

Improving Many-Task Computing in Scientific Workflows Using P2P Techniques

Jonas Dias¹, Eduardo Ogasawara^{1,2}, Daniel de Oliveira¹, Esther Pacitti³, and Marta Mattoso¹

¹Federal University of Rio de Janeiro, Rio de Janeiro, Brazil

²Federal Center of Technological Education, Rio de Janeiro, Brazil

³INRIA & LIRMM, Montpellier, France

{jonasdias, ogasawara, danielc, marta}@cos.ufrj.br
pacitti@lirmm.fr

Abstract

Large-scale scientific experiments are usually supported by scientific workflows that may demand high performance computing infrastructure. Within a given experiment, the same workflow may be explored with different sets of parameters. However, the parallelization of the workflow instances is hard to be accomplished mainly due to the heterogeneity of its activities. Many-Task computing paradigm seems to be a candidate approach to support workflow activity parallelism. However, scheduling a huge amount of workflow activities on large clusters may be susceptible to resource failures and overloading. In this paper, we propose Heracles, an approach to apply consolidated P2P techniques to improve Many-Task computing of workflow activities on large clusters. We present a fault tolerance mechanism, a dynamic resource management and a hierarchical organization of computing nodes to handle workflow instances execution properly. We have evaluated Heracles by executing experimental analysis regarding the benefits of P2P techniques on the workflow execution time.

1. Introduction

Over the last years the power of clusters has grown, which enabled the development of large-scale scientific experiments that are dependent of High Performance Computing (HPC) environments [1]. These experiments explore many executions of a set of related and dependent activities. Running these activities respecting their relations of dependencies and

also controlling data transfers between different resources require a high level management. Large-scale experiments are typically executed as scientific workflows [2], which is an abstraction that represents the chaining of activities to be executed. These activities consume input data by executing a computer program or scripts, in order to produce output data. The input and output data may be a set of parameters or files. Additionally, the output of a given activity can be consumed by following activities representing the workflow chaining. Scientific workflows are enacted by engines called Scientific Workflow Management Systems (SWfMS). Many SWfMS are available [3-5].

During the execution of an experiment, the same activity from a scientific workflow or even the whole workflow may be executed several times exploring different parameter combinations. Many of the workflow activities also process huge amounts of data. This intensive computation motivates the parallelization of a workflow activity execution. Furthermore, instead of letting scientists to specify how many processes a given activity should be parallelized, scientists should just specify a deadline, *i.e* the maximum time that an activity (or workflow) can execute in the cluster. This approach would be more flexible to scientists and adequate to the utility computing model [6]. However, it may require mechanisms for dynamically scheduling computing resources. High performance computing environments such as clusters, grids and clouds [7] are candidates to execute workflows in parallel.

A possible approach is to use MPI [8] to parallelize workflow activities. However, it may be unviable to add MPI logic inside complex and legacy code [9]. Parallelizing legacy programs can be very complex and it may demand a completely new code to be written. MPI does not support dynamic resource management either. This is problematic since activities are

¹ This work is partially sponsored by CNPq and CAPES

submitted as coupled parallel MPI processes. A single process failure aborts the whole group of activities [10]. Alternatively, submitting many uncoupled activities may also lead to a long waiting in queues. The success of a workflow execution depends on the successful execution of all running instances of the workflow activities. Thus, if one activity execution fails and is not automatically rescheduled, the whole workflow execution fails. Many activity rescheduling is very complex to be performed manually.

There is no uniform invasive MPI approach to parallelize workflow activities, since they belong to a heterogeneous set of programs, scripts and services from different sources. Thus, noninvasive parallelisms, *i.e.*, parallel approaches that do not make changes on the activity source code, such as parameter sweep or data parallelism, are more suitable to deal with workflow parallelization. Moreover, since these workflows may also generate a great amount of tasks, the new computational paradigm, known as Many-Task Computing (MTC) [11] seems to be adequate to support scientific workflow parallel execution [12].

Although MTC is a new and promising paradigm, there are still many open, yet important, issues to find adequate strategies for different types of scheduling and execution. Current MTC solutions [3,13-15] explore workflow parallelization, but they are somehow grid oriented and do not explore some characteristics of current clusters. For example, what are the winners approaches to deal with the dynamic behavior of computing nodes inside huge clusters? This issue is important since the number of cores per cluster is growing fast and, even with better machinery, as the number of electronic components increases, greater is the chance of a failure [16].

On this scenario, when a node of a cluster fails during the execution of a workflow activity, the MPI limitation causes a serious management issue. Sonmez *et al.* [17] analyze the performance of traditional scheduling policies on grids of multiple independent clusters and find that there is no single workflow scheduling policy with good performance on their investigated scenarios. They also noticed that head-nodes of the clusters usually get overloaded during workflow executions. That is the reason to believe that studies regarding scientific workflow execution using MTC on huge clusters are still necessary.

The huge number of nodes with possibly heterogeneous hardware and their dynamic behavior on clusters make us consider this scenario as similar to the ones found in Peer-to-Peer (P2P) computing. Consequently, P2P techniques may be useful on clusters to the management of workflow activities execution that demands MTC. Pacitti *et al.* [18] show that P2P techniques are useful on large-scale grids.

However, there are still not enough studies evaluating the usage of P2P techniques on huge clusters to process MTC workflow activities.

