Static and dynamic overprovisioning strategies for performance consistency in grids

Static and Dynamic Overprovisioning Strategies for Performance Consistency in Grids

Nezih Yigitbasi
Delft University of Technology, the Netherlands
M.N.Yigitbasi@tudelft.nl

Dick Epema
Delft University of Technology, the Netherlands
D.H.J.Epema@tudelft.nl

***Abstract*—It is not uncommon that grid users observe highly variable performance when they submit similar workloads at different times. From the users' point of view, such inconsistent performance is undesirable, and it leads to user dissatisfaction and confusion. We tackle this performance inconsistency problem using *overprovisioning* which is increasing the system capacity by a factor that we call the *overprovisioning factor* (κ). Although overprovisioning is not cost effective, its simplicity makes it the preferred method for providing performance guarantees. Hence in this work, we present a realistic investigation of overprovisioning in grids. Towards this end, first we present a performance and cost evaluation of static and dynamic overprovisioning strategies. We find that the dynamic overprovisioning strategy, for which we use computing clouds, provides better consistency with lower costs compared to static strategies, and overprovisioning beyond a certain value of κ (in our case κ=2.5) incurs significant costs without significant consistency improvements. Then, we design and evaluate a feedback-controlled system to dynamically determine κ to give performance guarantees to grid users. We show that our system determines κ dynamically and provides significant improvements over the initial system, as high as 67%, in the number of jobs that meet the performance requirements.**

I. Introduction

Users expect consistent performance from computer systems—when some interaction with an interactive application always finishes within 1 second, they are annoyed when suddenly the response time jumps to say 10 seconds. Likewise, when a certain Bag-of-Tasks (BoT) submitted to a grid has a response time of 5 hours, then the user will be surprised when a BoT with twice as many tasks (of a similar type as in the first BoT) takes say 24 hours. However, preventing such situations and providing consistent performance in grids is a difficult problem due to the specific characteristics of grids like the lack of support for advance reservations in many Local Resource Managers (LRMs), highly variable workloads, dynamic availability and heterogeneity of resources, and variable background loads of local users. In this work we investigate overprovisioning for solving the performance inconsistency problem in grids.

Overprovisioning can be defined as increasing the capacity, by a factor that we call the *overprovisioning factor*, of a system to better handle the fluctuations in the workload, and provide consistent performance even under unexpected user demands. Although overprovisioning is a simple solution for consistent performance and it obviates the need for complex algorithms, it is not cost effective and it may cause systems to be underutilized most of the time. Despite these disadvantages overprovisioning is the preferred method for providing performance guarantees in telecommunication systems [1], and it is also being used in large-scale distributed systems like modern data centers. Studies have shown that typical data center utilization is no more than 15-50% [2], [3], and telecommunication systems have roughly 30% [4] utilization on average.

A large body of work on providing predictable performance [5]–[7], and Service Level Agreements [8]–[10] already exists. What is missing so far from this research is a detailed realistic investigation of how we can achieve consistent performance in grids. In our recent work [11] we took the first step towards filling this gap with a performance evaluation of static overprovisioning strategies. In this work we extend our previous work, and we perform a realistic investigation of both static and dynamic overprovisioning strategies for achieving performance consistency in grids. Towards this end, our contribution is threefold:

1. We propose several overprovisioning strategies for multicluster grids, and we classify these strategies as static or dynamic based on when the resources are provisioned (Section III).
2. We assess the performance and the cost of the proposed strategies with realistic simulations. In our simulations we model the Dutch DAS-3 [12] multicluster grid and we use various synthetic workloads consisting of BoTs, which constitute the dominant application type in grids [13]. Our model includes the actual background load of other users, which is one of the causes of performance inconsistency. For our simulations with dynamic strategies, where we use clouds, we employ the resource model of the Amazon Elastic Compute Cloud (EC2) [14] (Section VI).
3. We design and evaluate a feedback-controlled system that exploits the elasticity of computing clouds to give performance guarantees to grid users, which is the user's perspective on our research problem. Our

This work was carried out in the context of the project Guaranteed Delivery in Grids (GUARD-G) (`http://guardg.st.ewi.tudelft.nl/`), which is supported by NWO.

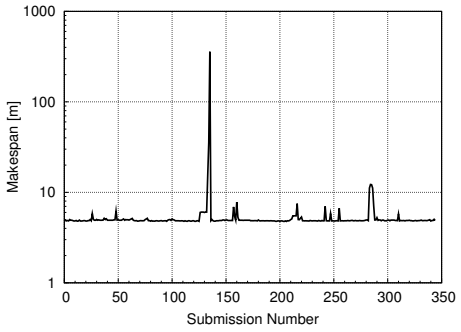

Figure 1. Evidence of the performance inconsistency in grids. The vertical axis has a logarithmic scale.

system overprovisions a grid dynamically using the user specified performance requirements and the measured system performance (Section VII).

II. Motivation

Grid users may observe highly variable performance when they submit similar workloads at different times depending on the system state. From the users' point of view, any variability in performance should only be caused by their own applications (due to modifications of the applications or inputs) and not by the system or by load due to other users. Hence, inconsistent performance is usually undesirable, and it leads to user dissatisfaction and confusion.

