Performance Analysis of Dynamic Workflow Scheduling in Multicluster Grids

Ozan Sonmez
Delft University of Technology
Delft, The Netherlands
o.o.sonmez@tudelft.nl

Nezih Yigitbasi
Delft University of Technology
Delft, The Netherlands
m.n.yigitbasi@tudelft.nl

Saeid Abrishami
Ferdowsi University
Mashhad, Iran
s-abrishami@ferdowsi.um.ac.ir

Alexandru Iosup
Delft University of Technology
Delft, The Netherlands
a.iosup@tudelft.nl

Dick Epema
Delft University of Technology
Delft, The Netherlands
d.h.j.epema@tudelft.nl

ABSTRACT

Scientists increasingly rely on the execution of workflows in grids to obtain results from complex mixtures of applications. However, the inherently dynamic nature of grid workflow scheduling, stemming from the unavailability of scheduling information and from resource contention among the (multiple) workflows and the non-workflow system load, may lead to poor or unpredictable performance. In this paper we present a comprehensive and realistic investigation of the performance of a wide range of dynamic workflow scheduling policies in multicluster grids. We first introduce a taxonomy of grid workflow scheduling policies that is based on the amount of dynamic information used in the scheduling process, and map to this taxonomy seven such policies across the full spectrum of information use. Then, we analyze the performance of these scheduling policies through simulations and experiments in a real multicluster grid. We find that there is no single grid workflow scheduling policy with good performance across all the investigated scenarios. We also find from our real system experiments that with demanding workloads, the limitations of the head-nodes of the grid clusters may lead to performance loss not expected from the simulation results. We show that task throttling, that is, limiting the per-workflow number of tasks dispatched to the system, prevents the head-nodes from becoming overloaded while largely preserving performance, at least for communication-intensive workflows.

1. INTRODUCTION

For convenience and cost-related reasons, scientists execute scientific workflows [4, 13] in distributed large-scale computational environments such as multicluster grids, that is, grids comprising multiple independent clusters. However, executing scientific workflows in grids is a dynamic process that raises numerous challenges. One of the most important of these is scheduling with incomplete or dynamic information about the workflows and the resource availability—the runtimes of, and the amounts of data transferred between workflow tasks may be unknown a priori, and the other grid users may impose a background load on the grid resources. It is the purpose of this paper to present a comprehensive and realistic investigation of the performance of a wide range of dynamic workflow policies in multicluster grids.

Multicluster grids are complex systems that are difficult to model in all their details, and many different workload scenarios may occur in practice. This means that on the one hand, experiments in real systems are called for, but on the other hand, as real experiments are often difficult to perform or repeat, simulations are needed—in this paper, we do both. For our experiments, we use the Dutch DAS-3 multicluster system equipped with our KOALA grid scheduler [28], which we have extended with several of the workflow scheduling policies presented in this paper. Our investigation considers various realistic, dynamic scenarios, such as scheduling small or large, communication- or computation-intensive, and single or multiple workflows.

A large body of work on workflow scheduling already exists. However, much of this work focuses on parallel computing environments [23], which are very different from grids, or considers static scheduling methods in which all tasks of a workflow are mapped to resources before its execution starts [11, 13, 27]. Recent research [24, 25, 42, 43] does address adaptive approaches in which scheduling decisions are revised at runtime, but it assumes the availability of perfectly accurate information about the workflow tasks and the system resources, which is not true for dynamic systems such as grids (see also our previous work [18, 32] for a thorough discussion of this topic). Moreover, few research results have been obtained for the realistic situations in which multiple workflows are submitted simultaneously to the grid, and when there exists contention for the system resources caused by non-workflow (background) system load. Finally, previous investigations have largely been based on either simulations or on real system experiments, but not on both.

Our main contribution is threefold:

1. We present a framework for dynamic workflow scheduling that includes a novel taxonomy of dynamic workflow scheduling policies based on the amount of (dynamic) information used (Section 2). We further map to this taxonomy seven workflow scheduling policies that cover the full spectrum of dynamic information use.

2. We investigate the performance of these seven policies in various realistic and dynamic scenarios using both simulations and real systems (Sections 4 and 5). One of the findings from our real-system experiments that does not show in the simulations is that the performance may be severely degraded because the head-nodes of the grid clusters may become overloaded due to the large number of workflow tasks and file transfers they have to manage.

3. To solve the problem of head-node overload, we analyze the performance of task throttling, that is, of limiting the per-workflow number of tasks concurrently present in the system. Our results indicate that task throttling can prevent head-node overload while not unduly decreasing the performance.

2. THE SCHEDULING FRAMEWORK

In this section we present a framework for dynamic workflow scheduling in multicluster grids.

2.1 Workflow Model

We assume a workflow to be represented by a directed acyclic graph (DAG), in which the nodes represent computational tasks and the directed edges represent communication. We call a task having no predecessor tasks an entry task, and a task having no successor tasks an exit task; a DAG may have several entry and exit tasks. We define the size of a workflow as the total number of its tasks.

We assume that a task depends on each of its predecessors by means of a file. An output file of a task can be the input file to several successor tasks. A task can be executed only after all its predecessor tasks are completed and all the corresponding files are available on its execution site.

2.2 Multicluster Grid Model

In our multicluster grid model, we assume that processors are grouped in clusters. The processors may have different performance across clusters, but within the same cluster they are homogeneous. Clusters are fully connected with network links that can be heterogeneous in terms of bandwidth. Each cluster has its own Local Resource Manager (LRM), and so its own local queue to which tasks arrive. Each LRM in the system executes the same scheduling procedure upon the arrival of new tasks and on the completion of tasks.

We consider a decentralized architecture for workflow scheduling in which each workflow submitted to the system is managed and scheduled by an individual broker/agent. This broker incorporates a workflow execution engine and employs a scheduling policy in order to map tasks to resources. The workflow engine submits tasks to the LRMs of the clusters according to some schedule, and initiates the necessary file transfers. Finally, we assume a monitoring service provides information about the status of the clusters (e.g., the numbers of idle resources and the loads).

Table 1: Mapping scheduling policies to our information availability framework. **U**, **K**, and **R** stand for information that is **U**nknown (or ignored), **K**nown a priori, and obtained at **R**untime by a policy, respectively.

Policy	Resource Info.			Task Info.	
	Status	Proc. Speed	Link Speed	Task Exec. Time	File Sizes
Round Robin	U	U	U	U	U
Single Cluster	R	U	U	U	U
All-Clusters	R	U	U	U	U
All-Cls. File-Aware	R	U	K	U	R
Coarsening	R	U	K	U	K
Cluster Min.	R	K	U	U	U
HEFT(-P)	R	K	K	K	K

2.3 A Taxonomy of Workflow Scheduling Policies

In this section we first propose a taxonomy for workflow scheduling policies based on the information they consider regarding resources and workflow tasks. We then map to our taxonomy seven dynamic workflow scheduling policies in most of which tasks are assigned to resources only after they become *eligible*, i.e., after all of their predecessor tasks are finished.

In our taxonomy, a particular piece of information regarding resources or workflow tasks can be either unknown (U), known a priori (K), or obtained at runtime (R). We consider the following pieces of information: for resources, their status, processing speed, and inter-cluster link speeds, and for tasks, their execution time and the sizes of their output files. Our taxonomy for workflow scheduling policies extends our previous taxonomy for bags-of-tasks scheduling policies [18] by considering more detailed resource and task information, and it differs significantly from previous efforts [23,41], since their taxonomies classify policies based on algorithmic approaches rather than on the information the policies take into account.

