Computing and Informatics, Vol. 33, 2014, 109–130

LOGOS: ENABLING LOCAL RESOURCE MANAGERS FOR THE EFFICIENT SUPPORT OF DATA-INTENSIVE WORKFLOWS WITHIN GRID SITES

David A. Monge

ITIC Research Institute
Universidad Nacional de Cuyo
ECT, Padre Jorge Contreras 1300
M5502JMA Mendoza, Argentina
e-mail: dmonge@itu.uncu.edu.ar

Carlos García Garino

ITIC Research Institute & Facultad de Ingeniería
Universidad Nacional de Cuyo
Centro Universitario
M5502JMA Mendoza, Argentina
e-mail: cgarcia@itu.uncu.edu.ar

Abstract. In this study we discuss how to enable grid sites for the support of data-intensive workflows. Usually, within grid sites, tasks and resources are administrated by local resource managers (LRMs). Many of LRMs have been designed for managing compute-intensive applications. Therefore, data-intensive workflow applications might not perform well on such environments due to the number and size of data transfers between tasks. To improve the performance of such kind of applications it is necessary to redefine the scheduling policies integrated on LRMs. This paper proposes a novel scheme for efficiently supporting data-intensive workflows in LRMs within grid sites. Such scheme is partially implemented in our grid middleware LOGOS and used to improve the performance of a well known LRM: HTCondor. The core of LOGOS is a novel communication-aware scheduling algorithm (PPSA) capable of finding near-optimal solutions. Experiments conducted in this study showed that our approach leads to performance improvements up to 52 % in the management of data-intensive workflow applications.

Keywords: Grid sites, data-intensive workflows, scheduling, HTCondor, DAGMan

1 INTRODUCTION

In the last years, several scientific areas based their experimentation and simulation processes using distributed computing technologies. In silico experiments became very popular extending their presence on many fields like astronomy, bioinformatics, earth sciences, high energy physics, health sciences and so on. Workflows were adopted in many of such research areas [1, 2, 3, 4] for guiding the experimental processes. The main reason for their acceptance is that workflows ease the design and management of applications. On the one hand, workflows permit the reuse of components (a.k.a. tasks) on different applications reducing the time and effort required during the development process. On the other hand, the identification of separate components facilitates the employment of distributed computing technologies for improving the global performance of applications. The latter possibility becomes crucial when the size of the applications is large.

Grid computing became a widely accepted technology for executing large-scale scientific applications [5, 6]. In recent times an increase in the amount of data to be processed by such type of applications has been observed. The growth of the volume of data to process (referred as *data deluge* in the literature) is affecting the way to do science, leading to the emergence of a new, *data-intensive science*, paradigm [2]. The actual efforts on data-intensive computing are focused on techniques for data streaming [7], data placement policies [8], quality of service requirements [9], and so on.

To ensure good performance of data-intensive scientific workflows it is required to develop data-aware management techniques not only focused on the Grid global level, but also within the existent grid sites. However, most of the application managers apply communication optimization techniques on an inter grid-site level disregarding the optimization of transfers within the sites.

In a grid site, executable tasks and resources are usually administrated by a local resource manager (LRM). Among the most known LRMs HTCondor [10], Torque [11] and GridEngine [12] can be cited. In general, LRMs apply load balancing and/or performance optimization policies, which only consider the impact of the computation workload of tasks. Such policies lead LRMs to perform well in the management of compute-intensive applications with loosely coupled or independent tasks; but, in the management of data-intensive applications where multiple dependencies are involved, LRMs might have a poor performance. Regardless of this limitation, LRMs are still very good for discovering resources, monitoring tasks, matchmaking tasks and resources, etc. Therefore, it is convenient to take advantage of these features and redefine the scheduling policies for supporting data-intensive applications efficiently.

The rest of this paper is organized as follows. In Section 2 we provide a review of the most salient works in the area of data-intensive computing in grid environments. In Section 3 we provide a review of the standard resource management policies within a well known LRM: HTCondor. In this section, we also describe the functioning of the DAGMan workflow manager. In Section 4 we describe our approach for improving the management of data-intensive workflows using HTCondor

DAGMan. In other words, we explain how to enable compute-intensive grid sites for efficient data-intensive computing. The details of the experimental environment are described in Section 5. Results are presented and analyzed in Section 6. Finally, conclusions and future work are given in Section 7.

2 RELATED WORK

There is a considerable amount of work in the area of data-intensive applications management in grid environments [5, 13, 14] as well as in the area of workflow based applications [15, 16]. In this section we provide a review of some strategies for supporting data-intensive applications in grid environments. We classify the related works into three categories: MapReduce applications, service-based workflows and component-based workflows.

MapReduce is programming model developed by Google [17] for processing of large-scale datasets across computer clusters. Such model consists of the definition of an application in terms of a pair of functions adopted from the functional programing paradigm: *map* and *reduce*. Apache Hadoop [18] is one of the most widespread implementations of the MapReduce model. There are also implementations for Grid, some of them are GridGain [19], an extension for the Kepler workflow system [20] and G-Hadoop [13]. MapReduce has proved to be an efficient programming model for data intensive applications. However, many scientific applications do not fit on this model and thus they cannot be expressed in terms of MapReduce. A second drawback of this model is that legacy systems are difficult to incorporate as part of the scientific application to develop. Therefore the developer is forced to design ad hoc mechanisms for such purposes.

The second approach for the development and execution of scientific applications is through the use of *service-based workflows*. Taverna [21] is a workflow engine part of the ^{my}Grid Project [22] for supporting experiments in the field of bioinformatics. Taverna is oriented to the design and execution of data-intensive and service-oriented workflows for grid. It provides a set of high level web services which can be used for the composition of workflows. There are some other initiatives for service-oriented scientific workflows [23]. However, in many research areas (e.g. those with high performance and scalability requirements to meet) it is not acceptable to express the applications in terms of services due to the overheads of the invocations. The second drawback is that in service workflows the data must be transfered to the resource where the code (i.e. services) is located. As the code cannot be moved and the data has to be transferred inevitably, the overall performance may suffer with data-intensive workflows.

