Delft University of Technology
Parallel and Distributed Systems Report Series

# GrenchMark: A Framework for Analyzing, Testing, and Comparing Grids

**A. Iosup and D.H.J. Epema**
{A.Iosup,D.H.J.Epema}@ewi.tudelft.nl

**report number PDS-2005-007**

TU Delft PDS

**ISSN 1387-2109**

Published and produced by:
Parallel and Distributed Systems Section
Faculty of Information Technology and Systems Department of Technical Mathematics and Informatics
Delft University of Technology
Zuidplantsoen 4
2628 BZ Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
reports@pds.twi.tudelft.nl

Information about Parallel and Distributed Systems Section:
http://pds.twi.tudelft.nl/

**Abstract**

Grid computing is becoming the natural way to aggregate and share large sets of heterogeneous resources. With the infrastructure becoming ready for the challenge, current grid development and acceptance hinge on proving that grids reliably support real applications, and on creating adequate benchmarks to quantify this support. However, grid applications are just beginning to emerge, and traditional benchmarks have yet to prove representative in grid environments. To address this *chicken-and-egg* problem, we propose a middle-way approach: create and run synthetic grid workloads comprising applications representative for today's grids. For this purpose, we have designed and implemented GrenchMark, a framework for synthetic workload generation and submission. The framework greatly facilitates synthetic workload modeling, comes with over 35 synthetic and real applications, and is extensible and flexible. We show how the framework can be used for grid system analysis, functionality testing in grid environments, and for comparing different grid settings, and present the results obtained with GrenchMark in our multi-cluster grid, the DAS.

# Contents

# List of Figures



# List of Tables

**Abstract**

Grid computing is becoming the natural way to aggregate and share large sets of heterogeneous resources. With the infrastructure becoming ready for the challenge, current grid development and acceptance hinge on proving that grids reliably support real applications, and on creating adequate benchmarks to quantify this support. However, grid applications are just beginning to emerge, and traditional benchmarks have yet to prove representative in grid environments. To address this *chicken-and-egg* problem, we propose a middle-way approach: create and run synthetic grid workloads comprising applications representative for today's grids. For this purpose, we have designed and implemented GrenchMark, a framework for synthetic workload generation and submission. The framework greatly facilitates synthetic workload modeling, comes with over 35 synthetic and real applications, and is extensible and flexible. We show how the framework can be used for grid system analysis, functionality testing in grid environments, and for comparing different grid settings, and present the results obtained with GrenchMark in our multi-cluster grid, the DAS.

# 1 Introduction

In the long term, Grid computing systems (Grids) aim at becoming the standard way of sharing heterogeneous resources, and of aggregating them into virtual platforms, to be used by multiple organizations and independent users alike. With the grid infrastructure starting to meet the requirements of such an ambitious goal [2], the current evolution of grids hinges on proving that it can run real applications, from traditional sequential and parallel applications to new, grid-only, applications. As a consequence, there is a clear need for generating and running workloads comprising grid applications for demonstration and testing purposes.

In this paper we present GrenchMark, a framework for synthetic workload generation and submission. With GrenchMark, we try to find a common ground for grid performance analysis and, in the flavor of the Parallel Workloads Archive[1], we offer the performance-oriented Grid community a tool that can help to bring together Grid performance evaluation approaches, towards the goal of building standard Grid benchmarks. Our main contributions are:

- A systematic approach to and a set of tools for generating synthetic grid workloads for analyzing, testing, and comparing common grid settings (Sections 2 and 4);
- Modeling and selecting a set of representative real and synthetic grid applications (Section 3), including applications that require co-allocation (Section 3.2).
- An experimental validation of our approach (Sections 5 and 6). In our setting, we use a multi-cluster environment, the DAS [1], the Koala[2] co-allocating grid scheduler [10], and the real applications included in the Ibis[3] Java-based Grid programming environment [14].

# 2 A Case for Synthetic Grid Workloads

There are three ways of evaluating the performance of a grid system: analytical modeling, simulation, and experimental testing. This section presents the benefits and drawbacks of each of the three, and argues for evaluating the performance of grid systems using synthetic workloads, one of the two possible approaches for experimental testing.

## 2.1 Analytical Modeling and Simulations

*Analytical modeling* is a traditional method for gaining insights into the performance of computing systems. Analytical modeling may simplify what-if analysis for changes in the system, in the middleware, or in the applications. However, the sheer size of grids and their heterogeneity make realistic analytical modeling hardly tractable.

[1] The Parallel Workload Archives makes various workload traces from real parallel production environments available at `http://www.cs.huji.ac.il/labs/parallel/workload/`

[2] Koala is developed at TU Delft, NL; more information about Koala is available at `http://www.st.ewi.tudelft.nl/koala/`.

[3] Ibis is developed at VU Amsterdam, NL, and is freely available from `http://www.cs.vu.nl/ibis/`.

*Simulations* may handle complex situations, sometimes very close to the real system. Furthermore, simulations allow the replay of real situations, greatly facilitating the discovery of appropriate solutions. However, simulated system size and diversity raises questions on the representativeness of simulating grids. Moreover, nondeterminism and other grid-specific forms of hidden dynamic behavior make the simulation approach even less suitable.

## 2.2 Experimental Testing

There are two ways to experimentally assess the performance of grid systems: *benchmarking* and using *synthetic grid workloads*. Note that current grids evolution prevent the use of traces of real grid workloads: the infrastructure changes too fast, leading to incompatible resource requests when re-running old traces. However, well-studied traces from production environments may be used for *what-if* analysis, for example to show that a parallel production machine can be replaced by a grid environment.

