A Performance Study of Grid Workflow Engines

Corina Stratan[‡], Alexandru Iosup[⋆], and Dick H.J. Epema[⋆]

[‡]*Politehnica University of Bucharest, Romania,* corina@cs.pub.ro

[⋆]*Delft University of Technology, The Netherlands,* A.Iosup@gmail.com, D.H.J.Epema@tudelft.nl

Members of the CoreGRID European Virtual Institute on Grid Resource Management and Scheduling

Abstract

To benefit from grids, scientists require grid workflow engines that automatically manage the execution of inter-related jobs on the grid infrastructure. So far, the workflows community has focused on scheduling algorithms and on interface tools. Thus, while several grid workflow engines have been deployed, little is known about their performance-related characteristics, and there are no commonly-used testing practices. This situation limits the adoption of the grid workflow engines, and hampers their tuning and their further development. In this work we propose a testing methodology for grid workflow engines that focuses on five characteristics: overhead, raw performance, stability, scalability, and reliability. Using this methodology, we evaluate in a real test environment several middleware stacks that include grid workflow engines, including two based on DAGMan/Condor and on Karajan/Globus.

1 Introduction

Grids have emerged as the software and hardware infrastructure for e-Science by integrating tens of thousands of computing resources into a shared infrastructure. To be truly useful for scientists, the grid infrastructure needs to be complemented with automated tools for running scientific applications, of which workflows of inter-related jobs are common example. So far, the workflows community has focused on efficient resource management solutions [1, 19, 28, 33], and on user-friendly interface tools [22, 23]. As a result, grid workflow engines (GWFEs) like DAGMan [28] or Karajan [32] have been deployed by a large community [29, 30]. However, there are no commonly-used testing practices for GWFEs, and the performance-related characteristics of GWFEs have rarely been investigated. To address this situation, in this work we propose and use a methodology for systematically testing GWFEs.

The lack of commonly-used testing practices is widespread in grids. Few testing tools have been developed, and those that have focus on generic testing features. However, recent tests of real grids reveal lower raw performance than expected from simulations [9], more functionality problems than previously believed [11], and lower reliability than contemporary large-scale parallel production facilities [10, 18, 27]. The situation is even more disturbing for GWFEs: there are no reported testing tools, large-scale performance studies, or comprehensive comparative studies. This reduces the ability GWFEs to gain industrial acceptance, makes system tuning work difficult and case-by-case only, and prevents scientists from identifying and focusing on meaningful research problems. For example, we show in Section 5 that tuning the system to reduce trivial overheads can significantly improve the system's performance, and that for some classes of workflows relatively little performance remains to be gained by improving the scheduling algorithms alone.

Our methodology for testing GWFEs relies on comprehensive load- and stress-tests performed in a real environment. It uses synthetically generated yet realistic workloads to obtain detailed information about five GWFE characteristics: the overhead, the raw performance, the stability, the scalability, and the reliability. To ensure the repeatability of the tests, we implement the testing methodology on top of existing generic grid testing tools. Last but not least, we build `Generic WFE`, a generic GWFE that can execute workflows atop several grid and cluster middleware stacks, and we use it as a comparison baseline for the other tested GWFEs. Using the proposed methodology, we are able to evaluate in a real environment four grid middleware stacks, of which two are based on DAGMan/Condor and on Karajan/Globus. Our main contribution is threefold:

- We propose the first comprehensive methodology for testing GWFEs (Section 3);
- We implement the proposed methodology, including `Generic WFE`, a baseline for GWFE comparison (Section 4);
- We evaluate in a real environment five performance-related characteristics for four grid middleware stacks (Section 5).

2 Grid Workflow Engines

In this section we present a brief overview of GWFEs.

2.1 A Generic Workflow Model

Throughout this work we use for grid workflows the model introduced by Coffman and Graham [4]. A workflow of tasks is represented as a directed graph G in which the nodes are computational tasks, and the directed edges represent communication; we denote the number of nodes and the number of edges by N and E, respectively. For simplicity, we consider only directed acyclic graphs (*DAGs*). A computational task can start only when all its predecessors, both computation and communication tasks, are finished.

The following terminology and definitions are borrowed from [4, 5]. We call *root* a node without predecessors and

leaf a node without descendents; a DAG may have several roots and leaves. We further define *a node's level*, derived from breadth-first traversal of the task graph from its roots, as the minimal size of a path from the top to this node (in number of edges); the level of a root is 0. Finally, we call the maximum level of a leaf in the graph as the *graph level*, which we denote by $L-1$; we set the *graph traversal height* to L. Note that the chains-of-tasks may be considered as degenerate workflows with L equal to the number of tasks, and that master-worker applications can be seen as workflows with $L = 3$ (including tasks for the initial splitting for the results gathering and post-processing). We further define the *branching factor* of a workflow as the ratio between its number of edges E and its number of nodes N.