Finally, the last approach to define and run scientific applications is using *component-based workflows*. This kind of workflows overcame some of the limitations of the above-mentioned approaches (e.g. it can include legacy components almost transparently, allow any structure of the tasks, can optimize the performance through code and data allocation, and so on).

Pegasus [24] is one of the most prominent workflow management systems and is part of the GriPhyN project [25]. Pegasus uses a partitioning and deferred planning scheme for the execution of large-scale workflows [26, 27]. It simplifies the structure of the application assuming that the generation of a data object is computationally more expensive than accessing to already processed results. Then, each partition is scheduled and submitted to DAGMan for execution. Among other workflow managers, ASKALON [28], or Swift [29] can be cited. In general, this kind of workflow managers focuses only on data optimizations at the Grid global level. Conversely, the work described on this paper focuses on data transfer optimizations within the grid sites.

Deelman [30] shows that there are a number of challenges associated to the management of scientific workflows. Among them, two important issues are the efficient scheduling of workflow tasks onto batch systems and the management of data transfers and storage. These are the two issues addressed by this study.

This paper proposes a data-aware near-optimal scheduling strategy for enabling LRMs in the management of data-intensive workflows in an efficient way. Unlike other work done in the area, we focus on optimizing the management of the data within the context of grid sites, rather than optimizations between grid sites. Such strategy is partially materialized in a new grid workflow system called LOcal Grid wOrkflow Scheduler (LOGOS). This work extends preliminary tests performed in previous studies of the authors [31, 32]. To compare our approach we selected the HTCondor DAGMan workflow manager that is used standalone in many grid settings and also as execution engine in other workflow systems such as Pegasus. Some recent applications can also be cited such as Bisque [4], a system that uses HTCondor for executing bioimage-analysis workflows.

3 RESOURCES MANAGEMENT WITH HTCONDOR DAGMAN

This section describes how resources and tasks are managed nowadays in a *de facto* standard LRM: the HTCondor middleware [10] (formerly Condor). It provides a high throughput computing (HTC) environment for the execution of applications in a distributed way. HTCondor defines a set of universes that make possible the management of different type of applications (e.g. scripts, binary executables, Java applications, MPI applications, etc.). HTCondor can also interact with the Globus Toolkit [33] for the submission of jobs to remote grid resources [34]. In addition, HTCondor is able to run workflow-based applications with tasks organized as nodes within a directed acyclic graph (DAG). The Directed Acyclic Graph Manager (DAGMan) [35] is a meta-scheduler capable of handling inter-task dependencies.

When a workflow is received, DAGMan submits the sub-tasks to HTCondor according to the precedence order defined in the DAG file. The process is as follows: DAGMan submits all the ready-to-execute tasks (i.e. tasks with all their predecessor tasks completed) to the HTCondor queue. Then, HTCondor schedules each task in the queue to the most adequate of the available resources. As tasks complete their

execution, DAGMan checks if there are more ready tasks and submits them to the HTCondor queue. This process continues until all the tasks in the DAG have been completed. This strategy is known as myopic scheduling and it is also implemented in other grid workflow managers like ASKALON [28].

There are two usual schemes for transferring data between HTCondor resources. The first one is using the Network File System (NFS). A typical deployment of NFS includes a central server that shares disk storage to a number of clients nodes. When running a workflow in DAGMan using the NFS scheme, tasks read the input data and write the data produced using a shared partition. Each read (write) operation performed by a client node implies the transfer of data from (to) the NFS server node. Therefore, if data transfers are not optimized, NFS may have communication overheads when the amounts of data to manage are big.

The second option for transferring data is using the scheme of *pre* and *post* scripts supported by DAGMan. In this case, the user is responsible for the data *stage-in* and *stage-out* operations for each task in the workflow. With this scheme, the data transfer optimizations are in charge of the user.

It is worth noting that, independently of the scheme used for data transfers, during the scheduling process the impact of communication times in the global performance is not considered at all. Such strategy may lead to resource assignments that bring with it transfers of large amount of data handicapping the total running time of the applications. This issue can be addressed by performing scheduling optimizations with emphasis on minimizing the communication times beside the computation times. In the next section we describe our strategy for addressing this concern.

4 ENABLING SITES FOR THE DATA GRID

In this section we present a workflow management strategy for enabling the support of data-intensive applications in LRMs. Such strategy is implemented in a new workflow engine: LOcal Grid wOrkflow Scheduler (LOGOS). LOGOS adopts the idea of workflow partitioning [26, 27] for execution within grid sites. In addition, it implements a near-optimal and communication-aware scheduling strategy designed to overcome the limitations of LRMs in the management of data-intensive workflows. Each site on the grid participates through a *grid site workflow manager* (GSWM) node, which serves as local agent for handling the execution of workflows. According to this scheme, it is possible to design cooperative management schemes based on the interaction of several GSWMs. Within a grid site, the GSWM node uses LOGOS for workflow scheduling, and the LRM for gathering resources information and executing the tasks. The general scheme for enabling grid sites for supporting data-intensive workflows is shown in Figure 1.

LOGOS has three components:

1. the scheduling algorithm,
2. the problem pre-processing component, and

3. the performance predictor.

The first component is the core of LOGOS, and it is represented by the performance prediction scheduling algorithm (PPSA). PPSA is a novel Branch and Bound scheduling algorithm which searches near optimal schedules. Section 4.1 describes the characteristics of this algorithm. The problem pre-processing (PP) component is in charge of preparing the scheduling problem for the scheduler. It includes features for workflow partitioning and resources selection, which are used to simplify the complexity of the original scheduling problem and therefore to accelerate the scheduling process. Regardless of the fact that the pre-processing of the problem is performed before the scheduling phase takes place, it is discussed later in Section 4.2 to clarify the explanation of both components (PPSA and PP). The last component is the performance estimation module (PEM). PEM provides the performance estimations that PPSA requires for constructing the task-resource mappings. The performance estimation process is explained in Section 4.3.

LOGOS works on the top of a LRM, which is in charge of retrieving information of resources availability, managing the tasks execution, monitoring tasks and resources, etc. In this study we use HTCondor DAGMan as the underlying platform for such purposes.

