Pilot-Job Provisioning through CREAM Computing Elements on the Worldwide LHC Computing Grid

Alexandre F Boyer, David R C Hill, Christophe Haen, Federico Stagni

Pilot-Job Provisioning through CREAM Computing Elements on the Worldwide LHC Computing Grid

Alexandre F. Boyer[1,2], David R.C. Hill[1], Christophe Haen[2], Federico Stagni[2]
{alexandre.franck.boyer, christophe.haen, federico.stagni}@cern.ch, david.hill@uca.fr
1 Université Clermont Auvergne, ISIMA, CNRS, LIMOS, Clermont-Ferrand, France
2 European Organization for Nuclear Research, Meyrin, Switzerland

Abstract

The push model, used to manage Computing Grid resources, raises several issues that prevent employing the resources at their full capacity and, thus, limits the Grid throughput. While the push model remains necessary to get access to the resources, this paper aims at addressing some of these issues to make better use of the Worldwide LHC Computing Grid. After a thorough analysis of the DIRAC Workload Management System and its design to assign Pilot-Jobs, we propose different optimizations to improve the submission system and the grid throughput. The obtained results show significant changes in performances and enable the generation of a larger number of Pilot-Jobs to exploit grid resources better.

1 Introduction

To manage the increasing volume of data coming from the LHC experiments, CERN mainly leverages the Worldwide LHC Computing Grid (WLCG). The grid is composed of many Sites geographically distributed across the world, that contain Storage Elements (SEs) to manipulate data, as well as Computing Elements (CEs), to transfer the workload. A CE delivers consistent services to interact with a Local Resource Management System (LRMS). ARC [8], HTCondor [7], and CREAM [1] are a popular choice of CEs among Sites. LRMSs, such as SLURM [17], mainly spread the workload among their Worker Nodes (WNs). Jobs going through the whole model, also called push model, may encounter issues due to the number of transfers to perform and to the volatile nature of the resources.

The LHCb collaboration has developed and adopted DIRAC [15] to interact with the large number of distributed resources offered by the WLCG. It combines a Workload Management System (WMS) to handle and orchestrate jobs among distributed and heterogeneous resources, and a Data Management System (DMS), which includes automated data replication with integrity checking, needed to deal with large volumes. DIRAC pioneered the Pilot-Job paradigm within the Large Hadron Collider (LHC) context [5] to overcome many problems inherent to the push model. Developers have built it as a general-purpose framework and offered it to several user communities that would also need to manage similar resources like Belle II [10], CTA [2], and others.

Some Virtual Organizations (VOs) such as LHCb still mostly depend on grid resources, which can only be provisioned by a specific software platform, called Site Director. The problem is that the Site Director has to use the push model to supply resources with Pilot-Jobs. This

approach suffers from latency but remains necessary to get access to the WLCG resources. Nevertheless, improvements can be proposed to the Site Director, to maximize the usage of the resources, and the submission of Pilot-Jobs.

After a presentation of the main concepts, such as the push model, the Pilot-Job paradigm and some of the Pilot-Job provisioning tools, we discuss the DIRAC Site Director and its current limitations. We focus on the Site Director usage and interactions with the CREAM CEs in the LHCb context. CREAM CEs transfer about 33% of the total number of submitted Pilot-Jobs. Then, we describe the solution proposed to increase the pilot throughput with such computing elements and we present our results and discussion.

2 Pilot-Jobs and Provisioning Tools

2.1 Issues of the Push Model

Despite their diversity, middleware programs interacting with grid resources deal with a similar abstract model known as the push model. It implies transfers at many levels to dispatch the jobs, first from a WMS to different CEs within Sites geographically distributed, then from these CEs to inner LRMSs, and finally to WNs that execute the workload. Pushing a job through the whole model would expose it to failures such as transfer issues, mismatching with a WN architecture. This model proved to be inefficient and error-prone according to Stagni et al. [14].

2.2 The Pilot-Job paradigm

Pilot-Job paradigm has been devised and implemented mostly to support computations across multiple distributed machines. As Turilli et al. underline [16], the Pilot-Job paradigm appeared as a real solution for solving the inefficient push model, and thus has been quickly adopted in the Grid Computing context. Several user communities started an immediate development not grounded on any architectural patterns, which led to a variety of similar Pilot-Job implementations.

Casajus and his colleagues of the LHCb DIRAC team [5], define Pilot-Job objects, also known as pilots, as "nothing more than empty resource reservation containers that are sent to the available computing resources with the final aim of executing the most appropriate pending payload in the central WMS queue". Pilots can perform basic sanity checks of the running environment before pulling a given payload to effectively run it on well-behaved and adapted resources. Pilots create an overlay network that masks the WMS components from the heterogeneity of the underlying resources.

2.3 Pilot-Job provisioning tools

Pilot provisioning tools aim at automating the submission of Pilot-Jobs to the resources, ensuring high-availability and maximizing the throughput of the job. A lot of WMSs integrate such tools to supply WNs with pilots.