2.2 An Overview of GWFEs in Practice

We now present an overview of GWFEs; for more details we refer to the survey of Yu and Buyya [34]. The DAGMan, integrated with the Condor resource management system, has a number of strong advantages, among which are language simplicity and a mechanism for recovery from failures; however, it also has shortcomings like the lack of a graphical interface and the inability to submit tasks to other resource management systems than Condor. Another workflow engine often used in Grid applications is Karajan, which introduces an XML-based language, has a graphical interface and can interoperate with all the versions of the Globus Toolkit. WS-BPEL, a workflow language widely adopted in enterprise environments, benefits of rich semantics, standardization and support from a number of major companies. In academic environments WS-BPEL is less popular, because of its complexity and of the lack of open-source engines with graphical editors. The open-source engine ActiveBPEL has been used however in several Grid projects. The Pegasus system has the ability to transform the user-defined abstract workflows into concrete workflows that map over the Grid resources; this strategy allows for different types of optimizations, like job clustering and using the Grid replication system to minimize the data staging. Other well-known workflow engines are Taverna [23], which is focused on biology applications, and Kepler [22].

3 A Methodology for Testing GWFEs

In this section we present a methodology for testing GWFEs.

3.1 Overview

Our methodology for testing GWFEs relies on four main ideas. First, and in contrast to evaluating only the raw performance, our methodology focuses on five system characteristics: the overhead, the raw performance, the stability, the scalability, and the reliability. The overhead is the time lost by the real system in comparison with an ideal system in which each system state-related operation is instantaneous.

The raw performance comprises the classical time-related performance metrics, e.g., the wait time of the tasks. The stability is the property of the system to behave identically when faced with the same workload. The scalability is the ability of the system to execute increasing amounts of work without depreciated performance. Reliability deals with the system performance and functionality when failures occur.

Second, we believe that the interactions between the middleware layers are too complex to be ignored. Thus, for the analysis of the system characteristics we test the complete middleware stack, that is, we do not test the GWFE in isolation.

Third, testing with a workload comprising a single workflow may not give the measure of the system behavior in production. Instead, we test using complete workloads. Faced with the current lack of published workload models, we present in Section 3.3 seven workload types (three of which are new), and discuss how to use them to generate test workloads for various system sizes. Also, Section 3.2 introduces new metrics that have been adapted to testing with complete workloads.

Fourth, testing in real environments, while time-consuming and potentially difficult to implement, is the only way to characterize all the five characteristics followed in this work; in comparison, other techniques fall short in several respects. Mathematical analysis is difficult to use for such complex middleware stacks, and for such large numbers of resources; in addition, real environment data is still needed to begin with the evaluation of the overhead, of the stability, and of the scalability characteristics. Simulation suffers from the real data problem to a lesser extent, due to its ability to explore various scenarios, but modeling is still a big problem for systems as complex as grids. We have also shown in our previous work that simulation tends to overestimate performance by underestimating the overheads [9]. Finally, analyzing existing system logs or traces can reveal the performance characteristics of the GWFEs. However, these logs and traces are scarce, only cover a limited amount of scenarios, and can at most hint towards the conditions that pose problems. The authors' own experience with log and trace analysis [9, 12, 24], including the recent analysis of grid workflow engine logs [24], also shows that the logged information is usually incomplete and sometimes even inconsistent with the information logged by the other layers of the middleware stack.

3.2 The Metrics

Below we describe the set of metrics used to investigate the five GWFE characteristics. We extend the use of existing metrics [31, 24] for testing with multi-workflow workloads, and we propose five new metrics. We use the following metrics defined in previous work:

1. **MakeSpan (MS)** defined for a workflow as the number of seconds elapsed since the earliest submission time of a root and until the latest completion time of a leaf.
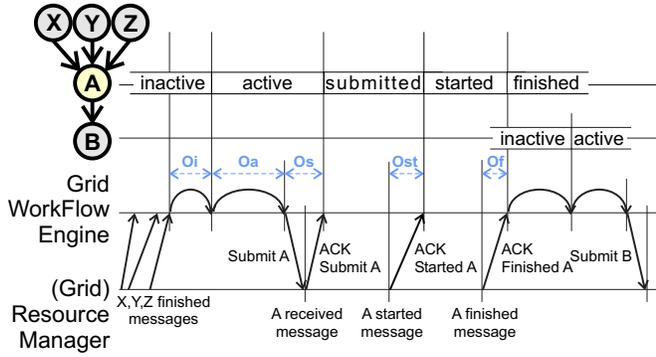
Figure 1. The execution of a workflow task, A. Boxes represent states. Continuous arrows represent information flows. The overhead components, i.e., the O_*, are also depicted.

2. **Sequential Path (SP)** [24] defined for a workflow as the MS of the execution of each of its tasks on a single machine dedicated to executing these tasks.

3. **Critical Path (CP)** [5] defined for a workflow as the time it takes to execute on a single, dedicated machine the shortest path from a root to a leaf in the workflow for which the delay of any of its tasks would cause a higher value of the MS.