Figure 1 shows evidence of the performance inconsistency in grids. In this experiment, we submit the same BoT consisting of 128 tasks periodically every 15 minutes to our multicluster grid DAS-3, which is usually underutilized. The graph shows the makespan in minutes for each submitted BoT. Since the system is mostly empty, we do not observe high variability in makespan for the first 130 submissions. However, we observe a significant variability between the 130th and 140th submissions, which is due to the background load created by other users, causing some tasks of the BoTs to be significantly delayed. The ratio of the maximum to the minimum makespan in this experiment is roughly 70! This result shows that even for a grid like DAS-3, which is a research grid, and hence usually underutilized, we may observe very strong performance inconsistencies.

It is a challenge to develop efficient solutions for providing consistent performance in grids due to their high level of heterogeneity and dynamic workloads. It is possible to tackle this problem at two levels: at the (global) scheduler level, and at the resource level. At the scheduler level, we need appropriate *mechanisms*, e.g., admission control, and (scheduling) *policies* to solve this problem. The problem can also be addressed at the resource level which consists of the computing nodes in the grid. In this work we investigate overprovisioning to solve this problem at the resource level.

III. Overprovisioning Strategies

We define overprovisioning as increasing the capacity of a system to provide better, and in particular, consistent performance even under variable workloads and unexpected demands. We define the *overprovisioning factor* κ as the ratio of the size of an overprovisioned system to the size of the initial system. Overprovisioning is a simple solution that obviates the need for complex algorithms. However, there are also some disadvantages of this solution. First, overprovisioning is of course a cost-ineffective solution. Second, overprovisioning may cause the system to be underutilized since resources may stay idle most of the time; however, the industry is used to low utilization in data centers where the average utilization is roughly 30% [4], and in telecommunication systems where the utilization is in the range 15-50% [2], [3].

To overprovision grids we propose various strategies, and we classify them as static or dynamic based on when the resources are provisioned. We summarize these *strategies* below:

- **Static Overprovisioning:** The resources are provisioned statically at system deployment time, hence before the workload arrives at the system. We distinguish:
  - **Overprovision Largest Site (`Largest`):** Only the largest site of the grid in terms of the number of processors is overprovisioned in this strategy.
  - **Overprovision All Sites (`All`):** All of the sites of the grid are overprovisioned equally.
  - **Overprovision Number of Sites (`Number`):** The number of sites of the grid is overprovisioned. The number of processors to deploy to the newly added sites are determined according to the overprovisioning factor.
- **Dynamic Overprovisioning (`Dynamic`):** Since fluctuations are common in grid workloads, static resource provisioning may not always be optimal. Therefore, we also consider a dynamic strategy where the resources are acquired/released in an on-demand fashion from a computing cloud. We use low and high load thresholds specified by the system administrator for acquiring/releasing resources from/to the cloud, which is also known as *auto-scaling* [14]. We continuously monitor the system and determine the load of the system periodically, where the period is also specified by the administrator. If the load exceeds the high threshold we acquire a new resource, and if the load falls below the low threshold we release a resource to the cloud.

The number of processors to be deployed to a specific site is determined by the overprovisioning factor κ and the overprovisioning strategy. For example, assume that a grid has N sites where site i has S_i processors, and that we use the `All` strategy for overprovisioning. Assume also that S is the size of the initial system, so $S = \sum_{i=1}^{N} S_i$. We want the size of the overprovisioned system $S' = \kappa S$, hence we set S'_i, the size of the overprovisioned site i, as $S'_i = \kappa S_i$. Thus, $S' = \sum_{i=1}^{N} S'_i = \sum_{i=1}^{N} \kappa S_i = \kappa S$. For the other strategies, the number of processors to deploy to attain a certain value

Table I
PROPERTIES OF THE DAS-3 CLUSTERS.

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

of κ is derived similarly.

IV. SYSTEM MODEL

A. System Model

In our simulations we exactly model our multicluster grid DAS-3 [12] which is a research grid located in the Netherlands. It comprises 272 dual-processor AMD Opteron compute nodes, and it consists of five homogeneous clusters; although the processors have different performance across different clusters, they are identical in the same cluster. The cluster properties are shown in Table I.

We assume that there is a Global Resource Manager (GRM) in the system interacting with the LRMs which are responsible for managing the cluster resources. The jobs are queued in the GRM's queue upon their arrival, and then dispatched to the LRMs where they wait for cluster resources. Once started, jobs run to completion, so we do not consider preemption or migration during execution.

When evaluating the `Dynamic` strategy, we assume that there is overhead for acquiring/releasing resources from/to the computing cloud. We have performed 20 successive resource acquisition/release experiments in the Amazon EC2 cloud with the `m1.small` instance type to determine the resource acquisition/release overheads [15]. We found that the minimum/maximum values for the resource acquisition and release overheads are 69/126 seconds and 18/23 seconds, respectively. We assume that the acquisition/release overhead for a single processor is uniformly distributed between these minimum and maximum values.