Condor [4], originally designed to allow users to execute tasks on a resource pool, was one of the first software to implement the Pilot-Job paradigm, under the name of Glideins, [9] to employ the grid resources via resource placeholders. It has been quickly complemented by GlideinWMS [13] to automate the provisioning of the Glideins. In the meantime, WMSs such as DIRAC [5], PanDA [11], and AliEN [3] have been developed and provide similar pilot

deployment features in spite of slight variations. Turilli et al. [16] further explain the Pilot-Job history.

Most of the Pilot-Job provisioning mechanisms aim at maximizing the throughput and minimizing the number of wasted resources by keeping a fixed amount of pilots in the Grid pool and continuously instantiating them while there are jobs to process. The tools usually generate pilots, that take the form of scripts, and send them to WNs via the grid architecture and the push model. To adjust the number of pilots within the grid and identify failures, some of them monitor the pilots. The characteristics of the jobs and priorities are matched with the attributes of the resources to achieve the best binding. Rubio-Montero et al. [12] and Turilli et al. [16] emphasize the commonalities but also the differences between several WMS in further details.

3 Characteristics of the Site Director

3.1 Operations of the Site Director

The Site Director is the DIRAC solution to supply WNs with Pilot-Jobs via the so-called push model. It works in cycles, executing the same logic at each iteration. An iteration consists in getting details about LRMS queues from a *Configuration* service first. Then, for each valid queue, the Site Director queries a *Matcher* service to get a list of jobs that could be executed on WNs associated with the given queue. According to the number of jobs that match the configuration n_j and the slots available in the queue S, the Site Director generates a certain amount of pilots n_p as scripts to run on the WNs:

$$n_p = \min(n_j, S)$$

A Site Director only submits the minimum number of pilots required, according to the jobs waiting in the queues, to avoid congesting the network. These scripts are finally pushed through the multiple grid components to reach the WNs, and the Site Director then registers the pilots in a database. To handle the inefficiencies of the push model, the Site Director monitors pilots to spot failures and provision resources accordingly. Figure 1 outlines the interactions between a Site Director, DIRAC services and a Site composed of one CE dealing with two queues.

It is worth noting that a Site Director is highly and dynamically configurable. Administrators can set up multiple instances that can run at the same time and manage specific Sites, CEs, and types of resources to split the workload. Additionally, they can tune the monitoring process to execute it every n_{update} cycle, set to 10 by default.

3.2 Constraints of the grid architecture

We carried out an analysis in DIRAC for the LHCb experiment to emphasize the different limitations of the Site Director that could cause latencies and prevent to submit as many pilots as needed to run jobs. Since we cannot profile the production environment, we draw on the DIRAC command-line interface, the web application, as well as the log files to get insight into the Site Directors.

The web application provides a pilot list along with a list of the jobs that they processed, as well as queues parameters. Despite pilots have been designed to fetch and run multiple jobs, most of the pilots, in the context of LHCb, handle a single job. Indeed, getting an accurate value of the time left allocated to a pilot is a complex operation due to the grid heterogeneity. Site

