

Overdimensioning for Consistent Performance in Grids

Nezih Yigitbasi, Dick Epema
Delft University of Technology

Abstract—Grid users may experience inconsistent performance due to specific characteristics of grids, such as fluctuating workloads, high failure rates, and high resource heterogeneity. Although extensive research has been done in grids, providing consistent performance remains largely an unsolved problem. In this study we use *overdimensioning*, a simple but cost-ineffective solution, to solve the performance inconsistency problem in grids. To this end, we propose several overdimensioning strategies, and we evaluate these strategies through simulations with workloads consisting of Bag-of-Tasks. We find that although overdimensioning is a simple solution, it is a viable solution to provide consistent performance in grids.

I. INTRODUCTION

Users expect consistent performance from computer systems—when a certain Bag-of-Tasks (BoT) submitted to a grid has a turnaround 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. In this study we investigate overdimensioning for solving the performance inconsistency problem in grids.

We define overdimensioning as increasing the capacity, by a factor that we call the *overdimensioning factor*, of a system to handle the fluctuations in the workload, and provide consistent performance even under unexpected demands. It has been used in many different systems like telecommunication systems, and data centers where the scarce resource is the bandwidth and processors, respectively. Although overdimensioning is a simple solution for consistent performance, it is not cost effective and causes systems to be under-utilized most of the time. Studies have shown that typical data center utilization is no more than 15-50% [1], and telecommunication systems have roughly 30% [2] utilization on average.

Previous work focuses either on providing predictable performance using advance reservations or prediction methods [3], [4], [5], or meeting the performance requirements specified by Service Level Agreements (SLA) [6], [7]. In contrast, in this study we present a detailed investigation of overdimensioning as a solution for achieving consistent performance in grids. To this end, we propose several overdimensioning strategies that are applicable in grids. Through simulations with various synthetic workloads consisting of BoTs, which

{M.N.Yigitbasi, D.H.J.Epema}@tudelft.nl. This work was carried out in the context of the Guaranteed Delivery in Grids (GUARD-G) project (<http://guardg.st.ewi.tudelft.nl/>), which is supported by NWO.

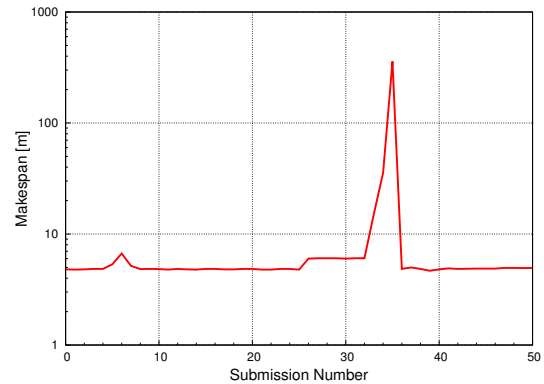


Fig. 1. Evidence of the performance inconsistency in grids. The vertical axis has a logarithmic scale.

constitute the dominant application type in grids [8], we assess the performance of the overdimensioning strategies with different scheduling policies.

II. PROBLEM DESCRIPTION

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 the application, not by the system. 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 [9], which is usually lowly utilized. 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 30 submissions. However, we observe a significant variability between the 30th and 40th submission, which is due to the high utilization of the system during these submissions 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 lowly utilized, we may observe very strong performance inconsistencies. It is the focus of our study to investigate overdimensioning to solve the performance inconsistency problem in grids.

III. OVERDIMENSIONING STRATEGIES

We define overdimensioning as increasing the capacity of a system to provide consistent performance under variable

workloads. We define the *overdimensioning factor* κ as the ratio of the size of an overdimensioned system to the size of the initial system. Although overdimensioning can be used for consistent performance, there are also some disadvantages of this approach. First, overdimensioning is of course a cost-ineffective solution. Secondly, the user requirements and the workload of a grid may also evolve in the future such that the overdimensioned system may not be able to meet the new demands. Hence, there is no guarantee that overdimensioning is a long-term solution for consistent performance. Finally, overdimensioning causes the system to be underutilized since resources will stay idle most of the time: the industry is used to low utilizations in data centers, and in telecommunication systems where the utilization is in the range 15-50% [1], and where the average utilization is roughly 30% [2], respectively.