We consider seven dynamic workflow scheduling policies, and in Table 1 we show how these policies map to our taxonomy. We describe these policies below in the order of increasing information usage:

1. Round Robin [10]: submits the eligible tasks of a workflow to the system clusters in round-robin order; the order of clusters is arbitrary.

2. Single Cluster: maps every complete workflow to the least-loaded cluster at its submission. The load of a cluster is defined as the total processor requirement of all jobs running or queued in the cluster normalized by the cluster size. This policy executes all tasks of a workflow in the same cluster in order to avoid inter-cluster file transfers. However, this policy may increase the makespan of a workflow when the number of eligible tasks, at any moment during its execution, is larger than the number of idle processors in the cluster.

3. All-Clusters: submits each eligible task to the least-loaded cluster. This policy can exploit idle resources across the grid; however, the performance may degrade due to inter-cluster file transfers.

4. All-Clusters File-Aware: submits each eligible task to the cluster that minimizes the transfer costs of the files on which it depends. The size of an output file can be known only after the task that creates it is completed. This policy gives preference to the clusters with idle processors.

5. Coarsening: is, in fact, a technique used in mul-

tilevel graph partitioning problems [22, 26] that iteratively reduces the size of a graph by collapsing groups of nodes and their internal edges. In each iteration, a group of nodes of the current graph is selected and is combined into a single coarser node. The edges of the original graph between nodes in the group disappear, but the edges that connect the corresponding coarse nodes remain in the new graph [21]. We use the Heavy Edge Matching (HEM) [21] coarsening technique to group tasks that are connected with heavy edges, i.e., task dependencies that correspond to relatively large files in a workflow. After coarsening, the remaining edges are conceivably be the task dependencies that correspond to relatively small files. Our approach is to execute all the tasks of a group in the same cluster (where the first task of that group is submitted to) with the aim to minimize the cost of inter-cluster file transfers. After a workflow is coarsened, it is scheduled with the All-Clusters File-Aware policy.

6. Cluster Minimization: submits as many eligible tasks as possible to a cluster until the cluster has no idle processors left before considering the next cluster. Therefore, it aims to reduce the number of inter-cluster file transfers by minimizing the number of clusters being used. It considers the clusters in descending order of their processing speeds, and hence, it gives preference to faster resources. If none of the clusters have idle processors, then the tasks are submitted to the cluster where the last task has been scheduled.

7. Heterogeneous Earliest-Finish-Time (HEFT) [36]: is a commonly cited list-scheduling heuristic [10,38,39]. This policy initially orders all the tasks of a worklflow in descending order of their *upward rank* values. The upward rank of a task is calculated as the sum of the execution time and the communication time (on a reference processor and with a reference bandwidth) of the tasks that are on this task's critical path, including itself. Then in this order the policy maps each task to a processor which ensures its earliest completion time.

In this work, we have modified the original HEFT policy such that it maps tasks to clusters instead of processors, hence it operates at the grid level, and we have changed the static task scheduling process to dynamic scheduling. Our implementation of the HEFT policy operates as follows. Among the eligible tasks, at each step, it selects the task with the highest upward rank value and assigns the selected task to the cluster that ensures its earliest completion time. We assume that an information service runs on each cluster that responds to queries regarding the estimated start times of workflow tasks. Then the completion time of a workflow task on a cluster is estimated as the sum of the estimated start time, the estimated delay of transferring input files on which it depends, to that cluster, and the execution time of the task on that cluster.

Although we assume the execution times of the workflow tasks are known a priori, we consider two kinds of prediction information regarding the execution times of non-workflow tasks (i.e., background load due to local users; see Section 3.3 for details), leading to two variations of this policy. The **HEFT** policy makes use of perfectly accurate task completion time predictions. In contrast, **HEFT-P** is provided task completion time predictions that may be inaccurate; the execution times of the non-workflow tasks submitted to a cluster are predicted using the Last-2 method [37], which predicts the execution time of a task as the average of the

last two previously observed task execution times. We refer to our previous work [32] for details of this prediction scheme. The prediction service simulates the scheduling policy of the LRM with the predicted task execution times and the actual execution times of the workflow tasks that have been submitted to the cluster in order to determine when a new workflow task will start. Although perfectly accurate estimations of task information is not realistic in grid settings [32], we use the HEFT policy in order to observe the performance that can be achieved if such information were available.

2.4 Task Throttling

In grids, overload conditions may occur due to the bursty nature of task arrivals, as in the case of executing large workflows. As a consequence, the execution performance of the applications may deteriorate to unacceptable levels; response times may grow significantly, and throughput may degrade. Furthermore, as we will demonstrate in Section 5.2, the real system may become unstable when executing large workflows, as the load on the head-nodes of the clusters severely increases due to the activity of the workflow execution engine [34], to the number of concurrent task submissions [6, 34], and to the excessive number of concurrent inter-cluster file transfers.

A common way to achieve efficient overload control in traditional distributed systems is to use admission control mechanisms [7,20]. In grids, however, instead of dropping or rejecting tasks, it may be sufficient to delay the submission of some of the tasks, in the expectation that the sites which to dispatch them will become less overloaded at a later time. For instance, the Condor system [35] allows [1] the system administrator to limit the total number of concurrent tasks in the system. We call this mechanism, *task throttling*, and apply it on a per-workflow basis. We define the *concurrency limit* as the maximum number of concurrent tasks that are dispatched (they can be running or queued) in a multicluster grid to all its clusters and at all times during the execution of the workflow.

Task throttling may have non-trivial effects on scheduling performance. For example, although with task throttling tasks may be delayed by the workflow execution engine, this may also lead to a decrease in the amount of inter-cluster communication, since there will be fewer concurrent tasks, and they will possibly be spread to fewer clusters. Consequently, the execution performance of the workflow may improve despite the additional delay of the tasks.

3. EXPERIMENTAL SETUP

In this section, we present the experimental setup used both in the simulations and the real experiments.

3.1 Experimental Environments

Our goal is to emulate, both in simulations and in practice, realistic scenarios in which we execute workflows on the DAS-3 [9], which is a multicluster grid system in the Netherlands. It consists of five clusters, and comprises 272 dual-processor AMD Opteron compute nodes. The properties of the clusters are shown in Table 2. Table 3 shows the average inter-cluster bandwidth (MB/s) as measured in real

---

Table 2: Properties of the DAS-3 clusters.

Location	Nodes	Speed [GHz]
Vrije University	85	2.4
U. of Amsterdam	41	2.2
Delft University	68	2.4
MultimediaN	46	2.4
Leiden University	32	2.6

Table 3: The inter-cluster link speeds (in MB/s) between the nodes of the DAS-3 clusters.

| Clusters	Vrije	Amsterdam	Delft	Multi.N	Leiden
Vrije	-	185	45	185	77
Amsterdam	185	-	53	512	115
Delft	45	53	-	10	53
Multi.N	185	512	10	-	115
Leiden	77	115	53	115	-

Figure 1: The overall utilization as well as the utilization in the individual clusters due to the background load considered in the simulation-based experiments. This load refers to the jobs submitted to the DAS-3 system during a period of three months (June-September 2008).

conditions (see [31] for details). On each cluster, the Sun Grid Engine (SGE) [14] operates as the local resource manager. SGE has been configured to run applications on the nodes in an exclusive fashion, i.e., in space-shared mode. As the storage facility, there is a separate distributed file system available on each of the clusters.

3.1.1 Simulated Environment

For our simulations, we have extended DGSim [17], which is our discrete event simulator for grids and for other large-scale distributed computing environments, with workflow execution capability.