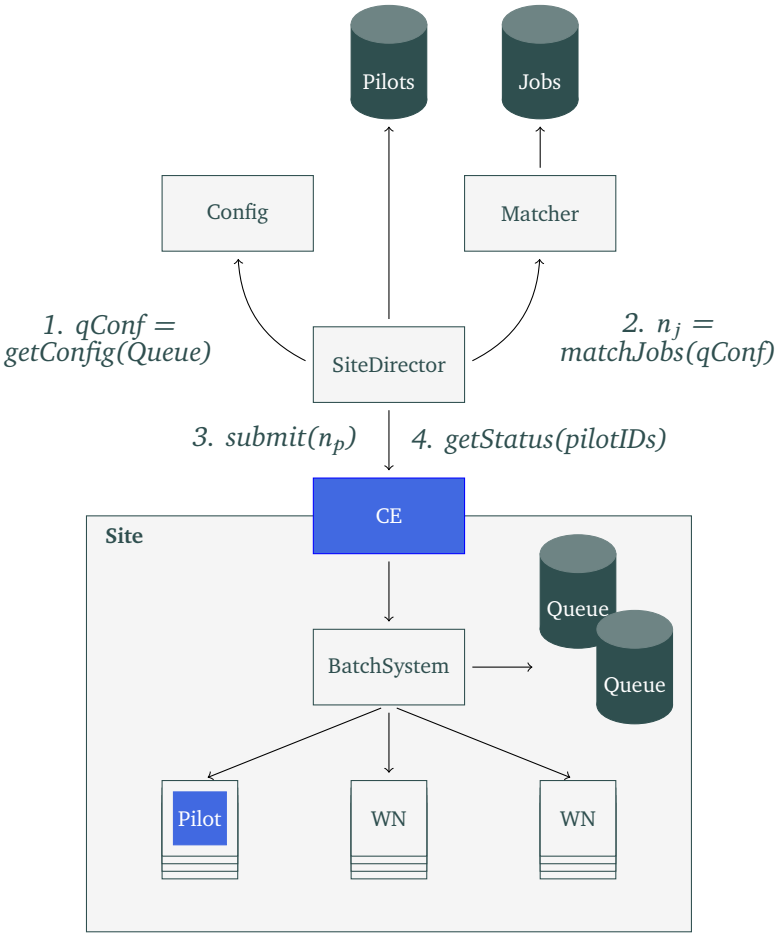


Figure 1: Interactions between a Site Director, DIRAC services and a Site

managers work with various LRMS types and versions and adjust specific features differently. Therefore, LHCb DIRAC administrators prefer to limit the number of jobs that a pilot can process, to avoid aborting the jobs that would run out of time. Thus, a Site Director generally submit a pilot per waiting job.

From the records, we also measured the duration that both 250 jobs and pilots take from their submission to their execution on a WN. Accurate values are only available on a small number of Sites, and thus results should be cautiously examined. The median duration to effectively bind a waiting job to a pilot is about 2 minutes, while the median duration to send and execute a pilot on a WN is 1 hour and 40 minutes and can last days in some cases. First, the medians demonstrate that jobs are rarely processed by pilots that were generated for this purpose. Second, the various queue levels, the scheduling policies of the Sites, and the communication overhead between required components to transfer the pilots are some of the factors inherent to the grid architecture and the push model that could also explain such durations.

Finally, queues may have a limited number of free slots S to carry pilots and run them, and the number of waiting jobs is often significantly superior to S.

3.3 Limitations due to the Site Director itself

As many different VOs are competing for limited resources, Sites managers may configure the LRMSs and their queues to accept a certain number of pilots. In this way, S the number of free slots in a queue is, in reality, composed of two parameters. We define *max pilots* as the

maximum number of pilots coming from a given queue, and bound to a VO, an LRMS can handle at the same time, and *max waiting pilots* as the maximum number of pilots, bound to a VO, that an LRMS can store in a given queue at the same time. To maximize the grid usage, Site Directors should fill the slots with pilots to reach *max pilots* in every queue they managed. As the number of WNs is limited, pilots stay a long time in the queues before running. Thus, the goal is to continuously submit pilots to maintain *max waiting pilots* in the LRMS queues.

The DIRAC command-line interface allows us to get a summary of the situation related to the pilots in the form of a JSON file. The file provides the number of pilots in a given state at a specific moment. *Running* is for pilots currently running on a WN while *Submitted* represents pilots that have been generated and are going to be queued. *Scheduled* covers pilots that are already queued, waiting for their submission on a WN. Both *Submitted* and *Scheduled* pilots are considered as waiting pilots, and their sum should reach *max waiting pilots*. In the same way, the sum of all pilots, irrespective of their status, could reach *max pilots*.

Figure 2 is the result of the command executed every 5 minutes for 12 hours and provides some of the plots generated. Plots only describe the activities of the pilots at a certain point in time, but we consider this sufficient to get a grasp of the limitations of the Site Director. In the same way, pilots can pass from *Submitted* to *Running* in less than 5 minutes, meaning some of them can only appear as *Running* on the plots, but this should not significantly impact the results. We notice that *max pilots* and *max waiting pilots* are reached but not maintained in the example. We can observe that the Site Director bound to the CE does not submit any pilot sometimes, whereas no pilot is queued and running pilots are decreasing through time. The web application can provide information about errors that could have occurred during the submission process, but nothing was reported for the studied CEs during this period. Thus, the limitation must come from the execution of the Site Director.