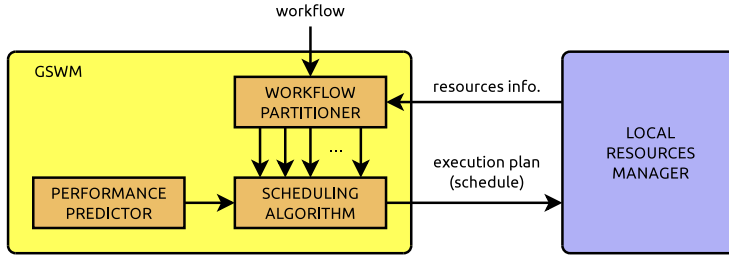


Figure 1. Block diagram of our scheme for scheduling data-intensive workflows on grid sites

4.1 Scheduling Strategy

This section presents the Performance Prediction Scheduling Algorithm (PPSA) integrated in LOGOS. PPSA belongs to the family of Branch and Bound algorithms. In general terms, it performs a tree-based search in the space of feasible schedules. This implies that the search is performed over all the possible combinations of task-resource mappings. As the optimal workflow-scheduling problem in grid is NP-complete, the explicit enumeration of all such combinations is impossible. As a consequence, PPSA is designed to explore portions of the search space only implicitly. This implicit exploration is performed by computing an estimated cost of the best solution that can be found within the current portion of the search space. Then, if the estimated cost is worse than the cost of the best schedule obtained

so far, the search process focuses on other portion of the space. Otherwise, if the estimation indicates that there is a possibility of finding good solutions, the search continues exploring such portion. Algorithm 1 shows the pseudocode of PPSA.

Algorithm 1 Pseudocode of PPSA

```
function PPSA(workflow, matches) returns a schedule
  inputs: workflow, a workflow DAG definition;
    matches, a list of matching resources per task in workflow;
  global: solution.mappings, the solution mappings;
    solution.cost, the estimated cost of solution;

  solution.mappings ← sequential solution on the best resource
  solution.cost ← cost of solution.mapping
  PPSA-Step({}, workflow, matches)
  return solution.mappings

procedure PPSA-Step(mappings, workflow, matches)
  inputs: mappings, a set of task-resource mappings; workflow; matches;
  global: solution.mappings; solution.cost;

  task ← select unmapped task
  resources ← order resources in matches[task]
  foreach resource in resources
    cost ← estimate workflow makespan
    if cost < solution.cost then
      register mapping (task,resource) in mappings
      if there are no more unmapped tasks in workflow then
        solution.mappings ← mappings
        solution.cost ← cost
      else
        PPSA-Step(mappings, workflow, resources)
      unregister (task,resource) from mappings
```

The PPSA function receives as inputs a description of the workflow to schedule and a list of the resources (*matches* list) able to run each task of the workflow. The algorithm starts by setting an initial solution and its related cost. We set as initial solution the serial schedule of the tasks in the fastest machine.

The algorithm has a recursive structure that explores the search space by expanding nodes of a tree in a depth-first order. Node expansions are performed through recursive calls of the PPSA-Step procedure. Each node is associated with a partial mapping of tasks and resources. Expanded nodes are evaluated to decide how the search process must continue.

For expanding a new node the algorithm selects an unmapped task (i.e. a task without resources assigned). The selection process is performed based on a policy

called Prior Critical Tasks (PCT). This policy prioritizes those tasks which are closer to the beginning of the workflow and have less lax time before delaying subsequent tasks. The selection process is performed by choosing to the unmapped task that has the highest priority value. The priority of a task i is computed according to:

$$priority_i^{PCT} = \frac{1}{start_i + slack_i}, \tag{1}$$

where $start_i$ is the start time of the task i, and $slack_i$ is the amount of time that the task i can be delayed without delaying any subsequent task. Tasks with a $slack = 0$ are *critical*. It means that such tasks cannot be delayed without delaying the finishing time of the workflow. It is worth noting that slack values and start times may change during the execution of PPSA because they are recomputed dynamically on each new generated mapping. Therefore, task priorities may also change between different scheduling steps.

Once a task is selected, it is necessary to set the evaluation order of the matched resources. The ordering is performed based on a policy named Maximum Capacity Resource (MCR). Such policy gives more priority to the most powerful resources. This ordering is established to evaluate first those mappings in which tasks finish earlier.

Once a new task mapping is selected, the partial schedule (node) is evaluated in terms of the estimated execution time of the entire workflow (hereafter estimated makespan). The makespan of a workflow is later finish time among all the tasks on it. Section 4.3 explains how makespan is computed. The estimated makespan of the node is used in combination with the cost of the best solution obtained so far to guide the search. If the estimated makespan exceeds the cost of the best solution, the search on such branch is abandoned because it will not lead to a better solution. This process in which portions of the search space are discarded without previous exploration is known as pruning. This characteristic is one of the most important points of PPSA, because it accelerates the search process. However, in some cases, the running time of PPSA may be long, attempting against the overall management time. In the next section we explain how to deal with this issue.

4.2 Algorithm and Problem Complexity

In general, Branch and Bound algorithms present exponential time complexity. In the case of PPSA, the general time complexity is given by $O(r^n)$, where r is the number of resources and n is the number of workflow sub-tasks. The exponential growth imposes a limit on the size of problems that PPSA can address before becoming computationally expensive (i.e. before the algorithm running time becomes longer than the gain obtained in the execution stage).

For preventing the downfall of general performance due to long running times of PPSA, we apply some strategies for reducing the complexity of the scheduling problem. The key for addressing this issue lies on the reduction of the number of

tasks and the number of resources. Our strategy consists of finding the solutions for a set of lower complexity problems instead of addressing one with high complexity.

