Improving Fairness in a Large Scale HTC System Through Workload Analysis and Simulation

Frédéric Azevedo[1], Dalibor Klusáček[2], and Frédéric Suter[1]

[1] IN2P3 Computing Center / CNRS, Lyon-Villeurbanne, France
firstname.lastname@cc.in2p3.fr
[2] CESNET a.l.e., Prague, Czech Republic
klusacek@cesnet.cz

Abstract. Monitoring and analyzing the execution of a workload is at the core of the operation of data centers. It allows operators to verify that the operational objectives are satisfied or detect and react to any unexpected and unwanted behavior. However, the scale and complexity of large workloads composed of millions of jobs executed each month on several thousands of cores, often limit the depth of such an analysis. This may lead to overlook some phenomena that, while not harmful at a global scale, can be detrimental to a specific class of users.
In this paper, we illustrate such a situation by analyzing a large High Throughput Computing (HTC) workload trace coming from one of the largest academic computing centers in France. The *Fair-Share* algorithm at the core of the batch scheduler ensures that all user groups are fairly provided with an amount of computing resources commensurate to their expressed needs. However, a deeper analysis of the produced schedule, especially of the job waiting times, shows a certain degree of unfairness between user groups. We identify the configuration of the quotas and scheduling queues as the main root causes of this unfairness. We thus propose a drastic reconfiguration of the system that aims at being more suited to the characteristics of the workload and at better balancing the waiting time among user groups. We evaluate the impact of this reconfiguration through detailed simulations. The obtained results show that it still satisfies the main operational objectives while significantly improving the quality of service experienced by formerly unfavored users.

1 Introduction

The analysis of workload traces is a common approach to understand and optimize the behavior of the system that manages the access to resources and the execution of jobs in a data center, i.e., the batch scheduling system. Most of the historical workload traces available in the Parallel Workload Archive (PWA) [6] were originally studied with such an objective in mind. More recently, a methodology to characterize HPC workloads and assess their heterogeneity based on the analysis of the workloads executed over a year on three systems deployed at NERSC was proposed in [12]. This study not only helps to understand the behavior of current HPC systems but can also be used to develop new scheduling

strategies for the forthcoming exascale systems. Similar studies mixing the characterization, modeling, and prediction of HPC workloads have been proposed in [5,17]. In [8], the authors proposed a complete reconfiguration of their batch scheduling systems, including the definition of the scheduling queues, based on a thorough analysis of the workload characteristics and detailed simulations. The effects of this reconfiguration have then been analyzed in [9].

The common point of all the aforementioned studies is to consider HPC workloads that are composed of parallel jobs spanning over multiple cores and multiple nodes. In this paper we study a large High Throughput Computing (HTC) workload trace coming from the Computing Center of the National Institute of Nuclear Physics and Particle Physics (CC-IN2P3) [14] which is one of the largest academic computing centers in France. One of the main characteristics of this workload trace is that it is composed of a vast majority of jobs running on only one core, i.e., Monte-Carlo simulations and analyzes made on experimental data. This trace is also much larger than HPC traces with nearly 3,000,000 jobs executed every month on 33,500 cores. Finally, this workload captures a mix of two different types of submissions. As one of the twelve Tier-1 centers engaged in the processing of the data produced by the Large Hadron Collider (LHC) at CERN, half of job submissions come from an international computing grid through a complex middleware stack. The other half is directly submitted to the batch scheduling system by users belonging to more than 70 scientific collaborations.

In a previous work, we described the process of simplification of the operation of this infrastructure by reducing the "human-in-the-loop" component in scheduling decisions [2]. The main contribution of this paper is that we conduct a thorough analysis of the workload processed at a real large scale HTC system to show that the current system configuration tends to favor the jobs coming through the grid. The fair-share algorithm implemented by the scheduler is thus not as fair as it seems. Indeed, jobs submitted by local users suffer from a significantly higher waiting time. Based on our observations, we propose and evaluate through simulation a drastic modification of the quotas and the configuration of the scheduling queues that aims at further improving the fairness of the produced schedule. The obtained results show that it still guarantees the satisfaction of the main operational objectives while significantly improving the quality of service experienced by the formerly unfavored users. To ensure the reproduction and further investigation of the presented results, and thus favor Open Science, we made our large workload trace available to the scientific community [3].