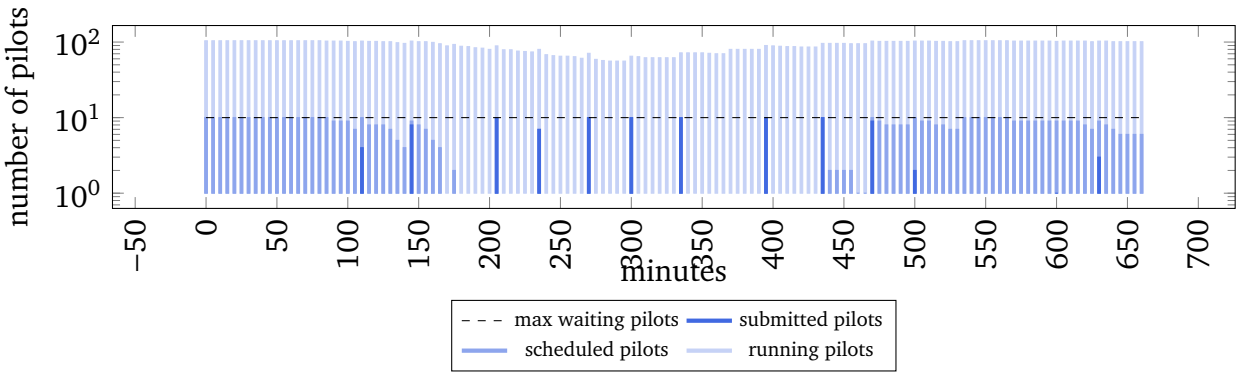


Figure 2: Status of the pilots supervised by a specific CREAM CE for 12 hours

In the LHCb context, each Site Director is bound to specific Sites and to a CE type to minimize the number of queues to manage per Site Director. DIRAC records its execution in a distinct log file where we can extract additional information. Each file consists of a suite of logs relative to the cycles. Each log contains a date as well as a message that can constitute a landmark to extract information of interest. We can retrieve details about the submission and monitoring tasks and their duration. To study the logs, we developed an analysis tool that draws on repeated messages and their dates across the files. Its purpose is to extract useful data from a given log file and summarize them into different graphs such as Figure 3.

Figure 3 describes the execution's length of a Site Directors handling a CREAM CE. The Site Director can spend around 500 seconds to make a cycle, with one time-consuming cycle

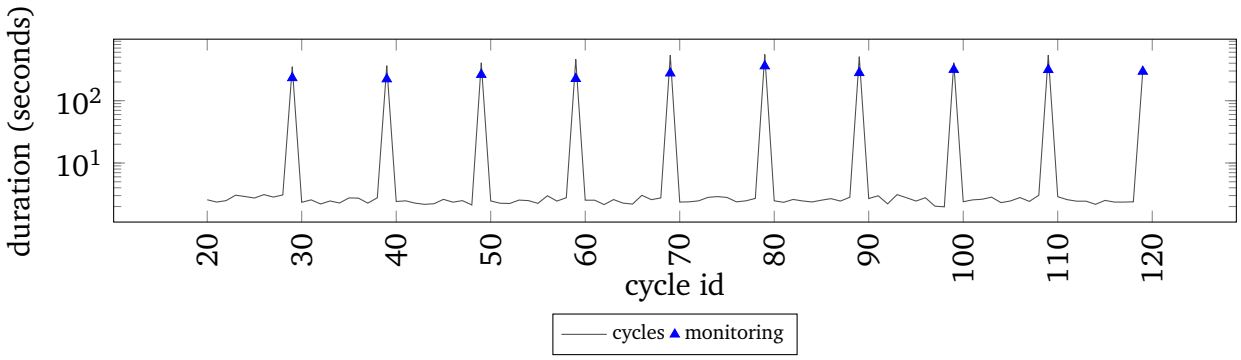


Figure 3: Duration of the cycles of a Site Director and their monitoring tasks

out of ten. This difference can vary according to the number of pilots managed by the Site Directors, the type of the supervised resources, their location, and their capabilities. Further studies in the logs highlighted a correlation between the cycles' length and the Site Director tasks, namely the submission and the monitoring of the pilots. The monitoring task, which happens one cycle out of ten, is the most time-consuming method within the cycles according to the example. By mapping the log messages with their location within the source code, it turns out that the communication between the DIRAC server and the CEs is the main cause of such consumption. Moreover, it seems that the submission process only happens at the same time as the monitoring task, while other cycles remain underused.

The combination of both Figure 3 and 2 suggests that the submission of pilots on a more frequent basis would help to maintain *max waiting pilots*. The main idea we will develop consists in increasing the submission rate and optimizing the monitoring process.

4 Optimization of the Site Director

4.1 Parallel interactions between the Site Director and CREAM CEs

To monitor the pilot statuses, a Site Director has to sequentially communicate with the CEs, via a communication interface adapted to the concerned CE type. Such interface takes the form of a plugin in DIRAC wrapping necessary tools to connect to a specific CE type. It allows the Site Director and other services to submit a pilot via a given CE, kill it, get its output and its status. These methods aim at sending requests to remote resources and waiting for data, which can be time-consuming and block the execution. Thus, as a Site Director may administer tens of CEs that could contain hundreds of pilots, parallel treatments should be privileged.

Connections should be processed simultaneously as much as possible to decrease or at least hide the waiting time to get remote data. Classical approaches to make an application parallel include processes and threads, but DIRAC has been written in Python and has to deal with the Global Python Interpreter (GIL). The GIL enables concurrency by preventing multiple threads from executing Python bytecodes at once, which does not benefit CPU-bound operations. In [6], we find further details about the GIL bottleneck concerning CPU-bound threads. Nevertheless, the interpreter releases the lock on I/O operations such as reading and writing in a file or connections to external resources, which is adapted to our needs. Indeed, the monitoring task would imply IO-bound threads. Connections to the CEs would create an opportunity to switch between threads and would minimize the stops in the program execution. Furthermore, and because of the GIL in Python, processes are privileged for CPU-bound operations. They would not prevent stops in the program and would highly depend on

the number of available CPUs on the DIRAC server. Figure 4 presents the current behavior and the expected one by using threads within the studied context. In each case, a Site Director connects to three CEs, ce_i, to get the pilot status. Each of them performs little CPU tasks before and after the connection. The central part of each ce_i represents the waiting time due to the connection. We expect threads to switch during I/O operations to avoid the program to stop, which would result in better execution time.

