Stochastic Tail-Phase Optimization for Bag-of-Tasks Execution in Clouds

Ana-Maria Oprescu, Thilo Kielmann, Haralambie Leahu
Dept. of Computer Science, VU University Amsterdam, The Netherlands
amo@cs.vu.nl, kielmann@cs.vu.nl, haralambie@gmail.com

***Abstract*—Elastic applications like bags of tasks benefit greatly from Infrastructure as a Service (IaaS) clouds that let users allocate compute resources on demand, charging based on reserved time intervals. Users, however, still need guidance for mapping their applications onto multiple IaaS offerings, both minimizing execution time and respecting budget limitations. For budget-controlled execution of bags of tasks, we built *BaTS*, a scheduler that estimates possible budget and makespan combinations using a tiny task sample, and then executes a bag within the user's budget constraints. Previous work has shown the efficacy of this approach. There remains, however, the risk of outlier tasks causing the execution to exceed the predicted makespan.**

In this work, we present a stochastic optimization of the tail phase for BaTS' execution. The main idea is to use the otherwise idling machines up until the end of their (already paid-for) allocation time. Using the task completion time information acquired during the execution, BaTS decides which tasks to replicate onto idle machines in the tail phase, reducing the makespan and improving the tolerance to outlier tasks. Our evaluation results show that this effect is robust w.r.t. the quality of runtime predictions and is the strongest with more expensive schedules in which many fast machines are available.

***Index Terms*—Scheduling, Budget, Task Replication**

I. INTRODUCTION

In computational science, parameter sweep or bag-of-tasks applications are as dominant as computationally demanding. An ideal match for these demands can be seen in commercial cloud offerings, like Amazon's EC2, where computers can be allocated ("rented") for given time intervals. The various commercial offerings differ not only in price, but also in the types of machines that can be allocated. While all machine types are described in terms of CPU clock frequency and memory size, it is not clear at all which machine type can execute a given user application faster than others, let alone predicting which machine type can yield the best price-performance ratio. The problem of allocating the right number of machines, of the right type, for the right time frame, strongly depends on the application program, and is left to the user.

This problem is addressed by BaTS, our budget-constrained scheduler for bags of tasks [1]. BaTS requires no a-priori information about task completion times. In a small, initial sampling phase, BaTS learns properties of the bag's runtime distribution for each considered cloud offering. BaTS provides the user with a list of makespan/cost options (with different combinations of machines) to choose from, ranging from the cheapest to the fastest offer. During the bulk execution, BaTS monitors progress and adjusts the machine configuration if necessary, based on the additionally gained information.

We have shown that BaTS allows a user to favor either money or time for the execution of a given bag. BaTS can enforce the execution to happen according to its predictions, however only within certain error margins, as all information is purely statistical, and late-scheduled outliers can always break a planned schedule. This is caused by BaTS focusing on the high-throughput phase of the computation, where average execution times for each machine type are sufficient to determine the overall makespan and budget.

In this paper, we focus on the tail phase of a bag's execution. Because individual tasks are indivisible and have individual completion times, the final task on each machine has a different completion time, leading to substantial amounts of unused compute power during the final phase (e.g., hour) of a computation. As the user has already paid for the remaining time of the increasingly idling machines, we have devised a scheme by which to utilize this idle capacity in order to speed up the overall computation, without incurring extra cost.

The contribution of this paper is a scheme that replicates running tasks onto idle rented machines whenever BaTS can predict, based on runtime statistics, that the replica will terminate before the already running task instance. We have evaluated our strategy by extensive simulation runs with thirty bags each of a normal runtime distribution, a multi-modal distribution, and a (heavy-tailed) Levy-truncated distribution. Our results show that our tail-phase replication indeed improves the overall makespan and reduces occurring violations of the estimated makespans, the extent of which depending on the given runtime distribution and machine configurations, the strongest effect being shown with more expensive schedules in which many fast machines are available. Also, the selection of tasks to be replicated is shown to be fairly robust w.r.t. the quality of the runtime estimations; even replicating randomly selected tasks performs only little worse than using perfectly-known completion times, albeit with higher error rates.

This paper is structured as follows. Sec. II describes the BaTS scheduler. Sec. III presents our new tail-phase task replication mechanism, that is evaluated in Sec. IV. Related work is discussed in Sec. V; we conclude in Sec. VI.