This paper proposes Heracles, an approach that uses P2P techniques to address many-task computing in scientific workflows on huge clusters such as a petascale supercomputer. Among its techniques, Heracles explores fault tolerance mechanisms, dynamic resource management and hierarchical node arrangement. Heracles deals with node failures, node overloading and dynamic node allocation to attend workflow execution time constraints. These characteristics are not explored by traditional cluster job schedulers (such as Torque [24]). This paper presents an initial evaluation of Heracles working inside an MTC scheduling scenario. Simulation studies analyze the impact of failure events on tasks execution on clusters. We have evaluated scenarios of having workflow activities producing 512, 1024, 2048 and 4096 tasks. The study results reinforced that Heracles' automatic rescheduling is an important feature on parallel workflow activity execution on clusters. Even with low failure rates, relying on manual rescheduling causes a great loss in the number of complete activities over time.

The remainder of this paper is as follows. Section 2 discusses background on P2P computing and workflow scheduling mechanisms; Section 3 presents Heracles, explaining its strategies using P2P techniques; Section 4 presents a proof of concept discussing the need to use Heracles. Section 5 concludes the paper.

2. Issues in Workflow Parallelization

P2P computing provides high scalability by forming heterogeneous and dynamic networks with decentralized control. P2P networks are also aware of churn events [19], since peers autonomy let them join and leave the network at anytime. Thus, P2P systems must be fault tolerant and implement dynamic resource management to provide quality of service.

In previous work [18], the authors discussed how P2P techniques might be combined to enhance large-scale grid data management. It was observed that different combinations of techniques can be envisioned for different grids. Thus, we believe that huge clusters may also take advantage of a given combination of P2P techniques. Additionally, other study [22] about workflow activities execution on P2P networks suggests that the hierarchical P2P network [20] is an indicated approach to schedule and execute workflow activities in parallel.

Workflow scheduling means that an activity of the workflow is scheduled to run on external resources

such as clusters or grids. The workflow parallelization means that the execution of an activity or some activities of the workflow is done in parallel to speed up the process. Nevertheless, even when an activity is scheduled to execute on an external resource, the control over the workflow execution is held by the SWfMS. The following sub-sections discuss those issues.

2.1 Workflow Scheduling and Parallelization

Scientific *in silico* experiments [23] are usually modeled as Scientific Workflow, which may be managed by different types of SWfMS. Some activities of the workflow can be configured to run on external resources, such as cluster and grids. Many papers discuss on workflow activity scheduling in parallel environments. DAGMan [24] and Pegasus [25], for example, are both SWfMS that became very popular due to their support for executing workflow activities on distributed resources. Recently, other studies propose other approaches for parallel workflow scheduling. Swift [3] is an environment used to specify a logical structure of a process using a script language that enables the registry of provenance data [26]. Additionally, Swift task scheduling can be done using Falkon [13], which is a many-task computing framework that sets apart the concept of acquiring resources from task submission to meet MTC requirements.

Each SWfMS has its particular characteristics, notation and language. They focus on different features such as scientific visualization, provenance or parallel execution. When experiments are modeled on a given SWfMS, the workflows stay dependent on specific technological issues. Besides, the representation on a particular SWfMS may overshadow the knowledge behind the modeled workflow. During an experiment, if scientists demand high performance, they need to re-model their workflows and possibly change the source code of their activities to obtain the desired level of parallelism and performance improvement. However, to re-model an entire workflow requires too much effort and may also lead to errors. Further, scientists are not required to be familiar to computer programming and/or parallelization methods. Thus, we believe that the parallel execution of the workflow activities should be done implicitly. An implicit approach tries to make the workflow parallel execution more transparent to the scientists without changing their applications. It means that the approach should be noninvasive, so it does not change the applications source code. This is very important since many scientific applications have very complex legacy code [9] that is very difficult to modify. The application may

also be proprietary, which means that scientists do not have access to its source code. Nevertheless, even without accessing the code, it is still possible to run these applications in parallel. Thus, the proposed approach brings transparency to the parallel execution of workflow activities, but the control over the workflow execution is still held by the SWfMS. Scientists may use their preferred SWfMS to schedule his workflow to run on a cluster through a MTC scheduler such as Falkon or Hydra. The scheduler may use Heracles approach to handle the execution of the activity in parallel providing transparency, load balancing and fault tolerance.

During the scientific experiment life cycle [27], the same experiment modeled as scientific workflows may run several times exploring different sets of parameters or input data. It is usually possible to run multiple parameter combinations simultaneously, in parallel, assigning instances of the workflow activity for each machine. This scenario is commonly called a parameter sweep case [15,28]. Data parallelism is also used in scientific workflows to increase the execution performance of the experiments, since the input data of a given application is fragmented into smaller pieces that are processed faster by the application. So, processing fragments in parallel leads to a faster overall execution. Each instance of these activities that process a different set of parameter or data fragment may be seen as a task. Yunhong Gu *et al.* [29] explore data parallelism in wide area networks using Sphere, a cloud computing system to run tasks that demands data parallelism. They consider network heterogeneity, load balancing and fault tolerance. However, they do not consider that the distributed tasks belong to scientific workflows, what reduces the degree of management of the experiment as a whole. Heracles uses parameter sweep and data parallelism approaches to provide a transparent parallelization, but it also uses other methods to enhance it as described on section 3.1.

