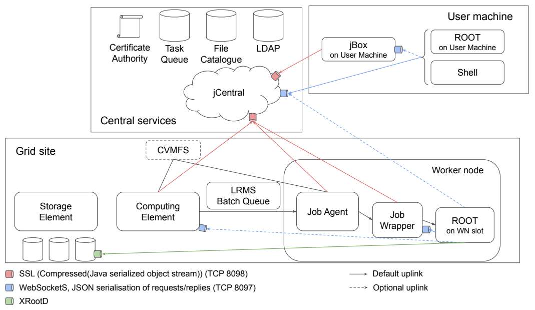

Figure 1: Diagram of the main components integrating the framework

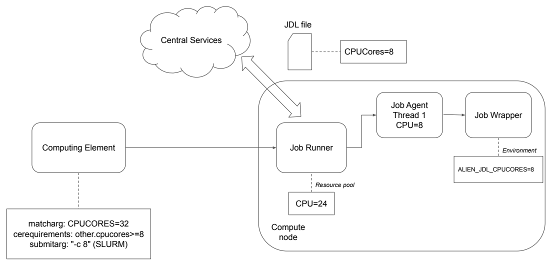

Figure 2: Diagram of the CPU Cores parameter definition and mechanism used for the execution of multi-core jobs.

3.1 Central Services

The framework has a centralised architecture, which means that job executed at different locations of the WLCG connect to the Central Services (CS) at CERN. The centralized management point dispatches payloads to run on sites, manages the input and output files required by the jobs, catalogues them and steers the clients to the appropriate Grid storage nodes to write their output to. Having central data and task management entities provides the system with load balancing capabilities between its multiple sites.

3.2 Grid site
The ALICE Grid is currently composed of 53 sites where jobs that study the physical phenomena are executed. Job scheduling takes into account the data locality (as per the central file catalogue) in order to minimize the IO latency and thus the resources consumed by the job.

3.2.1. Computing Element
The Computing Element (CE) is the gateway to each of the sites on the Grid. It runs on a persistent point of presence on the site called VOBox (Virtual Organization Box) and from there it submits generic jobs to the site Batch Queue via a Local Resource Management System (LRMS), upon instructions from the Central Services. The CE also performs monitoring and resource accounting functions.

3.2.2. Worker Nodes
The worker nodes are the machines where the jobs are executed, being configured with an environment containing all the needed tools. This environment can be set up either directly on the machine or in a container.

3.2.2.1 Job Agent and Job Wrapper

The generic jobs started by the LRMS launch Job Agent (JA) instances on the worker nodes. They connect to the Central Services and advertise their available resources, receiving an actual job that matches the constraints. The JA then forks a Job Wrapper (JW) that executes the actual payload inside a sandboxed directory or a container where available. The JA supervises the correct functioning of the JW and interacts with the CS to guarantee continuous monitoring and supervision of the resources used.

4 Implementing support for multi-core jobs
In the previously described framework architecture, the JA entity, in charge of making the requests for jobs to the CS, had no control over the resources used. Its only function was to perform the requests, all of them asking for the same amount of resources. In terms of CPU allocations, all jobs used a single CPU core. The JA as it was implemented posed limitations on managing multi-core slots.

4.1 Multi-core job matching and launching
To manage the new dimension of CPU cores resources we have introduced an intermediate entity called Job Runner (JR), replacing the direct calls to start JAs via LRMS with others that start the new component. JRs keep track of the available resources (CPU cores, memory and disk space) and spawn JA threads to pick up actual jobs from the queue (and subtract the respective amount of resources from the pool). This approach also reduces the amount of Java Virtual Machine processes on the worker node and consequently the memory footprint from the LRMS point of view.

The initial amount of resources is passed as environment variables, from the global LDAP configuration via the CE and the LRMS script. This is done in sync with the LRMS submission parameters, requesting and being aware of the same amount of CPU cores. For whole-node execution the JR can also be instructed to use all cores of the machine.

The actual payload sees as environment variables the amount of resources that it was granted. Upon payload completion the used resources are released and returned to the central resource pool and the job advertising / matching cycle repeats until the job runs out of allocated time.

Figure 2 above illustrates the definition of the involved parameters and the steps followed for the execution of multi-core tasks.

5 Integration of HPC resources in the Grid

Supercomputers have some particularities to which the framework must be adapted to for their correct integration. Some of their defining traits are their heterogeneous architectures and their local storage space limitations. They are best suited for executing massively parallel workloads and their job scheduling techniques are often very rigid and unpredictable.

Workflows are allocated partitions of the machine, at the minimum an entire worker node. The workflow is expected to efficiently and optimally utilise the node resources. Whole-node scheduling is used in conjunction with mechanisms that are capable of investigating the available resources and that can themselves launch their own workers, thus being able to get the most out of them. The renovated scheduling architecture described in 4.1 is perfectly suited for being used on supercomputers, particularly in conjunction with the whole-node scheduling concept.