4. **Speed-Up vs. Single Machine (SU-1)** [19, 24] defined for a workflow as the ratio between the MS of that workflow in the test environment and the workflow's SP.

To illustrate the components of the overhead for a single task, Figure 1 depicts a workflow with five tasks (A, B, X, Y, and Z), with an emphasis on task A's lifetime under a hypothetical execution process. Initially, A is "inactive" (it cannot be submitted for execution). Then, its predecessor tasks (i.e., tasks X, Y, and Z) finish executing, and after some time needed for task post-processing and internal state updates the GWFE changes A's state to "active", and starts task A's submission process. The task remains "active" until the grid resource manager acknowledges the submission; afterwards, A becomes "submitted". The task waits in the grid resource manager's queue, then starts executing, and after some delay due to messaging A's state becomes "started". Finally, the task finishes executing and after some more messaging delay A's state becomes "finished". From that point, the GWFE can begin executing task B. We define five overhead components: O_i, O_a, O_s, O_{st}, and O_f, which express the delays of the inactive to active transition, of the time elapsed between a task becomes active and the moment it is submitted to the grid resource manager, of the messaging needed to obtain an acknowledged submission, of the messaging needed to obtain a confirmation of the task's execution start, and of the messaging needed to perform the job clean-up and output transfer, respectively. The first four overheads manifest when the system is overloaded and/or when the GWFE manages its state inefficiently. The fifth overhead, O_f, manifests when the tasks have important amounts of data and/or when the GWFE's job clean-up and output transfer are inefficient.

We introduce in this work five new performance metrics (the MS-V, though also defined below, is introduced as a support metric and is not counted):

1. **Speed-Up vs. Perfect Grid (SU-inf)** defined for a workflow as the ratio between the MS of that workflow in the test environment and the workflow's CP. The SU-inf metric characterizes the achieved vs. the maximum achievable speed-up: only if its value is high over 1.0, improvements in the scheduling algorithm or system tuning can lead to significant performance improvements.

2. **Total Overhead (O)** defined for a workflow as the sum of the five overhead components discussed earlier. The O metric hints at how much the system tuning could improve the raw performance of the system.

3. **Head-Node Consumption** defined as the consumption of four resources on the head-node(s) of the system: the CPU usage expressed as a percentage, the memory usage expressed in MB, the number of processes used by the GWFE, and the number of sockets used by the GWFE. The head-node consumption is often ignored in simulation studies, which consider that the head-node can support an infinite amount of workflows (and their tasks). In practice, the head-node can quickly become the system's bottleneck, as shown by our previous work on testing grid resource managers [10], or by Henri Casanova's work on redundant job submissions [3].

4. **Internal Stability** defined for a workload as the ratio between the inter-quartile range (IQR) of the makespans of its workflows and their median makespan, expressed as a percentage. A system is internally (very) stable if its internal stability value is below 100% (10%).

5. **Overall Stability** We first define the Makespan Variability (MS-V) of a workload as the ratio between its MS and the range (minimum to maximum) of MS values for all its workflows. Then, we define the overall stability for a set of workloads executed in a test as the ratio between the maximum and the minimum MS-V value of any workload in the test; a value below 100% for this metric means the system is overall stable.

Finally, we define the MS, the SU-1, and the SU-inf metrics for multi-workflow workloads as the average of the metric's values for the workflows in the workload.

3.3 The Workloads

The system workload is a key part of any performance-related test [15]; in particular, the simulation work of Ahmad and Kwok [19] and our recent investigation of the performance delivered by a GWFE in a development environment [24] show that the properties of the workload's workflows have an important impact on the performance of the scheduling algorithms or engines.

3

Table 1. The job characteristics of the seven workflow-based workloads.

Wl. Type	Description	Characteristics N	L
S-1	sequence of tasks	30-50	30-50
S-2	parallel fork/join	30-50	5-15
S-3	bag-of-tasks	30-50	3
C-1	sameprob method [1]	30-50	5-15
C-2	samepred method [16]	30-50	5-15
C-3	layrprob method [33]	30-50	5-10
C-4	layrpred method [16]	30-50	5-10

Table 2. The characteristics of the four tested middleware stacks.

Middleware Stack	Description
DAGMan	DAGMan + Condor v. 6.8.3
Karajan	Karajan + Globus GT4 + Condor v. 6.8.3
Gen-Condor	Generic GWFE + Condor v. 6.8.3
Gen-SGE	Generic GWFE + Sun Grid Engine v. 6.1

Though desirable, it is difficult to define a realistic workload for GWFEs due to the scarcity of common practice reports. We therefore define seven testing workloads based on the following information sources: the only existing analysis of the traces of a real grid workflow engine [24], the GWFE community's research, the characteristics and the models of generic grid workloads [13, 14], and the workflow scheduling community's test workloads [1, 33, 19]. Table 1 summarizes the characteristics of the seven workloads used by our testing method; the first three (S-1, S-2, and S-3) are simple workload types, the last four (C-1 through C-4) are complex workload types. The first two workload types are instances of the chain-of-tasks (S-1) and of the master-worker (S-2) paradigms; S-3 is a hybrid in which the root node forks into several chains-of-tasks of short length that join at the end. The last four workload types have been defined by the workflow scheduling community [16] and have been used in several simulation-based workflow scheduling studies [1, 33, 19].