In our *computation model*, we employ the SPEC CPU benchmarks model for task execution time [33], that is, the time it takes to finish a task is inversely proportional to the performance of the processor it runs on. We assume that all LRMs of the clusters employ the FCFS policy. Once started, tasks run to completion, so we do not consider task preemption or task migration during execution.

In our *communication model*, we assume that the file transfer delay of sending a file from a predecessor to a successor task is zero if these two tasks are executed on the same cluster. If a task depends on a file created on a different cluster, we model the file transfer delay as the ratio of the file size and the bandwidth of the inter-cluster link. If a task depends on multiple files, then we model the total file transfer delay as the sum of the individual file transfer delays. This means that we consider the worst case scenario in which all file transfers are performed in series, although concurrent inter-cluster file transfers can perform better in reality [1]. Once a file is created in a cluster or transfered from another cluster, it remains in place until the whole workflow is executed; hence, we avoid redundant file transfers.

3.1.2 Real Environment

For our experiments in the DAS-3, we have extended our grid scheduler KOALA [28] with workflow execution support. To this end, we have designed and implemented two components: a workflow runner (WRunner), which can be considered as a workflow execution engine, and a workflow execution service (WES), which is responsible for interacting with the grid middleware. A separate WRunner is responsible for each workflow submission; it manages the dependencies between the tasks and the scheduling of workflow tasks. In our
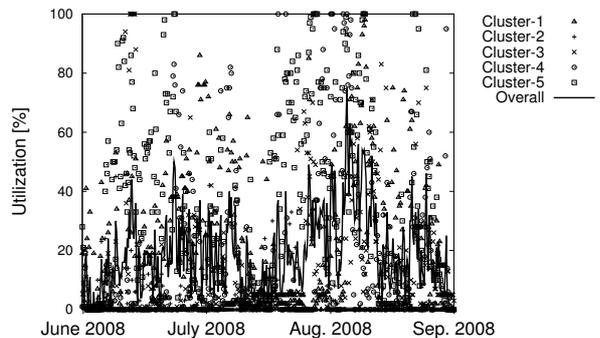
design, we use a service-based approach for better scalability. Hence, there is a single WES on the head-node of each of the clusters that initiates the execution of all workflow tasks scheduled to that cluster.

After registration to the scheduler, a WRunner obtains system-wide status information from the KOALA Information Service (KIS), and determines the execution sites of the tasks according to the scheduling policy it employs. For each task of the workflow, the WRunner copies input files and its executable to the execution site, and then delegates the execution of the task to the WES on this site. The WES is responsible for submitting the task to the middleware using the DRMAA [12] interface, and for monitoring its execution with the callbacks provided by the middleware. The WES provides the task status to WRunner upon its completion. If the task fails, the WRunner handles the failure by resubmitting the task. If the task finishes successfully, upon notification, the WRunner first transfers its output files to the submission site of the workflow, and then it updates the set of eligible tasks. The WRunner continues this process until all tasks of the workflow are completed.

3.1.3 Additional Considerations

Although we try to model the DAS-3 multicluster grid as realistically as possible, in our simulations there are some differences between our model and the real system:

- In our simulations, we do not model the head-nodes and the resource contention that may occur on the head-nodes in the real system due to the file transfers. This resource contention and the instability it causes has a significant effect on the performance; as we demonstrate in Section 5.2.

- In our simulations, we consider a task as finished when its execution completes; then, the eligible set of tasks is updated accordingly. But in the real experiments, we only consider a task as finished when its execution has completed and its output files have been transferred back to the submission site; and only then the eligible set of tasks is updated.

3.2 The Workflows

We use synthetic but realistic workflow applications that are publicly available in [4]. The applications are based on

Table 4: Categorization of the workflows.

Workload Type	Workflow Name-Size	Avg. Makespan on a Reference Cluster [s]	Avg. Total Size of the Output Files of a Workflow [MB]
wf-small1	CyberShake-30, 50,100	220	2246
wf-small2	Inspiral-30,50 Montage-100	1260	23
wf-large1	CyberShake-1000 Montage-1000	410	2866
wf-large2	SIPHT-1000 Inspiral-1000	3290	1150

four real scientific workflows: Montage, CyberShake, Inspiral, and SIPHT. These workflows are composed of several structural components such as pipeline, data distribution, data aggregation, and data redistribution. For each of these workflows, four synthetic applications are available[2], with sizes of 30, 50, 100, and 1000 tasks.

We categorize a synthetic workflow as *small*, if its size is at most 100, and as *large*, if its size is 1000. With small workflows, the number of tasks that become eligible concurrently can all fit in a single cluster in our system; however, with large workflows, more tasks may become eligible concurrently than can be accommodated by even the largest cluster. We further categorize the workflows according to their communication versus computation characteristics. Table 4 presents the categorization of the workflows and their characteristics. While the workflows in *wf-small1* and *wf-large1* are communication-intensive, the workflows in *wf-small2* and *wf-large2* are more computation-intensive.

3.3   The Workloads

In our experiments, we assume that two workloads are submitted to the system: a grid workload, which comprises the workflow applications submitted by grid users, and a local workload, which comprises the tasks submitted directly to the clusters by local users (background load). In our simulations, depending on the experimental scenario, we impose a background load together with a grid workload in order to attain realistic resource availability conditions. The background load refers to the jobs submitted to the DAS-3 system during a period of three months (June-September 2008). Figure 1 illustrates the system utilization of the background load. The corresponding workload trace is obtained from the Grid Workloads Archive [16]. In the simulations, the tasks that belong to the background load are submitted to the LRMs of their original execution locations. In our DAS-3 system, users may submit tasks directly to the LRMs, bypassing KOALA. We keep this non-workflow (background) load under control for performing controlled experiments in the real environment. To this end, during our real system experiments, we monitor the background load, and we maintain it between 30% and 40% in each cluster (which is not the case in our simulations).

We classify our experiments as either single workflow or multi-workflow scheduling. In the simulations of single workflow scheduling with background load, for each policy, each of the workflows is scheduled only once and the average results are presented per workload type (see Table 4). In the simulations with the background load, for each workload

type, we generate ten traces in each of which randomly selected workflow instances arrive at six-hour intervals during a period of three (simulated) months. For each policy-workload type pair, we run the ten corresponding experiments and present the average results. In the real system experiments (where there is always background load) of single workflow scheduling, for each policy, each of the considered workflows is executed ten times and we present the average results.

For multi-workflow scheduling, both for simulations and real experiments, several instances of the same workflow application are submitted simultaneously (the exact number varies across the experiments). We do not consider background load in the simulations of multi-workflow scheduling.

3.4   The Performance Metrics

To assess the performance of the workflow scheduling policies, we use the following traditional metrics [23]:

- The **Makespan (MS)** of a workflow in a grid is the time elapsed from its submission to the grid until the completion of its last task. For the multi-workflow scheduling experiments, in which a number of concurrent instances of a workflow are submitted to the grid, we consider the metric **Total Makespan**, which is the time elapsed from the submission of the instances until all the instances are completed.
- The **Normalized Schedule Length (NSL)** of a workflow in a grid is the ratio between its makespan and the time to execute (one of) its critical path(s).
- The **Wait Time** of a task is the time elapsed from when a task becomes eligible until the time it starts execution. It comprises two components: the File Transfer Delay **(FTD)**, which is due to waiting for input files to become available at the execution site, and the Queue Wait Time **(QWT)**, which is the time a task spends in the local queue of the cluster to which it was submitted.

In addition to these metrics, we also consider the Number of inter-cluster File Transfers per workflow **(NFT)**.

4.   SIMULATION RESULTS

In this section we present our simulation results of single workflow scheduling and multi-workflow scheduling, respectively.