This paper is organized as follows. Section 2 describes the computing infrastructure operated at CC-IN2P3 with details about the hardware, configuration of the batch scheduling systems, and operational constraints. In Sect. 3 we present and discuss the main characteristics of the workload executed on this infrastructure. Section 4 details the proposed modification of the system configuration and evaluates its impact on the quality of service and fairness experienced by the submitted jobs. Section 5 briefly explains how we produce and make available the workload trace used in this work. Finally, we conclude this paper and detail future work directions in Sect. 6.

Table 1: Characteristics of the nodes in the CC-IN2P3's HTC computing farm.

Model	Nodes	Cores/Node	Memory/Node	Cores
Xeon E5-2650 v4 2.20GHz	232	48	144 GB	11,136
Xeon Silver 4114 2.20GHz	240	40	128 GB	9,600
Xeon E5-2680 v2 2.80GHz	149	40	128 GB	5,960
Xeon E5-2680 v3 2.50GHz	124	48	144 GB	5,952
Xeon E5-2670 0 2.60GHz	24	32	96 GB	768
Xeon Silver 4114 2.20GHz	1	40	1,512 GB	40
Total	**770**			**33,456**

2 System Description

The IN2P3 Computing Center [14] is one of the largest academic computing centers in France. At the time of writing of this article, the CC-IN2P3 provides its users with about 33,500 *virtual* cores (i.e., hyper-threading is activated) on 770 physical nodes, whose characteristics are given in Tab. 1. In addition to this HTC computing farm, the CC-IN2P3 also offers resources for parallel, GPU-based, large memory, and interactive jobs that we ignored in this study.

These resources are managed by Univa Grid Engine (UGE v8.4.4) [16] which implements the *Fair Share Scheduler* first described in [7] and thus assigns priorities to all the unscheduled jobs to determine their order of execution. These priorities directly derive from the resources pledges expressed by the different user groups. As detailed in [2], each group has to provide an estimation of its computing needs as an amount of work, expressed in *Normalized HS06.hours* [11], to be done during each quarter of the following year. Once arbitration has been done with regard to the total available computing power, the respective share that has to be allocated to each group is converted into a consumption objective used by UGE to determine a fair-share schedule. This algorithm addresses one of the two main operational objectives of the CC-IN2P3: ensure that each user group is served according to its expressed resource request for the year.

In addition to this central scheduling algorithm, resources are organized in queues whose characteristics are given in Tab. 2. These queues are listed in the order in which they are considered by the job scheduler. They mainly differ by maximum allowed duration, both in terms of wallclock and CPU times, available memory and scratch disk space per job, and the type of jobs allowed to enter the queue, i.e., sequential or multi-core (denoted by the `mc_` prefix).

The *long* queues can both access the entire infrastructure. As it will be shown in Sect. 3, these two queues have to absorb the bulk of the workload. However, jobs are not really distinguished by their execution time in this configuration, with a minimal limit on execution time set to 48 CPU hours (or 58 hours) for all jobs. The *huge* queues are intended to jobs that need more memory or disk space while the access to the *longlasting* queues is limited to certain user groups.

Table 2: Names and upper limits (per job) of the queues. Queues are listed in the order in which they are considered by the job scheduler.

Queue name	CPU Time	Time	Memory	File Size	Pool size (in cores)
mc-long	48h	58h	3.6GB	30GB	33,456 (100%)
mc-huge	72h	86h	8GB	30GB	9,040 (27%)
mc-longlasting	202h	226h	3GB	30GB	19,800 (59%)
long	48h	58h	4GB	30GB	33,456 (100%)
huge	72h	86h	10GB	110GB	10,418 (31%)
longlasting	168h	192h	4GB	30GB	3,931 (12%)

We also see that all queues combined could virtually access more than three times the actual number of available cores. This configuration aims at achieving the highest possible utilization of the resources which is the second main operational objective of the center. Indeed, this ensures that load variations in the different scheduling queues cannot lead to leaving some cores idle.

