

# The Performance of Bags-of-Tasks in Large-Scale Distributed Systems

Alexandru Iosup  
Delft University of Technology  
Mekelweg 4, 2628 CD  
Delft, The Netherlands  
A.iosup@gmail.com

Ozan Sonmez  
Delft University of Technology  
Mekelweg 4, 2628 CD  
Delft, The Netherlands  
O.O.Sonmez@tudelft.nl

Shanny Anoep  
Delft University of Technology  
Mekelweg 4, 2628 CD  
Delft, The Netherlands  
S.Anoep@gmail.com

Dick Epema  
Delft University of Technology  
Mekelweg 4, 2628 CD  
Delft, The Netherlands  
D.H.J.Epema@tudelft.nl

## ABSTRACT

Ever more scientists are employing large-scale distributed systems such as grids for their computational work, instead of tightly coupled high-performance computing systems. However, while these distributed systems are more cost-effective, their heterogeneity in terms of hardware, software, and systems administration, and the lack of accurate resource information leads to inefficient scheduling. In addition, and in contrast to the workloads of tightly coupled high-performance computing systems, a large part of the workloads submitted to these distributed systems consists of large sets (bags) of sequential tasks. Therefore, a realistic performance analysis of scheduling bags-of-tasks in large-scale distributed systems is important. Towards this end, we introduce in this paper a realistic workload model for bags-of-tasks, and we explore through trace-based simulations the design space of scheduling bags-of-tasks. Finally, we identify three new scheduling policies that use only inaccurate information when scheduling, and we compare them against known classes of proposed scheduling policies.

## Categories and Subject Descriptors

- C.1.4 [Parallel Architectures]: Distributed architectures;
- C.4 [Performance of Systems]: Design studies;
- C.4 [Performance of Systems]: Modeling techniques;
- D.4.1 [Process Management]: Scheduling;
- D.4.7 [Organization and Design]: Distributed systems;
- D.4.8 [Performance]: Modeling and Simulation

## General Terms

Performance, Design, Algorithms

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HPDC'08, June 23–27, 2008, Boston, Massachusetts, USA.  
Copyright 2008 ACM 978-1-59593-997-5/08/06 ...\$5.00.

## Keywords

Bags-of-Tasks, batches of jobs, performance evaluation, workload modeling, scheduling, prediction, large-scale distributed systems, multi-cluster systems, trace-based simulation.

## 1. INTRODUCTION

Over the last decade, large-scale distributed computing systems have become the alternative to the traditional, tightly coupled high-performance computing systems for executing scientific tasks. While such distributed systems can be very cost-effective and easily scalable, due to resource heterogeneity and to the lack of accurate resource information, scheduling jobs in such systems can be a challenge. In addition, even though the local and wide-area interconnections in such systems have improved markedly and efficient wide-area communications libraries are now available, it turns out that a large fraction of the jobs in the workloads imposed on such systems is due to sequential applications, often submitted in the form of Bags-of-Tasks (BoT). In this paper we present a realistic and systematic investigation of the performance of scheduling BoTs in large-scale distributed computing systems.

Even though much computer-science research has been devoted to parallel processing, sequential jobs are still predominant in the real world of high-performance and high-throughput computing, and often, large numbers of sequential jobs (that share the same executable) are submitted in single groups. The reasons for this phenomenon are the relatively high network latencies, the complexities of parallel programming models, and the nature of the scientific computational work (e.g., repeated simulations, parameter sweeps). In recent work [23], we have presented evidence on the predominance and on the characteristics of BoTs in two categories of large-scale distributed computing systems: multi-cluster grids and large-scale cycle-scavenging systems (e.g., Condor-based systems). However, no workload model for BoTs is currently available. Without such a model, researchers employ unrealistic or real traces. The former may lead to tuning the system for cases that cannot exist in practice; the latter have limited use for platforms different from those where they were collected.

Scheduling BoTs to processors in dynamic large-scale dis-

tributed computing systems to achieve for instance the minimal (average) makespan, is a complicated problem. As a result, many scheduling policies for BoTs have been proposed for a variety of systems. However, most of these policies have been proposed for tightly-coupled systems, or at least, they do not consider heterogeneous resources. The few solutions that can be applied in practice in large-scale distributed computing systems [9, 12, 19, 28] assume that the scheduler has either no information or mostly accurate information at its disposal; the former are shown to be inefficient for specific workload characteristics [9, 28]. Adding to the lack of scheduling policies, in this work we show that with respect to information-awareness, these solutions do not cover the whole scheduling policies design space. Thus, there is a need for a realistic and systematic investigation of the performance of BoT scheduling in large-scale distributed computing systems.

This paper presents the following three contributions towards solving the problem of scheduling BoTs:

1. We propose a systematic approach to evaluating the scheduling of BoTs in large-scale distributed computing systems; with this approach we identify three classes of scheduling policies that have not been investigated previously (Section 2);
2. We propose a workload model for BoTs and we validate this model using long-term traces from seven large-scale distributed computing systems (Section 3);
3. We perform a realistic and comprehensive investigation of the performance of BoTs in large-scale distributed computing systems (Sections 4 and 5).

## 2. A SCHEDULING MODEL FOR BOTS

In this section we present a scheduling model for BoTs in large-scale distributed computing systems consisting of four components. In Section 2.1, we describe the models of the system and of the jobs submitted to it. The system model considers clusters of resources. In Section 2.2 we present the resource management architectures, which add structure to the set of clusters as to how their resources are jointly managed. Deciding which tasks to run where is in our model a two-step process: first from the waiting tasks in the system an *eligible set* is created using one of the *task selection policies* (Section 2.3), and then the tasks from the eligible set are mapped to resources using one of the *task scheduling policies* (Section 2.4).

Our model extends the current state-of-the-art in several ways. The resource management architecture and the task selection policies have not been explicitly included in previous BoT scheduling models. For the task scheduling policies, previous models [9, 28] have considered that the resource performance (e.g., speed, SPECInt2006 value [38]) or the task runtime are not (accurately) known by the scheduling policy, but that at least one of them is accurately known. In contrast to these models, our model considers that at the same time the information about both the processor performance and the task runtime may be inaccurate or even missing.

### 2.1 System and Job Model

In our model of large-scale distributed computing systems we assume that the computing resources (processors) are

grouped in clusters. The processors may have different performance across clusters, but within the same cluster they are homogeneous. The workload of the system consists of jobs submitted by various users; each of the jobs is a bag of sequential tasks (possibly only one). We employ the SPEC CPU benchmarks model for application execution time [38], that is, the time it takes to finish a task is inversely proportional to the performance of the processor it runs on (see also Section 4.1). Upon their arrival into the system the tasks are queued, waiting for available resources on which to be executed. Once started, tasks run to completion, so we do not consider task preemption or task migration during execution. Instead, tasks can be replicated and canceled, or migrated before they start.

### 2.2 Resource Management Architectures

The complete set of clusters is operated as one large-scale distributed computing systems using a resource management architecture which dictates how jobs are distributed across the resources in the system. In our model, each of the clusters has its own local resource manager (RM), but other RMs may be employed to build a complete architecture. A global resource manager (GRM) is an RM that can submit tasks for execution to another RM.

Each RM in the system executes the same scheduling procedure upon the arrival of new BoTs, on the completion of tasks, and also periodically. The scheduling procedure is as follows. First, the RM calls the task-selection policy, which selects from the RM’s current queue the *eligible set* of tasks. Then, the RM executes the eligible set using the task-scheduling policy, which in turn sorts the eligible set and/or ranks the resources to create a schedule. Only after all the tasks in the schedule are completed is a new eligible set generated.