4.1   Single Workflow Scheduling

We first evaluate the performance of the scheduling policies that we describe in Section 2.3 under various experimental scenarios. Then, we investigate the impact of task throttling on the performance of dynamic workflow scheduling when executing large workflows.

4.1.1   The Performance of the Scheduling Policies

In addition to our experimental setup (in Section 3), we applied the Coarsening policy only to wf-large1 and wf-large-2; the workflow sizes are shrunk to 100 and then scheduled with the File-Aware policy. For wf-small1 and wf-small2, Coarsening is exactly the same as the File-Aware policy. In addition, we applied the HEFT-P policy only when we impose background load in our experiments.

Figure 2 presents the performance of the workflow scheduling policies in terms of the average makespan and the average NSL, without and with background load (denoted by
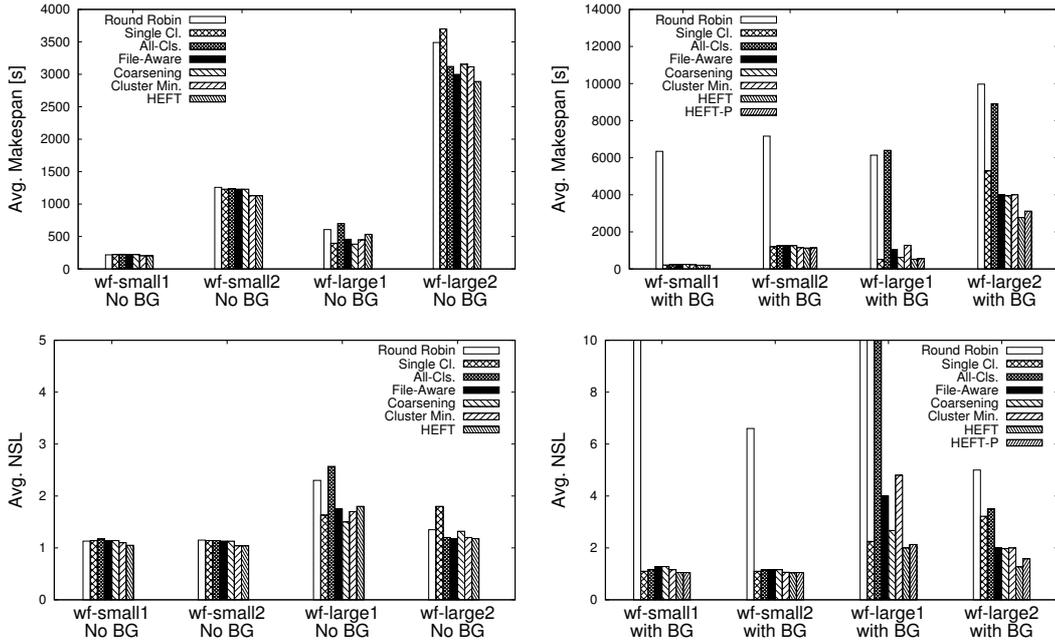
Figure 2: **Simulations, single workflow scheduling:** The performance of the workflow scheduling policies in terms of the average makespan and the average NSL without and with background load.
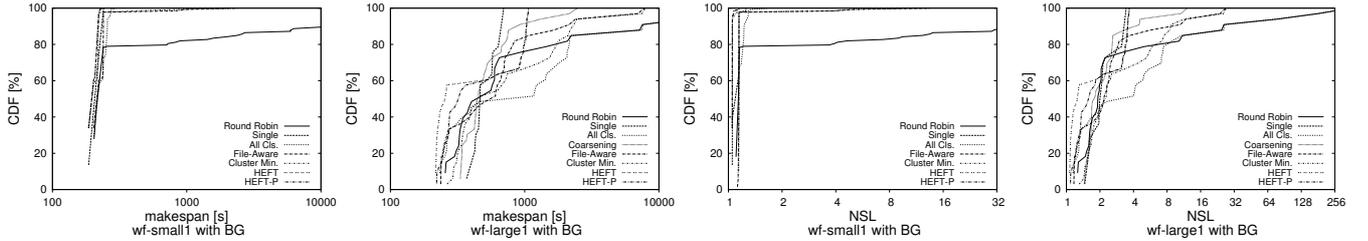

Figure 3: **Simulations, single workflow scheduling:** The cumulative distribution functions of the makespan and the NSL with background load. The horizontal axis has a logarithmic scale.

BG). Table 5 presents additional metrics such as the average task queue wait time, the average task file transfer delay, and the average number of inter-cluster file transfers performed per workflow. Finally, Figure 3 shows the cumulative distribution functions of the makespan and the NSL when background load is imposed (only for wf-small1 and wf-large1). Below, we present a separate discussion for each of the experimental scenarios.

wf-small, without BG Load

The very first noticeable result is that the performance variation is small among the policies. The reason is that, with any of the policies, except Round Robin, tasks are executed mostly in a single cluster. When the system is not loaded, for small workflows, selecting the appropriate cluster (e.g., according to processing speed) is a good strategy to attain a better performance. Since the Cluster Minimization policy takes into account the processing speed of the clusters, it is slightly better than the other policies, and yields almost identical performance to that of HEFT, which is fed by perfectly accurate resource and task information,

and which we expect to perform best in any of the scenarios.

wf-small, with BG Load

In this scenario we see that Round Robin performs much worse in comparison to the other policies. Round Robin maps tasks also to the most loaded clusters, and consequently, some tasks experience large queue wait times. As this policy distributes tasks across all clusters, it causes more inter-cluster file transfers than the other policies (see Table 5). The other policies perform similarly to each other, although Cluster Minimization, HEFT and HEFT-P yield slightly better performance.

wf-large, without BG Load

When executing large workflows, many tasks may become eligible simultaneously, which gives more room to make different scheduling decisions. Consequently, when scheduling large workflows, we observe that the performance varies considerably among the policies, and even their relative performance varies with different workloads (Figure 2). For the case of wf-large1, Round Robin and All-Clusters are the two

Table 5: **Simulations, single workflow scheduling:** The performance of the policies without and with background load, in terms of the average task queue wait time (QWT [s]), the average task file transfer delay (FTD [s]), and the average number of inter-cluster file transfers performed per workflow (NFT). '-' indicates that no experiment has been performed (see text).

Workload Type	Round Robin			Single Cl.			All-Cls.			File Aware		
	QWT	FTD	NFT	QWT	FTD	NFT	QWT	FTD	NFT	QWT	FTD	NFT
wf-small1	0	1.26	25	0	0	0	0	0.4	4	0	0	0
wf-small2	0	0.5	49	0	0	0	0	0.1	10	0	0	0
wf-large1	8	1.1	460	49	0	0	3.2	1.44	366	5	0.43	243
wf-large2	7.5	0.4	778	93	0	0	0.4	0.5	637	2	0.3	354
wf-small1+BG	525	1.23	26	0.03	0	0	0	1.2	6.7	3.3	0.01	0.1
wf-small2+BG	423	0.5	51	0.3	0	0	0.5	0.1	12	3	0.01	1.1
wf-large1+BG	682	1.2	455	113	0	0	573	3.1	392	42	0.4	229
wf-large2+BG	842	0.42	756	386	0	0	598	0.5	636	79	0.4	422

Workload Type	Coarsening			Cluster Min.			HEFT			HEFT-P		
	QWT	FTD	NFT	QWT	FTD	NFT	QWT	FTD	NFT	QWT	FTD	NFT
wf-small1	-	-	-	0	0	0	0	0	0	-	-	-
wf-small2	-	-	-	0	0	0	0	0.2	15	-	-	-
wf-large1	34	0.18	39	3.3	1.3	495	4	0.6	218	-	-	-
wf-large2	9.9	0.01	18	0.4	0.5	638	1.25	0.2	301	-	-	-
wf-small1+BG	-	-	-	1.7	0.1	1.2	0	0	0	0	0	0
wf-small2+BG	-	-	-	3.2	0.01	1.05	0	0.2	15	0.4	0.2	15
wf-large1+BG	93	0.3	58	42	1.4	466	9	0.5	290	9	0.5	290
wf-large2+BG	69	0.04	25.6	146	0.68	694	7.5	0.4	441	28	0.4	501

worst performing policies, and Single Cluster and Coarsening are the two best performing policies. On the other hand, for the case of wf-large2, Single Cluster performs the worst, while HEFT performs the best. We attribute this difference to the communication characteristics of the workloads (see Table 4).