MapReduce programming model is also explored in workflow scheduling as a particular case of data parallelism. Basically, a huge set of data is split into smaller pieces and mapped to be processed on compute nodes. The pieces are passed into the Splitter function (called Map) as (key, value) pairs. After the map function is executed, the intermediate values for a given output key are merged together into a list by an aggregation function (called Reduce). The reduce function combines the intermediate values into one or more final result related to the same output key.

Some related work in the literature proposes general solutions for managing workflows executions in large clusters. One example is GlideinWMS [30]. It is a general purpose Workload Management System (WMS) that relies on Condor software, with additional

glideinWMS specific code. Another approach is Corral [31], which is a system that runs a suite of real workflow-based applications including in astronomy, earthquake science, and genomics. Provisioning resources with Corral ahead of the workflow execution reduced the runtime of an astronomy application. Although these approaches propose solutions for workflow management in distributed environments, they still do not provide a transparent approach to submit, execute and gather provenance of workflow activities. They also do not follow the MTC paradigm to provide noninvasive parallelism. Therefore, based on the discussed studies, we believe that there are still open challenges to enhance workflow parallelism.

2.2 Challenges in Parallelizing Workflows

During an experiment setup, when scientists configure how the workflow instances may run, it may be error prone to specify technical factors that define how the environment must process the activities. The parallel execution of the workflow activities should be transparent for scientists just like query processing on distributed database systems [29].

Usually, on a parallel environment, scientists need to specify the number of processors they want to process their tasks. However, it is more transparent for the scientists if they could just define a deadline for their results. This approach fits the utility computing scenario where the users pay only for what they use [6]. Scientists would only specify what they want answered and when they need this answer. The SWfMS would submit this activity to a scheduler system capable to allocate resources on demand and deal with resource failure automatically. Thus, to attend time constraints defined by scientists, the scheduler must be able to manage the available resources dynamically. One initial approach would be to put a special module on the head-nodes of the clusters to control the scheduling and execution of workflow tasks dynamically. However, previous work [17] noticed that these nodes became overloaded due to the large number of workflow tasks and file transfers they have to manage on a dynamic scheduling scenario. Therefore, better strategies are still necessary to provide a transparent and reliable parallel execution for scientific workflows.

3. Heracles

Based on the discussed studies, analyzing their advances and limitations, we propose a new approach named Heracles. The goal of Heracles is to improve scientific workflow activities parallelization that

demands many-task computing on huge clusters. To achieve this objective, we use a set of P2P techniques and concepts to provide transparency, load balancing and quality of service. This is designed to support huge clusters systems that normally have a difficult job setup, centralized control and frequent hardware failures.

Heracles is designed to be implemented inside MTC schedulers for workflows (such as Swift/Falkon [3] or Hydra [12]) to control the parallelization of workflow activities. The integration with the SWfMS is a duty of the scheduler. Hydra, for example, already implements this SWfMS integration. Figure 1 represents an overview of Heracles structure. The SWfMS submits workflow activities to run on a cluster using an MTC scheduler. The many instances of workflow activities with different input data or parameters can be wrapped as tasks. Assuming that MTC schedulers for workflows are using Heracles, these tasks are not directly scheduled to run on the cluster. Instead, Heracles registers the metadata of each task on a distributed table [32] shared by a group of Heracles processes. A Heracles process is the actual object that is scheduled by the regular scheduler of the cluster to run in parallel. They are bootstrapped just like other jobs of the cluster and, once started, the processes use P2P techniques to control the tasks execution and gather all necessary provenance data requested by the MTC scheduler. The scheduler may, then, forward the provenance data to the SWfMS. Since Heracles processes are autonomous, they have components to handle the virtual P2P network overlay, to execute tasks and to gather metrics values and provenance data (execution monitor) during the task execution.

The advantage of implementing Heracles inside an MTC scheduler is that Heracles does not have to concern about important mechanisms such as data staging ones. Data staging allows you to stage data needed by a task before the task begins execution and to move data back to archives when a task has finished execution. However, data staging is a role of the MTC solutions for workflows (such as Swift/Falkon or Hydra). The following sections describe the main strategies used by Heracles.

3.1 Transparency

Heracles aims to improve the way scientists setup their activities to run in parallel. Instead of configuring the number of nodes of the cluster to be involved in the parallel activity, scientists should specify the deadline for the experiment results to provide a more flexible setup for parallelization. Heracles sets a number of computing nodes to compose the initial execution pool. This setup is based on the time constraint established

by scientists, the available resources and the information obtained from past executions (called provenance data [26]). This resource pool may grow or shrink depending on the time constraints. The decisions regarding these expansions are based on two metrics: partial efficiency (E'_p) for the set of p processors and the number of completed activities per time unit (C'_t). These metrics can be calculated from the distributed table available on Heracles process.

