Bag-of-Tasks Scheduling under Budget Constraints

Ana-Maria Oprescu, Thilo Kielmann
Department of Computer Science, Vrije Universiteit
Amsterdam, The Netherlands
{amo,kielmann}@cs.vu.nl

Abstract

Commercial cloud offerings, such as Amazon's EC2, let users allocate compute resources on demand, charging based on reserved time intervals. While this gives great flexibility to elastic applications, users lack guidance for choosing between multiple offerings, in order to complete their computations within given budget constraints. In this work, we present *BaTS*, our budget-constrained scheduler. BaTS can schedule large bags of tasks onto multiple clouds with different CPU performance and cost, minimizing completion time while respecting an upper bound for the budget to be spent. BaTS requires no a-priori information about task completion times, and learns to estimate them at run time. We evaluate BaTS by emulating different cloud environments on the DAS-3 multi-cluster system. Our results show that BaTS is able to schedule within a user-defined budget (if such a schedule is possible at all.) At the expense of extra compute time, signifcant cost savings can be achieved when comparing to a cost-oblivious round-robin scheduler.

1 Introduction

In *computational science*, *parameter sweep* or *bag of tasks* applications are as dominant as computationally demanding. The classic Condor [16] system is in widespread use to deploy as many application tasks as possible on otherwise under utilized computers, coining the term of *High Throughput Computing*, utilizing networks of idle workstations, cluster computers, and computational grids.

Common to such computing platforms is the model of sharing on a best-effort basis, without any performance guarantees, and commonly also free of charge. This cost-free, best-effort model has had a strong influence on the way bag-of-tasks applications have been deployed. Scientists simply grab as many machines as possible, trying to improve their computational throughput, while neglecting how quickly certain machines can perform the given tasks.

When Amazon announced EC2, its *Elastic Computing Cloud* [1], the era of cloud computing started to offer a different computing paradigm. EC2 and other commercial cloud offerings provide compute resources with defined quality of service (CPU type and clock speed, size of main memory, etc.) These computers can be allocated, and are charged, for given time intervals, typically per hour.

The various commercial offerings differ not only in price, but also in the types of machines that can be allocated. With EC2 alone, the several types of machines. While all machine offerings are described in terms of CPU clock frequency and memory size, it is not clear at all which machine type would execute a given user application faster than others, let alone predicting which machine type would provide 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.

In this work, we present *BaTS*, our budget-constrained scheduler. BaTS can schedule large bags of tasks onto multiple clouds with different CPU performance and cost. BaTS schedules such that a bag of tasks will be executed within a given budget (if possible), while minimizing the completion time. BaTS requires no a-priori information about task completion times, instead BaTS learns application throughput at runtime, using an initial sampling phase and a moving average throughout the computation.

We have emulated different types of clouds on the DAS-3 multi-cluster system. Here, our evaluation shows that BaTS is able to find schedules that fit into given, user-defined budget limits. Comparing to a budget-oblivious round-robin scheduler (RR), BaTS can schedule at much lower, and more importantly, limited cost. When allowed the budget consumed by RR, BaTS finds slightly slower schedules due to the initial, conservative sampling phase.

This paper is structured as follows. In Section 2, we present the BaTS scheduling algorithm. In Section 3, we evaluate its performance and limitations. Section 4 discusses related approaches before we draw conclusions and outline directions of ongoing work in Section 5.

2 Scheduling under Budget Constraints

BaTS is scheduling large bags of tasks onto multiple cloud platforms. The core functionality is to allocate a number of machines from different clouds, and to adapt the allocation regularly by acquiring or releasing machines in order to minimize the overall makespan while respecting the given budget limitation. The individual tasks are scheduled in a round-robin manner onto the allocated machines.

We assume that the tasks of a bag are independent of each other, so they are ready to be scheduled immediately. We also assume that the tasks can be preempted and rescheduled later, if needed by a reconfiguration of the cloud environment. Our task model incurs no prior knowledge about the task execution times. We assume that there is some completion time distribution among the tasks of a bag, but, a-priori, it is unknown to both the user and to the BaTS scheduling algorithm. The only information we require is the size of the bag (the total number of tasks that need to be executed).

About the machines, we assume that they belong to certain categories (like EC2's "Standard Large" or "High-Memory Double Extra Large") and that all machines within a category are homogeneous. The only information BaTS uses about the machines is their price, like "$0.1 per hour". Also, BaTS uses a list of machine categories (cloud offerings) and the maximum number of machines in each category, to which the user has access to. We use the term *cluster* for the machines of a category. (We do not, however, assume any kind of hardware clustering or co-location.)