In general, we observe that both Round Robin, All-Clusters, and Cluster Minimization perform many inter-cluster file transfers (see Table 5). Coarsening is a trade-off between low inter-cluster communication and high queue wait times. Although File-Aware is better than Cluster Minimization in terms of reducing inter-cluster communication, selecting faster resources pays off for Cluster Minimization even when the heterogeneity in the system is low (the ratio of the fastest processor to the slowest is 1.25 in our settings). Nevertheless, selecting the faster clusters may not be always an advantage if such clusters are much smaller than a slower cluster when scheduling large workflows, since in such a case tasks may be distributed to more clusters, and as a result the number of inter-cluster communications may increase.

wf-large, with BG Load

When background load is imposed, we observe a substantial increase in the average makespans and NSLs, respectively. The HEFT policy outperforms the other policies for both of the workloads. However, the performance of the HEFT policy worsens when the task completion time information is inaccurate (HEFT-P). File-Aware, Coarsening, and Cluster-Minimization all yield similar performance results when scheduling wf-large2; however, Coarsening is much better than the others when scheduling wf-large1, which includes more communication-intensive workflows. As a consequence of spreading tasks to many clusters, both Round Robin and All-Clusters suffer from file transfer delays as well as from large queue wait times.

4.1.2 The Impact of Task Throttling

In this section we evaluate the scheduling performance of the dynamic workflow scheduling policies when *task throt-*

Table 6: **Simulations, single workflow scheduling:** Percentage of change (Best, Worst) in the performance of File-Aware and Cluster-Minimization, when **task throttling** is applied, relative to the original performance of the policies.

Policy (Workload Type)	Change in the Makespan [%]		Change in the NSL [%]	
without BG	Best	Worst	Best	Worst
File-Aware (wf-large1)	-15	+16	-9	+25
File-Aware (wf-large2)	+17	+60	+18	+116
Cluster-Min. (wf-large1)	-4	+50	-20	+9
Cluster-Min. (wf-large2)	+11	+68	+16	+123
with BG				
File-Aware (wf-large1)	-50	-37	-40	-26
File-Aware (wf-large2)	+4	+51	+6.5	+80
Cluster-Min (wf-large1)	-30	-11	-27	-13
Cluster-Min (wf-large2)	+4	+43	+15	+75

tling is applied for each workflow submitted to the system.

To conduct our evaluation we modify our default setup as follows. We only consider the workloads that contain large workflows, and run them both with and without background load. As scheduling policies we select, from the policies that perform relatively well in the previous section, the File-Aware and the Cluster-Minimization policies. In our simulations, we do not model overload conditions for the head-nodes of the clusters; hence, we assess the impact of throttling on scheduling performance in ideal conditions. We use, in turn, three values for the concurrency limit: 50, 100, and 150. While these values are, as we show below, enough to understand the main impact of task throttling on the performance of scheduling, it is outside the scope of this work to assess the optimal concurrency limit.

Figure 4 shows the makespan performance of the policies for all values of the concurrency limit, as well as when no task throttling is applied. Table 6 presents the best and the worst performance results of the policies relative to their original performance out of all scenarios. We observe that the performance that can be attained with task throttling is related both to the communication character-
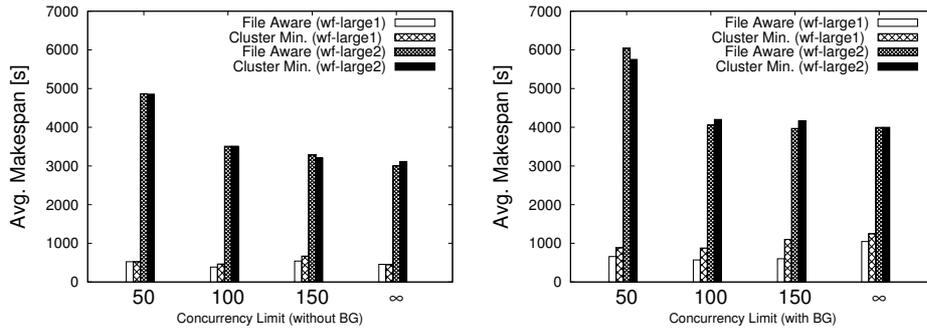
Figure 4: **Simulations, single workflow scheduling:** The average makespan for the File-Aware and the Cluster Minimization policies when **task throttling** is applied.

istics of the workflows, and to the utilization in the system (see Table 6). When scheduling workload wf-large1, which includes communication-intensive workflows, without background load, the makespan (NSL) performance can be improved by 15% (20%), while it can be improved by 50% (40%) when background load is imposed. On the other hand, when scheduling the workload wf-large2, which is more computation-intensive, the makespan (NSL) performance worsens for all the limit values, but the performance degradation is smaller when background load is imposed than without background load.

4.2 Multi-Workflow Scheduling

In the simulations of multi-workflow scheduling we consider the Single Cluster, the File-Aware (without and with task throttling), the Cluster Minimization (without and with task throttling), and the HEFT policies. We have used the CyberShake workflow application with sizes of 100 and 1000. For each of the policies that we consider, we submit either 5 or 50 concurrent instances of the CyberShake-100 application, and we submit 5 concurrent instances of the CyberShake-1000 application. When we apply task throttling, the concurrency limit per workflow submitted is set to 15 and 50 for the CyberShake-100, and CyberShake-1000 applications, respectively.

Figure 5 presents the performance of the scheduling policies in terms of the total makespan and the average NSL metrics. Table 7 presents additional metrics such as the grid-level delay due to the task throttling, average task queue wait time, the average task file transfer delay, and the average number of inter-cluster file transfers performed per workflow. For the small workflow (CyberShake-100) case, HEFT outperforms the other policies. Single Cluster and File Aware with task throttling yield similar performance results and they outperform the rest of the policies. For the large workflow case (CyberShake-1000), File Aware and Cluster Minimization both with task throttling have better total makespan performance than the other policies. In terms of NSL, File Aware and Cluster Minimization both without task throttling have the worst performance while the other policies achieve a similar performance.

With the Single Cluster policy, workflows are balanced across clusters, and no inter-cluster file transfer takes place. Task throttling decreases the amount of inter-cluster communication (see Table 7), hence both the performance of the File-Aware and the Cluster Minimization policies improve,

and they even outperform HEFT for the large workflow case. According to the results, for multi-workflow scheduling, a dynamic policy that takes inter-cluster communication into account and that also applies task throttling should be used. Alternatively, the Single Cluster policy may be preferred, depending on whether the application is communication- or computation-intensive and on the background load in the clusters, when the number of simultaneous workflow applications exceeds the number of clusters in the system.

4.3 Discussion

Our investigation shows that different system conditions, scenarios and workflow applications need different scheduling approaches in order to attain good application execution performance.