For example, let us suppose that scientists configure a given workflow activity composed by k tasks deadline to be w hours. Heracles decides to start the process on the cluster with p processing cores. After an arbitrary number n of finished activities, Heracles updates the number of completed activities per time unit (C'_t). To decide whether is necessary to expand or shrink the pool, it measures the partial efficiency (E'_p):

$$E'_p = \frac{\sum_{i=1}^n T_i}{p \cdot T_p} \quad (1)$$

Where T_i is the time spent processing activity i and T_p is the elapsed time. To calculate the new size p_{new} of the resource pool, Heracles uses the following formula:

$$p_{new} = \frac{1}{E'_p} \left(\frac{p \cdot (k - n)}{w \cdot C'_t} \right) \quad (2)$$

To illustrate the described strategy, consider $w = 72$ hours, $p = 32$ processors and $k = 10,000$ tasks. After 160 tasks have finished in about three hours, Heracles measures that $C'_t = 89.6$ tasks per hour. Heracles calculates an efficiency of $E'_p = 0.85$. So the new estimated size of the pool p_{new} is 57 processors. This value may be approximated considering the cluster infrastructure. For example, if each compute node has eight cores, p_{new} can be approximated by 64 since 64 is a multiple of 8. Figure 2 shows the evolution of C'_t , E'_p and p over time using the arbitrary example. Heracles starts with a small pool and then expands it to achieve a peak task-per-hour rate. Heracles also considers the amount of available resources on the cluster since there may be not enough compute nodes to allocate all p_{new} processors of the pool. However, based on p_{new} , it follows a tendency and, when possible, allocates (or frees) processing cores. Figure 2 shows that Heracles initial tendency is to obtain as many as possible processors on the pool. Yet, during the execution it seeks better efficiency.

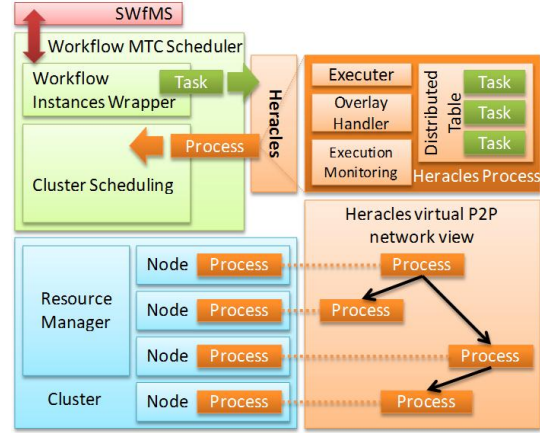


Figure 1: Heracles scenario overview

Using this approach, the resources usage tends to be more efficient, since activities with looser deadlines consume fewer resources to let activities with tighter deadlines to use more. Heracles uses the metrics to analyze and decide if it can reduce the number of resources being used in the case where the deadline is tending to be accomplished. This strategy makes it simpler and more transparent to scientists to setup their workflow activities, since they only need to set up a deadline. Heracles uses metrics to dynamically expand or contract the working resources to attend the time constraints.

3.2 Load Balancing

Clusters usually have centralized control on a head-node that controls submission and execution of all the cluster jobs on the working nodes. The head-node is the frontend for all the services available in the cluster machine. This approach makes it easier to manage a huge machine, even if there is more than one head-node. However, centralized approaches naturally lacks on load balancing. Thus, cluster head-nodes are powerful server machines capable to handle most applications that demand high performance computing. On a many-task computing scenario, though, the head-node of the cluster may become overloaded like Sonmez *et al.* [17] experienced on a workflow scheduling scenario. This overload happens because of the big amount of tasks that the workflow activities may generate. The head-node also has to deal with file transfers to stage in and stage out all the necessary data for each task. Provenance data needs also to be gathered during tasks execution and, since the working nodes network is private, this data may transit through the head node network.

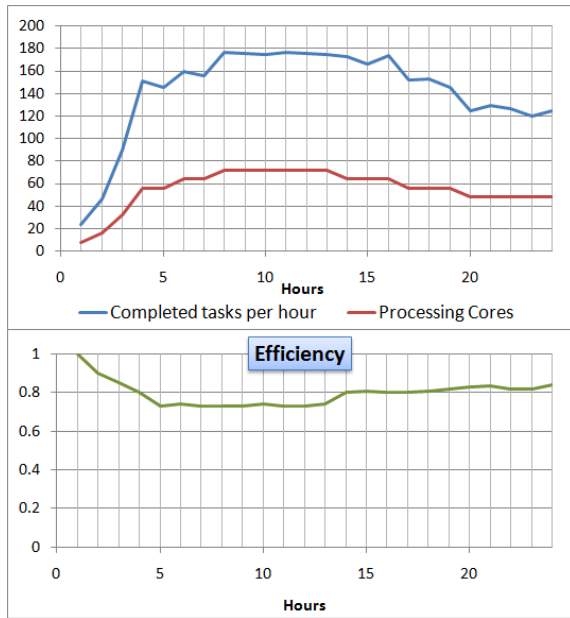


Figure 2: Example of the dynamic task scheduling using Heracles approach

Many of processes executed in the head-node cannot be moved to another resource, since the head-node is the only one responsible for many of the cluster services. However, the workflow scheduling application should not add extra responsibilities to the head-node. Thus, Heracles aims at improving load balancing distributing the task scheduling and execution management over the working nodes. To achieve this goal, Heracles establishes a hierarchical virtual P2P network over the resource pool reserved to execute the tasks. The hierarchical approach aims at expanding and shrinking the pool and to provide load balancing.