6 Resource isolation on a whole-node scheduling context

Because the new payload framework launches sub-process and coordinates them, instead of doing all the computation in a single process, we cannot be sure that a single payload would not launch as many processes as the system has free resources. We have implemented a mechanism which keeps the resource consumption within the bounds of the initial allocation.

6.1 CPU isolation

In our research scenario, we want to make sure that the CPU cores are only used by the process to which they have been assigned.

There is a variety of ways in which a job's CPU usage can be isolated such as *cgroups* versions 1 and 2, *isolcpus* and *taskset*. Each of these options handles CPU isolation in different ways, and some sites have already implemented some of them independently of the JAliEn framework. Below is a breakdown of the current status of the about 22000 different nodes that run ALICE jobs, collected by our probes attached to every job slot.

Other free procs	Our proc is free	Container platform	*cpuset* prefix	Host percentage
True	True	-	/	84.27%
False	False	-	/	5.41%
True	False	Docker	/docker	5.37%
False	False	-	/slurm	2.31%

Table 1: Results on the most four most populated resulting categories on sites characterisation.

The results of this study have enabled the classification of the hosts into different categories taking into account the CPU usage freedom of the external processes running in parallel to ours as well as of our process, the containerisation platform and the prefix of the path that has been assigned to the *cpuset*. The obtained results are shown in table 1. It can be observed that 84.27% of the hosts do not assign any specific CPU mask to control the CPU affinity of both our process and the processes that are running in parallel, they do not run in a container and they are assigned the default *cpuset* (/). Other relevant categories are, with 5.41% of the hosts, the cases where all processes are constrained to run on certain cores, and with 5.37% of the hosts, those running jobs in Docker containers.

6.2 Proposed solution
We propose to use the *taskset* command to pin each job to a set of CPU threads of the same size as the number of requested CPU cores, in order to ensure that jobs cannot overrun their allocation. We have implemented a mechanism that checks for the CPU threads that have already been pinned and, avoiding those, selects a set of cores on which to run a new job.

Although each situation has its own peculiarities, we can distinguish three main scenarios:

- Sites where no constraint is applied on the CPU cores to be used. In this case, we are free to make use of *taskset* and pin the payload processes to explicit cores.
- Sites where our processes are already constrained to run on specific cores. The task affinity that will be assigned to them may have been set either via its *cgroup* (*cpuset*) or via *taskset*. Although the configuration can be done using different tools, the CPU affinity masks of the processes can be seen using the *taskset* command.
- Sites where containers are being used. In these sites, *cpuset* or *taskset* should be configured by the system administrators because it is not possible to inspect external processes from inside the container, as they are isolated.

7 Results
The JR mechanism has enabled the execution of new workflows on new system types. In order to evaluate the updated job execution architecture, we have studied the number of jobs that have run so far using the new mechanism, how many of these jobs are multi-core jobs, and if the goal of reducing the amount of memory used per core has been accomplished.

7.1 Running multi-core jobs
We have chosen to analyse two different types of jobs, as they give us an overview of how multi-core jobs will run on the Grid when their usage increases popularity, and because they have run in a large enough number of nodes to get meaningful results.

These two types of jobs are:

- Reconstruction jobs which convert raw experiment data into AOD format. These jobs load into memory the compressed files in order to parse them, so they run on sites with a large amount of memory that can accommodate the raw files parsed;
- Organized analysis jobs, which use pipelines and start multiple processes that read events resulted from the aforementioned conversion or from other productions. These jobs access many files in parallel and do a large amount of I/O operations, so they require a fast connection between the computing nodes and the storage connected to the site.

These two types of jobs have different requirements, so they run on different Grid sites. Because of the fluid situation of the payload software, and the fact that it has not been run on a large scale, we will limit the observations to two queues: CERN-CORONA, that runs reconstruction jobs, and Wigner-KFKI-8core, that runs analysis jobs.

We noticed that while both CORONA and Wigner run 8 core jobs, these jobs do not use all the resources that the sites provide. The sites also have to run single core jobs in order to not waste CPU resources. This reveals that the Grid hasn't fully moved to the multi-core job infrastructure.

7.1.1. Memory usage per core
As expected, the amount of RAM used per job has increased from an average of 3.17 GB, to an average memory usage of 9.04 GB. That being the case, if we divide by the number of cores, we notice that the memory footprint per core has decreased to 1.13 GB. These results are in line with what we expected, meaning that we will be able to fit more computational power per

batch slot, because the resource that is lacking Grid-wide is mainly memory. By lowering the memory usage, we will be able to run fewer jobs that do more computational work.