II. THE BaTS SCHEDULER

BaTS is scheduling large bags of tasks onto multiple cloud platforms. The individual tasks are scheduled in a self-scheduling manner onto the allocated machines. An initial sampling phase computes a list of budget estimates providing the user with flexible control over budget and makespan. The execution phase allocates a number of machines from different clouds, and adapts the allocation regularly by acquiring and/or releasing machines in order to minimize the overall makespan while respecting the given budget limitation.

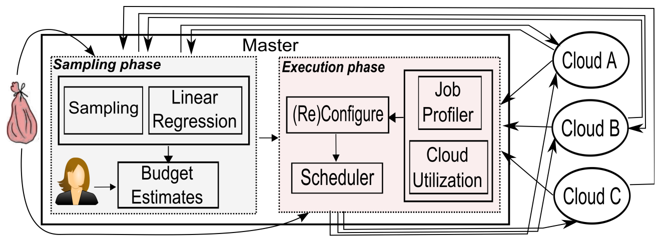

Fig. 1. BaTS sampling phase (left) and execution phase (right).

Our task model needs no prior knowledge about the task execution times; it only needs the total number of tasks. About the machines, we assume that they belong to certain categories (e.g. EC2's "Standard Large") and that all machines within a category are homogeneous. The only information BaTS uses about the machines is their (hourly) price.

Figure 1 sketches BaTS' overall system architecture. BaTS itself runs on a *master* machine, where the bag of tasks is available. Figure 1(left) sketches the sampling phase, where BaTS learns the bag's stochastic properties and uses linear regression to translate task completion times across clouds [1]. BaTS generates a list of budget estimates accordingly, reflecting execution speed and profitability (price/performance ratio). The user is then asked to select one of the budgets (e.g., faster or cheaper) corresponding to a desired schedule. The user's choice then determines the machines allocated by BaTS for the execution phase, shown in Figure 1(right). Here, BaTS allocates machines from various clouds and lets the scheduler dispatch the tasks to the cloud machines. Feedback, both about task completion times and cloud utilization, is used to reconfigure the clouds periodically, as needed.

Our very general task model does not allow any hard budget guarantees for the execution of the entire bag. Since BaTS has no a-priori information about the individual execution time of each task, we cannot guarantee that a certain budget will be definitely sufficient. The case might always occur that one or more outlier tasks, with exceptionally high completion time, might be scheduled only towards the end of the overall execution. In this paper, we present an optimization for the tail phase execution that reduces the severeness of this problem.

III. A Tail-Phase replication scheme

We assume that tasks are indivisible and have individual completion times, thus the final task on each machine has a different completion time. This yields significant amounts of unused, but paid-for compute power during the final phase of the computation. Our main idea is to utilize this already paid-for idle capacity and speed up the overall computation.

We propose a task replication scheme active only in the tail-phase. We define the beginning of the *tail* phase as the moment when the first machine becomes idle *and* there are no more unsubmitted tasks in the bag. The tail-phase ends when all tasks are finished. In the tail-phase, if a worker becomes idle while tasks are still executing, this worker will be used until the end of its paid-for period for executing replicated tasks.

Replication aims at using idle machines of relatively faster types than those currently executing a particular task. When *replicating* a task, BaTS starts a replica of a running task from the beginning, on an idle machine, while letting the original task running. The first one to complete will render the result.

To select replication candidates, BaTS compares the estimated remaining execution time of a task on the current machine (source machine) to the expected execution time of the same task when started on a machine that has become idle (target machine). If the task is expected to finish earlier on the target machine, it becomes a replication candidate. If a task has already multiple replicas, it enters the selection process with the minimum estimated remaining execution time among the replicas. Among the replication candidates, BaTS chooses the task with the largest expected remaining execution time.

The remaining execution time on the source machine is estimated using conditional expectation mechanisms. We take the discrete approach to conditional expectation and let BaTS construct the corresponding histogram during the *high-throughput* phase. The elapsed execution time of a task and the histogram are used to estimate the total execution time of that task. Using the regression coefficients (across clouds) computed during the sampling phase [1], BaTS converts task runtimes from different machine types to a canonical machine type (arbitrarily chosen from those available during the sampling phase). In this manner, it maintains a unified histogram of all finished task runtimes observed during the execution.

BaTS computes the estimated execution time of a task based on the histogram. It selects a subset of the histogram made up of task runtimes higher than the canonized elapsed runtime of the considered task. The execution time of the task on the canonical machine type is estimated as the average of the runtimes in the subset, weighted by their occurrences.