Heracles divides the resource pool into groups and elects one processor of the group to be the leader. On each pool expansion, new groups are formed. When a new group enters the hierarchy, it starts in the highest hierarchy level. As new groups enter the network, the older groups start to descend in the hierarchy. Since older groups are likely to leave the network when the pool shrinks, it seems to be an advantage to keep them on the bottom of the hierarchy level. Figure 3 (a) represents the complete resource pool. Each group is composed of 8 processors from the same compute node. Figure 3 (b) shows how the groups form the hierarchy. The numbers indicate the order that the groups were created and the darker squares are the leader processors of each group. At the same time that the hierarchy keeps the oldest groups at the bottom, it also tries to keep the hierarchy tree balanced.

Group leaders keep the list of tasks to be processed on a distributed table. When a processor finishes the

processing of a given task, it reports it on the table of its leader. The processor can then grab a new task published still to be executed on the distributed table. The group leaders are also responsible to gather provenance data of processed tasks. Centralizing the gathering process on the leader is important to save database transactions or disk writings operations depending where provenance data is written. Leaders are also important on the fault tolerant mechanism described in the next section. The leader of the highest level group is the one responsible to measure the partial efficiency and to make decisions regarding the expansion or shrinking of the resource pool.

The hierarchy approach improves load balancing distributing the scheduling and execution control. The groups of processors have a decentralized autonomy and report to their leader, while the leader reports to the upper level leader in the hierarchy. This structure also makes the dynamic resource management easier and enables and efficient fault tolerant mechanism.

3.3 Quality of Service

On typical cluster systems, if a working node fails during the execution of an activity, the activity is aborted. This procedure is indeed necessary if the activity is composed of a set of coupled MPI processes. When scientists submit a huge set of uncoupled tasks to be executed, some schedulers distinguish that singular failures affects only the task that was running on the node that failed. This is a better scenario, but scientists need to check every failed task and then resubmit them. These kinds of manual effort increase the chances to make mistakes and, commonly, scientists prefer to resubmit the whole set of tasks. However, the scheduler can automatically reschedule the tasks that failed. The tasks are, then, processed by another node just like in a P2P system.

It is possible to relate the failures on a cluster with churn events [19], which are very common on P2P networks. On huge clusters the frequency of failure events would be much smaller compared to a traditional P2P system. However, if scientists schedule many-tasks activities using a huge set of nodes from the same cluster, it is probable to experience failures. Since these failures are expected, we consider any failure on a task being processed as a churn event. After a churn event, the tasks assigned to the process or compute node that left the network should be assigned to another one. Fault tolerant mechanisms available on P2P systems are helpful to handle failures on cluster during many-task execution. Thus, to improve quality of service during workflow scheduling and execution on clusters, Heracles uses a fault tolerant mechanism.

Each processor core available on the resource pool of Heracles can be seen as a peer. A peer obtains tasks to process from the distributed table published on its group leader. The table has the list of all tasks: pending, running and finished ones. A pending task changes its state to running just after being scheduled to be processed by a peer. The group leader keeps an average of how much time a task takes to accomplish. If a task is taking too much to finish, it is possible that the responsible peer failed. The leader peer may, then, reschedule the task by changing its state to pending again.

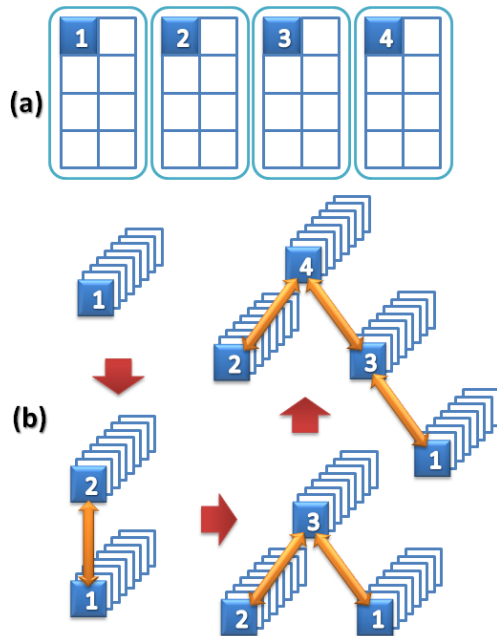


Figure 3: (a) Physical organization of the resources; (b) Resources mapping into the P2P groups by Heracles

On a worse scenario, the leader of a group or even the whole group may fail. This is possible since all the peers are processor cores which may be inside the same chip or compute node. Anyway, the leader failure is as bad as the group failure, since, without a leader, the peers within the group lose their communication with Heracles. Thus, on this scenario, the leader of the group one level above on the hierarchy has to notice the churn and automatically reschedule all the tasks once assigned to be processed by the group that has failed. This rescheduling may be done just changing the once running state of the tasks to pending. A group failure is easier to be noticed by other leaders in the hierarchy since they keep contact using overlay messages to maintain the distributed table update. If a leader observes that some other leaders are not executing their tasks and not responding overlay messages, it understands that a churn event happened

and decides to reschedule the tasks in the table that were assigned to that failing group.

It is notable that, if the group that failed is on an intermediary level of the hierarchy, the groups below that level do not lose the communication with Heracles structure since they keep information regarding other group leaders and can reposition themselves and later rebalance the hierarchical tree. Another important scenario is when the leader or the group on the highest level of the hierarchy fails. This would cause more trouble since the leader of the highest level is responsible by the dynamic resource management intelligence of Heracles. However, during the update of the distributed table, the leaders can notice that the leader above them failed. After detecting the failure, another group takes place of the highest hierarchy level and the tasks of the group are rescheduled.