*Benchmarking* is typically used to understand the quantitative aspects of running grid applications and to make results readily available for comparison. Benchmarks comprise a set applications representative for a class of systems, and a set of rules for running the applications as a synthetic system workload. Therefore, a benchmark is a single instance of a synthetic workload.

Benchmarks present severe limitations, when compared to synthetic grid workloads generation. They have to be developed under the auspices of an important number of (typically competing) entities, and can only include well-studied applications. Putting aside the considerable amounts of time and resources needed for these tasks, the main problem is that grid applications are starting to develop just now, typically at the same time with the infrastructure [12], thus limiting the availability of truly representative applications for inclusion in standard benchmarks. Other limitations in using benchmarks for more than raw performance evaluation are:

- Benchmarking results are valid only for workloads truly represented by the benchmark's set of applications; moreover, the number of applications typically included in benchmarks [8, 13] is typically small, limiting even more the scope of benchmarks;
- Benchmarks include mixes of applications representative at a certain moment of time, and are notoriously resistant to include new applications; thus, benchmarks cannot respond to the changing requirements of developing infrastructures, such as grids;
- Benchmarks make difficult either the evaluation of one particular system characteristic (high-level benchmarks), or the evaluation of a mix of characteristics (low-level benchmarks);

An extensible framework for generating and submitting *synthetic grid workloads* uses applications representative for today's grids, and enables the addition of future grid applications. This approach can help overcome the aforementioned limitations of benchmarks. First, it offers better flexibility in choosing the starting applications set when compared to benchmarks. Second, applications can be included in generated workloads, even when they are in a debug or test phase. Third, the workload generation can be easily parameterized, to allow for the evaluation of one or a mix of system characteristics.

## 2.3 Purposes of Synthetic Grid Workloads

We further present five reasons for using synthetic grid workloads.

a. *System design and procurement* Grid architectures offer many alternatives to their designers, in the form of hardware, of operating software, of middleware (e.g., a large variety of schedulers), and of software libraries. When a new system is replacing an old one, running a synthetic workload can show whether the new configuration performs according to the expectations, before the system becomes available to users. The same procedure may be used for assessing the performance of various systems, in the selection phase of the procurement process.

b. *Functionality testing and system tuning* Due to the inherent heterogeneity of the grids, complicated tasks may fail in various ways, for example due to misconfiguration or unavailability of required grid

middleware. Running synthetic workloads, which use the middleware in ways similar to the real application, helps testing the functionality of the grids and detecting many of the existing problems.

c. *Performance testing of grid applications* With grid applications being more and more oriented toward services or components, early performance testing is not only possible, but also required. The production cycle of traditional parallel and distributed applications must include early testing and profiling. These requirements can be satisfied with a synthetic workload generator and submitter.

d. *Comparing grid components* Grid middleware comprises various components, e.g., resource schedulers, information systems, and security managers. Synthetic workloads can be used for solving the requirements of component-specific use cases, or for testing the Grid-component integration.

e. *Building runtime databases* In many cases, getting accurate information about an application's runtime is critical for further optimizing its execution. For many scheduling algorithms, like backfilling, this information is useful or even critical. In addition, some applications need (dynamic) *on-site* tuning of their parameters in order to run faster. The use of historical runtime information databases can help alleviate this problem [11]. An automated workload generator and submitter would be of great help in filling the databases.

In this paper we show how GrenchMark can be used to generate synthetic workloads suitable for these five goals.

# 3 A Model for Synthetic Grid Workloads

This section presents a model for synthetic grid workloads.

## 3.1 Grid site and machine model

We assume grid systems comprising several computing sites. Sites are provided and maintained by individuals or institutions, and are dedicated to grid usage. Sites may have different usage policies, e.g., one site may dedicate 100% resources for running its local users' jobs, and only share the resources to the global grid community if no such jobs exist, while, at the other extreme, another site may offer all its resources at any time to anybody belonging to the global grid community, without discriminating between local and remote users.

Each site contains several computing resources, e.g., a site is a *cluster* of resources. Resources can be computational and storage resources or both at the same time. On each site, there is only one *gateway* (a machine used as an entry point to the system, from which jobs can be launched and files can be transferred to and from the cluster).

## 3.2 Grid applications model

In our model, we consider two types of applications that can run in grids, and may be included in synthetic grid workloads.

### 3.2.1 Unitary applications

This category includes single, unitary, applications. At most the job programming model must be taken into account when running in grids (e.g., launching a name server before launching an Ibis job). Typical examples include sequential and parallel (e.g., MPI, Java RMI, Ibis) applications.

### 3.2.2 Composite applications

This category includes applications composed of several unitary or composite applications. The grid scheduler needs to take into account issues like task inter-dependencies, advanced reservation and extended fault-tolerance, besides the components' job programming model. Typical examples include bags of tasks, chains of

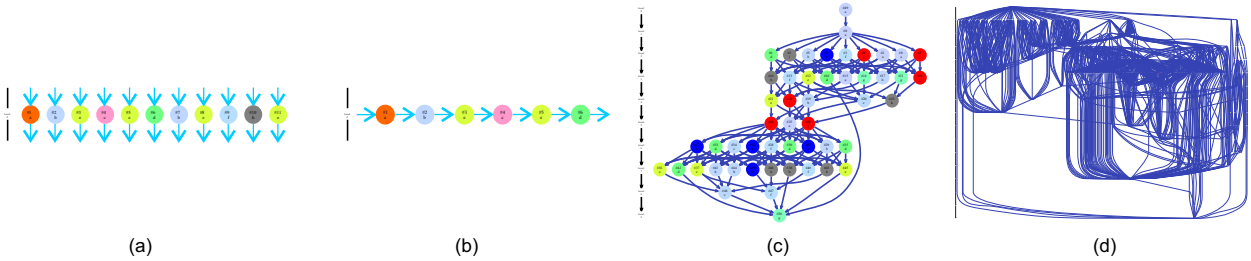


Figure 1: Composite applications generated with GrenchMark: (a) randomly generated *bag of jobs*; (b) randomly generated *chain of jobs*; (c) *DAG-based workflow* generated with a random method from the Standard Task Graph Set (`http://www.kasahara.elec.waseda.ac.jp/schedule/`); (d) *DAG-based workflow* generated from the SPEC benchmark `fpppp`, included in the Standard Task Graph Set.

tasks, DAG-based applications, and applications based on generic graphs (see Figure 1 for sample composite applications generated with GrenchMark).