The complexity reduction process is performed by the problem pre-processing module before the scheduling algorithm is invoked. The pre-processing module applies 3 actions:

1. partitioning of workflows,
2. resources selection, and
3. generation of pre-assignments:

- *Partitioning of workflows* is important to generate a set of sub-workflows of manageable size. This action is performed only when the workflow to manage is large. There is a two-fold reason for workflow partitioning. In the first place, workflows are partitioned to reduce the complexity o the scheduling problem as mentioned in the above paragraph. In the second place, by partitioning the workflow it is possible to alternate the scheduling and execution phases of the resulting sub-workflows, making the management process aware of the dynamism of the resources [24]. It is worth mentioning that the actual version of LOGOS does not implement any partitioning method; but we are considering the use of the Newman-Girvan algorithm for graph clustering [36].
- For each (sub-)workflow, a set of adequate resources is selected. The *resources selection* process consists on the definition of the adequate number of computers to consider in the scheduling of a given workflow. The proper size of the resources set is such that allows the execution of the maximum number of parallel tasks of the workflow. Within this stage only the resources that are compatible with the tasks are considered.
- Finally, the *generation of pre-assignments* consists on the reduction of the number of resources for a subset of the workflow tasks. For all the critical tasks in the workflow, the set of candidate machines is reduced to the most powerful ones. The generation of such pre-assignments reduces the number of task-resource mappings to be considered during scheduling, this way improving the running time of PPSA.

After applying the 3 actions introduced above, LOGOS has converted a high complexity problem into one or more sub-problems of lower complexity suitable for the characteristics of PPSA. It is worth mentioning that sub-workflows resulting from the partitioning stage could be managed in parallel on different grid sites if it is convenient.

4.3 Performance Estimation

As mentioned in Section 4.1, PPSA computes the workflow makespan for each partial or complete resources mapping to determine how the search process will continue.

As a consequence, PPSA requires accurate performance information for making proper resources selection. The performance estimation module (PEM) provides such information to PPSA.

The workflow makespan is computed by considering the duration of tasks and of data transfers. For the experiments in this paper, we use an analytical performance model that computes

1. the execution time of a task in a resource, and
2. the transfer time of data between two different tasks.

The model computes the execution time according to:

$$T_{ex}(i, r) = \kappa_i / PC(r), \tag{2}$$

where κ_i is the computation workload of the task i, and $PC(r)$ is the processing capacity of the resource r. Measurement units of κ_i and $PC(r)$ must be compatible (e.g. million of instructions and MIPS).

Transfer time is computed according to:

$$T_{tx}(i, j, r^i, r^j) = \begin{cases} \mu_{ij} / TR(r^i, r^j) & \text{if } r^i \neq r^j \\ 0 & \text{if } r^i = r^j \end{cases}, \tag{3}$$

where μ_{ij} is the size of the data to transfer from task i to task j, and $TR(r^i, r^j)$ is the available transfer rate between the resources r^i and r^j (which execute tasks i and j, respectively). The transfer time is assumed to be 0 when both tasks are mapped to the same resource.

Equations (2) and (3) can be applied only when the resources where tasks will be executed have been already selected. However, during the computation of workflow makespan, some tasks may not be assigned to resources. As a consequence, the processing capacities (PC) and transfer rates (TR) cannot be determined T_{ex} or T_{tx} either. In such cases the algorithm computes estimated values of the execution time ($\bar{T}_{ex}$) and the transfer time ($\bar{T}_{tx}$).

The estimated execution time of an unmapped task i is computed as follows:

$$\bar{T}_{ex}(i) = \kappa_i / \bar{PC}_i, \tag{4}$$

where $\bar{PC}_i$ is the average processing capacity of the resources compatible with such task, and κ_i is the computation workload of the task i.

The estimated transfer time of a message from an unmapped task i to an unmapped task j is computed as follows:

$$\bar{T}_{tx}(i, j) = \alpha \times \mu_{ij} / \bar{TR}_{ij}, \tag{5}$$

where $\bar{TR}_{ij}$ is the average transfer rate considering the resources compatible with tasks i and j, and α is the ratio between the number of tasks mappings that involve

different resources and the total number of them (i.e. the probability of mapping different resources for tasks i and j).

It is worth noting that this strategy computes average execution and transfer time estimations whereby the computed workflow makespan of a partial resources assignment might be over estimated. According to the literature this type of heuristics is non-admissible. The use of a non-admissible heuristics converts the optimization problem into an approximation one, and then the algorithm cannot ensure the optimal solution, but as a positive side effect, the running time of PPSA is considerably improved due to pruning.

5 EXPERIMENTAL ENVIRONMENT

This section describes the characteristics of the experiment conducted as part of this study. The experiment consists on measuring the performance of the workflow management schemes presented on this paper: the standalone HTCondor DAGMan (hereafter DAGMan), and the one that implements LOGOS in combination with DAGMan (hereafter LOGOS). In Section 5.1 the characteristics of a set of testing workflows are described. In Section 5.2 a description of the grid site is given. Then, in Section 5.3 the metrics for the evaluation of the results are explained.

5.1 Workflow Applications Description

For measuring the effectiveness of both schemes, we designed a set of workflows, which emulate the characteristics of real ones. Each workflow is represented by a directed acyclic graph $W = (\Gamma, \Delta)$, where Γ is the set of tasks, and Δ is the set of data dependencies between them.

Workflows used in this study comprise synthetic tasks that simulate real workload. Each task receives a single argument that indicates the number of operations to perform. Such operations consist of a set of numerical calculations. Tasks also produce output data used as inputs of subsequent child tasks. Such data do not contain any intelligible information but are adequate for the simulation of real transfers. Figure 2 presents a sample workflow.

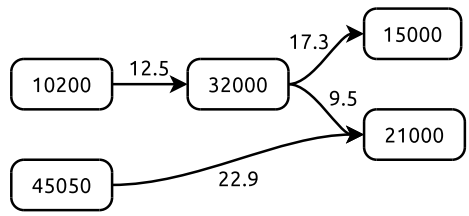


Figure 2. Workflow with 5 tasks and 4 data dependencies. Rectangles represent tasks indicating the number of operations to perform and edges represent data transfers indicating the sizes in MB

Because LOGOS still does not count with any partitioning method, workflows were generated with a proper size for their management. Therefore, it can be assumed that such workflows are the result of some partitioning method. Workflows were randomly generated according to the set of parameters explained below:

Number of tasks, n: the number of tasks comprised in the workflow. It defines its *size*.