To overdimension grids we propose the following **static** overdimensioning strategies which deploy additional processors to the existing grid infrastructure:

- **Overdimension Largest Site (Largest):** Only the largest site of the grid in terms of the number of processors is overdimensioned in this strategy.
- **Overdimension All Sites (All):** All of the sites of the grid are overdimensioned equally.
- **Overdimension Number of Sites (Number):** The number of sites of the grid is overdimensioned without changing the number of processors in each site. When the grid sites are of different size, this strategy can be applied by adding sites of the average size.

IV. SYSTEM MODEL

We assume a grid that consists of multiple homogeneous clusters; although the processors may have different performance across different clusters, they are identical in the same cluster. We assume that there is a Global Resource Manager (GRM) in the system interacting with several LRMs which are responsible for managing the resources of each cluster. The jobs are queued in the GRM's queue upon their arrival into the system, and then dispatched to the LRMs where they wait for available resources on which to be executed. Once started, jobs run to completion, so we do not consider preemption or migration during execution.

As the application type, we use BoTs, which are the dominant application type in grids [8]. To model the execution time of applications, we employ the SPEC CPU benchmarks [10]: 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 make use of any information about the system. 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 scheduling decisions. We consider two variants of dynamic scheduling:

TABLE I
THE DISTRIBUTIONS AND THE VALUES FOR THEIR PARAMETERS FOR THE REALISTIC BoT WORKLOAD MODEL DESCRIBED IN [12]. $N(\mu, \sigma^2)$ AND $W(\lambda, k)$ STAND FOR THE NORMAL AND WEIBULL DISTRIBUTIONS, RESPECTIVELY.

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

- **Dynamic Per Task Scheduling:** 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:** A separate scheduling decision is made for each BoT, and the whole BoT is sent to the single site which is the least loaded.
- **Prediction-based Scheduling** We consider only **Earliest Completion Time (ECT)** policy which makes use of historical data to predict the task runtimes. ECT submits each task to the cluster which is predicted to lead to the earliest completion time [11] taking into account the clusters' queues. When predicting the task runtimes, the historical data can be kept in two ways: user-based or batch-based [12]. In our study, we employ a user-based task runtime prediction policy. To predict the runtime of a task, we use the average of the runtimes of the previous two tasks [13], since this method is known to perform well in grid computing environments [14].

V. EXPERIMENTAL SETUP

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

A. Simulator and Assumptions

We have extended our event-based grid simulator DGSim [15] with the BoT scheduling policies described in Section IV. We simulate a homogeneous multi-cluster grid consisting of four sites with 50 processors each (this is the size of the initial system). Throughout our simulations we assume that there is no network communications overhead, and there exists no background load and all the load arrives directly to the GRM.

B. Workload

In this study we have performed experiments with two types of workloads: *realistic* and *simplified*. For the realistic workloads we use the BoT model described in [12]. The values for the important workload attributes are summarized in Table I. For the simplified workloads, we modify the realistic BoT model such that all tasks of the same BoT arrive at the same time, and that the BoT interarrival times and the task runtimes have exponential distributions.

For our experiments, we have generated ten synthetic 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 4125 BoTs, and 25K tasks, and the duration of each trace is roughly between 1 day and 1 week. To scale the traces for various loads, we used the method of modifying the interarrival times of the BoTs in the trace.

C. The Performance Metrics

To assess the performance of various strategies, we used **makespan (MS)** and the **Normalized Schedule Length (NSL)**. Makespan of a BoT is the difference between the earliest time of submission of any of its tasks, and the latest time of completion of any of its tasks. 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.

To assess the consistency provided by different strategies, we define and use two metrics which capture the notion of 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 is the stochastic variable representing the NSL of BoTs of size k .

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 employing the ECT policy, we use the accuracy, defined as in [13].

VI. EXPERIMENTAL RESULTS

In this section, we present the evaluation of the performance of the overdimensioning strategies that we proposed in Section III. For the box-whisker plots presented in this section, 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.

Impact of the scheduling policy on performance Figure 2 shows the makespan distribution for all policies when no overdimensioning is applied. Among the policies, the Dynamic Per Task policy has the best performance, and the ECT policy has the worst performance by far. When using the ECT policy, the prediction accuracy is around 40%, which is low since all tasks in a BoT arrive at the same time (see Section V-B), and

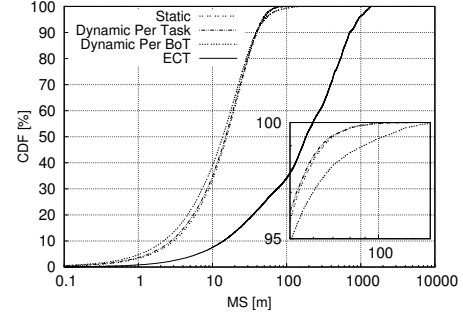


Fig. 2. The CDF of the makespan for the various scheduling policies. Note that the horizontal axis has a logarithmic scale.