7.2 Isolating CPU usage

For testing CPU isolation we have chosen to run a prototype of simulation jobs that haven't yet been fully deployed on the Grid. We use these jobs for testing because they are the most likely to run on more CPUs than required, as they use a forking mechanism that is prone to launching more than eight processes that could be eventually scheduled at the same time.

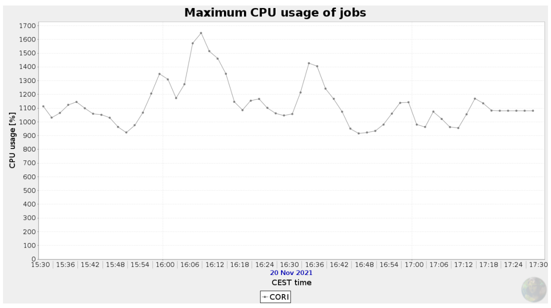

Figure 3: Per job CPU usage before isolating tasks

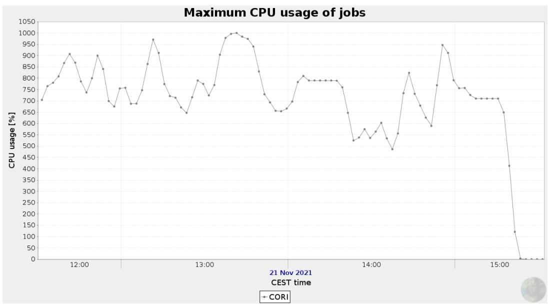

Figure 4: Per job CPU usage after isolating tasks

We notice from figure 3 that although multi-core jobs request eight cores, they actually use more than this amount of CPUs on average. There are jobs whose CPU usage goes up to 1600%, meaning that they fully use 16 cores. We expect this to be an issue that will escalate in the future, as more physicists start using the new framework.

Figure 4 represents the CPU usage of jobs after implementing the task isolation system. As expected, the CPU usage has decreased to close to 800%, equating to 8 CPUs working at 100%. In the graphic we can notice a peak of 1000%, but this is caused by the accounting method used by the monitoring software.

8 Conclusion

We implemented a mechanism capable of managing multi-core slots and even whole nodes allocated by ALICE Grid sites and using the given resources to launch single and multi-core jobs from the Grid with a user-defined granularity. This new feature allows the experiment to run the new generation of jobs, but it also allows the software to be deployed on new sites, such as supercomputers. Furthermore, this paper proposes a solution for slot fragmentation on the nodes and for CPU resources isolation.

Further research questions that arise from this paper are as follows:

- What is the impact of job containerization in regards to the job efficiency once the new framework has been broadly deployed on the Grid;
- How could Computing Element deployment automation be done on other supercomputers;

9 Bibliography

[1] D. Crooks, Paolo Calafiura, R. Harrington, Manikant Jha, T. Maeno, S. Purdie, H. Severini, S. Skipsey, V. Tsulaia, R. Walker, and A. Washbrook. Multi-core job submission and grid resource scheduling for atlas athenamp. *Journal of Physics Conference Series*, 396:2115–, 12 2012.

[2] A. Perez-Calero Yzquierdo, J. Balcas, J. Hernandez, F. Aftab Khan, J. Letts, D. Mason, and V. Verguilov. CMS Readiness for Multi-Core Workload Scheduling. *J. Phys. Conf. Ser.*, 898(5):052030, 2017.

[3] Federico Stagni, Andrea Valassi, and Vladimir Romanovskiy. Integrating lhcb workflows on hpc resources: status and strategies. *EPJ Web of Conferences*, 245:09002, 2020.

[4] Benjamin, Douglas, Childers, Taylor, Lesny, David, Oleynik, Danila, Panitkin, Sergey, Tsulaia, Vakho, Yang, Wei, and Zhao, Xin. Building and using containers at hpc centres for the atlas experiment. *EPJ Web Conf.*, 214:07005, 2019.

[5] Pérez-Calero Yzquierdo, Antonio. Cms strategy for hpc resource exploitation. *EPJ Web Conf.*, 245:09012, 2020.

[6] M. Martinez Pedreira, C. Grigoras, and V. Yurchenko. JAliEn: the new ALICE high-performance and high-scalability Grid framework. *EPJ Web Conf.*, 214:03037, 2019.

[7] P. Saiz, L. Aphecetche, P. Bunčić, R. Piskač, J.-E. Revsbech, and V. Šego. Alien—alice environment on the grid. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 502(2):437–440, 2003. Proceedings of the VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research.

[8] Jamie Shiers. The worldwide lhc computing grid (worldwide lcg). *Computer Physics Communications*, 177(1):219–223, 2007. Proceedings of the Conference on Computational Physics 2006.