Another important operational constraint of this system is related to the access to storage subsystems. High Energy Physics is a data-driven science, hence most of the jobs rely on locally stored data. Some user groups exhibit heavy I/O patterns that may become harmful to the storage subsystems in the worst case scenario where many jobs from these groups are executed concurrently. To prevent an overload of the storage subsystem, the number of concurrent jobs can be limited to a safe number for some groups by setting *Resource Quota Sets* (RQSs). The downside of this method is that it may increase the number of waiting jobs when this limit is reached. In such a case, operators can dynamically and temporarily modify the RQSs if the storage subsystems can cope with the extra load.

3 Workload Analysis

In this section we describe the main characteristics of the workload processed in November 2018 at CC-IN2P3. Over this month, 2,669,401 jobs were executed on the 33,456 available cores (or slots in the UGE vocabulary). We distinguish two sub-workloads depending on whether jobs are submitted to the batch system by *Local* users (1,174,078 jobs) or through a *Grid* middleware (1,495,323 jobs).

Figure 1 shows how many slots are simultaneously used. The dashed line indicates the total number of available slots while solid lines respectively show the overall utilization and which part of it comes from Grid and Local jobs.

This figure show that the main operational objective of the CC-IN2P3 is achieved with a global utilization well over 90%. We also see that this utilization is dominated by Grid jobs that use 3.45 times more slots than Local jobs while there are only 1.27 times more jobs coming through the grid than submitted by local users. About 28% of Grid jobs are multi-core, while 98% of Local jobs request only one core for their execution. There were less than 17,000 multi-core

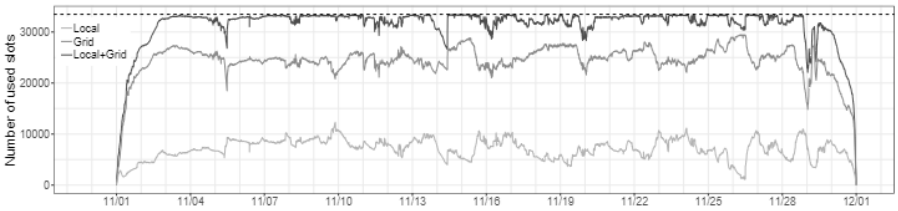

Fig. 1: Utilization of the resources, in terms of slots, over the considered period. The dashed line indicates the number of available slots, the other lines show the global utilization and the respective contributions of Local and Grid jobs.

Local jobs submitted over the considered 1-month period which required at most 16 cores. Most of multi-core Grid jobs fall in two categories. First, production jobs submitted by two LHC experiments (ATLAS and CMS) always require 8 cores. Second, two-core short-lived probe jobs are periodically submitted to check for site availability. They represent about 5% of the multi-core Grid jobs.

These two sub-workloads show several additional differences. The first one pertains to the expressed job duration. Figure 2 (left) shows a cumulative distribution function of the requested CPU time for Local and Grid jobs. There are only three values for the Grid jobs which correspond to the queue upper limits on CPU time (i.e., 48, 72, and 192 CPU hours). These predefined requirements are automatically added at the level of the Computing Element, i.e., the interface to the resources of a Grid site. The rationale is that most of the user groups that submit their jobs through the Grid manage their own workloads with pilot jobs [15]. Local jobs show a larger diversity in their CPU time requirements, even though about 35% of the jobs either require the upper limits of the queues or do not express any requirements. We also observe that a large fraction of Local jobs expresses requirements that are much lower than the queues upper limits. Nearly 40% of the jobs explicitly announce that they run for less than 12 hours but fall in the same queue as jobs potentially running for two days.