Density factor, δ: it is a ratio that represents the relation between the number of dependencies in a workflow and their maximum possible number. Formally, it is computed as $\delta = d/d_n^{\max}$, where d is the number of dependencies of the workflow, and $d_n^{\max}$ is the number of edges of a complete DAG of size n (i.e. the maximum number of dependencies of a workflow with n tasks). The latter value is computed as $d_n^{\max} = \frac{n \times (n-1)}{2}$.

During the generation of a workflow of size n, the δ parameter is used to compute the number of data dependencies d. According to n and δ, the number of data dependencies d in the workflow is computed as $d = d_n^{\max} \times \delta$.

The δ parameter is closely related with the degree of parallelism of the workflow. Higher values of δ produce workflows with more inter-task dependencies and tasks arranged in more linear structures. Higher values for δ also produce applications with more data to transfer. On the other hand, lower values of δ produce more parallel applications with less data to transfer.

Computation workload, κ: the computation workload is an integer parameter that describes the number of synthetic operations of a task.

Message size, μ: the μ parameter stands for the size of the messages to transfer between workflow tasks. Each message is generated with a μ value within the range defined in Table 1. The size of messages is measured in MB.

Table 1 summarizes the values of the parameters explained above used in workflow generation. Such values were selected to produce workflows with a structure of dependencies similar to those which can be found on the Standard Task Graph Set (STG) workflow repository [37].

Generation parameter	Value
Number of tasks (n)	$5, 10, \ldots, 50$
Density factor (δ)	$0.4, 0.6, 0.8$
Computation workload (κ)	$[10\,000, 60\,000]$
Message size (μ)	$[9.5, 28.6]$

Table 1. Set of parameters used for the generation of the 150 testing workflows

To analyze the characteristics of the generated workflows, we use the communication to computation ratio (CCR). The CCR is a typical metric used for describing parallel and distributed applications. This metric represents the proportion between the communication and the computation load of an application. In the

case of a workflow with n tasks and d dependencies, the CCR is computed as follows:

$$CCR = \frac{\sum_{i=1}^{d} \mu_i / \bar{TR}}{\sum_{j=1}^{n} \kappa_j / \bar{PC}}, \tag{6}$$

where μ_i is the size of the message that corresponds to the dependency i of the workflow, $\bar{TR}$ is the average transfer rate of the infrastructure used, κ_j represents the computation workload of the task j of the workflow, and $\bar{PC}$ stands for the average processing capacity of the infrastructure measured in number of synthetic operations per second. The numerator of this quotient represents the total transfer time of the messages on the workflow; and, the denominator stands for the total execution time of all the tasks. Note that the values of CCR are not only dependent on the characteristics of the workflow, but also on the underlying infrastructure.

5.2 Grid Site Description

The grid site used in this study comprises resources from 3 different types. There is a central node serving as Grid Site Workflow Manager (GSWM). The GSWM node uses LOGOS for the scheduling of workflows and HTCondor DAGMan for resources management. The size of the set of computing resources was set to 6, which is the maximum number of parallel tasks in the set of generated workflows (determined by the resources selection process). The resources were selected for keeping the heterogeneity within the grid site. The characteristics of the computing infrastructure are described in Table 2.

The information presented in individual columns is: resource type, processor type, memory, and processing capacity (PC). The values for PC of each node are the result of a benchmark process. This process consists of measurement of the average number of synthetic operations per second on each resource. The interconnection network has a nominal transfer rate of 1 Gbps. These values of PC and the nominal transfer rate are used as input for the performance model presented in Section 4.3.

Resource type	Processor	Memory	PC [op./s]
GSWM node*	Intel Core2 Duo 3.0 GHz	4 GB	-
Twister	Intel Core2 Duo 3.0 GHz	4 GB	1 784.18
Reloaded	Intel P4 HT 3.0 GHz	1 GB	598.61
Reloaded	Intel P4 HT 3.0 GHz	1 GB	598.95
Reloaded	Intel P4 HT 3.0 GHz	1 GB	598.87
Opteron	AMD Opteron 242 1.6 GHz	2 GB	995.16
Opteron	AMD Opteron 242 1.6 GHz	2 GB	1 079.47

*node with HTCondor DAGMan and LOGOS.

Table 2. Grid site computing infrastructure

5.3 Evaluation Metrics

To evaluate the performance of both execution schemes, we used the total execution time and the speedup metrics explained below:

- In the case of the LOGOS scheme, the total execution time (T^{LOGOS}) is the sum of the running time of 1. the scheduling algorithm, and 2. the effective execution time (i.e. the time required to manage the application) of a workflow using HTCondor DAGMan. In the case of the standalone DAGMan the total execution time (T^{DAGMan}) is equal to the effective execution time of the workflow.
- For comparing the performance of both approaches, we use the speedup metric. It is defined as the ratio of the total execution times obtained through the standalone DAGMan and the LOGOS strategies. The speedup S_i obtained in the execution of a workflow i, is computed according to:

$$S_i = \frac{T_i^{DAGMan}}{T_i^{LOGOS}}, \tag{7}$$

  where T_i^{DAGMan} is the total execution time of the workflow using DAGMan standalone and T_i^{LOGOS} is the total execution time of the workflow using PPSA for scheduling. A speedup value S can be also expressed as the percentage of improvement $S_{\%}$ computed as $S_{\%} = 100\% \times (1 - 1/S)$.

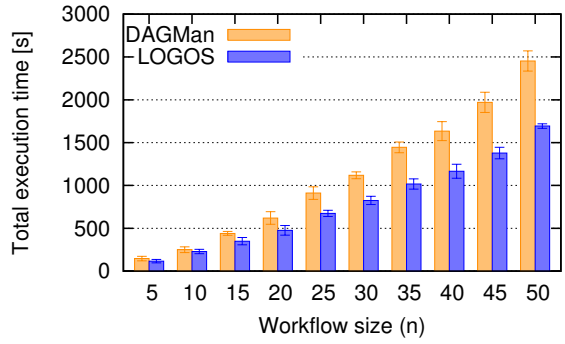
6 EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we present the results obtained from the performed experiment. 150 different workflows were executed using the scheduling schemes described in Sections 3 and 4, DAGMan and LOGOS, respectively. Results are presented and discussed separately for both metrics used.