In general, increasing information usage about the workflow tasks and the grid resources improves the performance of the dynamic workflow scheduling policies. Although HEFT, which is the omniscient policy, is unrealistic for grids, HEFT-P, which uses predicted resource availability information, is a good alternative to be used provided that the application characteristics are known a priori, and that the prediction of the cluster that will finish a task first is correct.

The policies that take inter-cluster communication into account achieve better performance than the policies that do not. For instance, the Coarsening policy, and the File-Aware policy with task throttling are good alternatives that can be considered in the absence of complete task and resource information when scheduling communication-intensive large workflows.

When many workflow applications are submitted together, balancing the workflows across clusters separately or applying task throttling improves the performance, since both approaches prevent high inter-cluster network traffic, which increases file transfer times when many tasks are distributed across clusters.

5. REAL SYSTEM RESULTS

In this section we present the results of the experiments that we performed in DAS-3. We only present the results for the Single Cluster, All-Clusters, and Cluster Minimization policies since according to our simulation results, Round Robin performs the worst, File-Aware has similar performance (in many cases) as Cluster Minimization, and HEFT requires accurate resource and task information, which we find unrealistic for grid environments. Finally, we inves-
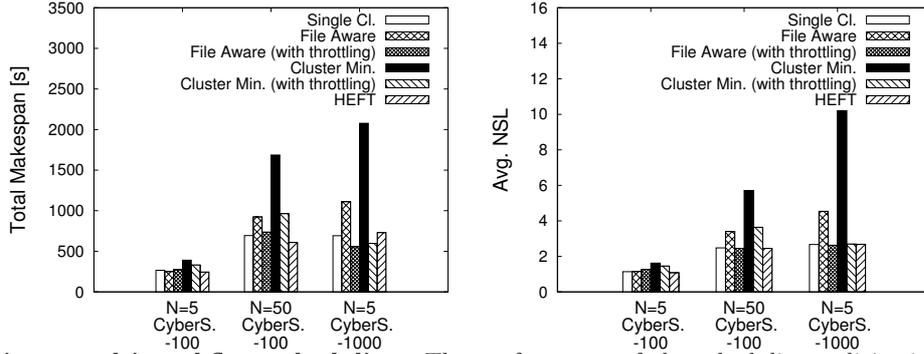
Figure 5: **Simulations, multi-workflow scheduling:** The performance of the scheduling policies in terms of the total makespan [s] and the average NSL. *N* denotes the number of application instances submitted together.

Table 7: **Simulations, multi-workflow scheduling:** The performance of the scheduling policies in terms of the average task queue wait time (grid-level + local, QWT [s]), the average task file transfer delay (FTD [s]), and the average number of inter-cluster file transfers performed per workflow (NFT).

	Single Cl.			File Aware			File Aware with Throttling			Cluster Min.			Cluster Min. with Throttling			HEFT		
CyberShake-100	QWT	FTD	NFT	QWT	FTD	NFT	QWT	FTD	NFT	QWT	FTD	NFT	QWT	FTD	NFT	QWT	FTD	NFT
N=5	0	0	0	0	0.02	1.8	21+0	0	0	0	29	12	63+0	6.22	12.8	0.24	0.13	7.2
N=50	124	0	0	95	1.63	30	71+28	2	21	198	30	43	97+36	7	28	121	0.49	24
CyberShake-1000																		
N=5	162	0	0	182	0.5	400	134+0	0.01	8	406	1	473	138+1	0.8	314	123	4.87	319

tigate the impact of task throttling with multi-workflow scheduling with the aim of improving system stability and responsiveness.

5.1 Single Workflow Scheduling

We first evaluate the performance of the selected workflow scheduling policies using the CyberShake-30 and the Montage-100 workflows for the wf-small1 and the wf-small2 workload type, respectively, and the CyberShake-1000 workflow for the wf-large1 workload type.

Table 8 shows the performance of the policies for single workflow scheduling experiments. We observe that the policies perform similarly for small workflows, confirming the simulations results for small workflows as shown in Figure 2 and Table 5. For the large workflow, Cluster Minimization has the best performance, significantly outperforming the Single Cluster and All-Clusters policies. Single Cluster has a higher queue wait time than the other policies, unlike the simulation results where All-Clusters has the highest queue wait time as shown in Table 5 (wflarge1 + BG). We attribute this difference to the much higher variability of the background load used in the simulations.

5.2 Multi-Workflow Scheduling

In this section we evaluate the performance of the selected scheduling policies, and the impact of task throttling on the performance of multi-workflow scheduling.

5.2.1 The Performance of the Scheduling Policies

For the multi-workflow scheduling experiments we submitted five instances of the same workflow application simultaneously and only once. Table 9 shows the performance of the policies for these experiments.

For small workflows, the relative performance order of the policies is the same as in the single workflow scheduling ex-
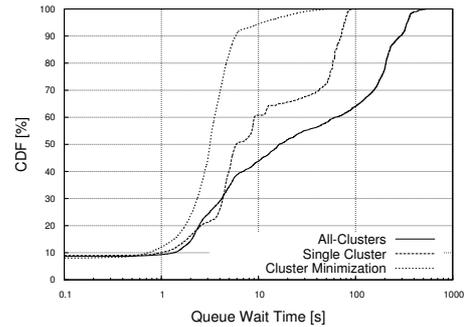

Figure 6: **Real system, multi-workflow scheduling:** Cumulative distribution function of the queue wait time of the CyberShake-1000 workflow tasks for all policies. The horizontal axis has a logarithmic scale.

periments, except for the Montage-100 workflow for which the Cluster Minimization policy has the worst performance. The reason is that Cluster Minimization does not balance the load well compared with the other policies, hence increasing the queue wait times.

For the large workflow, we observe that Single Cluster performs the best since all workflows that are submitted simultaneously are mapped to a separate cluster, hence distributing the load better than the other policies, and no inter-cluster file transfers take place. This result confirms the simulation results in Table 7 (last row). The All-Clusters policy has worse performance than the other policies. The reason is twofold. First, the All-Clusters policy causes the number of inter-cluster file transfers to increase. Secondly, with the All-Clusters policy, many tasks suffer a large wait time, which is shown in Figure 6. Unlike for the other poli-

Table 8: **Real system, single workflow scheduling:** The performance of the scheduling policies in terms of **average makespan** (MS [s]), average queue wait time (QWT [s]), average task file transfer delay (FTD [s]), and NSL.

Workload Type	Workflow	Single Cluster				All-Clusters				Cluster Min.			
		MS	NSL	QWT	FTD	MS	NSL	QWT	FTD	MS	NSL	QWT	FTD
wf-small1	CyberShake-30	244.35	1.38	4.98	0.99	251.40	1.42	4.19	1.00	247.46	1.39	5.16	1.01
wf-small2	Montage-100	1380.49	1.29	4.67	2.08	1390.72	1.30	4.62	2.14	1387.43	1.30	5.05	2.15
wf-large1	CyberShake-1000	1321.67	6.51	36.08	0.05	1101.39	5.42	4.96	0.02	718.70	3.54	5.09	0.02

Table 9: **Real system, multi-workflow scheduling:** The performance of the scheduling policies in terms of **total makespan** (MS [s]), average queue wait time (QWT [s]), average task file transfer delay (FTD [s]), and NSL.