We also consider in our model the notion of co-allocation: the simultaneous allocation of resources located in different grid sites to single applications which consist of multiple components. The components of co-allocated jobs usually interact with each other (e.g., consider the case of a tightly-coupled parallel application which happens to run on resources located in several grid sites). Unitary and composite jobs can both be co-allocated.

### 3.3 Grid workloads model

We argue that many of the potential Grid users are current HPC users, or are currently running large batches of jobs. For many working environments, and especially for the DAS, the amount of sequential jobs is significant or even dominant. Therefore, unitary applications with sequential or parallel structure, and composite applications with bag of tasks structure are to be preferred when creating grid workloads.

The workload structure is built using well-known statistical distributions for modeling resource requirements, with values extracted from real traces or selected from sets pre-defined by the workload designer. Jobs arrive dynamically in the system, and the inter-arrival time of the majority of jobs can be modeled with a statistical distribution. Jobs arrival can also be *bursty*, that is, many jobs may arrive in a very short time interval.

## 4 The GrenchMark framework

This section presents the GrenchMark framework.

### 4.1 Overview

GrenchMark is a synthetic grid workload generator and submitter. It is *extensible*, in that it allows new types of grid applications to be included in the workload generation, *parameterizable*, as it allows the user to parameterize the workloads generation and submission, and *portable*, as its reference implementation is written in `Python`.

The workload generator is based on the concepts of *unit generators* and of job description files (JDF) *printers*. The *unit generators* produce detailed descriptions on running a set of applications (*workload unit*), according to the workload description provided by the user. In principle, there is one unit for each supported application type. The *printers* take the generated workload units and create job description files suitable for grid submission. In this way, multiple unit generators can be coupled to produce a workload that can be submitted to any grid resource manager, as long as the resource manager supports that type of applications.

The grid applications currently supported by GrenchMark are sequential jobs, jobs which use MPI, and Ibis jobs. GrenchMark includes a set of over 35 synthetic and real applications. We have implemented four *synthetic applications*: `sser`, a sequential application with parameterizable computation and memory requirements, `sserio`, a sequential application with parameterizable computation and I/O requirements,

`smpi1`, an MPI application with parameterizable computation, communication, memory, and I/O requirements, and `swf`, an application implementing the job model used in the Standard Workloads Format, from the Parallel Workloads Archives (PWA). We use all the real applications and some of the synthetic applications included in the default Ibis distribution package. The reason is threefold: the Ibis applications closely resemble or are real-life parallel applications, they have been proven to run on a variety of grid settings, and they have been thoroughly described[4]. The Ibis applications come from the areas of physical simulations, parallel rendering, computational mathematics, state space search, bioinformatics, data compression, grid methods, and optimization. Currently, GrenchMark can submit jobs to Koala, Globus GRAM, and Condor.

The workload submitter generates detailed reports of the submission process. The reports include all job submission commands, the turnaround time of each job, including the grid overhead, the total turnaround time of the workload, and various statistical information.

## 4.2 Replaying traces with GrenchMark

GrenchMark can replay traces from various production environments. First, the user can load traces in the Standard Workload Format. Second, since a real trace contains tens of thousands of jobs, filtering out jobs and selecting the *first-N* jobs according to various criteria is essential for selecting a usable workload; GrenchMark can be used to shape and fit a real workload trace. Third, real traces cannot usually be replayed on another system than the one on which they were acquired, and even on the same system if it has changed. The latter happens more often in the case of grids, which are naturally evolving with their middleware, and therefore are very dynamic. GrenchMark can scale various aspects of the workload, e.g., the requested resources, or the job runtimes, and can help alleviate this problem [6].

## 4.3 Modeling workloads with GrenchMark

GrenchMark offers support for the following workload modeling aspects. First, it supports unitary and composite applications, and single-site and co-allocated jobs. Second, it allows the user to define various job inter-arrival times based on well-known statistical distributions. Besides the Poisson distribution, used traditionally in queue-based systems simulation, GrenchMark also supports uniform, normal, exponential and hyper-exponential, Weibull, log normal, and gamma distributions. Third, it allows the workload designer to combine several workloads into a single one (*mixing*). This allows for instance the inclusion of bursts, by combining a short workload with many jobs per time unit with a longer one, comprising fewer jobs per time unit. An additional use of workload mixing is in a *what-if* analysis that evaluates what will happen to a grid community if its resources would be shared with another group of users. In this case, the workload modeler can mix the typical workload of the two communities and evaluate whether the system can support both, under various job acceptance and execution policies.

## 4.4 The GrenchMark process

We define two use cases for the GrenchMark framework: in real world and in simulations. A workload generated by GrenchMark using only synthetic and well studied applications can be equally used in both cases.

Figure 2 depicts the typical process of using the GrenchMark framework in a real environment. First, the user describes the workload to be generated, as a formatted text file (1). Based on the user description, on the known application types, and on information about the grid sites, a workload is then generated by GrenchMark (2), and submitted to the grid (3). The grid environment is responsible for executing the jobs and returning their results (4). The results include not only job outcomes, but also detailed submission reports. Finally, the user processes all results in a post-production step (5). The use of GrenchMark for simulations is similar, with steps (3) and (4) possibly combined.