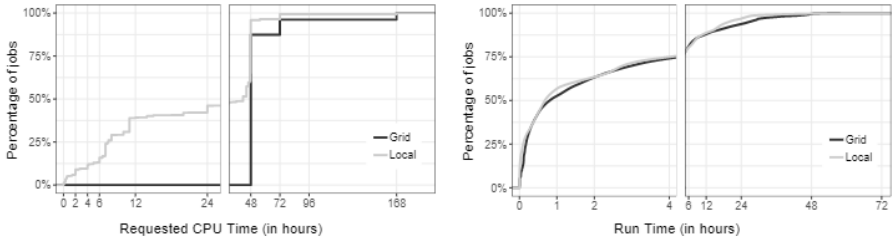

Fig. 2: Cumulative distribution functions of Local and Grid jobs requested (left) and consumed (right) run times.

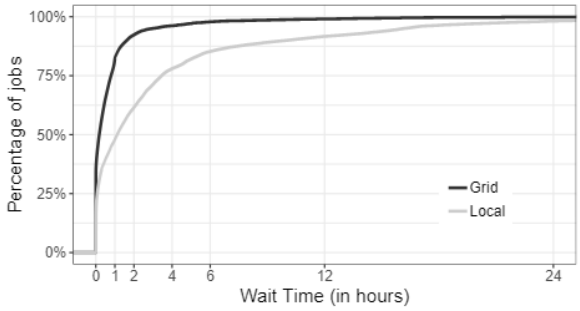

Fig. 3: Cumulative distribution functions of Local and Grid jobs wait time.

Figure 2 (right) shows the cumulative distribution functions of the actual run times of Local and Grid jobs. We can see that the distribution of job duration is very similar for both sub-workloads with a large proportion of jobs running for less than four hours. There is thus an important discrepancy between the characteristics of the jobs in terms of execution time and both the expression of requirements and the configuration of the scheduling queues.

Another important difference between the two sub-workloads is about how fast jobs can start their execution, i.e., how much time jobs have to wait in queues before starting. Figure 3 shows the cumulative distribution function of this waiting time for Local and Grid Jobs. We can see that while 99% of the jobs wait for less than a day, Local jobs tend to wait much more than Grid jobs. Indeed, more than 75% of the Grid jobs but only 50% of the Local wait for less than one hour. To understand the origin of such a discrepancy, we analyze the evolution of the number of jobs waiting in the queues over the considered period. Figure 4 shows this evolution which confirms that there are much more Local jobs waiting than Grid Jobs. On average, there are about 1,650 waiting Grid jobs but nearly 5,600 waiting Local jobs. More importantly, while the maximum number of waiting Grid jobs is always less than 4,000, there can be more than 40,000 Local jobs waiting in queues. Several factors can explain this difference.

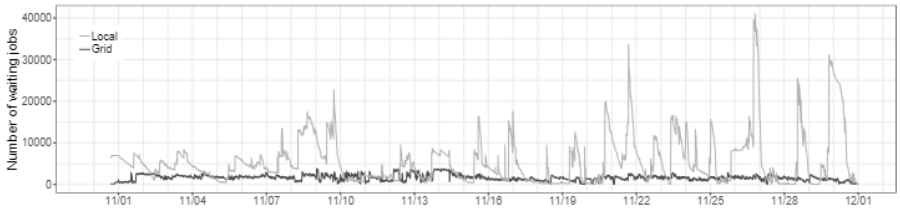

Fig. 4: Evolution of the number of waiting Local and Grid jobs.

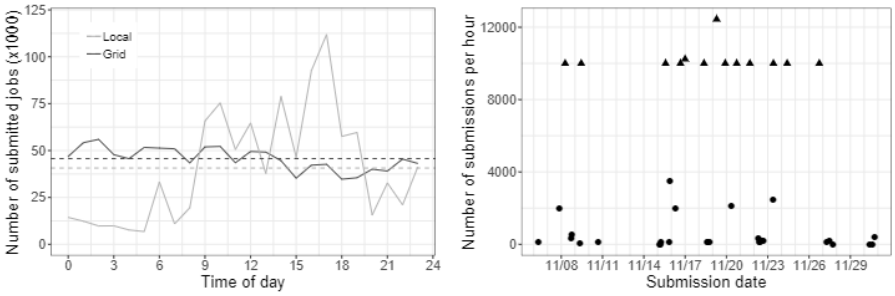

Fig. 5: Daily arrival rate for Local and Grid jobs on week days (left). Dashed lines depict the average number of submissions per hour. Submission pattern of a specific user group (right). Triangles correspond to submission bursts.