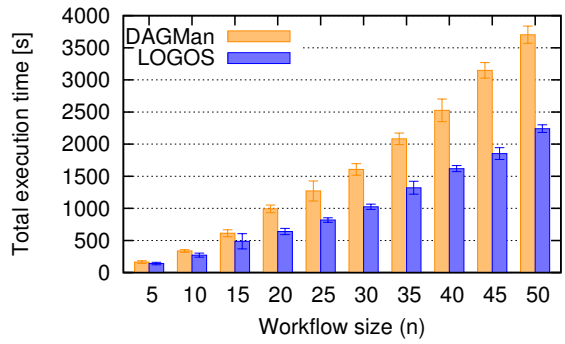
6.1 Total Execution Time

Figures 3 a), 3 b) and 3 c) present the total execution time obtained for workflows with density factors of 0.4, 0.6 and 0.8, respectively. Bars show the average execution time and the standard deviation for sets of workflows with the same number of tasks. It can be seen that the results show very small standard deviations. The low variability on these results indicates that both schemes present a predictable behavior.

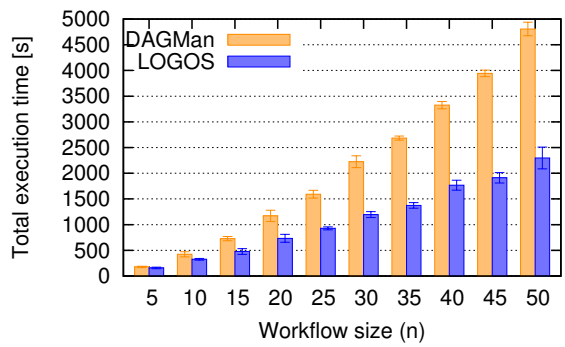
It can be observed from the analysis that LOGOS overcomes the DAGMan scheme in terms of total execution time. These improvements are achieved because of two factors. In the first place, PPSA (i.e. the scheduling algorithm of LOGOS) considers the cost of transfers during scheduling, conversely to its counterpart which only considers the cost of computation. In the second place, PPSA performs a search



a)



b)



c)

Figure 3. Total execution time for different workflow densities (less is better): workflows with density factor a) $\delta = 0.4$; b) $\delta = 0.6$; c) $\delta = 0.8$

of near-optimal schedules. Conversely, HTCondor DAGMan performs an opportunistic scheduling of tasks which considers only one of them at a time. These two aspects of the myopic scheme favor the bad performance on the execution of data-intensive workflows. In contrast, PPSA is free of these limitations and is able to achieve better performance on such kind of applications.

6.2 Speedups

Figure 4 a) presents average speedups for workflows with the same size and density. Results are grouped according to the values of the density factor δ for each workflow. It can be seen that the minimum speedups obtained were for workflows comprising 5 and 10 tasks. For such set of workflows, it is also noticeable that there is a small difference between speedups obtained for the different density factors. For workflows larger than 15–20 tasks the difference of speedup among workflow densities becomes more evident. Maximum speedups were obtained for the largest workflows (45–50 tasks). For such workflows, the difference of speedups among densities is the biggest.

Table 3 summarizes the minimum, maximum and average percentual speedups per workflow density. Minimum percentual speedups correspond to workflows with a number of tasks between 5 and 10. Maximum percentual speedups correspond to workflows with a number of tasks between 45 and 50.

Density	$S_{\%}^{\min}$	$S_{\%}^{\max}$	$S_{\%}^{\text{avg}}$
0.4	8.7 %	31.0 %	24.5 %
0.6	14.6 %	41.2 %	31.5 %
0.8	11.3 %	52.2 %	39.3 %

Table 3. Percentual speedups for each workflow density

In general, it can be observed that higher speedups are obtained when workflows have larger size. Such behavior is caused by the existence of more improvement possibilities in larger size workflows. PPSA explores a spectrum of solutions and therefore can get better schedules than those acquired by the myopic scheme which progressively constructs the final schedule.

It can also be observed that better speedups were obtained for workflows that have higher density factors. In general, higher density workflows present tasks arranged in quasi-linear structures. For such kind of workflows, the optimal schedule is similar to the sequential execution of the tasks in the fastest resource; and, due to the fact that the myopic algorithm is prone to map dependent tasks onto different resources, it forces unnecessary data transfers. Such kind of selection has very negative effects when critical tasks are affected because makespan is delayed inevitably. This effect is less important in the management of more parallel workflows because critical paths comprise a smaller number of tasks.

Figure 4 b) shows the relation between the CCR of workflows and the speedup obtained for each one. It can be seen from the regression analysis that better

speedups are obtained for workflows with larger values of CCR (i.e. larger amounts of data to process). This behavior corresponds to the characteristics of the two scheduling policies applied. In the case of the myopic scheme (DAGMan), transfer times are not considered during the selection of the resources. Therefore, when the number of data dependencies is increased (and thus the value of CCR increases), the performance of HTCondor DAGMan decreases. Conversely, PPSA performs data transfer optimizations and as a consequence it is able to find better schedules as the amount of data grows. Said in other words, when more data is involved in the workflow, PPSA performs well but the performance of DAGMan decays resulting in an increase of the speedup values computed.

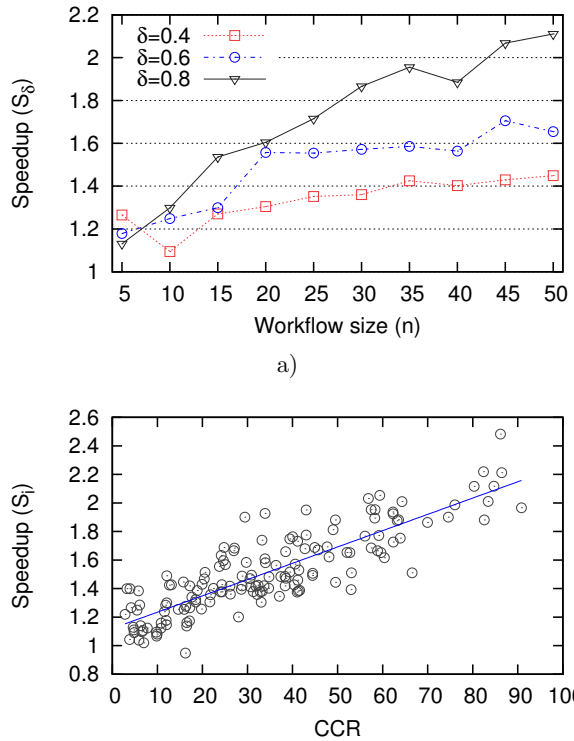


Figure 4. Speedup analysis: a) average speedups obtained for different workflow densities; b) regression between the communication to computation ratio (CCR) and the speedup computed for each workflow