In previous work [22], we have presented a taxonomy of resource management architectures; in our taxonomy clusters can work independently, or be combined into a centralized or decentralized architecture. In this work we use three resource management architectures, one based on independent clusters, one centralized, and one decentralized. Below we describe these architectures:

1. **SEparated Clusters (sep-c)** Each cluster operates separately with its own local RM and its own local queue to which jobs arrive. Each user can submit tasks to exactly one RM.
2. **Centralized Scheduler with Processor monitoring (csp)** Each cluster operates separately with its own local RM and its own local queue to which jobs arrive. In addition, a global RM with a global queue operates on top of the cluster RMs. The users submit tasks only to the global system queue. When the global RM observes that a cluster has idle resources, it moves some of the tasks on the global queue to that cluster. The information about the number of free processors is gathered periodically by a monitoring service.
3. **Condor-like, with Flocking (fcondor)** This models a Condor-like architecture with flocking capabilities [14]. Similarly to **sep-c**, each cluster operates separately with its own local RM and its own local queue to which jobs arrive. However, here each user can submit tasks to any RMs. A user keeps submitting tasks to the

same RM while that RM starts the tasks immediately; when tasks start to be queued, the user will switch to another RM, in round-robin order.

In our model, a GRM may submit at most one task at a time to another RM; the target RM will in this case receive BoTs with one task. This ensures that the GRM can control the order in which its tasks are considered at the remote RM. Note that for `sep-c` there is no GRM, but there exist local users.

### 2.3 Task Selection Policies

The RM uses its task selection policy to select from the RM’s local queue the *eligible set* of tasks. We investigate in this work seven task selection policies, the first two of which do not take into account the user who submits a task:

1. **S-T** The Select-Tasks policy selects all the tasks in the system. For each arriving BoT, the set of its tasks is added to the eligible set.
2. **S-BoT** The Select-BoTs policy selects BoTs in the order of their arrival. When the eligible set has become empty, the set of tasks of the selected BoT are considered the new eligible set.
3. **S-U-Prio** The Select-User-Priority policy assumes that each user in the system has a unique priority. It selects all the tasks of the user with the highest priority. Ideally, each user has a unique priority, which distinguishes the user’s resource usage rights from any other’s. In practice, a system will be configured with just a few (e.g., up to four) distinct priorities. We use in this work the ideal scenario; in many settings, its performance gives an upper bound of the performance of the practical scenario.
4. **S-U-T** The Select-User-Tasks policy aims at the equal sharing of resources among the system users: It first selects the user with the lowest resource consumption, and then it selects all the tasks of the selected user.
5. **S-U-BoT** Similarly to **S-U-T**, the Select-User-BoT policy aims at the equal sharing of resources among the system users: It first selects the user with the lowest resource consumption, then it orders the selected user’s BoTs in their order of arrival into an ordered set. From this set, the tasks of the first BoT are considered the new eligible set.
6. **S-U-GRR** The Select-User-Global-Round-Robin algorithm selects the next user in round-robin order. It then adds all waiting tasks of this user to the eligible set. If a user submits additional BoTs during his turn, the corresponding tasks will not be considered for selection during this turn.
7. **S-U-RR** The Select-User-Round-Robin policy is a variation of S-U-GRR, where only one task is selected per user at each round. Under S-U-RR, a BoT of size  $N$  will receive complete service after exactly  $N$  rounds. This algorithm is similar to the WFQ algorithm for scheduling packets over the network [35], with the main difference of tasks not having a known runtime at selection time, as opposed to network packets having a pre-assigned number of bits to transfer.

		Task Info.		
		K	H	U
Resource Info.	K	ECT [32], FPLT [33] MaxMin [9]	ECT-P*	FPF*
	H	DFPLT [12] MQD [28]	-	-
	U	STFR*	-	RR [19] WQR [12]

**Table 1: An information availability framework. K, H, and U stand for information known a-priori, based on historical data, and unknown, respectively. The scheduling policies marked with  $\star$  have not been previously studied in the context of BoT scheduling.**

For all task-selection policies that require it, the resource consumption is simply computed as the sum of the past usage and usage of currently running tasks. The usage of a task is computed as the CPU time spent by it until its completion (the current moment) for tasks that have (not yet) completed. Given the range of the runtimes of the tasks in our traces and experiments, a precision on the order of seconds is enough for computing and accurately measuring these CPU time values.

### 2.4 Task Scheduling Policies

There exist many scheduling policies for BoT workloads in large-scale distributed computing systems [9, 12, 19, 28]. We categorize such policies according to the information policy used for resources and tasks, where a piece of information can be either fully Known (K), known from Historical records (H), or fully Unknown (U). In this work we consider only two pieces of information (the *information set*): the performance of a processor and the execution time of a task (e.g., on a reference processor). Then, a task-scheduling policy can be characterized in terms of its information usage by a tuple  $(R, T)$ , with  $R$  and  $T$  the information policy for resources and for tasks, respectively. We map several scheduling policies to this characterization in Table 1. Most of the existing scheduling policies are either  $(U, U)$  or  $(K, K)$ ; in practice, Condor [39] uses by default an  $(U, U)$  policy, AppLeS supports several  $(U, U)$  and  $(K, K)$  heuristics [5], MyGrid implements an  $(U, U)$  policy [11]. There are no scheduling policies of types  $(U, K)$ ,  $(K, U)$ ,  $(U, H)$ ,  $(H, U)$ , and  $(H, H)$ .

To address the main goal of this work, a systematic approach to evaluating BoT scheduling in large-scale distributed systems, we propose one simple policy for each of the types  $(U, K)$ ,  $(K, U)$ , and  $(K, H)$ ; we will investigate policies of the type  $(H, H)$  in future work. The  $(U, K)$  and  $(K, U)$  policies give an upper bound of the achievable performance of  $(U, H)$  and  $(H, U)$  policies, respectively. The scheduling policies used in this work are described below:

1. **ECT**  $(K, K)$  The Earliest Completion Time policy assigns each task to the resource (cluster or processor) that leads to the earliest completion time possible. If the resource is a cluster, it also takes into account the cluster’s queue when computing the earliest completion time. It is a Gantt chart-based scheduling policy [10]; other examples of similar policies include MaxMin, MinMin, and (X)Sufferage [9].
2. **FPLT**  $(K, K)$  The Fastest Processor Largest Task policy assigns the largest task to the fastest processor available.

3. **RR** ( $U, U$ ) The Round-robin Replication policy first assigns all the tasks to processors, in the initial order of the eligible set. After finishing all tasks in the eligible set, it replicates tasks at most once on the resources that become available, in round-robin order.
4. **WQR** ( $U, U$ ) The Work Queue with Replication policy differs from RR in that it can replicate tasks several times instead of only once. The number of replicas is appended to name of the scheduling policy, e.g., WQR-1 replicates tasks once (and is identical to RR).
5. **DFPLT** ( $H, K$ ) The Dynamic Fastest Processor Largest Task policy assumes that the resource performance is dynamic over time. On the completion of a task, the performance of the resource on which the task was executed is (re-)computed, and the resource receives a performance rank. This policy assigns the largest task to the resource with the highest performance rank.
6. **ECT-P** ( $K, H$ ) The ECT with task runtime Prediction policy operates similarly to ECT, but uses predicted instead of real task runtime values.
7. **STFR** ( $U, K$ ) The Shortest Task First with Replication always assigns the shortest task first. After finishing all tasks in the eligible set, it replicates tasks at most once on the resources that become available, in round-robin order.
8. **FPF** ( $K, U$ ) The Fastest Processor First assigns the tasks in the initial order of the eligible set. Each task is assigned to the fastest available processor.

The information set can be extended to more dimensions than just two. However, the two selected pieces of information already foster non-trivial customization, e.g., the execution time of a task can be extended to include the job setup and removal. If the job execution model of the system does not allow for the decoupling of job data from the job execution, as is the case for many cluster managers used in practice, the execution time of a task can also include the data transfer time to/from the execution place. Similarly, the execution time can include the setup of a virtual environment that is needed for executing a job.

### 3. A WORKLOAD MODEL FOR BOTS

In this section we present a workload model for large-scale distributed computing systems that focuses on Bag-of-Tasks applications. Our workload model is useful for experiments in real environments, simulations, and mathematical analysis.