Let the elapsed execution time of a task on the source machine be τ_s, the elapsed execution time converted to the canonical machine type be τ_c, the estimated total execution time on the source machine be t_s, the estimated total execution time on the canonical machine type be t_c and the estimated total execution time on the target machine be t_t. Using to the linear regression assumption and given that t_c is computed as the conditional expectation of the random variable that is the task runtime, for values larger than τ_c, we can write:

$$\tau_c = \beta_{1_{c,s}} * \tau_s + \beta_{0_{c,s}} \quad ; \quad t_c = E[t | t > \tau_c]$$

Once we obtain t_c from the above formula, and since all the regression parameters $\beta_{1_{c,s}}$, $\beta_{0_{c,s}}$, $\beta_{1_{s,c}}$, $\beta_{0_{s,c}}$, $\beta_{1_{t,c}}$ and $\beta_{0_{t,c}}$ were computed during the sampling phase, we can apply again the linear regression assumption to obtain t_s and t_t

$$t_s = \beta_{1_{s,c}} * t_c + \beta_{0_{s,c}} \quad ; \quad t_t = \beta_{1_{t,c}} * t_c + \beta_{0_{t,c}}$$

A task is considered a replication candidate if the expected remaining execution time ($t_s - \tau_s$) is larger than the expected execution time on the target machine(t_t). The task with the largest expected remaining execution time among the candidates will be replicated on the target machine.

If there is no candidate when a machine becomes idle, BaTS has to decide whether to release the machine or to replicate a random task that is already being executed by a different machine. The former strategy represents the pure stochastic replication, while the latter represents a hybrid stochastic-random replication. We have implemented both strategies and we will compare their efficacy in the evaluation Sec..

IV. Evaluation

We evaluate our two stochastic tail phase strategies (a) pure stochastic replication and (b) stochastic-random replication by comparison to three alternative approaches: (c) using (in practice not available) perfect knowledge about task completion times for selecting tasks to replicate, giving the theoretical optimum. (d) selecting tasks for replication at random, modeling zero knowledge about completion times, and (e) not performing any tail optimization at all. The random approaches (b) and (d) keep all worker nodes active as long as tasks are executed by slower machines. Strategies (a) and (c) release workers if no replication candidates can be found.

A. Workloads

We use three different kinds of workloads, modeled after real bags of tasks, using a normal distribution, a Levy-truncated distribution, and a multi-modal mixture of distributions. Each workload contains 1000 tasks, with runtimes generated according to the respective distribution type.

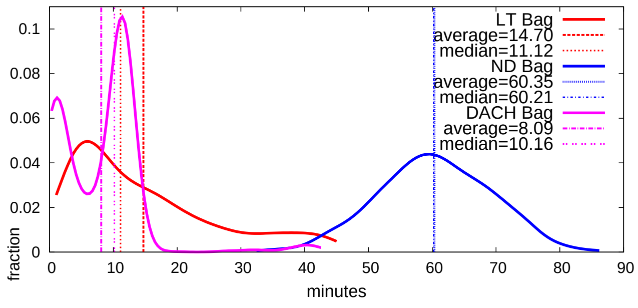

Fig. 2. Distribution types used for workloads generation: normal distribution (**NDB**) with an expectation of 60 min., Levy-truncated (**LTB**) with an upper bound of 45 min. and a real-world multi-modal distribution (**DACH**).

The work in [2] has shown that, in many cases, the intra-BoT distribution of execution times follows a normal distribution. Accordingly, we have generated a workload following the normal distribution $N(60, \sigma^2), \sigma = 4\sqrt{5}$ (in minutes, see Fig. 2), abbreviated as **NDB** ("normal distribution bag").

It has been shown [3] that some bags of tasks have a skewed distribution, bounded by some maximum value. To model such bags, we generate workloads according to a truncated Levy distribution with a scaling factor (τ) of 12 minutes and a maximum value (b) of 45 minutes, as shown in Fig. 2, abbreviated as **LTB** ("Levy-truncated distribution bag").

Another real application is taken from The First International Data Analysis Challenge for Finding Supernovae (DACH) [4]. The task runtimes depicted in Fig. 2 were obtained by running the entire workload on a reference machine. We model this workload as a mixture of distributions generating workloads of which task runtimes exhibit a root-mean square deviation from the real workload of less than 5%. We refer to the corresponding workload as a *multi-modal* distribution bag, abbreviated as **MDB** ("multi-modal distribution bag").