We now adapt in three steps these seven workload types to the task of testing GWFEs. First, we set for each workload type the characteristics of the workflows as shown in the columns N and L of Table 1. The values for N and L correspond to the real values encountered in at least one real system [24]. Second, to make these test workloads scalable, we recommend using workloads comprising a number of workflows (the *workload size*) and an average workflow task duration (the *task size*) that are proportional with the size and the performance of the tested system. Note that by combining multiple workflows into the same workload we can compensate for the limited range of the values of N. The task sizes encountered in practice may range from seconds to below 5 minutes for workflow tasks [24, 28], or from a few minutes to a few days for non-workflow grid jobs [13, 14] or for workflows with at most 3-4 tasks. Third, we define an arrival scenario that is typical to grids: bursty arrivals in which many workflows are submitted to the system in just a few minutes [21, 13, 14].

4 Experimental Setup

In this section we present the experimental setup.

4.1 The Test Environment

We used for testing a cluster of 15 common-of-the-shelf PCs, each having a dual-processor Pentium IV at 3.2GHz and 2GB RAM; the computers are interconnected by a 1 Gigabit Ethernet network. The environment is monitored by the MonALISA monitoring system [20] which has enabled us to gather resource consumption data from the workstations.

For the experiments we used and extended ServMark [2], a distributed performance evaluation tool for distributed environments. ServMark integrates two previously developed evaluation tools, DiPerF [26] and GrenchMark [10]. While GrenchMark is able to generate various types of synthetic Grid workloads, DiPerf coordinates the distributed testing, collecting performance metrics and providing performance statistics. ServMark combines the features of these two instruments and automates the time-consuming processes of performance testing, reliability testing and tuning for distributed systems. In order to evaluate the Grid workflow engines, we extended ServMark with modules that generate multiple types of workflow workloads, create submit files for the most common workflow engines and middlewares, and provide GWFE-related performance statistics.

The reason for selecting this test system size is twofold: it allows the tests to be performed in a reasonable amount of time (two weeks for the experiments in this work, with the duration proportional with the system size), and corresponds to the average cluster size for academic and small production clusters as reported by an extensive study [17].

4.2 The Tested Middleware Stacks

In the experiments presented in Section 5 we test four middleware stacks; Table 2 presents their structure. The first middleware stack, DAGMan, comprises the DAGMan GWFE and the Condor (grid) resource manager. Condor is one of the most used resource management systems: in 2006, Condor was used to manage at least 60,000 resources grouped in over 1750 pools [29]. DAGMan is the default workflow engine for Condor.

The second middleware stack, Karajan, comprises the Karajan GWFE, the Globus GT4 grid middleware, and Condor as the cluster resource manager. Globus is arguably the most used grid middleware; the Globus project counts over 5,000 file transfer servers and about 100,000 service head-nodes (a service head-node can manage anything from one cluster to one processor) [30].

The last two middleware stacks comprise our Generic GWFE operating on top of either Condor or the Sun Grid Engine (SGE) [7] as cluster resource managers. Our Generic GWFE is a simplistic implementation of a multi-threaded workflow engine, which attempts to submit tasks to the underlying middleware as soon as they become "ac-
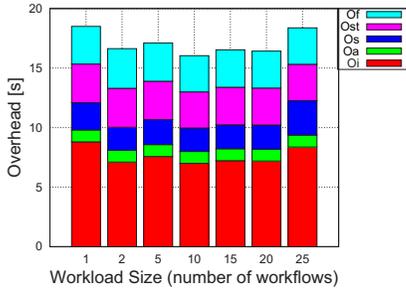
Figure 2. The overhead breakdown for seven workload sizes (from 1 to 25 workflows).
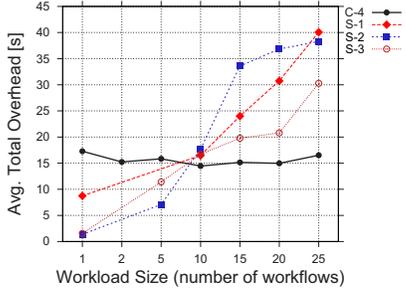

Figure 3. The impact of the workload type on the total overhead.

tive". Generic is not fault-tolerant, and does not have any of the advanced capabilities of a production GWFE. Thus, Generic can be used as the baseline GWFE.

5 Experimental Results

In this section we analyze five characteristics for four GWFEs: the overhead, the raw performance, the stability, the scalability, and the reliability.

5.1 The Overhead

To characterize the GWFE overhead we perform two sets of experiments. To better isolate the overhead, in each of these tests we execute jobs that take below 1 second to complete (i.e., "hello world" jobs).