B. Scheduling Model

As the application type we use BoTs, which are the dominant application type in grids [13]. To model the application execution time, we employ the SPEC CPU benchmarks [16]: the time it takes to finish a task is inversely proportional to the performance of the processor it runs on. We consider the following BoT scheduling *policies*, which differ by the system information they use:

- **Static Scheduling:** This policy does not use of any system information. Each BoT is statically partitioned across the sites where number of tasks sent to each site is proportional to the size of the site.
- **Dynamic Scheduling:** This policy takes the current state of the system (e.g., the load) into account when taking decisions. We consider two variants of dynamic scheduling:
  - **Dynamic Per Task Scheduling:** In this policy, a separate scheduling decision is made for each task of each BoT, and the task is sent to the site with the lowest load, where we define the load of a site as the fraction of used processors.
  - **Dynamic Per BoT Scheduling:** In this policy, a separate scheduling decision is made for each BoT, and the whole BoT is sent to the least loaded site.
- **Prediction-based Scheduling** We consider only one such policy:
  - **Earliest Completion Time (ECT):** This policy uses historical data to predict the task runtimes. With this policy each task is submitted to the cluster which is predicted to lead to the earliest completion time taking into account the clusters' queues. To predict the runtime of a task, we use the average of the runtimes of the previous two tasks [17], since this method is known to perform well in multicluster grids [6].

Table II
THE DISTRIBUTIONS AND THE VALUES FOR THEIR PARAMETERS FOR THE BOT WORKLOAD MODEL DESCRIBED IN [18]. $N(\mu,\sigma^2)$ AND $W(\lambda,\kappa)$ STAND FOR THE NORMAL AND WEIBULL DISTRIBUTIONS, RESPECTIVELY.

	Bag-of-Tasks		Task
	Inter-Arrival Time	Size	Average Runtime
	W(4.25,7.86)	W(1.76,2.11)	N(2.73,6.1)
Average	124.6 s	6.1	7859.7 s

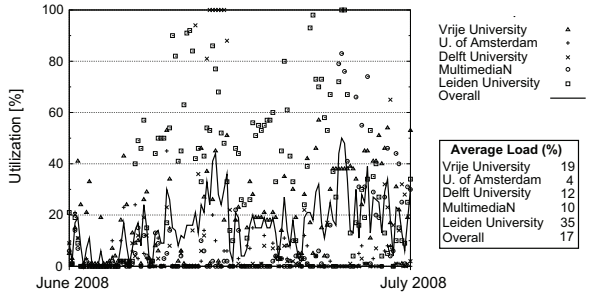

Figure 2. The overall utilization as well as the utilization in the individual clusters due to the background load, which consists of the jobs submitted to the DAS-3 system during June 2008.

V. EXPERIMENTAL SETUP

In this section we introduce our experimental setup. First, we describe the workload that we use in our simulations. Then, we describe our methodology and the metrics for assessing the performance and cost of the overprovisioning strategies.

A. Workload

We have performed experiments with BoT workloads that we generate using the realistic BoT model described in [18]. The values for the important workload attributes are summarized in Table II. These parameters are determined after a base-two logarithmic transformation is applied to the empirical data. In addition, in [18] the authors assume that the minimum BoT size is two, whereas we assume that single tasks are also BoTs with size one.

In our simulations we impose a background load together with the BoT workload in order to attain realistic scenarios. The background load consists of the jobs submitted to DAS-

3 during June 2008, and the corresponding workload trace is obtained from the Grid Workloads Archive [20]. Figure 2 shows the utilization of the background load. During the simulations, the background tasks are submitted to the LRMs of their original execution sites.

For our experiments, we have generated ten workloads for load levels ranging from 10% to 100% in the initial system in steps of 10%. Although we have performed simulations with all load levels, we present the results of load level 80% in Section VI which we think is representative, unless otherwise noted. Each workload contains approximately 1650 BoTs, and 10K tasks, and the duration of each trace is roughly between 1 day and 1 week.

B. Methodology

For assessing the static overprovisioning strategies, first, we evaluate the system with the aforementioned workloads, then we overprovision the system according to the strategy under consideration, and we use the same workload to assess the impact of the overprovisioning strategy. For the `Dynamic` strategy, a criterion has to be defined which determines when the system should acquire/release resources from/to the computing cloud. To this end, for the simulations with the `Dynamic` strategy, with the BoT workload that imposes 80% load on the system, we use a high threshold of 70% and a low threshold of 60% for deciding when to acquire and release additional resources, respectively. When using the `Dynamic` strategy, κ varies over time. Hence, in order to obtain comparable results in our simulations with the `Dynamic` strategy, we keep the average value of κ always in the $\pm$ 10% range of the specified value. For example, for $\kappa = 2.0$, when acquiring resources we do not exceed $\kappa = 2.2$, and when releasing resources we do not fall below $\kappa = 1.8$.

Finally, to obtain comparable results we assume that cloud resources have the same performance as the slowest grid cluster.