B. Experimental Setup

BaTS' non-deterministic scheduling requires statistically relevant results, consisting of many experiments. The real-world deployment of such experiments (like in our previous work [1], [5]) would have taken prohibitive amounts of time. Therefore, we have built a simulator for the behaviour of actual executions. At each simulation step, it determines the smallest execution time until a task is completed. At this simulated point in time, the executing machine becomes idle and thus a candidate to start the execution of another task. The simulator focuses on estimated vs. real task runtimes; it ignores machine reconfigurations and VM startup delays, allowing us to study the pure effect of the different tail phase strategies.

We have used 30 different instances of each of the three workload types and executed each instance 30 times for each tail phase optimization strategy using our simulator, avoiding bias by by individual bags or execution orders.

There are two different sets of experiments, using two different (extreme) budget types, taken from the set computed during the sampling phase (see Sec. II): *minimal+10%* and *fastest*. The minimal budget type is of no interest since it usually involves a single machine type, i.e. the most profitable one, rendering replication useless. For each bag instance and budget type we compute the estimated makespans.

For each execution we record the actual makespan and we compute the percentage it is faster or slower than the estimation. Based on these 30 data points (percentages) we compute an average percentage for each bag instance. These bag instance percentages are averaged to obtain the percentage representative for each bag distribution type and each budget type. We also record the minimum and maximum percentages for each distribution type, and the percentage of runs that exceed the predicted makespan. The experiment methodology is repeated for each tail phase strategy. All the results are shown in Table I, grouped by the bag distribution type and presented according to the budget type governing the respective set of experiments: *minimal+10%* and *fastest*.

We consider three cloud offerings, each with a maximum of 24 nodes readily available for renting. We emulate that one cloud ("A") executes tasks according to their generated runtime, another ("B") executes them three times as fast at four times the price per hour and the third one ("C") executes them twice as fast as "A" for the same price per hour.

TABLE I
RUNS EXCEEDING MAKESPAN ESTIMATION AND MAKESPAN VARIATION W.R.T. ESTIMATES (ALL IN %)

budget	strategy	NDB				LTB				MDB			
minimal+10%		exc.	avg	min	max	exc.	avg	min	max	exc.	avg	min	max
	Perfect	56.4	1.08	-3.29	7.50	52.2	4.53	-19.40	70.36	77.0	10.61	-27.36	40.56
	Stoch+Rand	56.9	1.11	-3.24	7.52	52.6	4.55	-19.35	70.38	77.0	10.61	-27.36	40.57
	Stochastic	57.1	1.13	-3.18	7.53	53.6	4.66	-19.01	70.50	77.0	10.67	-27.36	40.90
	Random	57.4	1.15	-3.20	7.56	52.6	4.55	-19.37	70.38	77.0	10.61	-27.36	40.57
	Without	75.4	2.39	-2.10	9.03	59.4	5.40	-17.46	72.51	77.0	10.93	-27.08	42.16
fastest		exc.	avg	min	max	exc.	avg	min	max	exc.	avg	min	max
	Perfect	88.0	3.47	-0.99	10.24	72.1	9.89	-15.70	78.81	92.2	24.14	-18.76	58.48
	Stoch+Rand	95.9	4.60	0.11	11.23	81.4	13.17	-12.96	84.18	92.8	27.32	-17.10	61.88
	Stochastic	96.4	4.87	0.36	11.77	82.8	14.30	-11.52	88.19	93.2	28.95	-16.44	63.35
	Random	100.0	6.61	1.86	13.48	83.2	14.90	-11.86	86.65	92.9	28.54	-15.98	63.57
	Without	100.0	11.52	6.63	18.59	97.3	29.14	-0.02	108.39	96.8	48.41	-5.00	93.22

For the *fastest* schedules, 24 machines of each type (A, B, and C) are used. For the *minimal+10%* schedules, between 1 and 5 machines of type A plus 19 to 24 machines of type C have been used (the exact configurations depending on the actual bag instances). Clearly, hardly any room for improvement is given by *minimal+10%* schedules, while *fastest* schedules provide ample opportunity for task replication.

C. Results and discussion

Positive percentages for *avg*, *min*, and *max* indicate that the estimated makespan has been exceeded by the respective amount of time (and vice versa). Like with the percentage of runs exceeding makespans (*exc.*), "low is good" in Table I.

A surprising result is that all replication strategies perform within 5% of the perfect replication, which raises the question of how much information about the task runtime distribution is actually needed for a successful replication strategy.