First, we assess the impact of the middleware stack on the total overhead and on the overhead components (see Section 3.2) for one workload type, C-4. Figure 2 shows the total overhead and the overhead breakdown only for DAGMan; we have obtained similar results for all the other middleware stacks. The overhead is significant: over 15 seconds per task. From the overhead components, the most important is O_i, followed by O_f and O_{st}. The influence of each overhead component does not vary greatly for various workload sizes. Thus, better performance can be expected through improvements in the speed of updating the state of finished tasks and of their successors.

Second, we assess the impact of the workload type on the total overhead for DAGMan. Figure 3 shows the total overhead for four workload types; we have obtained similar

Table 3. The performance metrics for two of the four tested middleware stacks.

Middleware	MS [s]	SU-1	SU-inf
DAGMan	$1,327 \pm 138$	0.38 ± 0.05	1.20 ± 0.08
Karajan	$1,111 \pm 154$	0.30 ± 0.04	1.19 ± 0.16

results for the other three workload types. For C-4, the total overhead does not vary significantly with the increase of the workload size. We attribute this behavior to the characteristics of the C-4 workloads, which do not stress the system. Unlike C-4, the workloads comprising simple workflows stress the system, and their total overhead increases with the workload size. For S-1 and for S-3, the total overhead threatens to render the system unresponsive for workload sizes of above 50 workflows.
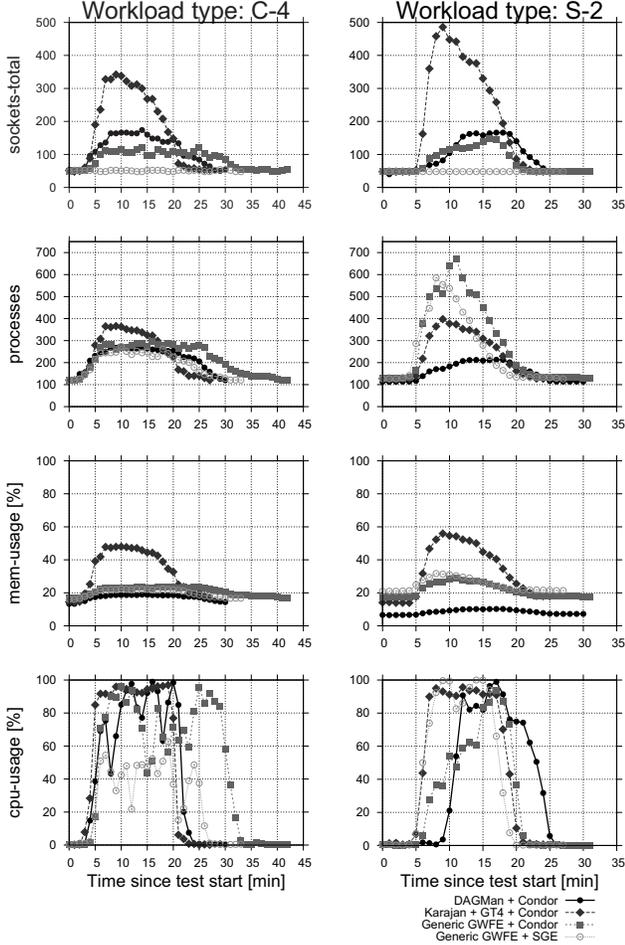
5.2 The Raw Performance

To characterize the GWFE performance we perform three sets of experiments; the first two are similar to the experiments for assessing the GWFE overhead, but the workload size is fixed to 10 workflows. In these performance experiments we run jobs with realistic durations, that is, with an average of 90 seconds for the first two, and with an average between 1 and 9 minutes for the third experiment. We assess in each experiment three performance metrics, MS, SU-1, and SU-inf, and for the head-node the consumption of its CPU, memory, process, and socket resources (see Section 3.2).

First, we assess the impact of the middleware stack on performance for one workload type, C-4. Table 3 shows the performance metrics for two of the four tested middleware stacks: DAGMan and Karajan (the real GWFEs). Karajan leads to a lower MS than DAGMan, by at most 15%. However, this higher performance comes at the cost of much higher resource consumption (see the next paragraph). The SU-1 metric of both Karajan and DAGMan is much lower than 1.0, which means that they are both efficient in comparison with a single machine. Finally, for both GWFEs the SU-inf metric is around 1.2: the performance improvement would be at most 20% over the current value, even if the scheduling algorithm would be optimal, and if the GWFE would operate on top of a grid system of infinite size and zero-latency for messaging. We conclude that for the C-4 workloads there is little to gain from adding new workflow scheduling algorithms to Karajan or DAGMan.

The head-node resource consumption for the first experiment is depicted in Figure 4 for C-4 and S-2; we have obtained similar results for all the other workload types. Karajan imposes the highest usage of any resource for C-4, and of the CPU, memory, and of the number of sockets for S-2. DAGMan has lower memory consumption and starts fewer processes than the other three middleware stacks; its CPU and socket are average. Finally, Gen-SGE consumes fewer head-node resources than Gen-Condor.