C. Performance Metrics

To evaluate the performance of the strategies, we use the makespan and the normalized schedule length as performance metrics. The makespan (MS) of a BoT is defined as the difference between the earliest submission time of any of its tasks, and the latest completion time of any of its tasks. The Normalized Schedule Length (NSL) of a BoT is defined as the ratio of its makespan to the sum of the runtimes of its tasks on a reference processor. Lower NSL values are better, in particular, NSL values below 1 (which indicates speedup) are desirable.

We also define and use two consistency metrics to assess different strategies. We define consistency in two dimensions: across BoTs of different sizes, and across BoTs of the same size. For assessing the consistency across BoTs of different sizes, we define

$$C_d = \max_{k,l} \frac{\bar{N_k}}{\bar{N_l}},$$

where N_k (N_l) is the stochastic variable representing the NSL of BoTs of size k (l).

To assess the consistency across BoTs of the same size, we define

$$C_s = \max_{k} CoV(N_k),$$

where $CoV(N_k)$ is the coefficient of variation of N_k. The system gets more consistent as C_d gets closer to 1, and C_s gets closer to 0. We also interpret a tighter range of the NSL as a sign of better consistency.

To evaluate the accuracy of the task runtime predictions when using the ECT policy, we use the accuracy, defined as in [17].

Finally, when evaluating the cost of the strategies, we use the *CPU-hours* metric which we define as the time in hours a processor is used. We believe that this metric is a fair indicator of cost independent of the underlying details like the billing model. When calculating the CPU-hours, we round up the partial instance-hours to one hour similar to the Amazon EC2 on-demand instances pricing model [14]. Although there are other costs like administration and maintenance costs of the resources, we neglect these costs, and we only focus on the resource usage.

VI. Experimental Results

In this section, we present the evaluation of the performance (Section VI-A) and cost (Section VI-B) of the overprovisioning strategies.

A. Performance Evaluation

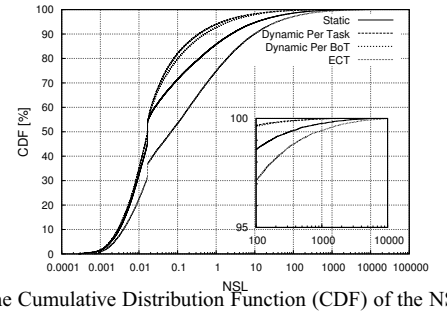

Figure 3. The Cumulative Distribution Function (CDF) of the NSL for the various scheduling policies. The horizontal axis has a logarithmic scale.

Impact of the scheduling policy on performance Figure 3 shows the NSL distribution for all policies when no overprovisioning is applied. Although the Dynamic Per Task and the Dynamic Per BoT policies have similar performance, the Dynamic Per Task policy performs slightly better. The ECT policy has the worst performance by far compared to other policies. When using the ECT policy, the prediction accuracy is around 40%, which is low since all tasks in a BoT arrive within a short time interval, and hence the same prediction error is made for all tasks. This low prediction accuracy leads to scheduling decisions that cause some BoTs to suffer high response times with the ECT policy.

Table III
SUMMARY OF CONSISTENCY VALUES FOR ALL STRATEGIES AND FOR DIFFERENT OVERPROVISIONING FACTORS (κ).

Overprovisioning Strategy	$\kappa = 1.0$ (NO)		$\kappa = 1.5$		$\kappa = 2.0$		$\kappa = 2.5$		$\kappa = 3.0$	
	C_d	C_s	C_d	C_s	C_d	C_s	C_d	C_s	C_d	C_s
`All`	29.59	12.05	15.13	10.54	4.72	9.33	2.64	7.36	2.62	5.38
`Largest`	29.59	12.05	16.88	11.57	3.67	9.27	2.63	7.38	2.63	5.58
`Number`	29.59	12.05	17.71	10.61	3.75	9.12	2.70	6.90	2.42	5.67
`Dynamic`	29.59	12.05	14.65	10.27	3.50	8.64	2.42	6.36	2.10	4.60