Random replication performs surprisingly well, because it gains in quantity what it lacks in quality: it *never releases* a worker until the workload has finished execution, unless all remaining tasks are executed by faster machines. Stochastic replication and perfect replication *always release* workers when no candidate tasks can be identified (correctly, or else due to prediction inaccuracy).

Table I shows that execution without replication performs already well on the normal distribution bags for the *minimal+10%* budget type. As there are only a few machines of a different (slower) type and thus only a few replication candidates, the replication strategies bring only a marginal gain. However, the *fastest* budget type allows for more machine types, and all replication strategies outperform execution without replication by $\approx 7\%$. Here, stochastic replication gains ground compared to the random replication. The stochastic+random replication comes second after perfect replication, outperforming the other strategies by $\approx 2\%$.

The Levy-truncated distribution bags present the same insensitivity to the replication strategy under the *minimal+10%*. Again, for the *fastest* budget, we notice that all replication strategies improve the default execution significantly (on average by 14.24% to 19.25%). The second-best strategy is the stochastic+random replication. Due to the distribution mode typically situated at smaller runtimes compared to the maximum runtimes, the stochastic replication has a clearer signpost early on during the replication stage. However, if the outlying tasks have not been observed yet, or only a few of them have, the candidate selection criterion may fail to recognize that a task would indeed benefit from replication. This is why adding random selection of tasks as replication candidates outperforms stochastic replication alone.

The multi-modal distribution bags under *minimal+10%* budget are the least sensitive to replication. With the *fastest* budget, all replication strategies improve the default execution on average by at least 19.46% (stochastic replication) and at most 24.27% (perfect replication), with the stochastic+random replication again being the second-best strategy, albeit less clearly than for the normal and Levy-truncated bags. The two distribution modes add an extra degree of difficulty when estimating task runtimes. The truncated mean may be in between the two distribution modes, which would lead to selecting the "wrong" (i.e. the shorter) task for replication. Also, the "gap" in the distribution situated at the 20–30 minutes interval contributes to lesser accuracy of the truncated mean for the stochastic prediction of a task's runtime.

V. RELATED WORK

Systems for executing bags of tasks are legion. [6], [7] have the most in common with BaTS by considering execution costs, but neither approach deals with tail-phase issues. BaTS' tail-phase optimizations employ task replication and task runtime prediction. We discuss these two topics in turn.

A successful replication strategy depends on several factors: the metric to improve (e.g., load-balance, makespan, cost), the available resources, and the type of information available about the tasks' runtimes. Thus, from desktop grids [8], [9] to supercomputers [10], to scientific grids and clouds [11], the replication strategies adopt different heuristics.

The work in [10] uses task replication within supercomputer allocations to improve machine utilization. Similarly, BaTS employs task replication during the tail phase of the execution, in order to reduce the overall makespan, at no extra costs. However, BaTS follows the perspective of a cloud user: focus on efficient task execution rather than machine utilization.

Replication is also used to deal with "straggler" tasks, in order to improve the overall makespan, while slightly increasing the number of extra resources [12]. BaTS, however, utilizes resources otherwise idle and already paid-for. Also, [12] does not clearly indicate when the replication should start.

Our work has more in common with the ExPERT framework [13], which constructs the Pareto-frontier of scheduling strategies from which it selects the one that best fulfills a user-provided utility function. However, we only consider reliable resources, for which the user has already paid and we do not set deadlines at task level; we care about the overall makespan.

SpeQuloS [14] is a framework for hybrid bag-of-tasks execution: the bulk of the execution is done on desktop machines, while the tail-phase is executed on cloud resources. We share the idea of predicting the makespan without per-task runtime knowledge. However, SpeQulos uses historical data obtained from previous executions, while BaTS only uses (more precise) information derived from the bag at hand.

Runtime prediction has been an active area of research for several decades [15], [16]. The larger part of such research investigates the task runtime as a function of its input. Similarly, the ExPERT framework [13] takes as input the time *when* the task has been submitted, aided by a general task-level set deadline. However, as BaTS assumes no a-priori information about individual task runtimes, we cannot employ such research results. The only input we consider is the time elapsed *since* a task has started its execution, along with the overall runtime distribution as BaTS observes it at runtime.

VI. Conclusions

In previous work [1], [5] we showed the efficacy of BaTS, enabling a user to execute a bag of tasks on multiple cloud offerings, favoring either cheaper execution or faster makespans, depending on the mix of machines from different cloud offerings. BaTS works well in most cases, despite the absence of any a-priori knowledge about task runtimes. However, the uncertain nature of the available information, and the latent risk of outlier tasks being scheduled too late to be compensated by other, short-running tasks prohibit any hard promises.