The most difficult step in using GrenchMark is step (1): describing the workload. To ease this task, we have designed a simple and extensible language; the workload designer is only concerned with the difficulty of designing representative workloads, rather than how to describe them. Figure 3 shows the workload

[4]For the complete list of publications related to Ibis applications, please visit `http://www.cs.vu.nl/ibis`

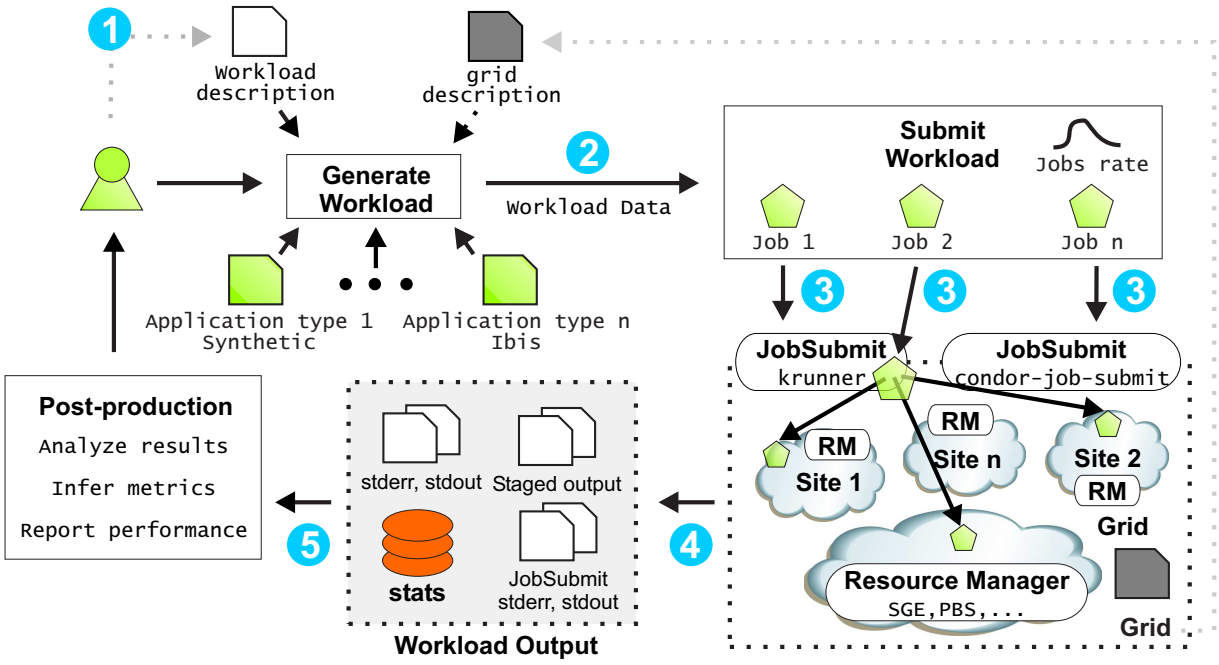


Figure 2: The GrenchMark process.

```
# File-type: text/wl-spec
#Jobs Type SiteType Total SiteInfo ArrivalTimeDistr OtherInfo
25   sser   single    1 *:? Poisson(120s)  StartAt=0s
25   sserio single    1 *:? Poisson(120s)  StartAt=60s
25   smpi1  single    1 *:? Poisson(120s)  StartAt=30s,ExternalFile=smpi1.xin
25   smpi1  single    1 *:? Poisson(120s)  StartAt=90s,ExternalFile=smpi2.xin
```

Figure 3: A GrenchMark workload description example.

```
SiteTypesWithWeights=nonfixed/30;fixed/50
SitesWithWeights=fs1;fs2;fs3;fs4
NComponentsWithWeights=1/50.0;3/16.2;5/5.0;10;15;2/10.0;4/10.0;8
TotalCPUsWithWeights=2/20.0;4/30.0;5;8;10;16;20;32
...
```

Figure 4: A GrenchMark workload specification language extension.

description for generating the `gmark+` test, comprising 100 jobs of four different types. The first two lines are comments. The next two lines are used to generate sequential jobs of types `sser` and `sserio`, with default parameters. The final two lines are used to generate MPI jobs of type `smpi1`, with parameters specified in external files `smpi1.xin` and `smpi2.xin`. All four job types assume an arrival process with Poisson distribution, with an average rate of 1 job every 120 seconds. The first job of each type starts at a time specified in the workload description with the help of the `StartAt` tag. For MPI jobs, the specified external file (tag `ExternalFile`) contains other application-specific parameters (see also Section 4.5 and Figure 4).

## 4.5 Extending the GrenchMark framework

GrenchMark has been designed with an incremental approach in mind, and facilitates future extensions. The framework design can be easily extended, for instance by adding various workload generation *notions* (e.g., users, virtual organizations). GrenchMark also offers a simple *plug-in system*, which can be used to add unit generators (new application types) and printers (support for other grid resource managers).

Another way the user can extend the workload generation process is to define a more refined workload specification language and use it from an already existing plug-in; the only requirement is that the extension language is based on *Key=Value* statements. A file written in an extension language is automatically parsed, and the data is available to the user when the plug-in is invoked. GrenchMark further offers mechanisms to parse simple values (e.g., `boolean`s, `string`s, `integer`s, and `float`s), and more complicated constructs (e.g., lists, and lists with weights). Figure 4 shows an example of a file written in an extended language, for generating a workload with co-allocated jobs. Line 1 defines the type of co-allocated jobs to be generated as a list with weights; *nonfixed* jobs (multi-site jobs with unspecified execution sites) have a lower weight than *fixed* jobs (multi-site jobs with specified execution sites). Line 2 defines the possible execution sites, as a list with weights. A default weight of 1.0 is automatically assigned to the elements for which the user has not specified the weight. Lines 3 and 4 use lists with weights to define the number of components and the number of processors for generated jobs. Jobs with 1 component and 4 processors are preferred.