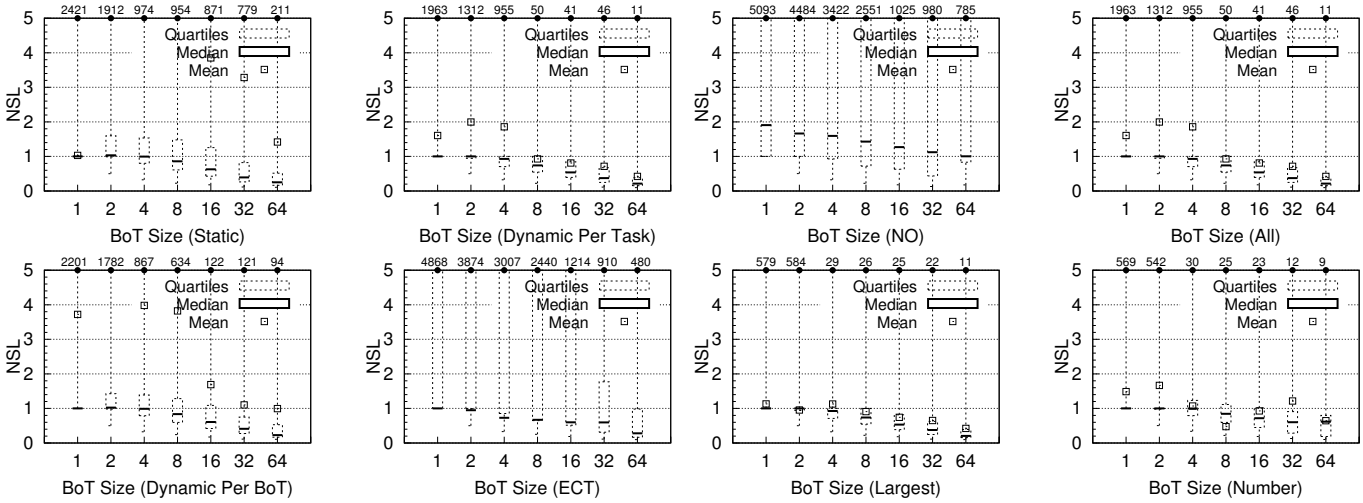

Figure 4. The impact of the scheduling policy on NSL when using the `All` strategy and $\kappa = 2.0$. The mean, and for ECT the third quartile is not always visible.

Figure 5. The NSL distributions for the static strategies (κ = 2.0). The third quartile is not visible for the initial system (NO).

Figure 4 shows the impact of the scheduling policy on the NSL when we use the `All` overprovisioning strategy and κ is 2.0. In this section, for the box-whisker plots, the values at the top of the graphs are the maximum values observed, which are probably outliers, so what we are really interested in are the mean/median values and the quartiles. We observe that as the policy uses more recent system information, the NSL improves (lower interquartile range), hence the NSL of the Dynamic Per Task and Dynamic Per BoT policies is better than that of the other policies.

Since the Dynamic Per Task policy has the best performance among the policies, we use this policy in the rest of our evaluation.

Performance and consistency of the overprovisioning strategies The NSL distributions for the static strategies are shown in Figure 5 and for the `Dynamic` strategy it is shown in the upper-right graph of Figure 7 when κ is 2.0. Corresponding consistency metric values are shown in column 3 of Table III, where the first column ($\kappa = 1.0$) shows the consistency values for the initial system (NO). Clearly, the consistency obtained with different strategies is much better than the initial system due to increased system capacity. We observe that the `Dynamic` strategy provides better consistency compared to static strategies (Table III) since this strategy is able to handle the spikes in the workload that the static strategies can not handle. The static strategies have similar performance, so when overprovisioning a grid statically what really matters is the overprovisioning factor.

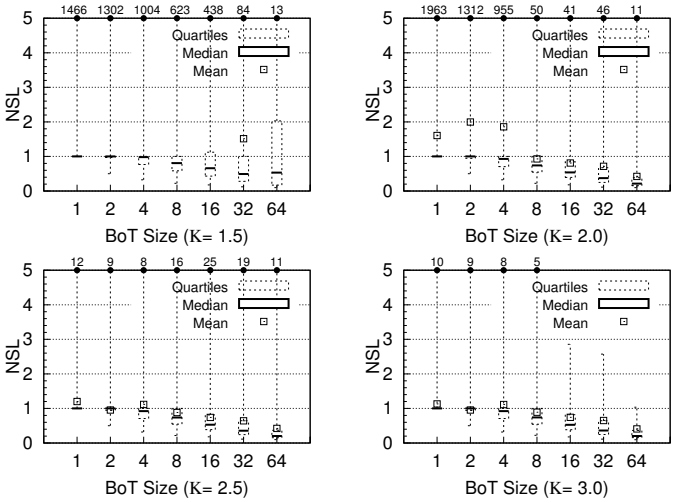

Figure 6. The effect of the overprovisioning factor κ on the NSL distribution with the `All` strategy for $\kappa = 1.5$ (upper left), $\kappa = 2.0$ (upper right), $\kappa = 2.5$ (lower left) and $\kappa = 3.0$ (lower right), respectively. Some of the mean values are not visible for the $\kappa = 1.5$ case.

However, since `Number` increases the number of sites in the grid, hence increasing the administration costs, `All` and `Largest` are the viable candidates among the static strategies.

Impact of the overprovisioning factor κ on consistency Figure 6 and Figure 7 show the effect of κ on consistency with the `All` strategy and the `Dynamic` strategy, respectively. Corresponding consistency metric values are shown in