First, they clearly differ by their submission patterns. Figure 5 (left) shows the daily arrival rates for Local and Grid jobs. The dashed lines depict the average number of submissions per hour which is quite similar for the two sub-workloads (respectively 40,666 and 45,695 jobs per hour). However, we clearly observe two different submission patterns. Local jobs follow a traditional "working hours" pattern, while Grid jobs are submitted at an almost constant rate.

Figure 5 (right) shows the submission pattern of a specific user group that leverages the "array job" feature of UGE that enables the submission of up to 10,000 related jobs in a single command. Then we clearly observe burst submissions which are typical and correspond to "production" periods. They are usually followed by "result analysis" periods of lower activity. The combination of these general and specific submission patterns amplifies the difference in the number of waiting jobs with Grid submissions that are controlled upstream by monitoring and the use of pilot jobs.

The other factors that contribute to the observed discrepancy are that the groups submitting jobs through the Grid have the highest priorities in the fair-share computation, they are the main source of multi-core jobs that benefit of higher priorities on the `mc-*` queues, and more stringent RQSs are set for Local users thus limiting the number of concurrent running jobs.

4 Revisiting the Configuration of the Batch System

The analysis conducted in the previous section confirmed that the main operational objectives, i.e., a maximal utilization of the computing resources and the respect of a fair sharing of resources derived from the pledges made by the different user groups, were achieved. However, we also observed that the overall fairness provided by the scheduling algorithm hides a more unfair behavior that has been overlooked by operators. We showed in Fig. 3 that jobs submitted by Local users wait much more than Grid jobs.

Some of the root causes of these larger waiting times for Local jobs, i.e., higher submission rates during the day, and burst submissions, can not be solved by the scheduling system, but we believe that the current configuration of the system amplifies the effects of these peaks in Local job submission.

The primary job discrimination factor currently used is whether a job is sequential or multi-core. Our analysis showed that 96% of the multi-core jobs are Grid jobs. Moreover, the queues dedicated to multi-core jobs have a higher priority and the `mc-long` has access to the entire set of cores. Then, an important fraction of Grid jobs (27%) is privileged with regard to the rest of the workload. A second important factor is that the user groups with the biggest shares are the main source of grid jobs, hence increasing the priority gap with Local jobs. Finally, while local jobs express an estimation of their execution time, the current configuration of scheduling queues does not leverage this information.

To better reflect the respective characteristics of the two identified sub-workloads and increase fairness by better balancing the waiting times across jobs, we propose a new configuration of the scheduling queues described in Tab. 3.

The first three queues are dedicated to Local jobs and are considered in increasing order of expressed job duration. Moreover, the shorter the jobs are in a queue, the more resources the corresponding queue can access. Indeed, short jobs will quickly release resources, and thus can use more slots at a given time without harming the overall throughput. Then, all the Grid jobs are now placed in a single dedicated `grid` queue. By considering this queue *after* those for Local jobs, we give a higher priority to Local jobs and then reduce their waiting times. Finally, we merged the queues for jobs with special requirements to remove the sequential/multi-core distinction. The number of slots a queue can access has been empirically chosen based on the utilization shown in Fig. 1. We aim at having a good tradeoff between preventing queue overload and ensuring a maximal utilization even under load variations within the sub-workloads.

To assess the impact of these modifications on the overall utilization and the waiting times experienced by jobs, we implemented a simulator using the *Alea job scheduling simulator* [1,10] which is based on the GridSim toolkit [13]. Alea allows for detailed simulations, supports several mainstream scheduling algorithms (e.g., FCFS, variants of Backfilling, Fair-Share) and commonly used

Table 3: Names and upper limits (per job) of the queues in the new configuration. Queues are listed in the order in which they are considered by the job scheduler.