Second, we assess the impact of the workload type on performance for DAGMan. Table 4 shows the performance

Figure 4. The head-node resource consumption for the four tested middleware stacks: (*left column***) for the C-4 workload; (***right column***) for the S-2 workload. Each row depicts the consumption of one type of resource.**

Table 4. The performance metrics for the seven tested workload types.

Wl.type	MS [s]	SU-1	SU-inf
S-1	$3,663 \pm 1,003$	1.37 ± 0.07	1.37 ± 0.07
S-2	$1,373 \pm 243$	0.43 ± 0.07	1.29 ± 0.09
S-3	927 ± 122	0.37 ± 0.09	2.09 ± 0.56
C-1	$1,747 \pm 411$	0.59 ± 0.22	1.24 ± 0.07
C-2	$1,314 \pm 348$	0.35 ± 0.13	1.23 ± 0.06
C-3	$1,199 \pm 278$	0.48 ± 0.15	1.25 ± 0.14
C-4	$1,327 \pm 138$	0.38 ± 0.05	1.20 ± 0.08

Table 5. The performance metrics vs. the average task runtime.

Task RunTime [s]	MS [s]	SU-1	SU-inf
1'30"	$1,061 \pm 81$	0.25 ± 0.02	1.48 ± 0.05
3'	$1,533 \pm 76$	0.23 ± 0.02	1.44 ± 0.04
4'	$1,918 \pm 119$	0.23 ± 0.02	1.40 ± 0.02
5'	$2,597 \pm 167$	0.24 ± 0.02	1.44 ± 0.05
9'	$4,497 \pm 253$	0.23 ± 0.03	1.41 ± 0.04

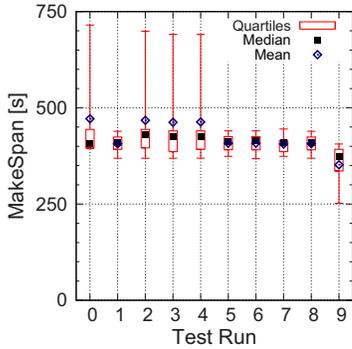gain from adding new workflow scheduling algorithms into DAGMan.

Third, we assess the impact on performance of the average task duration (t) for one workload type, C-4, and for one middleware stack, DAGMan; the results are summarized in Table 5. Expected, the MS of the workload increases when t increases. Notably, the SU-1 and the SU-inf performance metrics do not change significantly with t. We conclude that the expected speed-up is not correlated with the average task runtime, when the overheads are still comparable with the average task runtime.

### 5.3 The Stability, Scalability, and Reliability

In this section we present an overview of our experiments on the stability, on the scalability, and on the reliability of GWFEs.

We assess the stability of a middleware stack by running independently ten identical test runs. Each run comprises the execution of a C-4 workload with a size of 10 workflows/workload. Figure 5 depicts the MS distribution for the stability tests on DAGMan; we have obtained similar results for DAGMan using the other workload types. Five out of the ten runs (runs 1 and 5–8) hint towards the overall middleware stack's stability, and for them the system is internally very stable. However, the other five runs exhibit wide ranges of makespans, which makes the system overall unstable. A potential solution to make DAGMan overall stable is to extend it to a self-monitoring system that automatically adjusts to the system conditions to favor the most under-served workflows.

For scalability, we note that the main bottleneck in a busy system is often the head-node [10, 3]. Figure 4 shows the consumption for a moderate number of concurrent workflows (that is, 10) for the four middleware stacks. Karajan leads to the full utilization of the processor, to a number of 300-500 concurrently opened sockets (a common limit in today's operating systems is 1024), and to 50-60% memory usage. This indicates that Karajan would scale with difficulty to larger workload sizes. Following a similar

---

metrics for the seven test workload types. Expectedly, the MS varies greatly with the workload type: the tasks of bags-of-tasks workflows (S-3) are more suitable for execution in parallel than those of the chain-of-tasks workflows (S-1), and have a much lower MS; S-3 falls in between the two simple workload types. From the complex workload types, C-1 and to some extend C-2 are the most "difficult" and lead to the highest MSs. The SU-1 metric follows the same pattern: S-1 is better executed on a single machine than on the DAGMan middleware stack; at the other end, the S-3 workloads benefit the most from the grid execution. The SU-inf metric shows that DAGMan is already close to the lowest achievable MS, with the exception of the S-3 tasks, where there is still place for significant improvement (the SU-inf is high over 1.0); this improvement can be achieved either through new scheduling algorithms (e.g., like the ones proposed in our recent work [14]) or through reducing the overhead with system tuning (see Figure 3). Finally, for all the complex workloads (C-1 through C-4) there is little to

Figure 5. The makespan distribution for 10 independent runs of the C-4 workload using the DAGMan middleware stack. Each distribution is depicted as a *box-and-whiskers* plot with an additional point for the mean. Different runs exhibit similar mean and median, but very different value ranges.
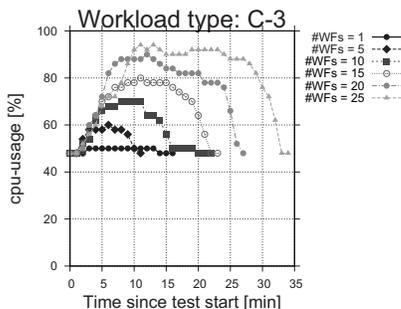
Figure 6. The CPU consumption on the head-node increases with the workload size.