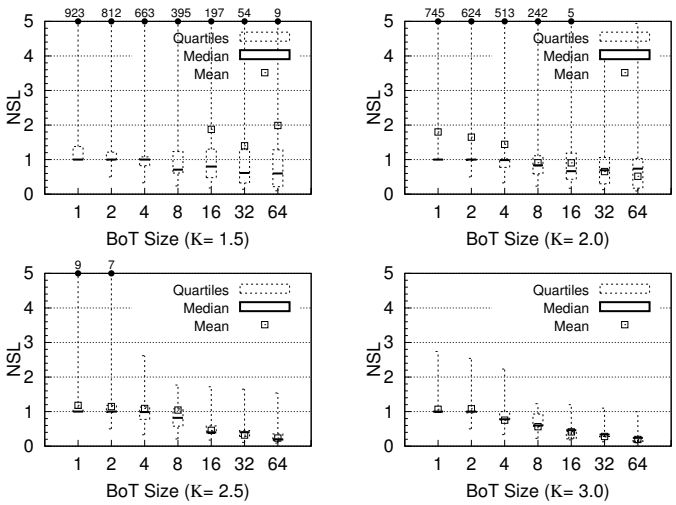

Figure 7. The effect of the overprovisioning factor κ on the NSL distribution with the `Dynamic` strategy for $\kappa = 1.5$ (upper left), $\kappa = 2.0$ (upper right), $\kappa = 2.5$ (lower left) and $\kappa = 3.0$ (lower right), respectively. Some of the mean values are not visible for the $\kappa = 1.5$ case.

Table III. As expected, we observe significant improvements in the overall consistency of the system with increasing overprovisioning factors. The outliers that we observe with smaller overprovisioning factors disappear with increasing overprovisioning factors since the overprovisioned system can handle these spikes. In particular, going from $\kappa = 2.0$ to 2.5 dramatically reduces the outliers. Also, the outliers are much smaller for the `Dynamic` strategy than the `All` strategy.

However, we observe minor improvements in consistency as κ increases beyond $\kappa = 2.5$: *the overprovisioned system with $\kappa = 2.5$ can already handle the variability in the workload*. Hence overprovisioning beyond a certain value of κ (in our case for $\kappa = 2.5$), which we call the *critical value*, incurs significant costs but does not improve consistency significantly. Therefore, to maximize the benefit of overprovisioning it is important to determine the critical value of the overprovisioning factor.

Finally, the consistency metrics converge to similar values as κ approaches 3.0 (see Table III). Although the system is overprovisioned significantly when $\kappa = 3.0$, there is still some variability in the performance which is probably due to the variability inherent in the workload.

B. Cost Evaluation

Due to the dynamic nature of grid workloads, static strategies may cause underutilization and hence increase the costs. The on-demand resource provisioning approach used with the `Dynamic` strategy overcomes these problems. In this section we evaluate the cost of the strategies for various overprovisioning factors to understand how much we can gain in terms of cost when using the `Dynamic` strategy. We use the CPU-hours metric described in Section V-C to assess the cost of the strategies.

Table IV shows the cost of the `All` and `Dynamic` strategies for different overprovisioning factors. In this table, we only report the results for the `All` strategy since the cost is the same for different static strategies for the same overprovisioning factor. Although the cost increases proportionally with κ, we do not observe proportional performance improvement as we already show in Section VI-A. This situation is due to the underutilization of resources caused by static allocation. When using the `Dynamic` strategy, there is a significant reduction, as high as 42%, in cost since the resources are only acquired on-demand, and they are not allowed to stay idle as with static overprovisioning. As κ increases, the number of idle resources in the cloud also increases, hence decreasing the cost reduction. As a result, we conclude that the `Dynamic` strategy provides better consistency with lower costs compared to static strategies.

Table IV
COST OF THE ALL AND DYNAMIC STRATEGIES IN TERMS OF CPU-HOURS.

κ	`All`	`Dynamic`	Reduction (%)
1.5	56655	32446	42.7
2.0	75540	49427	34.5
2.5	94425	69572	26.3
3.0	113310	85484	24.5

VII. DYNAMICALLY DETERMINING THE OVERPROVISIONING FACTOR

Up to this point, we evaluated the performance and cost of various strategies from the *system's perspective* with different overprovisioning factors and scheduling policies. In particular, our goal was to improve the *system's performance consistency*. We now approach our problem from the *user's perspective*, and we answer the question of how can we dynamically determine the overprovisioning factor to give performance guarantees to users. As our aim is to determine the overprovisioning factor and deploy additional processors *dynamically* to meet *user specified performance requirements*, in this section we only use the `Dynamic` strategy. Towards this end, we design a feedback-controlled system which exploits the elasticity of clouds to dynamically determine κ for specified performance requirements. Instead of a control-theoretical method, we follow an approach inspired by the controllers in the SEDA architecture [21].

For the controller operation, the administrator specifies various parameters shown in Table V. The `Window` parameter determines the number of BoTs that should be completed before the controller activation, hence, it determines how frequently the controller is activated and how fast it reacts to changes in the system performance. The `TargetMakespan` parameter determines the makespan target that the controller has to meet, and the

Table V
THE CONTROLLER PARAMETERS WITH THEIR CORRESPONDING DESCRIPTIONS.