Queue name	CPU Time	Time	Memory	File Size	Pool size (in cores)
local-short	6h	7h	4G	30G	20,000 (59.8%)
local-medium	24h	28h	4G	30G	15,000 (44.8%)
local-long	48h	58h	4G	30G	10,000 (29.9%)
grid	48h	58h	3.6G	30G	25,000 (74.7%)
huge	72h	86h	10G	110G	10,000 (29.9%)
longlasting	202h	226h	3G	30G	5,000 (14.9%)

Table 4: Distribution of job waiting times.

Workload	Scenario	Average	Percentiles			Maximum
			50^{th}	75^{th}	90^{th}	
Grid	Baseline	1h 10m	0s	8m 18s	1h 18m	15d 21h 54m
	Modified	1h 45m	0s	14m	2h 2m	14d 4h 33m
Local	Baseline	2h 3m	4m 30s	1h 40m	6h 40m	11d 21h 41m
	Modified	1h 58m	8s	1h 10m	6h 20m	4d 19h 6m

system restrictions such as queue limits and quotas. We implemented two variants of the scheduling system. The first one constitutes a *baseline* and aims at reproducing the current configuration of the system. It uses the same queue definitions, shares, and RQSs as those used by UGE. Moreover the simulation starts with a set of "dummy" jobs running to mimic the state of the system on Nov. 1, 2018 at 12 am. The *modified* variant only differs by the definitions of the scheduling queues to use those presented in Tab. 3. We compare in Tab. 4 the distribution of waiting times in the two sub-workloads in the *baseline* and *modified* simulated schedules.

The proposed modification achieves its objectives. The average waiting time which was almost two times shorter for Grid jobs than for Local jobs is now more balanced, yet still shorter for Grid jobs. The median and third quartile values show that for a large fraction of Grid jobs, the impact of our modification is negligible, i.e., an increase of 6 minutes only, while the waiting times of Local jobs is reduced by half an hour. At the 90^{th} percentile, the waiting of Grid jobs almost doubles, but remains three times less than that of the Local jobs. Finally, our modification reduces the maximum waiting time for both sub-workloads with a noticeable reduction of one week for the most waiting Local job.

Then we verify in Fig. 6 that both variants of the simulator can reproduce the main trends of the original schedule for each sub-workload, even though it

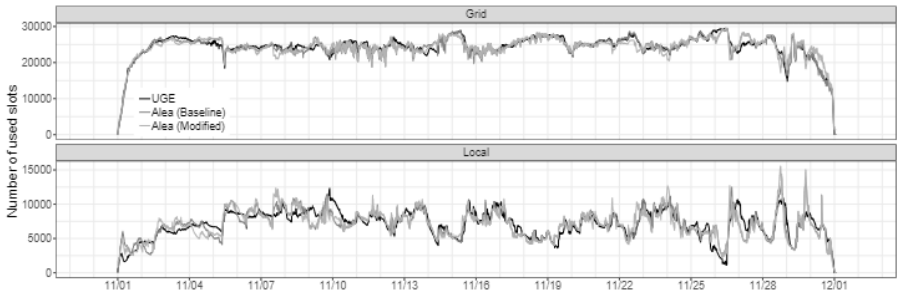

Fig. 6: Actual (UGE) and simulated (Alea) resource utilization, in slots, for Grid (top) and Local (bottom) jobs.

Table 5: Impact of RQS relaxation on job waiting times.

			Percentiles			
Workload	**Scenario**	**Average**	**50^{th}**	**75^{th}**	**90^{th}**	**Maximum**
Grid	Conservative	1h 53m	0s	16m	2h 21m	13d 15h 21m
	Extreme	1h 57m	4s	17m 41s	2h 47m	14d 4h 41m
Local	Conservative	1h 39m	2s	45m 40s	5h 8m	3d 16h 58m
	Extreme	1h 14m	1s	21m 55s	2h 30m	3d 23h 11m

does not capture all the configuration parameters of the real system. We can see that the *baseline* version can be used to evaluate the impact of the proposed modification and that the proposed modification achieves a very similar overall utilization and thus does not compromise the main operational objective.