# 5 Experimental setup

We used the DAS system[5] as an experimental environment. The DAS system comprises 5 clusters, with 32 up to 72 dual-processor SMP nodes each. A shared file system (NFS) is used within each cluster. Each cluster has one entry point (*gateway*), on which users can log and submit jobs, or retrieve their jobs' output. The DAS system does not discriminate between local and remote users, but has different resource allocation policies depending on the jobs' requested runtimes.

GrenchMark was used to generate and submit the workloads. We used both modeled and real (trace-based) workloads to generate the test workloads. Each workload was submitted in the normal DAS working environment, thus being influenced by the background load generated by other DAS users. Some jobs could not finish in the time for which they requested resources, and were stopped automatically by the Koala scheduler. This situation corresponds to users under-estimating applications' runtimes. Each workload ran between the submission start time and 20 minutes after the submission of the last job. Thus, some jobs did not run, as not enough free resources were available during the time between their submission and the end of the workload run. This situation is typical for real working environments, and being able to run and stop the workload according to the user specifications shows some of the capabilities of GrenchMark.

# 6 The results

This section presents the results obtained with GrenchMark. The experiments are not gauged to show a full analysis of a certain feature; instead, we try to show how GrenchMark can be used for various goals. The major difference between the two approaches is in the way results are analyzed, due to the difference in goals; e.g., in Table 2 we show a summary of runtimes for three applications (*performance testing*), but not the detailed analysis required for the complete performance characterization of these applications (*performance analysis*).

Unless otherwise stated, we use the success rate of the jobs as the performance metric for our experiments. The reason is twofold: we find that for current grids the ability to reliably run jobs is even more important than raw performance, and we argue (based on the results shown in Table 2) that GrenchMark results can also be used to extract other metrics (from the many available, see for instance [7]). A successful job is a job that acquires its requested resources, runs, finishes, and returns all results within the time allowed for the workload.

## 6.1 Grid system analysis

This section shows how GrenchMark can be used for two types of system analysis: performance testing (and analysis), and *what-if* analysis.

[5]Distributed ASCI Supercomputer, `http://www.cs.vu.nl/das2/`

Table 1: The experimental results for performance testing.

| Workload | Applications types | # of CPUs | Component number | Component size | Success Rate |
|---|---|---|---|---|---|
| `gmark1` | synthetic, sequential | 1 | 1 | 1 | 97% |
| `smpi1` | synthetic, MPI | 2-128 | 1-15 | 1-32 | 81% |
| `ibis1` | N Queens, Ibis | 2-16 | 1-8 | 2-16 | 70% |

Table 2: A summary of time and run/success percentages for different job types.

| Job name | Job type | Turnaround [s] Avg | Turnaround [s] Range | Runtime [s] Avg | Runtime [s] Range | Run | Run+ Success |
|---|---|---|---|---|---|---|---|
| `sser` | seq | 129 | 16–926 | 44 | 1–588 | 100% | 97% |
| `smpi1` | MPI | 332 | 21–1078 | 110 | 1–332 | 80% | 81% |
| `NQueens` | Ibis | 99 | 15–1835 | 31 | 1–201 | 70% | 85% |

### 6.1.1 Performance testing

We consider a *performance testing* case in which *the user wants to test the performance of one or more of his applications*. This situation occurs when developing an application, when testing one or more system design options. In all of these situations, the user wants to generate the workload with as little input as possible, submit the generated workloads, and have a detailed report on the obtained results within a certain time-frame.

Table 1 shows the structure of the three generated workloads, and the detailed success rate for running them. We use three types of applications: `sser` (synthetic sequential), `smpi1` (synthetic parallel, MPI), and an *N Queens solver* (synthetic parallel, Ibis). Each generated workload contains one application type, and comprises 100 different instances. For synthetic applications, we used combinations of parameters that would keep the run-time of the applications under 30 minutes, under optimal conditions. For the Ibis jobs (workload `ibis1`), 10-20% of the applications have severely underestimated runtime requests; this corresponds to the situation where some users severely underestimate the runtimes of complex applications [15]. Each job requests resources for a time below 15 minutes. The combination of run-time and resource request settings ensures that some applications would fail due to the user's own fault. The total submission time of any workload is kept under two hours, to show how GrenchMark can be used in a time-constrained test situation. To satisfy typical grid situations, jobs request resources for 1 to 15 components. As the DAS has only 5 sites, jobs with more than 5 components will have several components running at the same site. For parallel jobs, there is a preference for 2 and 4 components. For multi-component jobs there is a preference for 2, 4, and 16 processors. Various inter-arrival time distributions are used to generate the submission time of workload jobs. Component requests are either fixed (specifying the name of a grid site) or nonfixed (leaving the scheduler to decide).

The lower performance of parallel jobs (workloads `smpi1` and `ibis1`) when compared to one-processor jobs (workload `gmark1`), is caused by the parallel jobs need to allocate resource sets, as opposed to allocating single resources. Ibis jobs also suffer from the small time limit coupled with inaccurate estimations.

The turnaround time of an application can vary greatly (see Table 2), due to different parameter settings, or to varying system load. The variations in the application runtimes are due to different parameter settings. As expected, the percentage of the applications that are actually run (Table 2, column *Run*) depends heavily on the job size and system load. The success rate of jobs that *did* run shows little variation (Table 2, column *Run+Success*).