7 CONCLUDING REMARKS AND DISCUSSION

In this paper we proposed a scheme for enabling local resource managers for supporting data-intensive workflows in grid sites. Such scheme is implemented in our middleware for workflow management called LOGOS. We also presented a novel scheduling algorithm which considers tasks execution and transfer times. Tests were performed using HTCondor DAGMan for resources management, which is a widely used middleware in actual grid sites and workflow management systems. It was proven that our approach can achieve improvements on the running time that range from 8.7 % in the case of the smallest data-intensive workflows, and to 52.2 % for the largest ones. It was also determined that our algorithm is capable of finding better schedules for workflow applications where the communication workload predominates over the computation workload.

Despite of the good results obtained in this study, it is worth noting that given the computational complexity of the proposed scheduling algorithm (PPSA), in some cases its running time might deteriorate the overall performance. To tackle this issue it becomes critical to use a proper workflow partitioning method in order to reduce the complexity of the addressed problems.

Our future work includes incorporation and analysis of such adequate partitioning method for managing large-size workflow applications. We are planning the comparison of PPSA with other state-of-the-art scheduling strategies in the context of data grids. The Pegasus workflow system seems to be one of the most suitable candidates for such comparison. In particular, we aim to adapt PPSA for working with Pegasus and to compare it with the other algorithms included on it. In addition, the performance of LOGOS in the management of an actual data-intensive workflow application will be studied in the future.

Acknowledgments

This research is supported by the National Agency for Scientific and Technological Promotion (ANPCyT) through project No. PAE-PICT 2312. This study is also funded by the Argentine Ministry of Science and Technology (MINCyT), and the Czech Ministry of Education, Youth and Sports (MEYS), project Nos. RC0904 and MEB111005. The financial support provided by the SeCTyP-UNCuyo through projects No. 06B/194 and No. 06B/253 is gratefully acknowledged. The first author wants to thank the National Scientific and Technological Research Council (CONICET) for the granted fellowship. Finally, the authors want to thank the anonymous reviewers for their valuable comments and suggestions that helped to improve the quality of this paper.

REFERENCES

[1] Taylor, I. J.—Deelman, E.—Gannon, D. B.—Shields, M.: Workflows for e-Science: Scientific Workflows for Grids. Springer London, 1st edition, December 2007.

[2] Hey, T.—Tansley, S.—Tolle, K.: The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Research, Redmond, Washington, October 2009.

[3] Šimo, B.—Habala, O.—Gatial, E.—Hluchý, L.: Leveraging Interactivity and MPI for Environmental Applications. Computing and Informatics, Vol. 27, 2008, No. 2, pp. 271–284.

[4] Kvilekval, K.—Fedorov, D.—Gaur, U.—Goff, S.—Merchant, N.—Manjunath, B. S.—Singh, A.: Bisque: Advances in Bioimage Databases. IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 35, 2012, No. 3, pp. 56–64.

[5] Furano, F.: Large Databases on the GRID. Nuclear Instruments and Methods in Physics Research, Vol. A623, 2010, pp. 855–858.

[6] Kussul, N.—Shelestov, A.—Skakun, S.—Kravchenko, O.—Gripich, Y.—Hluchý, L.—Kopp, P.—Lupian, E.: The Data Fusion Grid Infrastructure: Project Objectives and Achievements. Computing and Informatics, Vol. 29, 2010, No. 2, pp. 319–334.

[7] Zhang, W.—Cao, J.—Zhong, Y.—Liu, L.—Wu, C.: Grid Resource Management and Scheduling for Data Streaming Applications. Computing and Informatics, Vol. 29, 2010, No. 6+, pp. 1193–1220.

[8] Amer, M. A.—Chervenak, A.—Chen, W.: Improving Scientific Workflow Performance Using Policy Based Data Placement. In Proceedings of the 13th IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2012), Chapel Hill, NC, USA, 2012, pp. 86–93.

[9] Słota, R.—Nikolow, D.—Skałkowski, K.—Kitowski, J.: Management of Data Access with Quality of Service in PL-Grid Environment. Computing and Informatics, Vol. 31, 2012, No. 2, pp. 463–479.

[10] Thain, D.—Tannenbaum, T.—Livny, M.: Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience, Vol. 17, 2005, pp. 323–356.

[11] Torque Resource Manager: Available on: `http://www.adaptivecomputing.com/products/open-source/torque/`.

[12] Oracle Grid Engine: Available on: `http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html`.

[13] Wang, L.—Tao, J.—Ranjan, R.—Marten, H.—Streit, A.—Chen, J.—Chen, D.: G-Hadoop: MapReduce Across Distributed Data Centers For Data-Intensive Computing. Future Generation Computer Systems (to appear).

[14] Skillicorn, D.—Talia, D.: Mining Large Data Sets on Grids: Issues and Prospects. Computing and Informatics, Vol. 21, 2002, No. 4, pp. 347–362.

[15] Deelman, E.—Chervenak, A.: Data Management Challenges of Data-Intensive Scientific Workflows. In Proceedings of the 2008 8th IEEE International Symposium on Cluster Computing and the Grid, Washington, DC, USA, May 2008, pp. 687–692.

[16] Kumar, V. S.—Kurc, T.—Ratnakar, V.—Kim, J.—Mehta, G.—Vahi, K.—Nelson, Y. L.—Sadayappan, P.—Deelman, E.—Gil, Y.—Hall, M.—Saltz, J.: Parameterized Specification, Configuration and Execution of Data-Intensive Scientific Workflows. Cluster Computing, Vol. 13, 2010, No. 3, pp. 315–333.