Parameter	Description
`Window`	Number of BoTs completed before controller activation
`TargetMakespan`	The target makespan
`ReleaseThreshold`	The makespan threshold used to release cloud resources

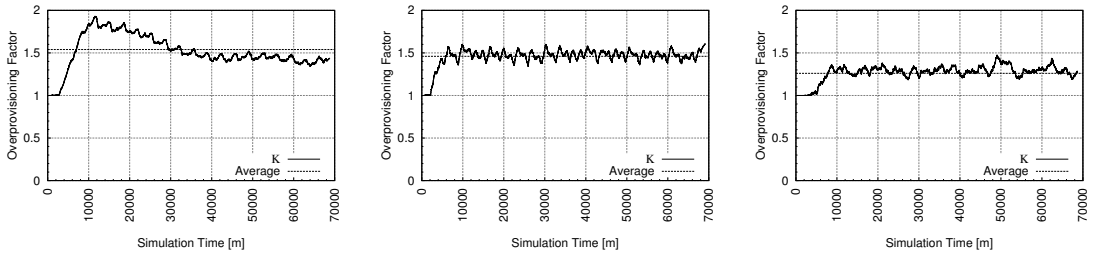

Figure 8. Overprovisioning factor over time and the average overprovisioning factor for the [ReleaseThreshold-TargetMakespan] values of [250m-300m] (left), [700m-750m] (center), and [1000m-1250m] (right).

ReleaseThreshold parameter determines the makespan threshold the controller uses to release cloud resources. The aim of the controller is to meet the TargetMakespan while at the same time avoid wasting resources when unneeded using the ReleaseThreshold. When specifying the TargetMakespan and ReleaseThreshold parameters, we use the 90th percentile of the makespan. This metric has two advantages compared to other metrics like the average or maximum: it better characterizes the makespan distribution, and we also believe that it reflects the user-perceived performance of the system better. To determine the sensitivity of the controller to the parameters of Table V, we have performed various simulations with different parameter values except for the Window parameter, for which we use the value of ten BoTs.

In our architecture, the controller treats the system as a black-box, and it measures the performance of the system at each activation using the historical performance data of the most recently completed BoTs. At each activation, if the measured performance exceeds the TargetMakespan value, the controller instructs the acquisition of a resource from the cloud. Similarly, if the measured performance falls below the ReleaseThreshold value, the controller instructs the release of a resource to the cloud. The provisioning of resources are performed one by one, and we leave as future work to determine the optimal number of resources to provision simultaneously.

To evaluate our design, we simulate the DAS-3 grid and we use the Dynamic Per Task scheduling policy without any background load. For these simulations, to empirically show that the controller stabilizes, we use an approximately one and a half month long workload consisting of 32860 BoTs, and the average BoT makespan for the workload in the initial system (without the controller) is roughly 3120 minutes (m). In our simulations we evaluate three different scenarios for loose and tight performance requirements. To this end, we use the [ReleaseThreshold-TargetMakespan] values of [250m-300m], [700m-750m], and [1000m-1250m] from tight to loose makespan performance requirements, respectively.

Figure 8 shows the overprovisioning factor over time for the different performance requirements. Initially, there are no resources used from the cloud, hence $\kappa = 1$. The controller uses fewer cloud resources as the performance requirements get looser, resulting in lower overprovisioning factors compared to tight performance requirements. In addition, the average overprovisioning factor is smaller for loose performance requirements. It is also remarkable to note that when the performance requirements get tighter, there is only a rather small increase in the overprovisioning factor.

Table VI shows the number of BoTs that meet the specified performance requirements (having a makespan less than the TargetMakespan value) without and with the controller, and the improvement (%) in the number of BoTs with the controller over the system without the controller. There is a significant improvement as high as 67% when the performance requirements are tight. The improvement gets smaller as the performance requirements get looser, as expected, since the system without the controller is already able to meet such loose performance requirements.

VIII. Related Work

We classify the previous work into two categories where the primary focus is either on predictable performance or Service Level Agreements (SLA). Although an extensive body of research focused on these research problems, there is no detailed investigation of how we can achieve consistent performance in grids. In our recent work [11], we took the first step towards filling this gap and we evaluated the performance of static overprovisioning strategies. In this work we extend our previous work by evaluating the performance and cost of both static and dynamic overprovisioning strategies with realistic simulations. We summarize the related work below.

Related work on predictable performance Various studies investigated advance reservations [22]–[24], which is known to increase predictability reserving resources in advance, and therefore providing the requested resources exactly when needed. We believe that advance reservations can also be used for providing consistent performance since the reserved resources are guaranteed to be available when needed (as-

Table VI
Number of BoTs (out of 32860) that meet the specified performance requirements without (w/o) and with (w) the controller, and the resulting improvement (% of 32860) over the system without the controller.

[ReleaseThreshold-TargetMakespan]	w/o Controller	w Controller	Improvement
[250m-300m]	4732	26849	67%
[700m-750m]	5900	21870	48%
[1000m-1250m]	6959	20219	40%

suming no failures occur).

As another solution to provide predictable performance, researchers investigated prediction methods to predict various parameters like job runtimes and queue wait times. These predictions have been used for various decisions in the system like scheduling or admission control [5]–[7].

The primary focus of these studies is on providing *predictable* performance by using advance reservations and predictions. In contrast, our primary focus is on providing *consistent* performance using overprovisioning.