Workload Type	Workflow	Single Cluster				All-Clusters				Cluster Min.			
		MS	NSL	QWT	FTD	MS	NSL	QWT	FTD	MS	NSL	QWT	FTD
wf-small1	CyberShake-30	328.92	1.85	4.61	1.01	331.89	1.87	5.46	1.02	330.14	1.86	3.57	1.02
wf-small2	Montage-100	2195.24	2.06	15.82	2.21	2377.00	2.23	17.29	2.27	3568.16	3.35	48.16	2.58
wf-large1	CyberShake-1000	1413.15	6.96	26.21	0.06	2859.89	14.08	103.41	0.12	1940.47	9.55	6.56	0.08

cies, around 37% of the tasks experience queue wait times of more than 100 seconds. We also observe a significant difference in median performance: the performance of the Cluster Minimization policy is roughly twice better than that of the Single Cluster policy, and the performance of the Single Cluster policy is roughly twice better than that of the All-Clusters policy. The performance difference between the policies is more significant in the real experiments than in the simulations. We attribute this observation to the resource contention in many layers of the system (e.g., in the network, and file system), which gets even worse for large workflows.

One of the main reasons for poor performance for multi-workflow submissions of large workflows is that the head-nodes become overloaded at such a scale. Figure 7 (top) shows the average CPU load of the Delft cluster head-node as reported by the system for the multi-workflow scheduling experiments with the CyberShake-1000 workflow. The CPU load is the value reported by the system's `top` utility. A high CPU load exceeding 100% can result in the incapacity of the system to perform even the simplest operations such as opening a socket or a file. This is the reason why we observe long delays in initiating file transfers (not included in FTD), as the workflow engine connects to the head-node of the execution site which is not responsive in overload situations. To overcome the problems related to head-node overload, we next investigate throttling as a possible solution.

5.2.2 The Impact of Task Throttling

In this section we evaluate the impact of task throttling on the performance of multi-workflow scheduling. To this end, we submit five instances of the CyberShake-1000 workflow application simultaneously and only once. We use 25, 50, 100 and 150 as the concurrency limits. In contrast with the experiments presented in Section 4.1.2, we use here only three of the five DAS-3 clusters, due to the unavailability of the two other clusters.

We first look at the impact of the concurrency limit on performance. Figure 7 (bottom) depicts the total makespan with and without task throttling applied. As the concurrency limit increases from 25 to 150, we observe a decrease in the total makespan, converging as the concurrency limit increases to the performance of the system when no throttling is applied.

Second, we investigate the effects of using task throttling on the head-node. Figure 7 (top) shows the average CPU load of the Delft cluster head-node, when the cluster acts both as the submission site and as one of the execution sites. We observe in the figure a stable period where the CPU load is high due to the large number of running tasks in the system. Although the throttled system with a concurrency limit of 150 tasks and the initial system yield a similar total makespan (see Figure 7 (bottom)), the system with task throttling exhibits a factor of 2 improvement in the average CPU load, which consequently improves the system stability and responsiveness. When the concurrency limit is set to 25, the average CPU load shows a further substantial decrease, leading to a factor of 4 improvement over the system without task throttling, but then the total makespan increases noticeably because of the low concurrency limit.

To conclude, task throttling with appropriate concurrency limits prevents head-nodes being overloaded and simultaneously preserves the execution performance of the workflow applications. This fact motivates future research on determining appropriate throttling mechanisms and their associated parameters (e.g., the concurrency limit).

5.3 Discussion

For single workflow scheduling, the different policies have similar performance for small workflows. However, for large workflows, the policies have different performance, and in particular, the policies that minimize the inter-cluster communication have better performance than the policies that do not.

For multi-workflow scheduling, selecting a single cluster per workflow for execution yields the best performance. Policies distributing the tasks across clusters have worse performance due to the increased inter-cluster communication. In addition, for large workflows, head-nodes may get overloaded, which consequently threatens the performance; task throttling alleviates this problem.

6. RELATED WORK

An extensive body of research has focused on scheduling workflows in traditional parallel systems, addressing both homogeneous [23] and heterogeneous [5] sets of processors. The scheduling methods are usually static, that is, all tasks
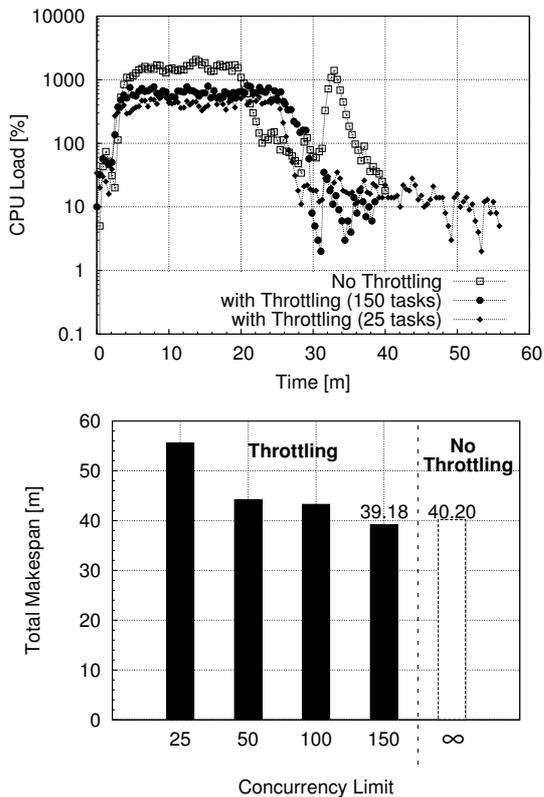
Figure 7: **Real system, multi-workflow scheduling:** The average CPU load of the Delft cluster head-node when 5 instances of CyberShake-1000 are submitted simultaneously (top), and the total makespan with and without throttling applied (bottom). The vertical axis of the top graph has a logarithmic scale.

are mapped to processors before execution of the workflow starts, and they assume that perfectly accurate information is available about the communication and the computation characteristics of the tasks. A classification of static scheduling approaches is presented in [36]. Such scheduling solutions, however, cannot be applied directly to multicluster grids. First, they operate at the processor level, while in grids the tasks are submitted to the local resource managers. Second, they do not consider the dynamic resource availability experienced in grids, which also makes accurate predictions of computation and communication costs difficult. Nevertheless, several studies [24, 25, 42, 43] adapted previous static scheduling methods by revising scheduling decisions at runtime taking the grid dynamics into account.

Currently, there are several scientific workflow management systems that can operate in grids. Some of the well-known ones are the Condor DAGMan [8], Pegasus [10], Karajan [38], Kepler [2], and Askalon [39]. They employ various types of static and/or dynamic scheduling methods. For more details we refer to the survey of Yu and Buyya [40].

Although most of the related work in grids deals with single workflow scheduling [11, 13, 15, 27], there are some studies that also address multi-workflow scheduling; by comparison, our work puts forth a more comprehensive investigation. Zhao and Sakellariou [44] present a method that combines several workflows into a single workflow, then prioritizes the

tasks and maps them to resources using a static scheduling method. Iverson et al. [19] demonstrate that scheduling each competing workflow with a dynamic policy in a decentralized way improves the overall performance.

In addition, several researchers have addressed data-aware workflow scheduling, in which large data sets associated with scientific workflows are taken into account when scheduling tasks. Park and Humphrey [29] propose a bandwidth allocation technique to speed up file transfers. Shishir and Ann [3] present several data staging techniques for data intensive workflows, and demonstrate that decoupled data staging can reduce the execution time of workflows significantly. Arun et al. [30] evaluate a dynamic method that minimizes the storage space needed by workflows by removing data files at runtime when they are no longer needed.

In summary, our work complements and extends previous work in three main ways. First, we consider dynamic policies with various information availabilities. Secondly, we consider both single and multi-workflow scheduling in our performance evaluation. Finally, we perform simulations with realistic scenarios, and we validate our findings through experiments in the DAS-3 multicluster grid.