[17] Dean, J.—Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, Vol. 51, 2008, No. 1, pp. 107–113.

[18] Apache Hadoop: Available on: `http://hadoop.apache.org/`.

[19] Ivanov, N.—Setrakyan, D.: Real Time Big Data Processing with GridGain. Electronic book, 2012. Available on: `http://www.gridgain.com/book/book.html`.

[20] Wang, J.—Crawl, D.—Altintas, I.: Kepler + Hadoop: A General Architecture Facilitating Data-Intensive Applications in Scientific Workflow Systems. In Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, WORKS '09, New York, NY, USA, 2009, pp. 12:1–12:8.

[21] Missier, P.—Soiland-Reyes, S.—Owen, S.—Tan, W.—Nenadic, A.—Dunlop, I.—Williams, A.—Oinn, T.—Goble, C. A.: Taverna, Reloaded. In Michael Gertz and Bertram Ludscher, editors, Scientific and Statistical Database Management, Vol. LNCS 6187 Springer-Verlag, 2010, pp. 471–481.

[22] ^{my}Grid Project: Available on: `http://www.mygrid.org.uk/`.

[23] Slominski, A.: Adapting BPEL to Scientific Workflows. In Taylor, I. J., Deelman, E., Gannon, D. and Shields, M. editors, Workflows for e-Science: Scientific Workflows for Grids. Springer London, 1st edition, December 2007.

[24] Deelman, E.—Singh, G.—Su, M.-H.—Blythe, J.—Gil, Y.—Kesselman, C.—Mehta, G.—Vahi, K.—Berriman, G. B.—Good, J.—Laity, A.—Jacob, J. C.—Katz, D. S.: Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming Journal, Vol. 13, 2005, No. 3, pp. 219–237.

[25] GriPhyN: Available on: `http://griphyn.org/`.

[26] Lee, K.—Paton, N. W.—Sakellariou, R.—Deelman, E.—Fernandes, A. A. A.—Mehta, G.: Adaptive Workflow Processing and Execution in Pegasus. In Proceedings of the 2008 3rd International Conference on Grid and Pervasive Computing – Workshops, Washington, DC, USA, 2008, pp. 99–106.

[27] Chen, W.—Deelman, E.: Partitioning and Scheduling Workflows Across Multiple Sites with Storage Constraints. In Proceedings of the 9th international conference on Parallel Processing and Applied Mathematics (PPAM '11), Vol. LNCS 7204 Springer-Verlag, 2012, pp. 11–20.

[28] Fahringer, T.—Jugravu, A.—Pllana, S.—Prodan, R.—Seragiotto, C.—Truong, HL.: ASKALON: A Tool Set for Cluster and Grid Computing. Concurrency and Computation: Practice and Experience, Vol. 17, 2005, No. 2–4, pp. 143–169.

[29] Wilde, M.—Hategan, M.—Wozniak, J. M.–Clifford, B.—Katz, D. S.—Foster, I.: Swift: A Language for Distributed Parallel Scripting. Parallel Computing, Vol. 37, No. 9, 2011, pp. 633–652.

[30] Deelman, E.: Grids and Clouds: Making Workflow Applications Work in Heterogeneous Distributed Environments. International Journal of High Performance Computing Applications, Vol. 24, 2010, No. 3, pp. 284–298.

[31] Monge, D. A.—García Garino, C.: Improving Workflows Execution on DAGMan by a Performance-driven Scheduling Tool. In R. Orozco and A. Fernández, editors, Proceedings of the 3rd Symposium on High-Performance Computing in Latin America, 39 JAIIO, Vol. 3, 2010, pp. 3271–3285. Available on: `http://www.39jaiio.org.ar/sites/default/files/39jaiio-hpc-09.pdf`.

[32] Monge, D. A.—Bělohradský, J.—García Garino, C.—Zelezný, F.: A Performance Prediction Module for Workflow Scheduling. In A. R. Mendarozqueta, M. Marciszack and M. A. Groppo, editors, Proceedings of the 4th Symposium on High-Performance Computing in Latin America, 40 JAIIO, Vol. 4, 2011, pp. 130–144.

[33] Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems. Journal of Computer Science and Technology, Vol. 3779, 2006, pp. 2–13.

[34] Frey, J.—Tannenbaum, T.—Livny, M.—Foster, I.—Tuecke, S.: Condor-G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing, Vol. 5, 2002, pp. 237–246.

[35] Couvares, P.—Kosar, T.—Roy, A.—Weber, J.—Wenger, K.: Workflow Management in Condor. In Ian J. Taylor, Deelman E., Dennis B. Gannon, and Matthew Shields, editors, Workflows for e-Science: Scientific Workflows for Grids, Springer London, 2007, chapter 1, pp. 357–375.

[36] Newman, M. E. J.—Girvan, M.: Finding and Evaluating Community Structure in Networks. Physical Review E, Vol. 69, No. 5, 2004, pp. 026113.

[37] Standard Task Graph Set: Kasahara Laboratory, Dept. of Electrical, Electronics and Computer Engineering, Waseda University. Available on: `http://www.kasahara.elec.waseda.ac.jp/schedule/index.html`.

David A. Monge received the engineering degree in information systems from Universidad Tecnológica Nacional, Mendoza, Argentina, in 2007 and received the Ph. D. degree in Computer Science at UNICEN, Tandil, Buenos Aires, Argentina in 2013. He is member of the Information and Communication Technologies Institute (ITIC), Universidad Nacional de Cuyo, Argentina. His research interests include workflow applications, grid, clouds, performance prediction and data mining.

Carlos García Garino graduated in engineering at University of Buenos Aires, Argentina in 1978 and received a Ph. D. degree from Universidad Politécnica de Cataluña, Barcelona, Spain in 1993. Currently he is Full Professor at the School of Engineering and Head of the ITIC Research Institute, Universidad Nacional de Cuyo, Argentina. His research interests include computational mechanics, computer networks, and distributed computing. He has more than 50 papers published in scientific journals and proceedings of international conferences carried out in Argentina, Brazil, Chile, Canada, Spain, France, Portugal, Belgium and Japan.