While four decades of research lead to many valuable models for computing system workloads [36, 31, 29], these studies focus on modeling single (sequential or parallel) tasks. There is no study that models explicitly BoTs in the context of large-scale distributed computing systems. However, we have shown in previous work [23] that the workloads of various types of large-scale distributed computing systems, and in particular of cluster-based grids, are dominated by BoTs, both in number of tasks included in such groupings and in resource consumption for their execution. Having analyzed in previous work the characteristics of BoTs, we present in this section the first explicit model of BoTs in large-scale distributed computing systems.

Trace ID	System		Trace	
	Name	Size [CPUs]	Duration [Years]	Size [tasks]
T1	DAS-2	400	1.5	1.1M
T2	Grid'5000	~2500	2.5	1.0M
T3	NGS	378	3	0.6M
T4	AuverGrid	475	1	0.4M
T5	SHARCNET	6,828	1	1.1M
T6	LCG	24,515	0.03	0.2M
T7	NorduGrid	~2000	2	0.8M

**Table 2: Characteristics of the seven grid traces used to validate the BoT workload model.**

### 3.1 Model Overview

The model for BoTs proposed in this paper focuses on four aspects: the submitting user (Section 3.2), the BoT arrival patterns (Section 3.3), the BoT size (Section 3.4), and the intra-BoT (task) characteristics (Section 3.5). For each aspect we have selected several important characteristics.

For each characteristic, the modeling goal is to find a well-known statistical distribution that can be later sampled to generate realistic characteristic values. Seven grid workload traces from the Grid Workloads Archive (GWA) [1] are used to validate the BoT model. Table 2 summarizes the characteristics of the seven systems and their traces. Six of the traces presented in this work have been collected for periods of over one year, five traces collect data for over half a million jobs, and four traces have been collected from systems with over a thousand processors. We conclude that the traces used in this work are representative for the workloads of large-scale distributed computing systems; for a complete description of the traces we refer to the GWA web site and to [25]. The seven traces do not necessarily include information on BoTs. Our previous work [23] discusses how to extract such information from logs that contain only information on independent tasks.

The real (trace) data corresponding to each of the characteristics are fitted to the following candidate distributions, each of which has low complexity [16] and is used extensively in the analysis of computer systems: exponential, hyper-exponential, normal, log-normal, gamma, and Weibull. The fitting process uses the Maximum Likelihood Estimation (MLE) method [2], which delivers good accuracy for the large data samples specific to workload traces. Then, goodness-of-fit tests are used to assess the quality of the fitting for each distribution, and to establish a best fit for each of the model parameters. For each candidate distribution with the parameters found during the fitting process, we formulate the hypothesis that the data are derived from it (*the null-hypothesis* of the goodness-of-fit test). We use the Kolmogorov-Smirnov test (KS-test) [30] for testing the null-hypothesis. The KS-test statistic  $D$  estimates the maximal distance between the CDF of the empirical distribution of the input data and that of the fitted distribution. The null-hypothesis is rejected if  $D$  is greater than the critical value obtained from the KS-test table. The KS-test is robust in outcome (i.e., the value of the  $D$  statistic is not affected by scale changes, like using logarithmic values). The KS-test has the advantage over other traditional goodness-of-fit tests, like the t-test or the chi-square test, of making no assumption about the distribution of the data (note: Pearson's chi-square test is applied to binned data (e.g., a data

Trace ID	User Ranking	Bag-Of-Tasks			Task	
		IAT	Daily Cycle	Size	ART	RTV
T1	Z(1.25,333)	W(4.06,7.91)	G(2.62,0.13)	W(1.75,2.91)	N(1.78,3.87)	W(1.64,10.21)
T2	Z(1.39,481)	W(4.27,8.42)	W(1.57,20.54)	G(2.47,1.64)	N(2.31,4.97)	N(5.54,9.56)
T3	Z(1.30,379)	W(4.94,8.48)	W(1.64,25.42)	W(2.02,1.58)	N(3.50,3.51)	G(1.79,0.29)
T4	- (see text)	W(3.87,7.33)	W(1.72,25.49)	W(1.78,1.51)	G(3.55,0.47)	N(6.41,11.73)
T5	Z(1.25,412)	W(4.17,7.65)	W(2.44,28.99)	W(1.37,1.89)	N(3.06,7.45)	W(1.93,13.65)
T6	Z(1.32,216)	W(4.05,6.48)	W(1.71,23.86)	N(1.33,2.71)	LN(1.82,0.34)	W(2.85,14.21)
T7	Z(1.36,387)	N(1.97,8.00)	W(1.62,22.18)	W(1.80,2.17)	N(2.76,9.04)	W(2.86,16.58)
Avg	Z(1.31,368)	W(4.25,7.86)	W(1.79,24.16)	W(1.76,2.11)	N(2.73,6.1)	W(2.05,12.25)

**Table 3: The parameter values for the best fits of the statistical distributions to the BoT model for the seven studied traces. N, LN, W, and G stand for the normal, lognormal, Weibull, and gamma distributions, respectively. Z stands for the Zipf distribution with two parameters:  $\alpha$  and the number of unique users (ranks).**

histogram), but the value of the test depends on the how the data is binned). The KS-test can disprove the null-hypothesis, but *cannot* prove it. However, a lower value of  $D$  indicates better similarity between the input data and data sampled from the theoretical distributions. We use this latter property to select the best fits.

For each model characteristic, the candidate distribution with the lowest  $D$  value is selected for each workload trace; the parameters of the best fit distributions are recorded as an instance of the model of the respective trace. The selected distributions for each trace and their parameters are depicted in Table 3. Trace T4 does not include user information.

We define a theoretical "average system" as the system that has the average properties of the seven systems considered in this work. Using the average system properties we can generate synthetic yet realistic traces, without using a single real system as a reference. We build the average system properties as follows. For each model characteristic, a candidate distribution that has the lowest average  $D$  value over all seven traces is selected as the average system fit. When for two candidate distributions the difference of their  $D$  values is below 0.01, the distribution closest to the average system fit is selected. The data for each trace are then fit independently to the candidate distribution, resulting in a set of best fit parameters. The parameters of the average system represent the average of this set. The 'Avg' row in Table 3 presents the parameters of the average system.

Similarly to the average workload models built for other types of systems [31], we cannot claim that the model of our average system workload, including the parameter values, represents the user behavior of an actual system. Instead, the main strength of this model is that it represents a common basis for the traces from which it has been extracted. By varying the parameters of the model characteristics, traces that are statistically similar to the GWA traces can be obtained for any system. By using the selected model parameters (the 'Avg' row), results obtained by different researchers can be compared.

## 3.2 Submitting User

In many computing environments, a small number of users dominate the workload [13, 21]. The Zipf distribution is used in many fields for characterizing "rank data", where the relative frequency of a given rank is determined by the distribution function [34]. The Zipf distribution was fitted to the user ranking based on their relative job submission frequencies. First, the users are ranked based on number of submitted jobs in descending order (the lowest rank is equal

to the number of unique users in the trace). The probabilities associated with each rank are calculated by dividing the number of submitted jobs for each user of that rank by the total number of jobs in the trace.

## 3.3 BoT Arrival Patterns

The BoT arrival patterns are modeled in two steps: first the inter-arrival time (IAT) between consecutive BoT arrivals during peak hours, and then the IAT variations caused by the daily submission cycle.

Similar to the results in [21], the hours between 8AM and 5PM are found to be "peak hours", with significantly more arrivals than during the rest of the day. Only the data for BoTs arriving during the peak hours are considered when modeling the IAT. According to established modeling practice [31], a logarithmic transformation with base 2 is applied to these data to reduce the range and the effect of extreme values; this does not affect the quality of the data fitting. The Weibull distribution is selected as the average system fit and as the best fit for six of the seven traces.

The daily cycle is modeled similarly to [31]. First, the day is split into 48 slots of 30 minutes each, then the number of BoT arrivals during each of the 48 slots is counted, and finally this data set is fitted against the candidate distributions. The Weibull distribution is again selected as the average system fit; it is also the best fit for six of the seven traces.