The hierarchical structure and the distributed table create a scenario to support the fault tolerance mechanism. Unless all the group leaders fail at the same time, the automatic reschedule of the tasks grants that all the workflow activities would be executed. This mechanism improves the quality of the scheduling and execution services of a huge cluster system.

4. Case study

Heracles is an approach that enables the scheduling of activities based on deadline and is resilient of churn occurrences. As an initial evaluation of Heracles working inside a MTC scheduling scenario, we present a simulation study as a proof of concept. Our purpose is to analyze the impact of churn events on tasks execution on clusters. In our study, a hypothetical cluster receives many workflow activities to be executed. Activities are decomposed into tasks that may suffer with churn events. We have evaluated scenarios of having workflow activities producing 512, 1024, 2048 and 4096 tasks. The tasks produced could be classified as small, medium and big. Smaller tasks run in an average time of 1 hour each, while large tasks run in an average time of 4 hours each. The study was evaluated on a time frame of seven days and considers that an average of 2000 activities was submitted on the cluster per day. The churn event frequency is 0.01, which means that one percent of the activities fail. On this scenario we have analyzed two different activity rescheduling approaches: the manual and the automatic.

The manual rescheduling approach assumes that scientists submits its activities and checks its status every twelve hours after the submission. If the activity is finished and present failures, it is manually rescheduled to run again all tasks from that particular

activity. The automatic approach automatically reschedules only the tasks that have failed, so that when an activity is finished, it is necessarily complete.

The first analyzed scenario considers activities that produce small tasks. These activities take from 3 hours, when they have 512 tasks, to 24 hours, when they have 4096 tasks, to execute. Figure 4 (a) shows the percentage of complete activities for manual (blue column) and automatic (red column) rescheduling. The graphics group the activities by their number of tasks. As expected, the results show that manual rescheduling do not scale well, even for activities with small tasks. This inefficiency is caused by the time that many activities wait to be rescheduled and to the fact that when any single task fails, the whole activity is resubmitted. A new submission is also prone to churn events. Automatic reschedule presents better results completing more activities and being also less sensitive to the increasing number of tasks.

Figure 4 (b) presents the results for activities that produce medium tasks. Medium tasks take from 6 hours, when they have 512 tasks, to 48 hours, when they have 4096 tasks, to execute. Figure 4 (c) represents activities with big tasks. Big tasks take from 12 hours, when they have 512 tasks, to 72 hours, when they have 4096 tasks, to execute. Both results reinforces our conclusions that manual reschedule do not scale. In the worst case scenario, the manual reschedule produces only about 10% of complete activities while the automatic approach produces more than 50%. It is also important to reinforce that the manual approach requires the effort of scientists to have the activity rescheduled in order to complete all tasks. Using the automatic approach, scientists only need to wait until the activity finishes. At the time, the activity is complete. In the results presented in Figure 4, the automatic approach does not present 100% of complete activities because the time frame of seven days is not enough to process all the tasks from the activities submitted on the last days.

Since the advantage of automatic rescheduling is clear when compared with the manual approach, we also made an extra study analyzing the automatic rescheduling sensitivity to the churn frequency. Figure 5 presents our results with the churn frequency varying from zero to two percent. This second study is also modeled on a time frame of seven days and 2000 activities are submitted by day. We used activities with 4096 medium tasks.

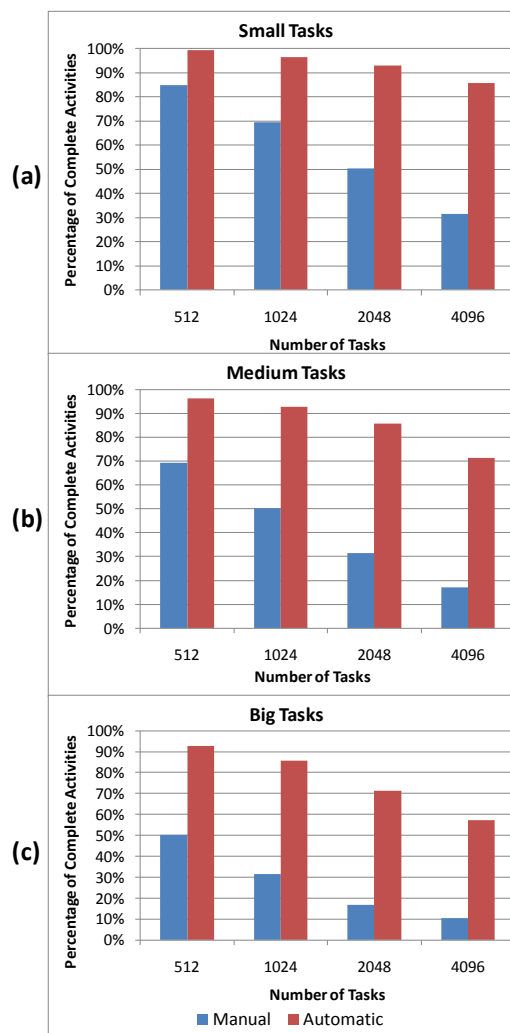


Figure 4: Percentage of complete activities using manual and automatic rescheduling per activity with (a) small task size, (b) medium tasks size and (c) big task size

The results presented in Figure 5 shows that the churn impact on the automatic rescheduling process is small. It is expected to have worse results with greater churn rates since more tasks need to run again. However, the percentage of complete activities decreases only in 0.5% while the churn frequency grows from 0.25% to 2%.