In this paper, we presented several stochastic optimizations for BaTS that utilize the otherwise idling machines during the tail phase of an execution. As these machines have already been paid for by the user up until the end of the next interval, they can be exploited to reduce the makespan at no extra costs.

We presented an in-depth performance evaluation based on our BaTS simulator that allows execution of bag instances with the same statistical task runtime distributions. We investigated bags with normal distributions, with (heavy-tailed) Levy-truncated distributions, and with multi-modal distributions.

Our main result is that replicating tasks in the tail phase onto already rented, but otherwise idle machines reduces the overall makespan of a bag, and thus reduces the risk of the execution exceeding the predicted makespan. The strength of this effect strongly depends on both the runtime distribution of the bag and on the set of available machines. Higher execution budgets typically allow more room for improvement, due to the larger number of fast machines in the pool. This is a positive result: a user willing to spend more money has a stronger emphasis on short makespans than a user preferring lower budgets.

We evaluated the sensitivity of the replication decisions to the quality of the runtime estimation by comparing our stochastic runtime estimations both to (theoretical) perfectly known task runtimes, and to selecting tasks for replication at random. Surprisingly, our results show that the differences between the three are small, whereas the stochastic runtime estimations perform more consistently than random selections. We can thus conclude that our task replication approach is fairly robust to the quality of the estimations. This makes the histogram constructed during the high-throughput phase of the computation a sufficient tool to predict task runtimes, rendering more elaborate and computationally expensive techniques for identifying the exact runtime distribution unnecessary.

To summarize, our tail-phase optimizations improve the execution of bags of tasks by our BaTS scheduler, by exploiting otherwise idling machines. As the user paid for these machines anyway, these improvements come without additional costs.

Acknowledgements: The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement FP7-ICT-257438.

References

[1] A.-M. Oprescu, T. Kielmann, and H. Leahu, "Budget estimation and control for bag-of-tasks scheduling in clouds," *Parallel Processing Letters*, vol. 21, no. 2, pp. 219–243, 2011.

[2] A. Iosup, O. Sonmez, S. Anoep, and D. Epema, in *HPDC '08*.

[3] T. N. Minh, L. Wolters, and D. Epema, "A realistic integrated model of parallel system workloads," ser. CCGRID'10.

[4] "The First International Data Analysis Challenge for Finding Supernovae," http://www.cluster2008.org/challenge/, held in conjunction with IEEE Cluster/Grid 2008.

[5] A.-M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *CloudCom 2010*, pp. 351–359.

[6] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *Grid 2010*.

[7] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, and Y. Yang, "A compromised-time-cost scheduling algorithm in swindew-c for instance-intensive cost-constrained workflows on a cloud computing platform," *Int. J. High Perform. C.*, vol. 24, no. 4, pp. 445–456, 2010.

[8] D. Kondo, A. A. Chien, and H. Casanova, "Scheduling task parallel applications for rapid turnaround on enterprise desktop grids," *J. Grid Comput.*, vol. 5, no. 4, pp. 379–405, 2007.

[9] J. Wingstrom and H. Casanova, "Probabilistic allocation of tasks on desktop grids," in *IPDPS'08*.

[10] T. G. Armstrong, Z. Zhang, D. S. Katz, M. Wilde, and I. T. Foster, "Scheduling many-task workloads on supercomputers: Dealing with trailing tasks," in *MTAGS 2010*.

[11] W. Lu, J. Jackson, J. Ekanayake, R. S. Barga, and N. Araujo, "Performing large science experiments on azure: Pitfalls and solutions," in *CloudCom 2010*.

[12] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI'04*.

[13] O. A. Ben-Yehuda, A. Schuster, A. Sharov, M. Silberstein, and A. Iosup, "Expert: Pareto-efficient task replication on grids and a cloud," in *IPDPS'12*.

[14] S. Delamare, G. Fedak, D. Kondo, and O. Lodygensky, "Spequlos: A qos service for bot applications using best effort distributed computing infrastructures," in *HPDC'12*.

[15] M. A. Iverson, F. Özgüner, and L. C. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *IEEE Trans. Computers*, vol. 48, no. 12, pp. 1374–1379, 1999.

[16] S. Verboven, P. Hellinckx, F. Arickx, and J. Broeckhove, "Runtime prediction based grid scheduling of parameter sweep jobs," in *APSCC'08*.