## 3.4 BoT Size

Similarly to IAT modeling, a base-two logarithmic transformation is applied before fitting to the batch size (i.e., the number of tasks in a BoT). The Weibull and the normal distributions are tied, followed closely by gamma; the Weibull distribution is selected as the average system fit due to the higher number of best fits for individual traces: five against one.

The average BoT size per trace is between 5 and 50, while the maximum BoT size can be on the order of thousands. Depending on size, we define nine classes of BoTs: of size 2-4, of size 5-9, of size 10-19, of size 20-49, of size 50-99, of size 100-199, of size 200-499, of size 500-999, and of size 1000 and over. We also define the Small (Medium) class encompassing the BoTs that fall in the classes 2-4 and 5-9 (10-19 and 20-49). We expect the size-based classes to have different performance characteristics under different scheduling configurations.

## 3.5 Intra-BoT Characteristics

The modeled intra-BoT characteristics are the average task runtime (ART) and the task runtime variability (RTV).

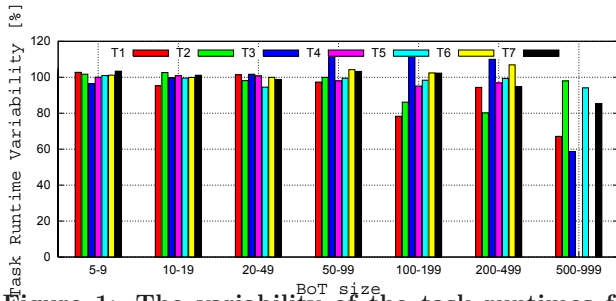


Figure 1: The variability of the task runtimes for each size-based class of BoTs between 2 and 1000, relative to the average variability of the aggregated Small and Medium BoT classes. For a trace and BoT size range, values closer to 100% denote variability closer to the median BoT variability of the trace.

Similarly to IAT modeling, a base-two logarithmic transformation is applied before fitting to the task runtime. The normal distribution is the average system fit and the best fit for five of the seven traces.

The intra-BoT task runtime variability is defined as the variance of runtimes of the tasks belonging to the same BoT. The Weibull distribution is the average system fit and the best fit for four of the seven traces.

### 3.6 Correlation Between BoT Size and Intra-BoT Characteristics

We now look at the correlation between the BoT size and the intra-BoT characteristics. We evaluate for each class of BoTs the task runtime variability and the task runtime. Figure 1 depicts the variability of the task runtimes for each size-based class of BoTs between 2 and 1000, relative to the average variability of the aggregated Small and Medium BoT classes (which form the majority of the BoTs). There is no significant difference between the various BoT classes. We have obtained similar results for the task runtime case (not depicted here for brevity). We conclude that the BoT size and the intra-BoT characteristics considered in our model are not strongly correlated.

### 3.7 Task Runtime Predictions

The high intra-BoT task runtime variability indicates that common prediction techniques may perform poorly. The user submits a new BoT to the system. The prediction problem is to predict for each task inside the BoT the base runtime, that is, the duration of executing that task on a base computing resource.

We define the relative inaccuracy of a prediction as  $\frac{|pred-real|}{real} \times 100\%$ , where *real* (*pred*) is the real (predicted) value. We further define the inaccuracy of a predictor for a given workload characteristic as the average inaccuracy for that characteristic. Good solutions to the prediction problem lead to low task runtime inaccuracy. Previous work on BoT scheduling has considered that the inaccuracy of a predictor has an upper limit of 100% [9]. We show in this section that this is not the case, and that values of 10000% (two orders of magnitude higher) are common. We investigate the impact of this finding in Section 5.3.

To evaluate the expected task runtime inaccuracy we use six predictors that are widely used for performance prediction in large-scale distributed and in other computing systems [37]; Table 4 describes them. We consider two types of input data for each predictor: batch and user. The batch input only lets the predictor use information obtained during

Model	Description
AdpExpSmooth	Adaptive exponential smoothing (Trigg and Leach)
AutoRegress	General autoregressive, $p=5$
ExpSmooth	Exponential smoothing, $\alpha = 0.5$
LastValue	Last observed value
Mean(10)	Average of last ten observed values
Mean	Average of all previously observed values

Table 4: The six time-series models used in this work. For details we refer to [7].

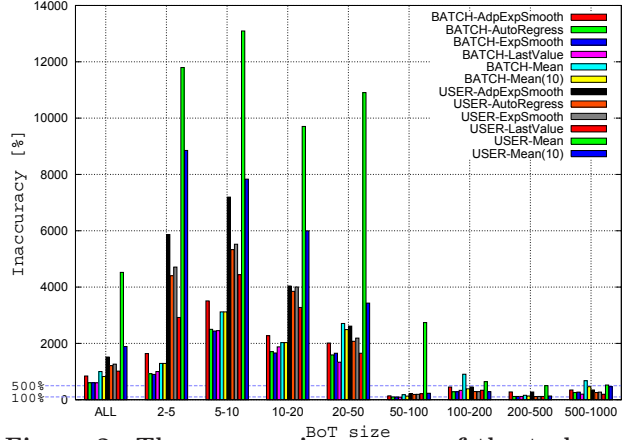


Figure 2: The average inaccuracy of the task runtime predictors.

the execution of the current BoT, which corresponds to the situation of a BoT manager being started for each BoT. The user input lets the predictor use all previous information for the same user, which corresponds to the situation of having a BoT manager per user (e.g., default in Condor-based systems [39]). The average prediction results for traces T1 and T2 are depicted in Figure 2. The user-based predictors are generally more inaccurate than their batch-based counterparts. The AdpExpSmooth and Mean predictors are more inaccurate than the rest. These two observations indicate that the user submission changes significantly over time. This may correspond to the habit of non-computing research groups to submit their jobs through the help of their technical support, thus through a single system user. The LastValue and the ExpSmooth (based on the last two values) with batch input perform are the best overall predictors. However, even the best predictor exhibits an average (maximum) inaccuracy of over 600% (30,000%!).

## 4. THE EXPERIMENTAL SETUP

This section describes the setup of the experiments in Section 5.

### 4.1 The Simulator

The experiments are performed in a simulated environment encompassing two multi-cluster environments, the DAS and Grid’5000 grids, for a total of 20 clusters and over 3500 processors, which is motivated by the authors’ current work on inter-operating the corresponding real environments. The simulation environment is the DGSim [22, 26] discrete event simulator for grids and for other large-scale distributed computing environments. The reason for using DGSim is threefold. First, DGSim already provides much of the simulated environment used in this work, e.g., the

combined DAS and Grid’5000 system, the resource management architectures (see Section 2.2). Second, the scheduling component of DGSim can be easily extended with the features required for this work. DGSim was extended for this paper with the task-selection and task-scheduling policies described in Section 2. In addition, DGSim was extended to support heterogeneous processing speeds; the processing speed of the resources used in simulation correspond to the SPECInt2006 values [38] of the real DAS and Grid’5000 resources, and their relative performance ranges between 1.0 and 1.75. Third, when compared to the previous simulation tools, DGSim focuses more on the simulation process, with better support for synthetic trace generation and use, and for exploring large design spaces [26].

DGSim reports a variety of performance metrics, such as the utilization, the per-task wait and response time, the per-task slowdown, the goodput (expressed as the total processing time of the jobs that finish successfully), and the finished tasks (expressed as the percentage of tasks that finish, from the tasks in the workload). For this paper, the simulator was extended to report the following BoT-related metrics:

**Makespan (MS)**, which for a BoT is defined as the difference between the earliest time of submission of any of its tasks, and the latest time of completion of any of its tasks.

**Normalized Schedule Length (NSL)**, which for a BoT is defined as the ratio of its Makespan and the sum of its tasks’ runtime on a reference processor. The NSL is the extension of the slowdown used for jobs in traditional computing systems [17]. Lower NSL values are better, in particular NSL values below 1 are desired.