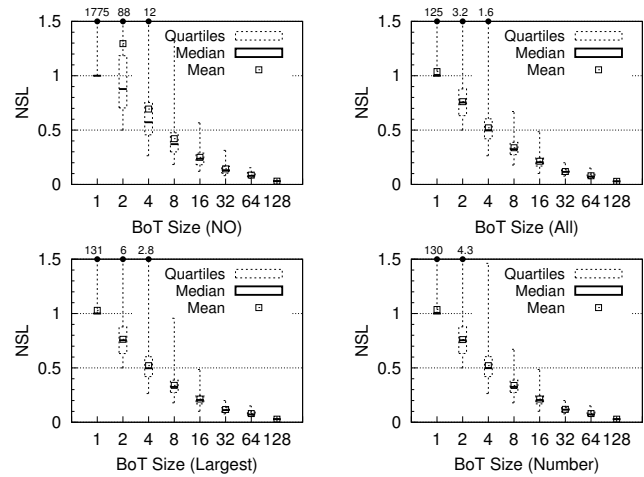


Fig. 3. The NSL distributions for the static overdimensioning strategies ($\kappa = 2.0$).

hence the same prediction error is made for all tasks. 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 overdimensioning strategies The NSL distributions for the static overdimensioning strategies are shown in Figure 3 when κ is 2.0 and the Dynamic Per Task scheduling policy is used. Corresponding values for the consistency metrics are shown in column 3 of Table II, where the first column ($\kappa = 1.0$) shows the consistency values for the initial system (NO). The consistency obtained with different overdimensioning strategies is much better than the initial system, as expected. Among the static strategies we observe similar performance although All performs slightly better. In addition, All and Largest are viable alternatives to Number since Number increases the administration, installation and maintenance costs as it increases the number of sites in a grid.

The consistency of the overdimensioned system ($\kappa = 1.5$) improves significantly compared to the initial system. However, if the system is overdimensioned beyond $\kappa = 1.5$, we observe minor improvements in consistency compared to the overdimensioned system with $\kappa = 1.5$: the overdimensioned system with $\kappa = 1.5$ can already handle the fluctuations in the

TABLE II
SUMMARY OF CONSISTENCY VALUES FOR ALL STRATEGIES AND FOR DIFFERENT κ VALUES WITH THE SIMPLIFIED WORKLOAD MODEL.

Overdimensioning 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	56.83	21.19	16.71	15.77	14.44	3.99	13.96	1.71	13.90	1.71
Largest	56.83	21.19	17.05	19.81	14.27	1.71	14.02	1.71	13.86	1.71
Number	56.83	21.19	16.71	10.78	14.43	5.24	13.97	3.33	13.89	1.71

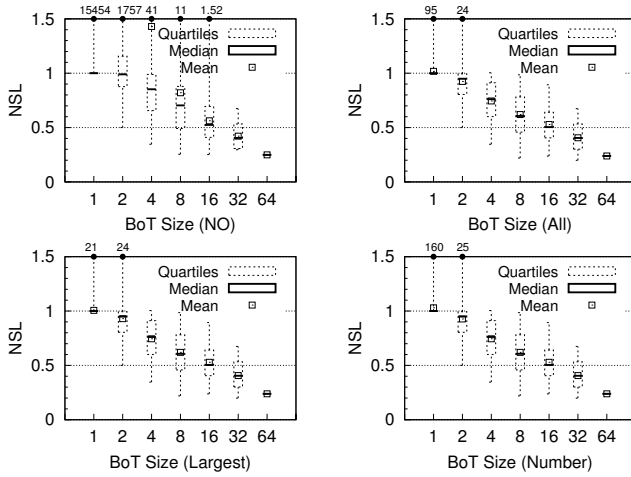


Fig. 4. Comparison of the NSL distributions for the static overdimensioning strategies with the realistic workload model ($\kappa = 2.0$).

workload. We claim that overdimensioning beyond a certain value of κ (in our case for $\kappa = 1.5$), which we call the *critical value*, incurs significant costs but does not improve consistency significantly compared to the overdimensioned system with the critical overdimensioning factor. Hence, it is important to determine the critical value of the overdimensioning factor to maximize the benefit of overdimensioning.