### 6.1.2 *What-if* analysis

We consider three use cases for illustrating GrenchMark's capabilities for *what-if* analysis : system change, grid inter-operability, and special situations. In the first two cases we assume that the environments under scrutiny have been thoroughly studied and their workloads modeled.

First, we consider the case where a testing environment would be placed under (much) more strain. Our testing environment, the DAS system, has actually *hot-swapped* its resource manager, because it could not cope with the increasing number of job submissions. The work of all the DAS community was negatively

Table 3: The results for the first case of *what if* analysis.

| Workload | Submit rate vs. original | # of jobs | Job size (# of CPUs) | Submit Errors |
|---|---|---|---|---|
| DAS2-FS3-1x | 1x | 14 | 1-50 | 0% |
| DAS2-FS3-10x | 10x | 71 | 1-50 | 0% |
| DAS2-FS3-25x | 25x | 102 | 1-50 | 4% |
| DAS2-FS3-50x | 50x | 426 | 1-50 | 24% |
| DAS2-FS3-100x | 100x | 739 | 1-50 | 41% |

Table 4: The results for the second case of *what if* analysis.

| Workload (the PWA index is given in paranthesis) | # of jobs | Component no. | Component size | Success Rate |
|---|---|---|---|---|
| DAS2-FS3 (15) + OSC (10) | 1103 | 1-4 | 1-32 | 76% |
| DAS2-FS3 (15) + CTC (4) | 863 | 1-8 | 1-10 | 70% |
| DAS2-FS3 (15) + SDSC'96 (3) | 873 | 1-8 | 1-32 | 77% |

affected for a period of two weeks. Such a change could have been prevented if the question *What if the current users would submit 10 times more jobs in the same amount of time? Or 50 times, or 100 times...* would have been answered at the system installation, or during a quiet period. Furthermore, the project sponsoring this research (see Acknowledgements) has as its main objective making the DAS infrastructure easily available to the Dutch academics, and requires this *what-if* question to be answered before the new settings are released to the public. We take a thoroughly studied trace [9] of the DAS system and re-run it in the new environment. The trace was recorded when the previous resource manager was still in place, and contains over 425,000 jobs run throughout 2003. To limit the testing period, we consider only the jobs submitted to one of the five DAS clusters (over 65,000), limit the workloads to at most 1000 jobs, and ensure that the resulting workloads have a submission span of maximum three hours. All these requirements were met through the workload definition language; no tool external to GrenchMark, e.g., MySQL, was used. We also scale the jobs submission times to make them 10, 25, 50, and 100 times smaller than the original submission times. Table 3 details the filtered and scaled traces and shows the percentage of the submit errors, from the total number of submitted jobs. We conclude that the new resource management system (including Koala) can handle an up to 10 times increase in the submission rate, for the user base characterized by the input traces.

Second, we consider the case where our testing environment would also be used by the users of another environment and we want to find out *what is the success rate of the jobs submitted by these combined communities?*. This situation occurs when from two existing production environments one environment is put temporarily or permanently out of production, and only one environment remains to run the submissions of jobs from both user communities. For the extended use case in which two environments share their resources, suffices for the user to run the generated workload on both systems at the same time; for this purpose, GrenchMark allows a generated workload to be accurately replayed on different systems (even if those systems are simulated). We took the same trace as in the first *what-if* analysis case, and combined it with either of the OSC, the CTC, or the SDSC'96 traces from the Parallel Workloads Archive (PWA). We selected just the jobs with runtimes below 900 seconds. We define a *trace job set* as a set consecutive jobs from a trace for which the submission interval is 3 hours. To solve the problem of not selecting the most demanding, nor the least demanding trace job set, we first select the *top 100 sets*, sorted by their number of jobs (*size*). Then, we compute the average number of jobs for the top 100 sets, and we choose the set whose size is the closest to this average. Table 4 shows the detailed structure of the generated workloads, and the success rate for running them. The observed success rate was above 70% for all tests. We conclude that the new DAS (including Koala) can handle the proposed combinations of communities, with a reasonable success rate for submitted jobs.

Third, we consider the case where *the user wants to test the outcome of the system being suddenly subjected to a large number of submitted jobs (bursts)*. Using workloads based on synthetic applications, we generated two bursts, and for each a background load, filling 25% and 5% of the system's capacity, respectively. During the tests, the background load generated by other users was between 0% and 5%. We restrict our tests to one of the DAS clusters (fs4). Figure 5 shows the system utilization graph during the period when the

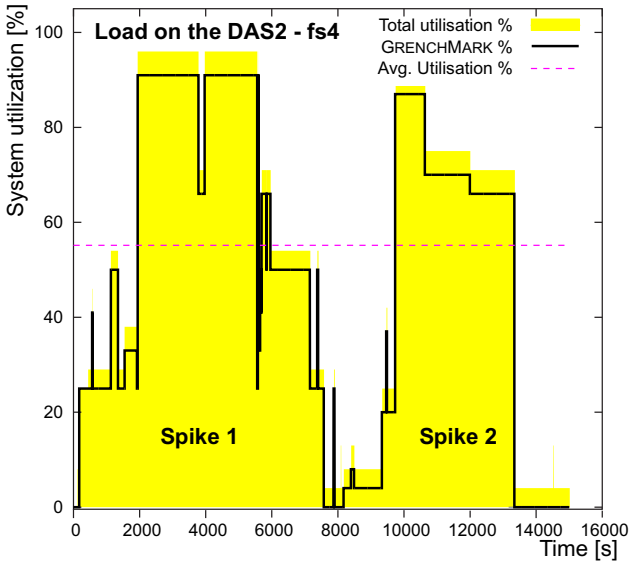


Figure 5: The utilization of the DAS fs4 cluster for *bursty* submissions.