## 4.2 The Workloads

Each of the 20 clusters of the combined system receives an independent stream of jobs (input workload). The input workloads used in our experiments are either one month-long traces collected from the individual grids starting at identical moments in time (*real traces*), or synthetic traces that reflect the properties of grid workloads (*realistic traces*). In [22] we argue that most research on scheduling in grids uses unrealistic synthetic traces for experimental purposes. We use throughout this article the formulation *realistic traces* in place of *realistic synthetic traces* to accentuate the difference between the traces used in this work and the unrealistic synthetic traces.

The need for realistic traces is twofold. First, traces coming from one system cannot be used unmodified on a different system [15, 18] (e.g., the job submission depends on the original circumstances); the seven traces used in this work originate from seven different grids. Furthermore, modifying the real traces (e.g., by scaling or duplicating their jobs) may lead to input that does not actually represent a realistic trace; scheduling results are highly sensitive to such changes [15]. Second, given the size of the explored design space (see Section 5), performing the experiments for each real trace becomes unmanageable. The use of real traces is required for validation purposes, to give evidence that results obtained with real and with realistic traces are similar.

Unless otherwise noted, the realistic traces use the parameter values of the average system given in row ”Avg” of Table 3. Our workloads have between 20,000 and 175,000 tasks, with an average of over 60,000 tasks (the median is

over 64,000 tasks). Thus, our experimental workloads fulfill the realism and the size requirements for the accurate simulation of schedulers [18].

## 4.3 Simulation Assumptions

We have made the following five assumptions in the experiments (a full description and motivation for the first four is given in [22]). First, we assume that the network between the clusters is perfect and has zero latency. Second, because all tasks are sequential, all sites employ the FCFS policy, *without* backfilling. Third, multi-processor machines (which do occur in the DAS and Grid’5000) behave and can be used as sets of single-processor machines without a performance penalty. Fourth, all load arrives directly at the RMs of the resource management architecture which is in place, with no jobs bypassing those RMs. Finally, we assume that there are no resource failures, because in light of our model for cluster-based grids [24] and of the model for desktop grids of Kondo et al. [27], an environment with resource failures is equivalent to a smaller environment, provided that the average resource availability duration is not lower than the runtime of the jobs.

## 5. THE PERFORMANCE OF BAG-OF-TASKS

In this section we present an investigation of the performance of BoTs in large-scale distributed computing systems. We cover with over 1200 trace-based simulations a design space with five axes and more than 2 million points: We consider 7 task-scheduling policies,  $7^P$  tuples of values describing the workload (with  $P$  the number of parameters in the workload model and 7 the number of values of each), 2 values for information inaccuracy (yes or no), 8 task-selection policies, and 3 resource management architectures; the exploration further uses 3 types of workloads (real, realistic, synthetic), for 7 system loads. To explore this design space efficiently, we assess in the subsections below in turn with a set of experiments for each parameter the impact on performance of varying the parameter along one of the axes of the design space; Table 5 shows an overview of what each experiment covers.

Most of the results present average values of the BoT MS and/or NSL metrics under various system loads, and as such are mostly useful to the system administrator or to the user submitting a workload with characteristics closely match those of the average workload. However, Sections 5.1 and 5.2 present detailed results for different BoT sizes and task runtimes; thus, they are also useful to understanding the performance perceived by users whose workload characteristics are significantly different than the average workload.

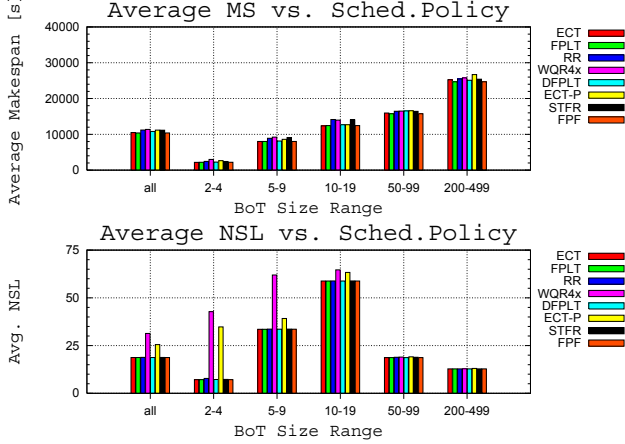
### 5.1 The Impact of the Task Scheduling Policy

In this section we assess the impact on performance of the scheduling policy using the setup described in the first line of Table 5. Our main findings are summarized below.

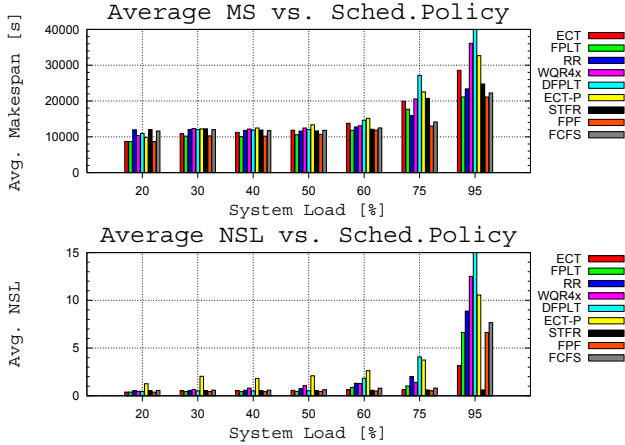
**Small and Medium-Sized BoTs have a high NSL even under low load** (Figure 3) We simulate the task-scheduling policies in a csp/S-T system under real load of 25%. Figure 3 shows that even with this low load, under csp/S-T the BoTs of sizes 5-9 and 10-19 have a higher than average NSL, while the average MS increases monotonically with the BoT size. This indicates that their users get higher than expected wait times for their tasks compared to other

Section	Res.Mgmt. Architecture	Selection Policy	Scheduling Policy	Workload Characteristics	System Load	Information Inaccuracy
Section 5.1	csp	S-T	<b>all</b>	real, realistic	~25%, 20-95%	no
Section 5.2	csp	S-T	all	<b>synthetic</b>	60%	no
Section 5.3	csp	S-T	all	realistic	20-95%	<b>yes</b>
Section 5.4	csp	<b>all</b>	FPLT	realistic	20-95%	no
Section 5.5	<b>all</b>	S-T	FPLT	realistic	20-95%	no

**Table 5: The design space coverage of the experiments presented in this section. The characteristics in bold indicate the main focus of each section.**



**Figure 3: The performance of the scheduling policies in large-scale distributed computing systems with csp/S-T, under *real* load.**



**Figure 4: The performance of the scheduling policies in large-scale distributed computing systems with csp/S-T, under *realistic* load.**

system users. The main beneficiaries are the users submitting very small (size 2-4) or very large BoTs (size over 200).

**Comparison of policies with different information types** Figure 4 shows the performance of the scheduling policies under realistic load varying between 20% and 95%. The  $(K, K)$  policies have the best balance between MS and NSL. Moreover, FPLT has the lowest overall MS, and ECT has the second lowest overall NSL. As the policies that include unknown  $(U)$  information models rely on the quality of their uninformed heuristic, their performance ranges from surprisingly good to surprisingly poor. The  $(U, K)$  policy STFR has

the best overall NSL, but average MS. The  $(K, U)$  policy FPF has the lowest overall MS, but poor NSL. The  $(U, U)$  policy RR has a good MS, but poor NSL. The WQR4x has poor MS and NSL, especially at high load. The prediction-based policies (those that have at least one information model of type  $H$ , i.e., DFPLT and ECT-P), perform consistently worse than the other scheduling policies.

**Design guidelines for scheduling policies** The transition from information type  $K$  to type  $H$  can be costly: FPLT outperforms DFPLT (transition  $(K, K)$  to  $(H, K)$ ), and ECT outperforms ECT-P (transition  $(K, K)$  to  $(K, H)$ ). However, it may be necessary due to real system requirements. While we have not explored in this paper the complete transition from  $K$  via  $H$  to  $U$ , the comparison of policies with different information types above shows evidence that when changing to information types  $H$  and  $U$ , it is better to do so for the piece of information that has the lowest variation. In our simulated system, the variation in resource performance is low (i.e., the ratio between the performance of the fastest and the slowest processor is below 2), whereas the task runtime variation is high (reaches values over a thousand). Thus, for the system we simulated, policies of type  $(U, *)$  are preferable.