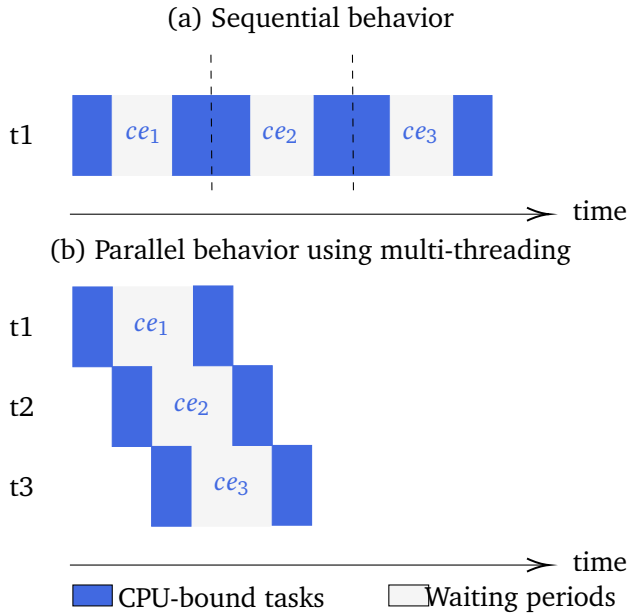


Figure 4: Duration for a Site Director to get pilot status in three CEs

4.2 CREAM communication interface and proxies

Even though getting pilot status in each CE simultaneously would ease the monitoring of the pilots and, thus, allow the submission of a larger number of them, it remains incomplete. Indeed, CEs may interact with hundreds or even thousands of pilots, and some of the communication interfaces could be better optimized. We have focused on CREAM resources that LHCb DIRAC mainly leverages to deal with inner LRMSs.

Pilots need to get proxies to run jobs. A proxy has a limited lifetime and the execution of some jobs may exceed it, which can lead to its expiration and the abortion of the jobs. To address this issue, before getting the status of a pilot, most of the communication interfaces perform a check of the proxy lifetime left and renew it if necessary. This takes the form of a connection again. The communication interface attached to CREAM does not perform this checking and renews the proxies of chunks of pilots in multiple requests every cycle involving the monitoring, which remains unnecessary. Renewing them every n cycles, n being larger than m the number of cycles to wait before invoking the monitoring would reduce the amount of time spent to monitor the pilots on this kind of CEs occasionally.

4.3 Pilot-Job Submission rate

A Site Director regulates the number of pilots to submit in a given LRMS queue according to the number of pilots running, scheduled and submitted, related to this queue, and also the limit values it can support, namely *max pilots* and *max waiting pilots*:

$$pilotsToSubmit = min((max\ pilots - pilots), (max\ waiting\ pilots - waiting\ pilots)) \quad (1)$$

In Equation (1), *pilots* represents the current number of pilots related to the queue, irrespective of their status, and *waiting pilots* the current number of submitted and scheduled pilots. It computes this number before each submission on a given queue to fill in every slot. Then, as the resources should be fully exploited for a while, the Site Director stops submitting in a given queue during 10 cycles. This could also explain the lack of submitted pilots sometimes, previously seen in section 3.3. Site Directors often submit in every queue during a same cycle, just before monitoring the pilots. Site Directors spending a lot of time monitoring pilots could probably submit new pilots during the following cycle as some of the submitted pilots could be already running on the WNs at the end of the operation. Furthermore, a cycle has a minimum duration of 120 seconds by default, even if the Site Director does not use it, which is the case of nine out of ten cycles. Therefore, we have introduced a new configuration option that intends to tune the number of cycles to wait before submitting in a given queue. This would allow us to submit a smaller amount of pilots more often to better meet the pressure demand.

5 Performance Results and Discussion

5.1 Results selection

Getting the overall benefit of the update is a complex operation. Indeed, scheduling policies of the queues may change over time. In the same way, DIRAC administrators may modify parameters related to the Site Directors, the Sites, the CEs, and the queues. They can also add or remove resources. Therefore, to complete the study, we have investigated data such as the monitoring time and the number of pilots submitted per hour, on all the Site Director running in production dealing with CREAM CEs, during 6 months, including one month after the update. The parameter to increase the pilot submission pace has not been used at this time yet.

We removed data related to the Site Directors instantiated or deleted during this period, as it would skew the study. Administrators have also modified a certain number of queues, their nature, and their parameters. Thus, we removed the Site Directors involved in these adjustments, only when their impact is significant. This statement concerns 7 Site Directors.