Table 5: The results for system functionality testing.

| Workload | Types of applications | # of CPUs | Component no. | Component size | Success Rate |
|---|---|---|---|---|---|
| `gmark+` | synthetic, seq. & MPI | 1-128 | 1-15 | 1-32 | 85% |
| `ibis+` | various, Ibis | 2-32 | 1-8 | 2-16 | 65% |
| `unitary` | `gmark+` & `ibis+` | 1-32 | 1-8 | 1-32 | 90% |

two burst workloads are submitted. The system performed correctly under strain; the maximum system utilization is 95%, of which 90% is generated by the burst workloads. Two spikes can be clearly identified on the utilization graph. As expected, running the first burst workload (spike 1) takes longer, due to the higher background utilization of the system.

## 6.2 Functionality testing in grid environments

This section presents two cases for showing how GrenchMark can be used for *functionality testing*: system functionality testing, and periodic system testing.

### 6.2.1 System functionality testing

We consider the case where the user wants to *test the system's capability to reliably run various types of jobs at the same time*. We combine the workloads used in the case of performance testing and obtained mixed workloads. Table 5 displays the composition of the two experimental workloads, and the success rates for their execution. The `gmark+` workload comprises a mix of parallel and sequential synthetic applications also used in workloads `gmark1` and `smpi1` (see Table 1). As expected, the success rate of `gmark+` is lower than for `gmark1`, but higher than for `smpi1`. The success rate fluctuates with the number of `smpi1` jobs: the lower this number, the lower the failure rate (sequential jobs are more stable). The same considerations can be applied for the `ibis+` and `unitary` workloads. For the `ibis+` workload, the success rate is 15% lower than normal due to severe runtime underestimations (see also Section 6.1.1). We conclude that the DAS and Koala can reliably run the GrenchMark jobs.

### 6.2.2 Periodic system testing

We consider a case where the user wants to *periodically test a system*. This type of testing helps identifying system problems, or can be used for obtaining the current performance of various system components. We

Table 6: Results of the comparison of success rates for single-site vs. co-allocated jobs.

| Workload | Types of applications | # of CPUs | Component no. | Component size | Success Rate |
|---|---|---|---|---|---|
| `single-site` | synthetic, MPI | 2-32 | 1 | 2-32 | 98% |
| `coallocated` | synthetic, MPI | 2-32 | 1-8 | 1-16 | 77% |
| `coallocated+` | synthetic, MPI | 2-128 | 1-15 | 1-16 | 71% |

Table 7: Comparison of success rates for unitary vs. composite jobs with or without execution fault tolerance.

| Workload | # of Jobs/ Sub-Jobs | Success rate when fault tolerance is ON | OFF |
|---|---|---|---|
| `unitary` | 100/100 | 90% | 100% |
| `composite` | 10/108 | 70-90% | 100% |

scheduled a set of simple test workloads for daily submission, from various DAS gateways. This helped to quickly identify and solve a critical deployment problem: the configuration file of Koala on one of the gateways was incorrect and caused the submission test to fail; if left uncorrected, users submitting jobs from that gateway would have been unable to submit some of their jobs through that gateway.

## 6.3 Comparing grid settings

This section presents two situations in which GrenchMark can be used for *comparing* grid settings.

### 6.3.1 Single site vs. co-allocated jobs

*We consider a comparison between the success rates of single-site and co-allocated jobs in a grid environment without reservation capabilities.* We devised three workloads, one with single-site jobs, one with the same jobs co-allocated at various sites, and one with larger jobs co-allocated at various sites. Table 6 shows the detailed structure and the success rate for the three workloads. The single-site jobs (workload `single-site`) have a higher success rate than co-allocated jobs (workloads `coallocated` and `coallocated+`), in this case by over 20%. This is due to the atomic reservation problems of co-allocation: local users can acquire resources selected for co-allocation just before they are claimed by the co-allocating scheduler. Since the average load in the DAS system is low (e.g., below 25% [9]), small and large co-allocated jobs have almost the same success rate (enough machines are available).

### 6.3.2 Unitary vs. composite jobs

We consider a *comparison between the success rate of unitary and composite applications in a grid environment without reservation capabilities, and with or without fault tolerance.* We considered only DAG-based composite applications for which the sub-jobs are unitary applications. We define a job that *can run* as a job for which all its dependencies have been met. Since Koala does not support the execution of composite jobs, we built a simple execution tool, which executes jobs that can run as soon as possible. The execution tool also allows for flexible fault tolerance schemes; we use here a simple *retry failed* mechanism with sub-job granularity. For different workflow types, execution tools, and mechanisms, we refer to [16]. Table 7 shows the success rate for unitary and composite jobs, with or without fault tolerance. Without fault tolerance, the success rate of composite jobs drops dramatically if the first sub-jobs in the DAG's topological sort fail. With fault tolerance, for a system with a high success rate for jobs, e.g., the DAS, the simple retry mechanism yields complete reliability in running correct jobs.

# 7 Related work

A significant number of projects have tried to tackle the Grid performance assessment problem from different angles: modeling workloads and simulating their run under various environment assumptions [15, 5, 3], attempting to produce a representative set of grid applications like the NAS Grid Benchmarks [8], creating

synthetic applications that can assess the status of grid services like the GRASP project [4], and creating tools for launching benchmarks and reporting results like the GridBench project [13]. GrenchMark is the natural complement to these approaches, by offering a much larger application base, more advanced workload modeling features, and the ability to replay existing workload traces. In addition, GrenchMark can be used for much more than just Grid performance evaluation (see Section 6).