## 5.2 The Impact of the Workload Characteristics

In this section we assess the impact on performance of the workload characteristics using the setup described in second line of Table 5. For each such characteristic, we generate a synthetic workload imposing a system load of 60% using the workload model described in Section 3, so that the values of the characteristics have the desired statistical properties (e.g., a median BoT size of 25). Our main findings are summarized below.

**Burstiness leads to poor performance in csp/S-T** (Figure 5) We vary the BoT arrival pattern with as possibilities a daily cycle based on the workload model (Realistic), the pattern with all BoTs arriving at the beginning of the simulation ("All at T=0"), and the pattern with all BoT interarrival times being equal (Evenly Spread). Figure 5 shows that the extreme of burstiness ("All at T=0") leads to much higher average MS and NSL values when compared to the other two patterns. The Evenly Spread case is more favorable than the Realistic case for four of the scheduling policies by about 10%, and for WQR4x (FPLT) even by 20% (48%).

**Impact of the BoT size** (Figure 6) From the realistic workload model proposed in Section 3, we vary the BoT size to achieve median values between 10 and 50, while keeping the coefficient of variation of the BoT size constant. Figure 6 shows that the MS increases with the increase of the median of the BoT size. Relative to a BoT median size of 10, the MS increase ranges from 37% for ECT to over 80% for WQR4x, which indicates that for some policies, the average MS is less

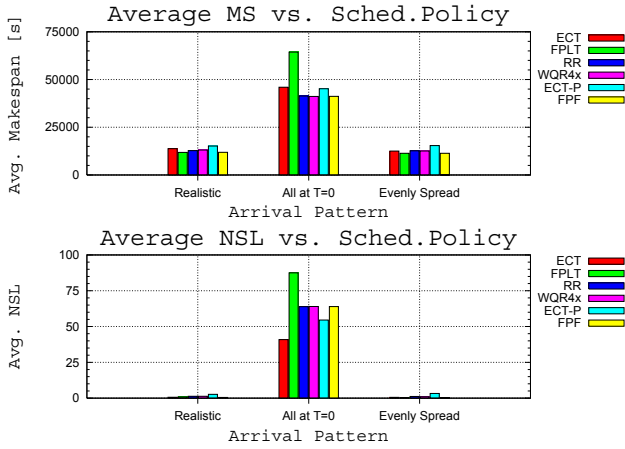


Figure 5: The performance of the scheduling policies in large-scale distributed computing systems with csp/S-T, under various BoT arrival patterns.

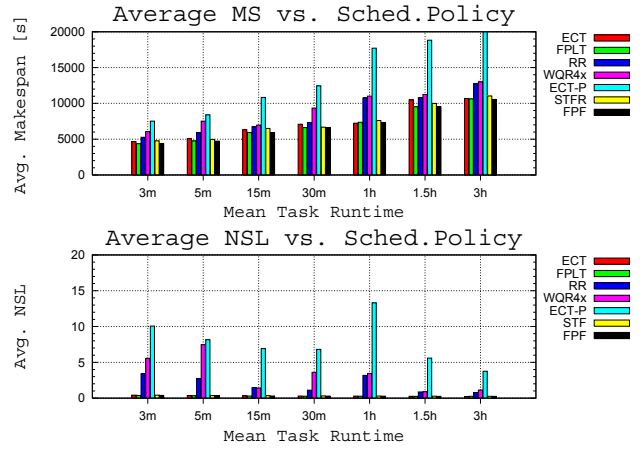


Figure 7: The performance of the scheduling policies in large-scale distributed computing systems with csp/S-T, for various mean task runtimes.

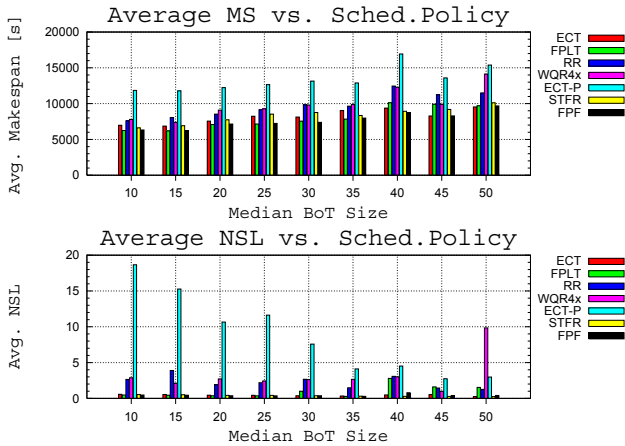


Figure 6: The performance of the scheduling policies in large-scale distributed computing systems with csp/S-T, for various median BoT sizes.

dependent from the BoT size than for others. The average NSL decreases for ECT-P and STFR with the increase of the BoT size; the other policies do not exhibit this performance improvement trend.

**Longer tasks lead to better NSL** (Figure 7) From the realistic workload model proposed in Section 3, we vary the task runtime to achieve mean values between 3 minutes and 3 hours. Figure 7 shows that the NSL tends to decrease with the increase of task runtime.

### 5.3 The Impact of the Dynamic System Information

We have shown in Section 3.7 that the LastValue predictor with batch input has good performance relative to the other predictors, while still having an average (maximum) prediction inaccuracy of over 600% (30,000). We have also shown in Section 5.1 that the prediction-based policies perform consistently worse than the other scheduling policies. In this section we assess the impact of various inaccuracy values under the assumption of null overall inaccuracy, that

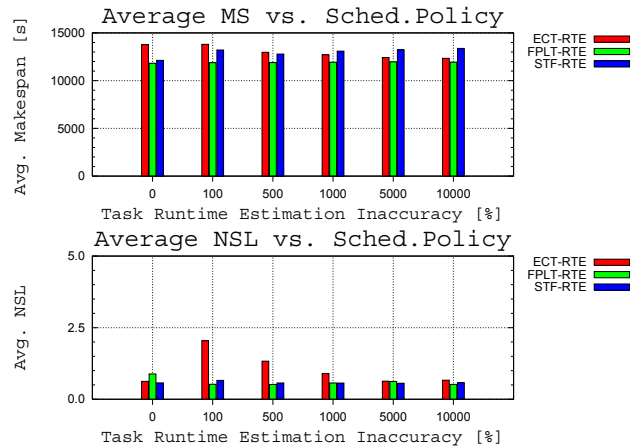


Figure 8: The performance of the scheduling policies in large-scale distributed computing systems with csp/S-T, when the inaccuracy of task runtime estimations varies.

is, we make the optimistic assumption that while any individual estimation may be highly inaccurate, the average estimation inaccuracy is 0%.

**Under null overall inaccuracy, accurate per-task information is not needed to schedule well** (Note that this finding may lead to new types of predictors for BoT scheduling that would not be useful in the context of independent tasks.) We first set the maximum inaccuracy  $I$  to a value between 0% (perfect information) and 10000% (high inaccuracy). Then, for each task runtime estimation we sample the estimation inaccuracy  $E$  from the uniform distribution  $[-I, +I]$ ; the task runtime is set to  $\max(R + (E/100) \times R, 1)$ , where  $R$  is the actual task runtime, and any task is at least 1 second long. Figure 8 shows that in general, under null overall inaccuracy the MS and NSL vary little with the increase of inaccuracy. Only for ECT we observe an increase of the NSL; surprisingly, this increase diminishes with the increase of the maximum inaccuracy. We attribute this fact to ECT's sensitivity to underestimations of the task runtime.