We also propose to investigate the impact of a relaxation of the RQSs that are applied to local user groups and limit their access to resources. We make the assumption that the bigger is the original RQS, the bigger an acceptable increase can be. Indeed, a group constrained to use very few resources is more likely to cause I/O issues than a group allowed to use a large number of slots. We thus classify local user groups in three categories (i.e., 0-5%, 5-10%, and 10+%) according to the fraction of resources they can use. We propose two scenarios: a *conservative* one in which RQSs are respectively increased by 5%, 10% and 20%, and an *extreme* scenario in which increases are of 100%, 200%, and 300%. Table 5 shows a further reduction of the waiting time of Local jobs with a moderate impact on Grid Jobs. The *extreme* relaxation leads to the fairest schedule with similar waiting times up to the 90^{th} percentile. However, it may be too extreme to be accepted by the system administrators. From these experiments, we can conclude that the new scheduling queues in Tab. 3. combined with a conservative relaxation of the RQSs is a good candidate for production.

5 Workload Trace Production and Availability

The workload trace used in this work has been produced by extracting and combining information from different tables of the Accounting and Reporting Console (ARCo) of Univa Grid Engine. We first curated and anonymized the data and then converted the trace into the Standard Workload Format (SWF) [4] used by the PWA [6]. As this format does not allow us to log shares or RQSs, we accompany the trace with additional files. To ensure the reproduction and further investigation of the presented results, we made an experimental artifact that comprises the workload trace, its additional files, the code of the simulator, the simulation results and a companion document describing the data analyses [3].

6 Conclusion and Future Work

The operation of a large data center is a complex task which is usually driven by a few selected operational objectives. In the case of the IN2P3 Computing Center,

these objectives are to ensure a maximal utilization of the computing farm and the respect of a sharing of the resources proportional to the requests expressed for the year by the different user groups. To meet these objectives, operators rely on the implementation of the Fair-Share algorithm proposed by the Univa Grid Engine batch scheduling system and benefit from the main characteristic of the processed workload to be composed of a vast majority of sequential jobs.

Thanks to the batch scheduling system, the objective of an overall fairness across user groups, in terms of resource utilization, is achieved. However, the deeper analysis of the workload and its processing that we proposed in this paper showed that the current configuration of the system leads to a significant unfairness, in terms of job waiting time. We identified the root causes of this unfairness and proposed a pragmatic reconfiguration of the scheduling queues and quotas to address this issue. Detailed simulations of the original and modified configurations show that the proposed modifications better balance the waiting times across jobs without hindering the overall utilization of the system. The main operational objective is thus still achieved and the quality of service is improved for local users. Jobs submitted through the grid, which were previously favored now experience waiting times on par with those of the local jobs.

We are aware that the presented analysis and proposed optimizations are tightly coupled to the very specific use case of the CC-IN2P3. The applicability of our findings to other systems can thus be legitimately questioned. However, we believe that two important and more general lessons can be drawn from this study. First, the behavior and performance of a job and resource management system are not only driven by the sole scheduling algorithm. This central component is the most studied in the job scheduling literature, but we shown that other components such as the scheduling queues and resource quotas can be key factors too and have to be included in performance studies. Second, the configuration of a batch system is a slowly evolving process and decisions made at a given time may last beyond an important evolution of the workload and impede the operation. Regular analyses of the workload and revisits of the system configuration should thus be seen as good operation practices.

As future work, we would like to improve the simulator to better mimic the behavior of the system in production. A faithful replay of the original workload will allow us to better measure the impact of potential modifications of the system, and thus ensure that these modifications can be safely applied in production. We also plan to refine the configurations of the queues and investigate which parameters could be further tuned, e.g., handle array jobs more specifically, to improve the quality of service experienced by CC-IN2P3's users.

Acknowledgements

We kindly acknowledge the support provided by MetaCentrum under the program LM2015042 and the project Reg. No. CZ.02.1.01/0.0/0.0/16_013/0001797 co-funded by the Ministry of Education, Youth and Sports of the Czech Republic. We also thank L. Gombert, N. Lajili, and O. Aidel for their kind help.