5.2 Relationship between monitoring duration and number of pilots submitted

We investigated a possible correlation between the monitoring time and the number of pilots submitted per hour. From data that we get, we computed a percentage representing the gain or the loss brought by the changes:

$$\frac{(mean\ after\ update - mean\ before\ update)}{mean\ before\ update}.$$

Figure 5 illustrates a scatter plot, where each point represents a Site Director.

They take between 80 and 97% less time to perform monitoring but submit between -8 and 26% more pilots per hour. Two Site Directors submit fewer pilots since the update. The first one submits pilots to a single queue, and thus the usage of the threads does not have any effect. The second one is handling 3 queues, but one of them is dealing with a larger number of pilots, which probably creates an imbalance in the duration of the thread. One of them is

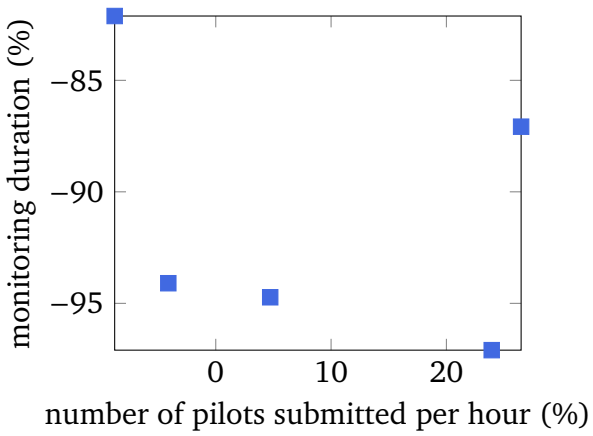


Figure 5: Possible effects of the changes on the Site Directors

likely to spend much more time than others. The other Site Directors are supervising a large number of queues, which could explain such differences in terms of pilots submitted.

Figure 6 provides further details that could explain the results. We examined the pilots' activity within a CE, using the command-line interface. We requested a pilots' activity summary every 5 minutes for 12 hours. To get more robust data, we replicated the operation three times, before and after the update. We also removed outliers caused by errors, such as temporary connection issues with the Sites. The first 3 box plots group designates the number of pilots by status before the update, and labels end with the letter *b* for *before*. The other represents the number of pilots by status after the update, and labels end with the letter *a* for *after*. In each group, *Su*, for *Submitted*, provides data about submitted pilots. *Sc* is the number of scheduled pilots, while *R* designates the running ones. *mp* and *mwp* stand for *max pilots* and *max waiting pilots*, respectively.

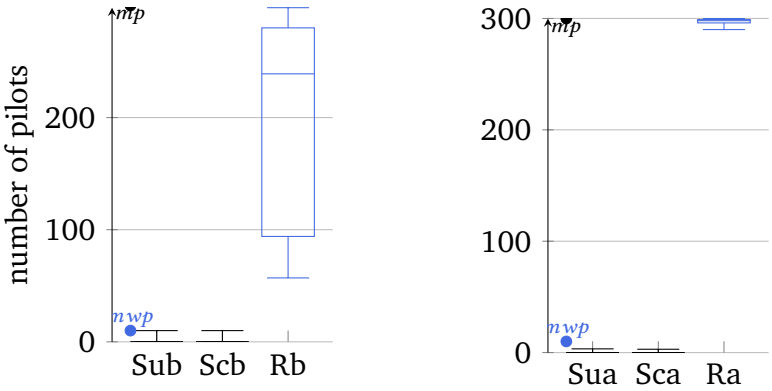


Figure 6: Number of submitted (Su), scheduled (Sc), and running (R) pilots supervised by a specific CREAM CE, before (left) and after (right) the update (median, $1st$, $25th$, $75th$, $99th$ percentiles)

The median of *Ra* was already close to *max pilots*, but the update helped to reach it and maintain it. The CE is submitting a maximum number of pilots in the WNs. In the meantime, the median of *Sc* is lower since the update. Pilots were probably waiting less time in the queues when we recorded the values, or the running ones are keeping resources a longer time, which decreases the submission rate.

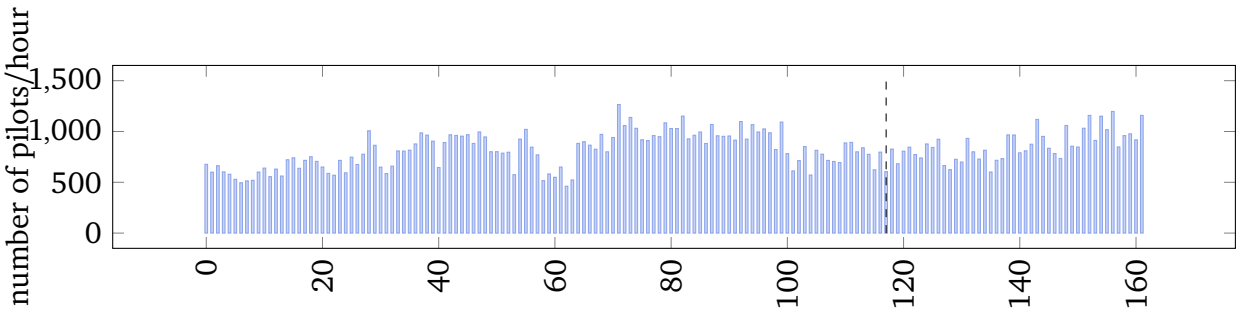
The optimizations that we made seem to have an impact on the number of pilots submitted. To better understand the results, we should consider the number of queues that a Site

Director manages, the original gap between the number of pilots submitted and the queues limits, namely *max waiting pilots* and *max pilots*, as well as the duration of the jobs to process. Nevertheless, we should carefully interpret such results and probably wait more time before confirming this impact, due to the heterogeneity of the resources and the number of modifications that could have occurred, not at our knowledge.