7. CONCLUSION

The performance of grid workflow scheduling policies affects an increasing number of scientists. To understand this performance, in this work we have conducted a comprehensive and realistic performance evaluation of dynamic workflow scheduling policies in multi-cluster grids. We have first introduced a scheduling taxonomy based on the amount of information used in the scheduling process, and mapped seven scheduling policies that span the full information spectrum to this taxonomy.

Secondly, we have investigated the performance of these policies in realistic scenarios using both simulations and real system experiments. Overall, we found that different system conditions and workflow applications need different scheduling approaches in order to attain good application execution performance. Therefore, we believe it is important in grids, as we do with our KOALA grid scheduler [28], to support various scheduling policies from which the users can benefit considering the characteristics of their applications and the system capabilities. Alternatively, a scheduling mechanism can be implemented that switches dynamically the scheduling policy or the associated parameters, based on the system state and the workflows to be scheduled. We also found that, for scheduling communication-intensive workflows, the scheduling policies that take into account intercluster communication achieve better performance than the policies that do not. For example, the Coarsening policy, and the File-Aware policy with task throttling, that is, limiting the number of tasks concurrently present in the grid, are two good options that can be considered in the absence of complete task and resource information.

Thirdly, our real system experiments have revealed performance problems that did not show in the simulations. In particular, we found that the head-nodes of real grid clusters may become unstable as the workflow size increases, leading to much lower performance. To solve this problem, we have analyzed the performance of task throttling, and we have shown that this approach keeps the system stable while delivering good performance.

For the future, we intend to analyze how support for user-

defined scheduling policies affects the performance of grid workflow scheduling, and to find automatically the optimal concurrency limit for task throttling.

8. REFERENCES

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The globus striped gridftp framework and server. In *Supercomputing*, pages 54–64, 2005.

[2] I. Altintas et al. Kepler: An extensible system for design and execution of scientific workflows. In *SSDBM*, pages 21–23, 2004.

[3] S. Bharathi and A. Chervenak. Data staging strategies and their impact on the execution of scientific workflows. In *DADC*, page 5, 2009.

[4] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, Mei-Hui, and S. K. Vahi. Characterization of scientific workflows. In *Workshop on Workflows in Support of Large Scale Science, in conjunction with Supercomputing*, 2008.

[5] T. D. Braun et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *JPDC*, 61(6):810–837, June 2001.

[6] H. Casanova. On the harmfulness of redundant batch requests. *High-Performance Distributed Computing, International Symposium on*, 0:255–266, 2006.

[7] L. Cherkasova and P. Phaal. Session based admission control: a mechanism for improving performance of commercial web sites. In *Workshop on Quality of Service, IEEE/IFIP event*, 1998.

[8] Condor DAGMan. http://www.cs.wisc.edu/condor/dagman/.

[9] The Distributed ASCI Supercomputer. http://www.cs.vu.nl/das3/.

[10] E. Deelman et al. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13(3):219–237, 2005.

[11] E. Deelman et al. Task scheduling strategies for workflow-based applications in grids. In *CCGRID*, pages 759–767, 2005.

[12] Distributed Resource Management Application API. http://drmaa.org.

[13] R. Duan, R. Prodan, and T. Fahringer. Run-time optimisation of grid workflow applications. In *GRID*, pages 33–40, 2006.

[14] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *CCGRID*, page 35, 2001.

[15] L. He, S. A. Jarvis, D. P. Spooner, D. Bacigalupo, G. Tan, and G. R. Nudd. Mapping dag-based applications to multiclusters with background workload. In *CCGRID*, pages 855–862, 2005.

[16] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema. The grid workloads archive. *FGCS*, 24(7):672–686, 2008.

[17] A. Iosup, O. Sonmez, and D. Epema. DGSim: Comparing grid resource management architectures through trace-based simulation. In *Euro-Par*, volume 5168 of *LNCS*, pages 13–25, 2008.

[18] A. Iosup, O. O. Sonmez, S. Anoep, and D. H. J. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *HPDC*, pages 97–108, 2008.

[19] M. A. Iverson and F. Özgüner. Hierarchical, competitive scheduling of multiple dags in a dynamic heterogeneous environment. *Distrib. Sys. Engineering*, 6(3):112–, 1999.

[20] R. Iyer, V. Tewari, and K. Kant. Overload control mechanisms for web servers. In *Workshop on Performance and QoS of Next Generation Networks*, pages 225–244, 2000.

[21] G. Karypis and V. Kumar. Multilevel graph partitioning schemes. In *Int. Conf. Par. Proc.*, pages 113–122, 1995.

[22] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *JPDC*, 48:71–95, 1998.

[23] Y.-K. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *JPDC*, 59(3):381–422, 1999.

[24] J. Lee, S. Chin, H. Lee, T. Yoon, K. Chung, and H. Yu. Adaptive workflow scheduling strategy in service-based grids. In *GPC*, pages 298–309, 2007.

[25] K. Lee, N. W. Paton, R. Sakellariou, E. Deelman, A. A. A. Fernandes, and G. Mehta. Adaptive workflow processing and execution in pegasus. In *GPC-WORKSHOPS*, pages 99–106, 2008.

[26] N. Mansour, R. Ponnusamy, A. Choudhary, and G. C. Fox. Graph contraction for physical optimization methods: a quality-cost tradeoff for mapping data on parallel computers. In *Supercomputing*, pages 1–10, 1993.

[27] L. Meyer, D. Scheftner, J. Vöckler, M. Mattoso, M. Wilde, and I. Foster. An opportunistic algorithm for scheduling workflows on grids. In *VECPAR*, volume 4395 of *LNCS*, 2006.

[28] H. H. Mohamed and D. H. J. Epema. KOALA: a co-allocating grid scheduler. *CCPE*, 20(16):1851–1876, 2008.

[29] S.-M. Park and M. Humphrey. Data throttling for data-intensive workflows. In *IPDPS*, pages 1–11, 2008.

[30] A. Ramakrishnan et al. Scheduling data-intensive workflows onto storage-constrained distributed resources. In *CCGRID*, pages 401–409, 2007.

[31] O. Sonmez, H. Mohamed, and D. Epema. On the benefit of processor co-allocation in multicluster grid systems. *IEEE TPDS*, 2009. (To Appear).

[32] O. Sonmez, N. Yigitbasi, A. Iosup, and D. Epema. Trace-based evaluation of job runtime and queue wait time predictions in grids. In *HPDC*, pages 111–120, 2009.

[33] SPECCPU Team. SPEC CPU2006. Standard Performance. http://www.spec.org/cpu2006/.

[34] C. Stratan, A. Iosup, and D. H. J. Epema. A performance study of grid workflow engines. In *GRID '08: Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 25–32, Washington, DC, USA, 2008. IEEE Computer Society.

[35] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *CCPE*, 17(2-4):323–356, 2005.

[36] H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE TPDS*, 13(3):260–274, 2002.

[37] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE TPDS*, 18(6):789–803, 2007.

[38] G. von Laszewski and M. Hategan. Java CoG Kit Karajan/Gridant workflow guide. http://www.cogkit.org.

[39] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Rec.*, 34(3):56–62, 2005.

[40] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, 2005.

[41] J. Yu, R. Buyya, and R. Kotagiri. *Workflow Scheduling Algorithms for Grid Computing*. Springer, Berlin Germany, 2008.

[42] Z. Yu and W. Shi. An adaptive rescheduling strategy for grid workflow applications. *IPDPS*, 0:115, 2007.

[43] Y. Zhang, C. Koelbel, and K. Cooper. Hybrid re-scheduling mechanisms for workflow applications on multi-cluster grid. In *CCGRID*, pages 116–123, 2009.

[44] H. Zhao and R. Sakellariou. Scheduling multiple dags onto heterogeneous systems. In *HCW*, April 2006.