Table 6. The makespan increases when hardware failures occur more often.

Failure inter-arrival [min]	no failure	60	20	5
Makespan [min]	72	77	151	390

reasoning, we note that DAGMan would scale much easier than Karajan, in particular with regard to the memory usage (which it heavily optimizes). Figure 6 shows how the CPU consumption increases with the workload size for DAGMan.

As shown in our previous work [12], resource failures happen on a regular basis in grid environments. Reliability has become an imperative requirement for grid middleware, and workflow engines must provide ways to handle failures as well. In a grid workflow, failures may occur at several levels: at the hardware level (e.g., workstation crash, network failure), at the task level (e.g., unhandled exception), and at the workflow level (e.g., data movement unsuccessful). While some grid middlewares are able to recover the jobs after hardware failures, the workflow engines have various ways of handling higher-level failures.

At the middleware level, both Condor and the SGE have job checkpointing mechanisms, but Condor handles hardware failures even when checkpointing is not enabled, by re-scheduling the job if the connection with the worker sta-

tion is lost. The workflow engines treat failures either by keeping a physical state log (DAGMan, Karajan) or through language-specific exception handling (ActiveBPEL).

Our reliability experiment analyzes the impact of hardware failures on the completion time of a workload. We use our (fault-intolerant) Generic GWFE with the Condor middleware, and cause at regular time intervals the crash of one station for a duration of 5 minutes. Table 6 shows the MS for three interval values, and for the case when no failure occurs. Expectedly, the MS increases with the decrease of the interval; for an interval of 5 minutes the MS increases fivefold in comparison to the no-failure scenario (mainly due to the default period for which Condor waits until attempting to reschedule the job, i.e., 20 minutes).

6 Related Work

Our work on testing GWFEs relates to work on performance assessment tools and to performance studies.

Recently, much attention has been given to tools for performance assessment in real environments [26, 10, 8]. However, these research efforts have been directed mainly to building generic testing solutions, and to providing distributed testing capabilities. Our tools leverage from DiPerF and from GrenchMark the high-performance testing and the test management and orchestration; we extend these two tools with GWFE testing capabilities.

Closest to our work, three recent studies evaluate in real environments the raw performance of cluster resource managers [6], the functionality and the reliability of a grid resource manager [10], and the overhead of one GWFE [25]. In contrast to the first two studies, our work focuses on GWFEs, and on five system characteristics. In contrast to the latter study, we evaluate several GWFEs, and assess more system characteristics.

Several performance studies have tried to compare workflow schedulers using simulation; the study of Ahmad and Kwok covers twenty such schedulers [19]. In comparison, our work extends the number of covered characteristics, uses more realistic workloads where multiple workflows are competing for the same resources, and focuses on real measurements in place of simulation.

7 Conclusion and Future Work

GWFEs are promising to couple the grid infrastructure to the demand for automated execution of large number of inter-dependent tasks. Until now, researchers have focused on creating interfaces to make this coupling easy-to-use, or on developing algorithms for more efficient use of the grid infrastructures. However, in-depth and comparative studies of the existing solutions are needed to extend the achieved results, and to gain industrial acceptance. In this work we have addressed this issue through an in-depth performance study of the performance of four GWFEs. First, we have introduced a methodology for testing GWFEs based on realistic and repeatable testing in real environments, and capable of characterizing five GWFE aspects: the overhead, the raw performance, the stability, the scalability, and the

reliability. Then, we have adapted and extended an existing generic grid performance evaluation tool to the requirements of the testing methodology. Finally, we have used the newly created tools to perform a comparative study of four middleware stacks that include two of the most widely used GWFEs, DAGMan and Karajan. Notably, we are able to show that these tools are already close to the optimal raw performance, but that significant improvements can be made to reduce their overhead, and to improve their stability, their scalability, and their behavior when facing failures.

For the near future, we plan to test more GWFEs, including some of the user-friendliest, e.g., Triana and Kepler. We also plan to investigate ways to make the testing results available to the performance-aware community, towards a common platform for the exchange of performance results.

References

[1] V. Almeida, I. M. M. Vasconcelos, J. N. C. Árabe, and D. A. Menascé. Using random task graphs to investigate the potential benefits of heterogeneity in parallel systems. In *SC*, pages 683–691, 1992.

[2] M. Andreica, N. Tapus, A. Iosup, D. Epema, C. Dumitrescu, I. Raicu, I. Foster, and M. Ripeanu. Towards ServMark, an architecture for testing grids. Technical Report TR-0062, CoreGRID, Nov 2006.