The study results reinforced that automatic rescheduling is an important feature on parallel workflow activity execution on clusters. Even with low churn rates, relying on manual rescheduling causes a great loss in the number of complete activities over time. Automatic rescheduling, though, scales much better. Since Heracles also aims at providing a transparent and fault tolerant mechanism to process activities, we believe that this proof of concept

strengthen our intentions to improve MTC scheduling and execution for scientific workflow activities.

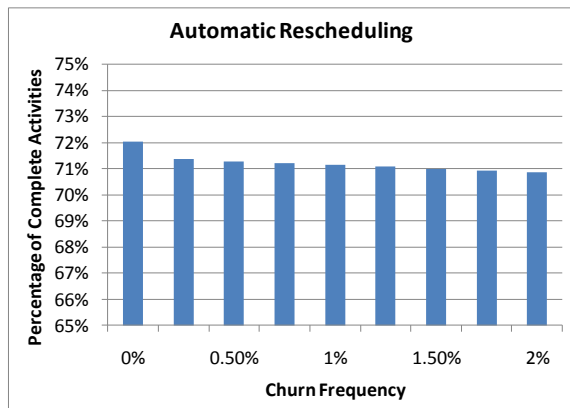


Figure 5: Automatic rescheduling sensitivity to different churn frequencies

5. Conclusions

High performance computing clusters have a vast set of computing nodes. They empower the development of large scale scientific experiments. These experiments explore different executions of scientific workflows; traditionally managed by SWfMS. Parallelizing workflow activities may produce a huge number of tasks demanding a many-task computing approach. However, scheduling and executing MTC activities on huge clusters may also suffer with churn events, poor load balancing and usability issues. To improve MTC scheduling and execution for scientific workflows, P2P techniques may help providing fault tolerant mechanisms and dynamic resource management.

We present the Heracles approach based on well known P2P techniques, such as distributed tables and hierarchical topologies, combined with traditional parallelism approaches for many-task computing, such as parameter sweep and data parallelism. Heracles aims to handle the execution of many tasks from scientific workflows on distributed resources using P2P techniques. Heracles can be used inside workflow schedulers. It takes the control over the tasks execution and gathers provenance data to report to the workflow scheduler. Heracles provides transparency, since scientists just need to inform a time constraint when they need their activities processed. It also enhances load balancing establishing a hierarchical P2P overlay on the cluster resources and a fault tolerant mechanism to be aware of churn events during the activities execution.

An initial proof of concept showed that churn events decreases the performance of activities

execution and manual rescheduling does not scale at all. Automatic rescheduling, though, performed better and proved itself not very sensitive to the churn frequency growth. Thus, we believe that Heracles approach is a necessary attempt to improve MTC in scientific workflows execution. However, Heracles does much more than a simple automatic rescheduling, so we believe that future work must analyze the advantages that MTC schedulers can achieve when using Heracles approach. Additionally, another fault tolerance approach to be evaluated in future versions of Heracles is to use redundant executions [33].

6. References

- [1] E. Deelman, D. Gannon, M. Shields, e I. Taylor, 2009, Workflows and e-Science: An overview of workflow system features and capabilities, *Future Generation Computer Systems*, v. 25, n. 5, p. 528-540.
- [2] D.A. Brown, P.R. Brady, A. Dietz, J. Cao, B. Johnson, e J. McNabb, 2007, "A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis", *Workflows for e-Science*, Springer, p. 39-59.
- [3] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, e M. Wilde, 2007, Swift: Fast, Reliable, Loosely Coupled Parallel Computation, In: *Services 2007*, p. 206, 199
- [4] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, e S. Mock, 2004, Kepler: an extensible system for design and execution of scientific workflows, In: *SSDBM*, p. 423-424, Greece.
- [5] S.P. Callahan, J. Freire, E. Santos, C.E. Scheidegger, C.T. Silva, e H.T. Vo, 2006, VisTrails: visualization meets data management, In: *Proc. SIGMOD 2006*, p. 745-747, USA.
- [6] J. Yu, R. Buyya, e C.K. Tham, 2005, Cost-Based Scheduling of Scientific Workflow Application on Utility Grids, In: *Proceedings of the First International Conference on e-Science and Grid Computing*, p. 140-147
- [7] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, e J. Good, 2008, On the use of cloud computing for scientific workflows, In: *IEEE Fourth International Conference on eScience (eScience 2008), Indianapolis, USA*, p. 7-12
- [8] W.D. Gropp, 2001, Learning from the Success of MPI, *cs/0109017* (Set.)
- [9] P. Chan e D. Abramson, 2008, A Programming Framework for Incremental Data Distribution in Iterative Applications, In: *Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, p. 244-251
- [10] W. Gropp e E. Lusk, 1995, Dynamic process management in an MPI setting, In: *Parallel and Distributed Processing, 1995. Proceedings.*

Seventh IEEE Symposium on Parallel and Distributed Processing, 1995. Proceedings. Seventh IEEE Symposium on, p. 530-533