Related work on SLAs In [8], authors propose a dynamic offloading architecture similar to our feedback-controlled system to meet the SLAs. In their architecture they have idle servers in the system, and they deploy additional applications to meet the specified performance requirements; in contrast we use computing clouds instead of keeping processors idle. In their work, the authors perform experiments with web workloads, in contrast we use realistic grid workloads to perform a more detailed investigation than theirs to investigate various tight and loose performance requirements.

In [9], authors present an approach for QoS aware resource management in grids using online performance models. They show that negotiated SLAs are satisfied with the proposed approach. In contrast, we use overprovisioning to meet the performance requirements. In addition, our work targets multicluster grids whereas their work targets service-based grids on which services are deployed, and the workload consists of service requests.

IX. Conclusion

Providing consistent performance in grids is a difficult research problem. In this work we investigated overprovisioning to solve this problem, and we have performed a realistic evaluation of overprovisioning in grids. Although our main focus is on grids, we believe that the main ideas are also applicable to other large-scale distributed systems.

We presented a realistic evaluation of various overprovisioning strategies with different overprovisioning factors (κ) and scheduling policies. We find that beyond a certain value for the overprovisioning factor (in our case $\kappa = 2.5$) there is only slight improvement in consistency with significant additional costs. In addition, the `Dynamic` strategy provides better consistency with lower costs compared to static strategies. Finally, to dynamically determine the overprovisioning factor to give performance guarantees to users, we designed and evaluated a feedback-controlled system exploiting the elasticity of clouds. Through various simulations for loose and tight makespan performance requirements, we showed that our system provides significant improvements over the initial system, as high as 67%, in the number of BoTs that meet the specified performance requirements.

References

[1] A. Pras, R. V. D. Meent, and M. Mandjes, "Qos in hybrid networks - an operators perspective," in *13th International Workshop on Quality of Service*, 2005.

[2] A. Andrzejak, M. Arlitt, and J. Rolia, "Bounding the resource savings of utility computing models," HP, Tech. Rep., 2002.

[3] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *IEEE Computer*, vol. 40, pp. 33–37, 2007.

[4] O. Andrew, "Data networks are lightly utilized, and will stay that way," *Review of Network Economics*, vol. 2, pp. 210–237, 1999.

[5] M. Dobber, R. van der Mei, and G. Koole, "A prediction method for job runtimes on shared processors: Survey, statistical analysis and new avenues," *Performance Evaluation*, vol. 64, pp. 755–781, 2007.

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

[7] R. Wolski, "Experiences with predicting resource performance on-line in computational grid settings," in *Performance Evaluation Review*, 2006, pp. 575–611.

[8] A. Leff, J. T. Rayfield, and D. M. Dias, "Service-level agreements and commercial grids," *IEEE Internet Computing*, vol. 7, pp. 44–50, 2003.

[9] S. Kounev, R. Nou, and J. Torres, "Autonomic qos-aware resource management in grid computing using online performance models," in *2nd Int'l Conference on Performance Evaluation Methodologies and Tools*, 2007, pp. 1–10.

[10] R. Al-Ali, K. Amin, G. von Laszewski, O. Rana, D. Walker, M. Hategan, and N. Zaluzec, "Analysis and Provision of QoS for Distributed Grid Applications," *J.of Grid Computing*, vol. 2, pp. 163–182, 2004.

[11] N. Yigitbasi and D. Epema, "Overdimensioning for consistent performance in grids," in *CCGRID*, 2010, pp. 526–529.

[12] "The distributed asci supercomputer," http://www.cs.vu.nl/das3/.

[13] A. Iosup, C. Dumitrescu, D. Epema, H. Li, and L. Wolters, "How are real grids used? the analysis of four grid traces and its implications," in *GRID*, 2006, pp. 262–269.

[14] Amazon, "http://aws.amazon.com/ec2/," 2008.

[15] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, "C-meter: A framework for performance analysis of computing clouds," in *Int'l Workshop on Cloud Computing*, 2009.

[16] "SPEC CPU2006 Benchmark," [Online]: http://www.spec.org/cpu2006/.

[17] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates," *IEEE Trans. on Parallel and Distributed Systems*, vol. 18, pp. 789–803, 2007.

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

[19] A. Iosup, O. Sonmez, and D. Epema, "Dgsim: Comparing grid resource management architectures through trace-based simulation," in *EuroPar*, 2008, pp. 13–25.

[20] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema, "The grid workloads archive," *FGCS*, vol. 24, pp. 672–686, 2008.

[21] M. Welsh and D. Culler, "Adaptive overload control for busy internet servers," in *USITS*, 2003, pp. 26–28.

[22] W. Smith, I. T. Foster, and V. E. Taylor, "Scheduling with advanced reservations," in *IPDPS*, 2000, p. 127.

[23] C. Castillo, G. N. Rouskas, and K. Harfoush, "Efficient resource management using advance reservations for heterogeneous grids," in *IPDPS*, 2008, pp. 1–12.

[24] D. C. Nurmi, R. Wolski, and J. Brevik, "Varq: virtual advance reservations for queues," in *HPDC*, 2008, pp. 75–86.