5.3 Evolution of the number of Pilot-Jobs submitted

Figure 7 presents the number of pilots that these Site Directors submitted. On one hand, we have the pilots successfully submitted, and on the other hand, the failed ones. The dashed line designates the date of the update. In the first case, the average number of pilots successfully submitted per hour was 809 and increased to 869 after the update, which represents an increase of 7.42%. The standard deviation slightly increased from 58 to 139 as well. We cannot see a noticeable transition at this scale, but Site Directors have submitted an increasing number of pilots over time. In the second case, the number of failed submissions increased right after the update. Further investigations in the logs suggest errors within the CEs and queues but remain unclear. These errors were already existing before the update, the changes that we made probably ease the submission process, even within these queues, which highlight them.

(a) Successfully submitted



(b) Failed to be submitted

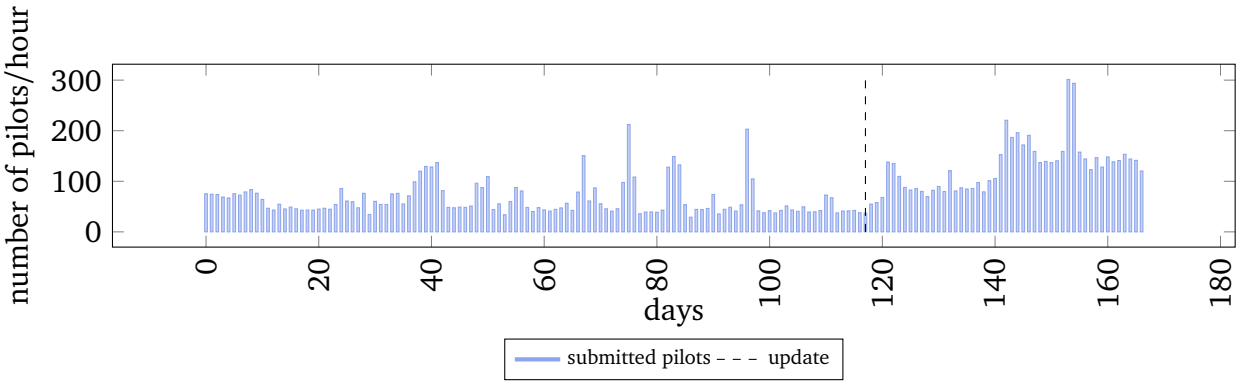


Figure 7: Number of pilots submitted per hour

Assessing the number of jobs that pilots, coming from the concerned Site Director, processed, would not be relevant. Indeed, this number has varied multiple times during 6 months, and thus the number of jobs processed per hour would not represent the impact of this study.

5.4 Other solutions considered along with this study

Administrators may instantiate new Site Directors to split resources across them. Having one Site Director per CE would likely provide better results than multi-threads, but would partially help since it would make the maintenance part difficult. Indeed, in the context of LHCb, we have hundreds of CEs, administrators would not be able to manage so many Site Directors.

In the same way, one could think about isolating the monitoring part of the Site Directors into a specific agent. On one hand, this would prevent the stops occurring in the execution of the Site Director and would ease the Pilot-Job submission. On the other hand, it would only shift the monitoring issues elsewhere and would continue affecting the Site Directors.

Finally, in section 3.2, we have demonstrated that jobs were rarely processed by pilots generated for this purpose, which may call into question the need to check the presence of jobs, before instantiating the pilots. Indeed, a Site Director only generates a limited number of pilots according to the jobs available that could run in a given resource, as well as the number of free slots. When pilots are finally running, this limited number may not reflect the number of jobs previously available as other pilots from different Site Directors may have already processed the jobs. This case happens can occur in specific Sites. Moreover, scheduling policies of some Sites prioritize VOs submitting the most pilots. Limiting the number of pilots in such a way could increase the scheduling duration. Thus, one could imagine a Site Director strategy consisting in continuously sending pilots in the queues, that would slow down production rates in the case that pilots do not fetch any job. Then, Site administrators could adapt the strategy to the situation, and the scheduling policy of the Sites.