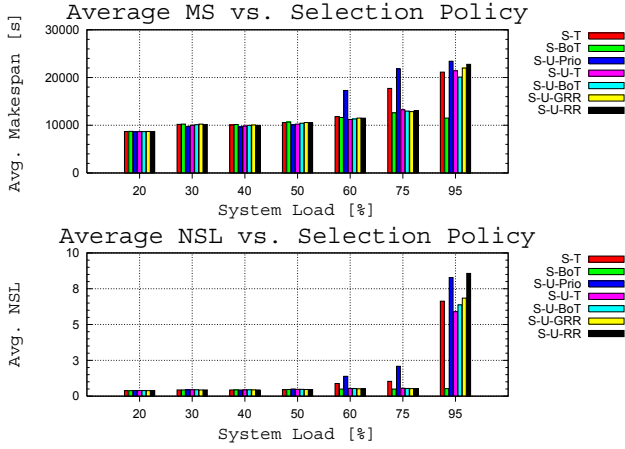


Figure 9: The performance of the task selection policies in large-scale distributed computing systems with `csp/FPLT`, under realistic load.

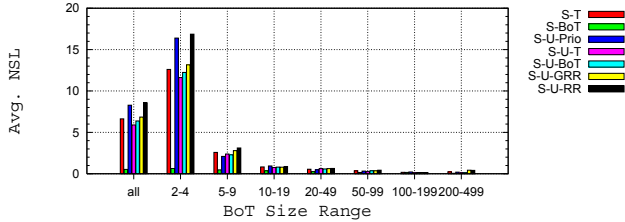


Figure 10: The NSL of the task selection policies for different BoT sizes in large-scale distributed computing systems with `csp/FPLT`, when a realistic load of 95% is imposed on the system.

## 5.4 The Impact of the Task Selection Policy

In this section we assess the impact of the task-selection policy on the system performance. We use the setup described in the fourth line of Table 5: a `csp` resource management architecture, the `FPLT` ( $K, K$ ) task-scheduling policy, and realistic workloads that subject the system to a load between 20% and 95%. Our main findings are described below.

**The task-selection policy is important only in busy systems** Figure 9 shows that for loads up to 50%, the performance of the system is almost identical for the seven task-selection policies. This is not surprising: with a ( $K, K$ ) scheduling policy, the system resources are harnessed efficiently when much spare capacity exists. The fact that `FLTP` does not use replication is also important in establishing a load of 50% as the threshold beyond which the task-selection policy does have an impact on performance. With a policy that does use replication, this threshold would be lower, in proportion to the average number of replications per task. Figure 9 also shows that the task-selection policy has an important impact on performance for loads of 50% and higher; this impact increases with the load.

**S-BoT has much better performance than the other task-selection policies** Figure 10 depicts the NSL of all the task-selection policies for various system load. For loads of 60% and higher, the average NSL (MS) for `S-BoT` is up to 16 (2) times lower than the average NSL (MS) of the other

policies. In particular, `S-BoT` outperforms the task-selection policy most commonly used in practice, which is `S-T`. We attribute the differences to the design of the `S-BoT` policy, which greatly favors the small BoTs that are common in the workloads of large-scale distributed computing systems. This is confirmed by the set of columns corresponding to 95% load in Figure 10.

**System fairness costs: accounting system or performance** (Figure 9) Introducing fairness into a resource management system in general reduces the aggregate performance of the system. We compare the four task-selection policies studied in this paper that consider fairness: `S-U-T`, `S-U-BoT`, `S-U-GRR`, `S-U-RR`. The first three of these may all lead to one user blocking the system for a long time regardless of what the other users do (e.g., when sending one or more large BoTs and getting selected); `S-U-RR` does not suffer from this problem if all the users submit equally large BoTs. While system blocking is a possibility, our study shows that this does not happen in practice often enough to be significant. From the four policies, `S-U-T` performs the best. The ordering of BoTs by arrival time employed by `S-U-BoT` does not lead to better performance. The round-robin ordering of users employed by `S-U-GRR`, which does not require an accounting system to be present, leads in turn to up to 15% higher NSL than `S-U-T`. Finally, `S-U-RR`, which also does not require an accounting system, has up to 45% higher NSL values than `S-U-T`. To conclude, to have fairness a system must either setup an accounting system, or it will pay in lower performance.

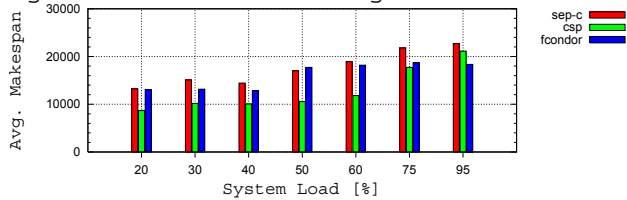
**System QoS costs: level vs. performance** (Figure 9) Offering guarantees about the response time of the submitted tasks is expensive in terms of performance in a system that does not support advance resource reservation. We compare the three task-selection policies presented in this paper that support QoS: `S-U-Prio`, `S-U-GRR`, `S-U-RR`. `S-U-Prio` guarantees that the tasks of the user with the highest priority from the users currently having queued tasks will be selected next. `S-U-GRR` guarantees that a user’s task will be selected for execution at most  $n$  rounds after submission, where  $n$  is the total number of users in the system, both with and without queued jobs. `S-U-RR` guarantees that at least one of a user’s queued tasks is selected in the next  $n$  rounds; the rounds of `S-U-RR` are on average much shorter than those of `S-U-GRR`, as `S-U-RR` selects only one task per round. `S-U-Prio` and `S-U-RR` have similar performance. Compared to `S-U-GRR`, their performance is lower, by up to 20%. To conclude, there is a trade-off between the weak QoS guarantees but higher performance of `S-U-GRR`, and the stronger QoS guarantees but lower performance of `S-U-Prio` and `S-U-RR`.

## 5.5 The Impact of the Resource Management Architecture

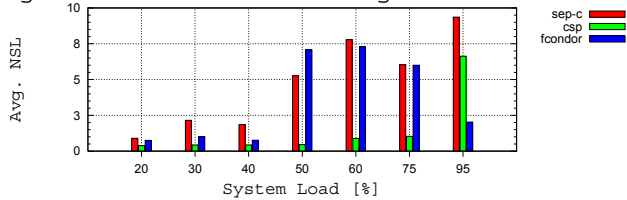
In this section we assess the impact of the resource management architecture on the system performance. We use the setup described in Table 5 (i.e., an `S-T` selection policy, and the `FPLT` ( $K, K$ ) scheduling policy), and realistic workloads that subject the system to a load between 20% and 95%.

**Centralized, separated, or distributed?** (Figure 11) The centralized policy `csp` achieves the best performance. Conversely, the independent clusters policy `sep-c` achieves the worst performance. The distributed architecture `fcondor` exhibits mixed performance results: expectedly, for loads

Average MS vs. Resource Management Architecture



Average NSL vs. Resource Management Architecture



**Figure 11: The impact of the system resource management architecture on performance under realistic load, with the S-T() selection (scheduling) policies.**

below 50% its performance is similar to that of `csp`, and, surprisingly, for loads above and including 50% its performance is similar to that of `sep-c`. We observe that for loads above and including 50% `fcondor` and `sep-c` cannot complete all tasks; for the 95% load they complete only 44% and 53% of the tasks, respectively. We attribute the inability of `fcondor` to complete more tasks to its fostering of "natural competition": most of the tasks that would get blocked in a central architecture (until other tasks are finished) are submitted in the distributed architecture to the clusters, where they compete with each other for occupying the idle processors.

## 6. RELATED WORK

This work stands at the intersection of three directions of research: workload modeling, design of BoT scheduling policies, and design of complete BoT scheduling systems.

We have discussed throughout the text the research most closely related for workload modeling [31, 29] and for design of BoT scheduling policies [9, 12, 19, 28]. For the latter, the policies have been previously evaluated through simulation in independent clusters environments (i.e., `sep-c` in Section 2) using a per BoT scheduling algorithm (i.e., algorithm `S-BoT`); the workload did not resemble the workloads of real large-scale distributed computing systems. Closest to our work, Casanova et al. [9] compare the performance of several BoT scheduling policies in a `sep-c` environment. They also present one class of selection policies (`S-BoT`). In comparison with these results, our work focuses on the systematic evaluation Bag-of-Tasks scheduling in large-scale distributed computing systems (resource management architecture, scheduling algorithm, scheduling policy), and on using realistic workloads.