References

1. Alea 4: Job scheduling simulator (Feb 2019), https://github.com/aleasimulator
2. Azevedo, F., Gombert, L., Suter, F.: Reducing the Human-in-the-Loop Component of the Scheduling of Large HTC Workloads. In: Proc. of the 22nd Workshop on Job Scheduling Strategies for Parallel Processing. LNCS, vol. 11332, pp. 39–60. Springer (2019)
3. Azevedo, F., Klusáček, D., Suter, F.: Companion of the Improving Fairness in a Large Scale HTC System Through Workload Analysis and Simulation article (2019), Available at: https://doi.org/10.6084/m9.figshare.8197823
4. Chapin, S., Cirne, W., Feitelson, D., Patton Jones, J., Leutenegger, S., Schwiegelshohn, U., Smith, W., Talby, D.: Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. In: Proc. of the 5th Workshop on Job Scheduling Strategies for Parallel Processing. LNCS, vol. 1659, pp. 67–90. Springer (2000)
5. Emeras, J., Varrette, S., Guzek, M., Bouvry, P.: Evalix: Classification and Prediction of Job Resource Consumption on HPC Platforms. In: Proc. of the 19th and 20th International Workshops on Job Scheduling Strategies for Parallel Processing. LNCS, vol. 10353, pp. 102–122. Springer (2017)
6. Feitelson, D., Tsafrir, D., Krakov, D.: Experience with using the Parallel Workloads Archive. Journal of Parallel and Distributed Computing **74**(10), 2967–2982 (2014)
7. Kay, J., Lauder, P.: A Fair Share Scheduler. Communications of the ACM **31**(1), 44–55 (Jan 1988)
8. Klusáček, D., Tóth, Š.: On Interactions among Scheduling Policies: Finding Efficient Queue Setup Using High-Resolution Simulations. In: Proc. of the 20th International Conference on Parallel Processing (Euro-Par 2014). LNCS, vol. 8632, pp. 138–149. Springer (2014)
9. Klusáček, D., Tóth, Š., Podolníková, G.: Real-Life Experience with Major Reconfiguration of Job Scheduling System. In: Proc. of the 19th and 20th International Workshops on Job Scheduling Strategies for Parallel Processing. LNCS, vol. 10353, pp. 83–101. Springer (2017)
10. Klusáček, D., Tóth, v., Podolníková, G.: Complex Job Scheduling Simulations with Alea 4. In: Proc. of the 9th EAI International Conference on Simulation Tools and Techniques (Simutools'16). pp. 124–129. ICST, Prague, Czech Republic (2016)
11. Michelotto, M., Alef, M., Iribarren, A., Meinhard, H., Wegner, P., Bly, M., Benelli, G., Brasolin, F., Degaudenzi, H., De Salvo, A., Gable, I., Hirstius, A., Hristov, P.: A comparison of HEP code with SPEC 1 benchmarks on multi-core worker nodes. Journal of Physics: Conference Series **219**(5), 052009 (2010)
12. Rodrigo, G., Östberg, P., Elmroth, E., Antypas, K., Gerber, R., Ramakrishnan, L.: Towards Understanding HPC Users and Systems: A NERSC Case Study. Journal of Parallel and Distributed Computing **111**, 206–221 (2018)
13. Sulistio, A., Cibej, U., Venugopal, S., Robic, B., Buyya, R.: A toolkit for modelling and simulating data Grids: an extension to GridSim. Concurrency and Computation: Practice & Experience **20**(13), 1591–1609 (2008)
14. The IN2P3 /CNRS Computing Center: http://cc.in2p3.fr/en/
15. Turilli, M., Santcroos, M., Jha, S.: A Comprehensive Perspective on Pilot-Job Systems. ACM Computing Survey **51**(2), 43:1–43:32 (2018)
16. Univa Corporation: Grid Engine. http://www.univa.com/products/
17. You, H., Zhang, H.: Comprehensive Workload Analysis and Modeling of a Petascale Supercomputer. In: Proc. of the 16th International Workshop on Job Scheduling Strategies for Parallel Processing. LNCS, vol. 7698, pp. 253–271. Springer (2013)