6 Conclusion

This paper has explained the inefficiency of the push model inherent to the grid architecture involving multiple queues as well as volatile and heterogeneous resources. It also exposes several advantages the Pilot-Job paradigm has brought to the computing grid community to increase the throughput of the jobs, but also the limitations of the DIRAC provisioning tool. The two main features provided by the Site Director, namely the ability to submit pilots and the duty to monitor them afterward, have been presented in detail.

An analysis performed on the LHCb DIRAC instance enabled to follow the interactions between the Site Directors and the CREAM CEs. This study has underlined the need to optimize the monitoring mechanism to reduce the Site Director latency and thus to generate more pilots to meet an increasing demand. The monitoring involves a lot of long I/O operations with the CEs that blocks the main execution. The program has used the Python GIL that only allows one thread to run at a time and switches the execution between the threads during I/O operations primarily. Each CE has then been attached to a thread to minimize the time wasted by the I/O operations.

Site Directors also deal with interfaces to communicate with the resources that have been refined. Major improvements include the decrease of proxies renewal in the CREAM CEs since they are not mandatory.

We measured an overall gain of 7% of the number of pilots submitted per hour on the Site Directors examined. Changes seem more efficient when DIRAC deal with a large number of short jobs, and the Site Directors with a large number of queues and pilots. At this time, this significant improvement of the grid potential has to be carefully interpreted. Indeed, many external variables may affect the outcome.

Future Research

Future work will focus on the optimization of further types of CEs such as Arc and HTCondor, and the fine-tuning of the Site Directors configuration according to the state of the system to continue easing the throughput. We will also investigate new Site Director strategies consisting in continuously sending pilots in the queues.

Acknowledgements

I would like to thank Vladimir Romanovskiy for his precious advice about the Site Director mechanisms.

References

[1] P. Andreetto, S. Bertocco, F. Capannini, M. Cecchi, A. Dorigo, E. Frizziero, A. Gianelle, F. Giacomini, M. Mezzadri, S. Monforte, and et al. Status and developments of the cream computing element service. *Journal of Physics: Conference Series*, 331(6):062024, Dec 2011.

[2] L Arrabito, C Barbier, R Graciani Diaz, B Khélifi, N Komin, G Lamanna, C Lavalley, T Le Flour, JP Lenain, A Lorca, M Renaud, M Sterzel, T Szepieniec, G Vasileiadis, and C Vuerli. Application of the DIRAC framework to CTA: first evaluation. *Journal of Physics: Conference Series*, 396(3):032007, dec 2012.

[3] S. Bagnasco, L. Betev, P. Buncic, F. Carminati, F. Furano, A. Grigoras, C. Grigoras, P. Mendez Lorenzo, A. J. Peters, and P. Saiz. The alice workload management system: Status before the real data taking. *Journal of Physics: Conference Series*, 219(6):062004, Apr 2010.

[4] Allan Bricker, Michael Litzkow, and Miron Livny. Condor technical summary. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1992.

[5] Adrian Casajus, Ricardo Graciani, Stuart Paterson, Andrei Tsaregorodtsev, and the Lhcb Dirac Team. Dirac pilot framework and the dirac workload management system. *Journal of Physics: Conference Series*, 219(6):062049, Apr 2010.

[6] Beazley David. Inside the python gil. [Online] Available: http://www.dabeaz.com/python/GIL.pdf, May 2009. [Accessed: 11-Jan-2020].

[7] Thain Douglas, Tannenbaum Todd, and Livny Miron. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[8] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced resource connector middleware for lightweight computational grids. *Future Generation Computer Systems*, 23(2):219–240, Feb 2007.

[9] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, Jul 2002.

[10] Hideki Miyake, Rafal Grzymkowski, Radek Ludacka, and Malachi Schram. Belle ii production system. In *Journal of Physics: Conference Series*, volume 664, page 052028. IOP Publishing, 2015.

[11] Paul Nilsson. The panda system in the atlas experiment. In *Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research — PoS(ACAT08)*, volume 70, page 027. SISSA Medialab, Oct 2009.

[12] A. J. Rubio-Montero, E. Huedo, F. Castejón, and R. Mayo-García. Gwpilot: Enabling multi-level scheduling in distributed infrastructures with gridway and pilot jobs. *Future Generation Computer Systems*, 45:25–52, Apr 2015.

[13] Igor Sfiligoi. glideinwms—a generic pilot-based workload management system. *Journal of Physics: Conference Series*, 119(6):062044, Jul 2008.

[14] Federico Stagni, Andrei Tsaregorodtsev, A. McNab, and C. Luzzi. Pilots 2.0: Dirac pilots for all the skies. *Journal of Physics: Conference Series*, 664(6):062061, Dec 2015.

[15] Andrei Tsaregorodtsev. Dirac distributed computing services. *Journal of Physics: Conference Series*, 513(3):032096, Jun 2014.

[16] Matteo Turilli, Mark Santcroos, and Shantenu Jha. A comprehensive perspective on pilot-job systems. *ACM Comput. Surv.*, 51(2):43:1–43:32, Apr 2018.

[17] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.