Consistency of the overdimensioning strategies with realistic workloads Up to this point, we have used the simplified workload model that we described in Section V-B. To confirm our findings, we have also performed simulations with the realistic workload model. Figure 4 shows the consistency of the static strategies when $\kappa = 2.0$.

We observe that the consistency of the system improves significantly as the system is overdimensioned. This results confirms our simulation results for the simplified workload which are shown in Table II. Hence we conclude that overdimensioning is indeed a viable solution for providing consistent performance in grids.

VII. CONCLUSION

In this study we have investigated overdimensioning, which is a simple yet effective solution for providing consistent performance in the context of grids. Although our main focus is on grids, we believe that the general ideas are applicable to other large-scale distributed systems.

We have investigated the performance and consistency of the overdimensioning strategies with different scheduling policies, and for various overdimensioning factors. We found that up to

a certain value for the overdimensioning factor, which we call as the *critical value*, the consistency of the system improves significantly. However, beyond the critical overdimensioning factor, we observe slight performance improvements with significant additional costs. For the future, we plan to investigate mechanisms and policies to give performance guarantees to grid users.

REFERENCES

- [1] A. Andrzejak, M. Arlitt, and J. Rolia, "Bounding the resource savings of utility computing models," HP, Tech. Rep., 2002.
- [2] O. Andrew, "Data networks are lightly utilized, and will stay that way," *Review of Network Economics*, vol. 2, no. 3, pp. 210–237, 1999.
- [3] W. Smith, I. T. Foster, and V. E. Taylor, "Scheduling with advanced reservations," in *Scheduling with Advanced Reservations*, 2000, p. 127.
- [4] M. Siddiqui, A. Villazn, and T. Fahringer, "Grid allocation and reservation - grid capacity planning with negotiation-based advance reservation for optimized qos," in *SC. ACM Press*, 2006, p. 103.
- [5] D. C. Nurmi, R. Wolski, and J. Brevik, "Varq: virtual advance reservations for queues," in *HPDC '08: Proceedings of the 17th Int'l Symposium on High Performance Distributed Computing*. ACM, 2008, pp. 75–86.
- [6] S. Kounev, R. Nou, and J. Torres, "Autonomic qos-aware resource management in grid computing using online performance models," in *ValueTools '07: Proceedings of the 2nd Int'l Conference on Performance Evaluation Methodologies and Tools*, 2007, pp. 1–10.
- [7] R. Al-Ali, K. Amin, G. von Laszewski, O. Rana, D. Walker, M. Hategan, and N. Zaluze, "Analysis and Provision of QoS for Distributed Grid Applications," *Journal of Grid Computing*, vol. 2, no. 2, pp. 163–182, 2004.
- [8] 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 '06: Proceedings of the 7th IEEE/ACM Int'l Conference on Grid Computing*, 2006, pp. 262–269.
- [9] "The distributed ascii supercomputer," 2009, <http://www.cs.vu.nl/das3/>.
- [10] "SPEC CPU 2006. Standard Performance," 2009. [Online]. Available: <http://www.spec.org/cpu2006/>
- [11] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *HCW*, 1999, pp. 30–44.
- [12] A. Iosup, O. Sonmez, S. Anoop, and D. Epema, "The performance of bags-of-tasks in large-scale distributed systems," in *HPDC '08: Proceedings of the 17th Int'l Symposium on High Performance Distributed Computing*, 2008, pp. 97–108.
- [13] D. Tsafir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, 2007.
- [14] O. Sonmez, N. Yigitbasi, A. Iosup, and D. Epema, "Trace-based evaluation of job runtime and queue wait time predictions in grids," in *HPDC '09: Proceedings of the 18th ACM Int'l Symposium on High Performance Distributed Computing*. ACM, 2009, pp. 111–120.
- [15] A. Iosup, O. Sonmez, and D. Epema, "Dgsim: Comparing grid resource management architectures through trace-based simulation," in *Euro-Par '08: Proceedings of the 14th Int'l Euro-Par Conference on Parallel Processing*, 2008, pp. 13–25.