The modeling/simulation approach is almost exclusively based on traces which are now part of the Parallel Workloads Archive. The major hurdle for this approach is to prove the representativeness of simulation results for real grid environments.

In [8], the authors propose a small set of parallel applications as Grid benchmarks. Simple workloads are defined for the applications, in that the running parameters and the order in which the applications are to be run are fixed. The drawbacks of this approach are that the applications are only representative for a restricted research area (here, computational fluid dynamics), make very little use of Grid components (only Grid-enabled MPI and a scheduler), and cannot adapt to the dynamic behavior of Grids (they require fixed resource sizes, and have no fault-tolerance, migration, or check-pointing features).

In [4], a small set of applications are specifically designed to test specific aspects of Grids functionality (*probes*). The applications assume the existence of common Grid components, like a global information system, or a file-transferring service. No attempt to form workloads with these applications is made.

In [13], a benchmark launching tool is proposed. This tool has the ability to launch benchmarks and display their results, and can be coupled with many of the existing HPC benchmarks. However, it has very limited workload modeling features, and cannot replay real traces.

# 8 Conclusion and future work

This paper has presented GrenchMark, a framework for synthetic grid workloads generation and submission. The framework supports various workload modeling primitives, comes with over 35 synthetic and real applications, and is portable, flexible, and extensible. We have designed and implemented GrenchMark, and have deployed it in the DAS, our 400-processors grid environment.

We have shown evidence that GrenchMark can be successfully used for grid system analysis, functionality testing in grid environments, and comparing different grid settings. We have presented various examples of how GrenchMark can be used for performance tests, what-if analysis, system functionality tests, periodic system tests, a single vs. co-allocated jobs comparison, and a unitary vs. composite jobs comparison. We have shown how these tests can be extended for different goals than those of this paper, e.g., other performance metrics and other types of tests. Last, but certainly not least, GrenchMark was instrumental in making the Koala grid scheduler reliable on the DAS, and releasing it for the DAS user community.

We are currently extending GrenchMark with support for malleable jobs. For the future, we plan to use GrenchMark for testing and comparing more grid settings, both in simulations and real situations.

# Availability

The official GrenchMark web site, including documentation and a freely available distribution, is located at: `http://grenchmark.st.ewi.tudelft.nl/`.

# Acknowledgments

This work was carried out in the context of the Virtual Laboratory for e-Science project (`www.vl-e.nl`), which is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W), and which is part of the ICT innovation program of the Dutch Ministry of Economic Affairs (EZ). Other people have contributed as well, and should not be forgotten. We thank Hashim Mohamed and Wouter Lammers, for their work on Koala, and Jason Maassen and Rob van Nieuwpoort, for their work on Ibis.

# References

[1] H. E. Bal et al. The distributed ASCI supercomputer project. *Operating Systems Review*, 34(4):76–96, October 2000.
[2] F. Berman, A. Hey, and G. Fox. *Grid Computing: Making The Global Infrastructure a Reality*. Wiley Publishing House, 2003.
[3] A. I. D. Bucur and D. H. J. Epema. Trace-based simulations of processor co-allocation policies in multiclusters. In *Proc. of the 12th IEEE HPDC*, pages 70–79. IEEE Computer Society, 2003.
[4] G. Chun, H. Dail, H. Casanova, and A. Snavely. Benchmark probes for grid assessment. In *IPDPS*. IEEE Computer Society, 2004.
[5] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On advantages of grid computing for parallel job scheduling. In *CCGRID*, pages 39–49. IEEE Computer Society, 2002.
[6] C. Ernemann, B. Song, and R. Yahyapour. Scaling of workload traces. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *JSSPP*, volume 2862 of *LNCS*, pages 166–182. Springer, 2003.
[7] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In D. G. Feitelson and L. Rudolph, editors, *JSSPP*, volume 1459 of *LNCS*, pages 1–24. Springer, 1998.
[8] M. Frumkin and R. F. V. der Wijngaart. Nas grid benchmarks: A tool for grid space exploration. *Cluster Computing*, 5(3):247–255, 2002.
[9] H. Li, D. Groep, and L. Wolters. Workload characteristics of a multi-cluster supercomputer. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *JSSPP*, LNCS, vol.3277, pages 176–194. Springer, 2004.
[10] H. Mohamed and D. Epema. Experiences with the koala co-allocating scheduler in multiclusters. In *Proc. of the 5th IEEE/ACM Int'l Symp. on Cluster Computing and the GRID (CCGrid2005)*, May 2005.
[11] W. Smith, I. Foster, and V. Taylor. Predicting application run times with historical information. *J. Parallel Distrib. Comput.*, 64(9):1007–1016, 2004.
[12] A. Snavely, G. Chun, H. Casanova, R. F. V. der Wijngaart, and M. A. Frumkin. Benchmarks for grid computing: a review of ongoing efforts and future directions. *ACM SIGMETRICS Perform. Eval. Rev.*, 30(4):27–32, 2003.
[13] G. Tsouloupas and M. D. Dikaiakos. GridBench: A workbench for grid benchmarking. In P. M. A. Sloot, A. G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, *EGC*, volume 3470 of *LNCS*, pages 211–225. Springer, 2005.
[14] R. V. van Nieuwpoort, J. Maassen, G. Wrzesinska, R. Hofman, C. Jacobs, T. Kielmann, and H. E. Bal. Ibis: a flexible and efficient java-based grid programming environment. *Concurrency & Computation: Practice & Experience.*, 17(7-8):1079–1107, June-July 2005.
[15] A. M. Weil and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):529–543, 2001.
[16] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD Rec.*, 34(3):44–49, 2005.