- [11] I. Raicu, I. Foster, e Yong Zhao, 2008, Many-task computing for grids and supercomputers, In: *Workshop on Many-Task Computing on Grids and Supercomputers*, p. 1-11, Austin, Texas.
- [12] E. Ogasawara, D. Oliveira, F. Chirigati, C.E. Barbosa, R. Elias, V. Braganholo, A. Coutinho, e M. Mattoso, 2009, Exploring many task computing in scientific workflows, In: *MTAGS 09*, p. 1-10, Portland, Oregon.
- [13] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, e M. Wilde, 2007, Falkon: a Fast and Light-weight task executiON framework, In: *SC07*, p. 1-12, Reno, Nevada.
- [14] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, e K. Kennedy, 2005, Task scheduling strategies for workflow-based applications in grids, In: *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, p. 759-767 Vol. 2
- [15] S. Smachat, M. Indrawan, S. Ling, C. Enticott, e D. Abramson, 2009, Scheduling Multiple Parameter Sweep Workflow Instances on the Grid, In: *e-Science, 2009. e-Science '09. Fifth IEEE International Conference one-Science, 2009. e-Science '09. Fifth IEEE International Conference on*, p. 300-306
- [16] J. Dias e A. Avelada, 2010, HPC Environment Management: New Challenges in the Petaflop Era, In: *9th International Meeting on High Performance Computing for Computational Science*, Berkeley, CA, USA.
- [17] O. Sonmez, N. Yigitbasi, S. Abrishami, A. Iosup, e D. Epema, 2010, Performance Analysis of Dynamic Workflow Scheduling in Multicluster Grids, In: *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC'10)/HPDC*, p. 49-60, Chicago, Illinois, USA.
- [18] E. Pacitti, P. Valduriez, e M. Mattoso, 2007, Grid Data Management: Open Problems and New Issues, *Journal of Grid Computing*, v. 5, n. 3, p. 273-281.
- [19] D. Wu, Y. Tian, K. Ng, e A. Datta, 2008, Stochastic analysis of the interplay between object maintenance and churn, *Computer Communications*, v. 31, n. 2 (Fev.), p. 220-239.
- [20] E.K. Lua, J. Crowcroft, M. Pias, R. Sharma, e S. Lim, 2005, A Survey and Comparison of Peer-to-Peer Overlay Network Schemes, *IEEE Communications Surveys and Tutorials*, v. 7, p. 72-93.
- [21] V. Lo, D. Zhou, Y. Liu, C. GauthierDickey, e J. Li, 2005, Scalable Supernode Selection in Peer-to-Peer Overlay Networks, In: *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, p. 18-27
- [22] E. Ogasawara, J. Dias, D. Oliveira, C. Rodrigues, C. Pivotto, R. Antas, V. Braganholo, P. Valduriez, e M. Mattoso, 2010, A P2P Approach to Many Tasks Computing for Scientific Workflows, In: *9th International Meeting on High Performance Computing for Computational Science*, Berkeley, CA, USA.
- [23] G.H. Travassos e M.O. Barros, 2003, Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering, In: *Proc. of 2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering, Roma*
- [24] C. Team, 2005, *DAGMan: A Directed Acyclic Graph Manager*, July 2005.
- [25] E. Deelman, G. Mehta, G. Singh, M. Su, e K. Vahi, 2007, "Pegasus: Mapping Large-Scale Workflows to Distributed Resources", *Workflows for e-Science*, Springer, p. 376-394.
- [26] J. Freire, D. Koop, E. Santos, e C.T. Silva, 2008, Provenance for Computational Tasks: A Survey, *Computing in Science and Engineering*, v.10, n. 3, p. 11-21.
- [27] M. Mattoso, C. Werner, G.H. Travassos, V. Braganholo, L. Murta, E. Ogasawara, D. Oliveira, S.M.S.D. Cruz, e W. Martinho, 2010, Towards Supporting the Life Cycle of Large Scale Scientific Experiments, *IJBPIIM*, v. 5, n. 1, p. 79-92.
- [28] E. Walker e C. Guiang, 2007, Challenges in executing large parameter sweep studies across widely distributed computing environments, In: *Workshop on Challenges of large applications in distributed environments*, p. 11-18, Monterey, California, USA.
- [29] Yunhong Gu e R. Grossman, 2008, Exploring data parallelism and locality in wide area networks, In: *Workshop on Many-Task Computing on Grids and Supercomputers*, p. 1-10
- [30] D. Bradley, I. Sfiligoi, S. Padhi, J. Frey, e T. Tannenbaum, 2010, Scalability and interoperability within glideinWMS, *Journal of Physics: Conference Series*, v. 219, n. 6, p. 062036.
- [31] G. Juve, E. Deelman, K. Vahi, e G. Mehta, 2010, Experiences with resource provisioning for scientific workflows using Corral, *Sci. Program.*, v. 18, n. 2, p. 77-92.
- [32] R. Akbarinia, E. Pacitti, e P. Valduriez, 2007, Data currency in replicated DHTs, In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, p. 211-222, Beijing, China.
- [33] J. Dean e S. Ghemawat, 2008, MapReduce: simplified data processing on large clusters, *Commun. ACM*, v. 51, n. 1, p. 107-113.