[3] H. Casanova. Benefits and drawbacks of redundant batch requests. *J. Grid Comput.*, 5(2):235–250, 2007.

[4] E. G. Coffman Jr. and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Inf.*, 1:200–213, 1972.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1989.

[6] T. A. El-Ghazawi, K. Gaj, N. A. Alexandridis, F. Vroman, N. Nguyen, J. R. Radzikowski, P. Samipagdi, and S. A. Suboh. A performance study of job management systems. *C&C:P&E*, 16(13):1229–1246, 2004.

[7] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *CCGrid*, pages 35–39. IEEE CS, 2001.

[8] W. Hoarau, S. Tixeuil, and L. Silva. Fault-injection and dependability benchmarking for grid computing middleware. In S. Gorlatch and M. Danelutto, editors, *Integrated Research in GRID Computing*, pages 119–134. Springer, 2007.

[9] A. Iosup, C. Dumitrescu, D. H. J. Epema, H. Li, and L. Wolters. How are real grids used? the analysis of four grid traces and its implications. In *GRID*, pages 262–269. IEEE CS, 2006.

[10] A. Iosup and D. H. J. Epema. GrenchMark: A framework for analyzing, testing, and comparing grids. In *CCGrid*, pages 313–320. IEEE CS, 2006.

[11] A. Iosup, D. H. J. Epema, A. Karp, P. Couvares, and M. Livny. Build-and-test workloads for grid middleware: Problem, analysis, and applications. In *CCGrid*, pages 205–213. IEEE CS, 2007.

[12] A. Iosup, M. Jan, O. O. Sonmez, and D. H. J. Epema. On the dynamic resource availability in grids. In *GRID*, pages 26–33. IEEE CS, 2007.

[13] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema. The Grid Workloads Archive. *Future Generation Comp. Syst.*, 24(7), Feb. 2008.

[14] A. Iosup, O. Sonmez, S. Anoep, and D. H. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *HPDC*, 2008. (accepted).

[15] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling,*. Wiley Interscience, May 1991.

[16] H. Kasahara. Standard Task Graph Set. [Online] Available: `http://www.kasahara.elec.waseda.ac.jp/schedule/`, Apr 2008.

[17] Y.-S. Kee, H. Casanova, and A. A. Chien. Realistic modeling and synthesis of resources for computational grids. In *SC*, page 54. IEEE CS, 2004.

[18] O. Khalili, J. He, C. Olschanowsky, A. Snavely, and H. Casanova. Measuring the performance and reliability of production computational grids. In *GRID*, pages 293–300. IEEE CS, 2006.

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

[20] I. Legrand et al. MonALISA: an agent based, dynamic service system to monitor, control and optimize grid based applications. In *CHEP*, Interlaken, Switzerland, 2004.

[21] H. Li, M. Muskulus, and L. Wolters. Modeling job arrivals in a data-intensive grid. In E. Frachtenberg and U. Schwiegelshohn, editors, *JSSPP*, volume 4376 of *LNCS*, pages 210–231. Springer-Verlag, 2006.

[22] B. Ludäscher et al. Scientific workflow management and the Kepler system. *C&C:P&E*, 18(10):1039–1065, 2006.

[23] T. M. Oinn et al. Taverna: lessons in creating a workflow environment for the life sciences. *C&C:P&E*, 18(10):1067–1100, 2006.

[24] S. Ostermann, A. Iosup, R. Prodan, T. Fahringer, and D. Epema. On the characteristics of grid workflows. In *Integrated Research in Grid Computing*, pages 431–442, Apr 2008.

[25] R. Prodan and T. Fahringer. Overhead analysis of scientific workflows in grid environments. *IEEE Trans. Parallel Distrib. Syst.*, 19(3):378–393, 2008.

[26] I. Raicu, C. Dumitrescu, M. Ripeanu, and I. T. Foster. The design, performance, and use of DiPerF: An automated distributed performance evaluation framework. *J. Grid Comput.*, 4(3):287–309, 2006.

[27] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *DSN*, pages 249–258. IEEE CS, 2006.

[28] G. Singh, C. Kesselman, and E. Deelman. Optimizing grid-based workflow execution. *J. Grid Comput.*, 3(3-4):201–219, 2005.

[29] D. Thain, T. Tannenbaum, and M. Livny. How to measure a large open-source distributed system. *C&C:P&E*, 18(15):1989–2019, 2006.

[30] The Metrics Project. Globus metrics. Tech.Rep. v1.4, Globus, May 2007.

[31] H. L. Truong, S. Dustdar, and T. Fahringer. Performance metrics and ontologies for grid workflows. *FGCS*, 23(6):760–772, 2007.

[32] G. von Laszewski and M. Hategan. Workflow concepts of the java cog kit. *J. Grid Comput.*, 3(3-4):239–258, 2005.

[33] T. Yang and A. Gerasoulis. Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE Trans. Parallel Distrib. Syst.*, 5(9):951–967, 1994.

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