In contrast to our simulation-based approach, the theory of divisible loads [6] proposes mathematical analysis tools to assess the performance of various BoT scheduling algorithms for several distributed and centralized resource management architectures [4, 3]. However, work using this theory [4, 3, 20, 8] does not consider the information policies, most of the selection policies, and many of the resource management architectures considered in this work; they also do not consider realistic workloads.

## 7. CONCLUSION AND FUTURE WORK

While distributed computing is becoming increasingly common, its resource scheduling is currently inefficient, due mainly to a lack of understanding of real user demands, and of scheduling alternatives. Bags-of-tasks have been identified as the scientists' tool of choice, but there is no workload model that explicitly includes bags-of-tasks. In this work we make a realistic and systematic investigation of the performance of bags-of-tasks scheduling solutions in large-scale distributed computing systems. We first propose a taxonomy of scheduling policies that focuses on information availability and accuracy, and we propose three new classes bag-of-tasks scheduling policies; for each new class we also propose a simple task scheduling policy. Then, we introduce a realistic workload model that focuses on bags-of-tasks, and we validate this model using seven long-term workload traces taken from large-scale distributed computing systems of various size and application. Finally, we explore the large design space of bag-of-task scheduling in large-scale distributed computing systems along five axes: the scheduling policy, the input workload, the information policy, the scheduling algorithm, and the resource management architecture.

For the future, we plan to extend our analysis for systems in which bags-of-tasks coexist with parallel tasks. We also plan to design more scheduling heuristics, particular of the kind that do not assume the a priori existence of any information and build their information set dynamically (the class  $(H, H)$  in our proposed taxonomy).

## Acknowledgements

This work was carried out in the context of the Virtual Laboratory for e-Science project ([www.vl-e.nl](http://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. We would also like to thank the anonymous HPDC reviewers for their comments.

## 8. REFERENCES

- [1] The Grid Workloads Archive. [Online] <http://gwa.ewi.tudelft.nl>, Mar 2008.
- [2] J. Aldrich, R. A. Fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, 12(3):162-176, 1997.
- [3] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert. Centralized versus distributed schedulers for multiple bag-of-task applications. In *IPDPS*. IEEE CS, 2006.
- [4] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: Results and open problems. *IEEE Trans. Parallel Distrib. Syst.*, 16(3):207-218, 2005.
- [5] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. M. Figueira, J. Hayes, G. Obertelli, J. M. Schopf, G. Shao, S. Smallen, N. T. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the grid using apples. *IEEE Trans. Parallel Distrib. Syst.*, 14(4):369-382, 2003.
- [6] V. Bharadwaj, D. Ghose, and T. G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Comput.*, 6(1):7-17, Jan. 2003.
- [7] P. J. Brockwell and R. A. Davis. *Introduction to time series and forecasting*. Springer, 1996.
- [8] T. E. Carroll and D. Grosu. A strategy proof mechanism for scheduling divisible loads in bus networks without control processors. In *IPDPS*. IEEE CS, 2006.

- [9] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *HCW*, pages 349–363. IEEE CS, 2000.
- [10] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The apples parameter sweep template: User-level middleware for the grid. In *SC*, 2000.
- [11] W. Cirne, D. P. da Silva, L. Costa, E. Santos-Neto, F. V. Brasileiro, J. P. Sauvé, F. A. B. Silva, C. O. Barros, and C. Silveira. Running bag-of-tasks applications on computational grids: The mygrid approach. In *ICPP*, pages 407–. IEEE CS, 2003.
- [12] D. P. da Silva, W. Cirne, and F. V. Brasileiro. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In H. Kosch, L. Böszörményi, and H. Hellwagner, editors, *Euro-Par*, volume 2790 of *LNCS*, pages 169–180. Springer-Verlag, 2003.
- [13] A. B. Downey and D. G. Feitelson. The elusive goal of workload characterization. *Performance Evaluation Review*, 26(4):14–29, 1999.
- [14] D. Epema, M. Livny, R. Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors: Load sharing among workstation clusters. *Future Gener. Comput. Syst.*, 12:53–65.
- [15] 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-Verlag, 2003.
- [16] M. Evans, N. Hastings, and B. Peacock. *Statistical Distributions*. Wiley, 3rd edition, 2000.
- [17] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In *IPPS/SPDP*, volume 1459 of *LNCS*, pages 1–24, 1998.
- [18] E. Frachtenberg and D. G. Feitelson. Pitfalls in parallel job scheduling evaluation. In D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, editors, *JSSPP*, volume 3834 of *LNCS*, pages 257–282. Springer-Verlag, 2005.
- [19] N. Fujimoto and K. Hagihara. Near-optimal dynamic task scheduling of independent coarse-grained tasks onto a computational grid. In *ICPP*, pages 391–398. IEEE CS, 2003.
- [20] N. Garg, D. Grosu, and V. Chaudhary. An antisocial strategy for scheduling mechanisms. In *IPDPS*. IEEE CS, 2005.
- [21] A. Iosup, C. Dumitrescu, D. H. Epema, H. Li, and L. Wolters. How are real grids used? The analysis of four grid traces and its implications. In *GRID*, pages 262–270. IEEE CS, 2006.
- [22] A. Iosup, D. Epema, T. Tannenbaum, M. Farrellee, and M. Livny. Inter-operating grids through delegated matchmaking. In *SC*. ACM Press, 2007.
- [23] A. Iosup, M. Jan, O. Sonmez, and D. Epema. The characteristics and performance of groups of jobs in grids. In *Euro-Par*, volume 4641 of *LNCS*, pages 382–393. Springer-Verlag, 2007.
- [24] A. Iosup, M. Jan, O. Sonmez, and D. H. Epema. On the dynamic resource availability in grids. In *GRID*, pages 26–33. IEEE CS, 2007.
- [25] A. Iosup, H. Li, M. Jan, S. Anoop, C. Dumitrescu, L. Wolters, and D. Epema. The grid workloads archive. *Future Gener. Comput. Syst.*, Feb. 2008. (in print).
- [26] A. Iosup, O. Sonmez, and D. Epema. The DGSim trace-based simulator. TU Delft, Tech. Rep. PDS-2008-003, Mar 2008. ISSN 1387-2109.
- [27] D. Kondo, G. Fedak, F. Cappello, A. A. Chien, and H. Casanova. Characterizing resource availability in enterprise desktop grids. *Future Gener. Comput. Syst.*, 23(7):888–903, 2007.
- [28] Y. C. Lee and A. Y. Zomaya. Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. *IEEE Trans. Computers*, 56(6):815–825, 2007.
- [29] 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.
- [30] H. W. Lilliefors. On the Kolmogorov-Smirnov test for the exponential distribution with mean unknown. *Journal of the American Statistical Association*, 64:387–389, 1969.
- [31] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *J. Parallel Distrib. Comput.*, 63(11):1105–1122, 2003.
- [32] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop*, pages 30–, 1999.
- [33] D. A. Menascé, D. Saha, S. C. S. Porto, V. Almeida, and S. K. Tripathi. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. *J. Parallel Distrib. Comput.*, 28(1):1–18, 1995.
- [34] M. E. J. Newman. Power laws, pareto distributions and zipf’s law. *Contemporary Physics*, 46:323, 2005.
- [35] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case. *IEEE/ACM Trans. Netw.*, 2(2):137–150, 1994.
- [36] R. F. Rosin. Determining a computing center environment. *Commun. ACM*, 8(7):463–468, 1965.
- [37] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos, editors, *KDD*, pages 426–435. ACM, 2003.
- [38] SPECCPU Team. SPEC CPU2006. Standard Performance Evaluation Corporation, Mar 2008. [Online] Available: <http://www.spec.org/cpu2006/>.
- [39] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Elsevier Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.