Our cost model assumes that machines can be allocated (reserved and charged for) for given machine reservation cycles, called *accountable time unit* (ATU), expressed in minutes. For simplicity, we currently assume that the ATU is the same for all clusters, e.g., sixty minutes. Each cluster, however, has its own cost per ATU per machine, expressed in some currency.

Figure 1 sketches the BaTS system architecture. BaTS itself runs on a *master* machine, likely outside a cloud environment. Here, the bag of tasks is available. BaTS allocates machines from various clusters and lets the scheduler dispatch the tasks to the cluster machines. Feedback, both about task completion times and cluster utilization is used to reconfigure the clusters periodically.

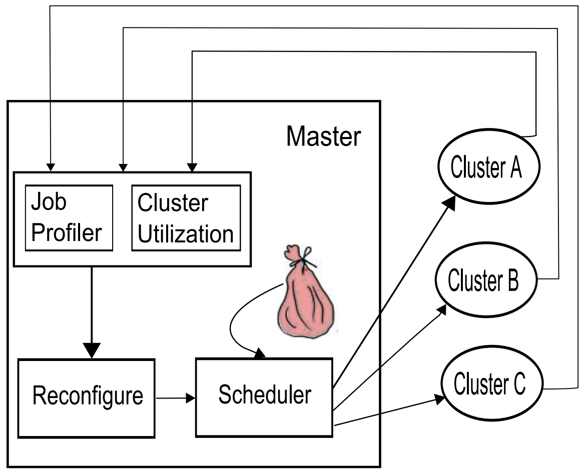

Figure 1. BaTS system architecture.

2.1 Profiling task execution time

BaTS learns execution time-related information by constantly observing the runtimes of submitted tasks. The basic idea is to estimate an average of the task execution time for each cluster. For this purpose, BaTS uses a cumulative moving average mechanism. Based on these estimates, BaTS decides which combination of machines would satisfy the budget constraint and optimize the makespan. As described in Section 2.2, we also provide the decision loop with feedback from monitoring the actual progress made in the bag-of-tasks execution.

We profile the task execution time on a per-cluster level. In an initial sampling phase, we use a small sample set as an initial subset of data points; one set per cluster. The size n of the sample set can be computed with respect to a certain confidence level, based on the canonical statistical formula for sampling with replacement [7]:

$$n = \left\lceil \frac{N * z_\alpha^2}{z_\alpha^2 + 2 * (N-1)\Delta^2} \right\rceil,$$

where n is the sample size, N is the size of the bag, $\Delta \in \{0.10, 0.15, 0.20, 0.25\}$ are typical values for the error level, $\alpha \in (0,1)$ is the confidence level and z_α is related to the Gaussian cumulative distribution function, usual values being: $z_{0.90} = 1.65$, $z_{0.95} = 1.96$ and $z_{0.99} = 2.58$. To correctly approximate the sampling with replacement model, N must be much larger than n; in practice, $n \leq 0.05 * N$ provides the desired property. However, n has an upper bound given by $\left\lceil \frac{z_\alpha^2}{2*\Delta^2} \right\rceil$, as shown in Figure 2.

From the moment at which all sample tasks of a cluster are finished, we derive an average task execution time per cluster (T_i, expressed in minutes), computed as a modified cumulative moving average [8] of task execution times seen so far:

$$T_i = \frac{\sum_{k=1}^{jobs_{running}} \tau_k + rt_{done}}{jobs_{running} + jobs_{done}},$$

We use the execution times of the sample tasks as indicators for all tasks from the bag. For this purpose, we maintain an ordered list of the execution times from the sample.

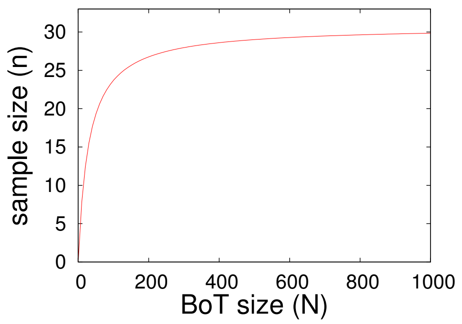

Figure 2. Sample size variation w.r.t. bag size; $\alpha = 0.95$, $z_{0.95} = 1.96$, $\Delta = 0.25$

Whenever T_i is computed later during the run, a task j submitted on cluster i which has not finished execution at this time is estimated (τ_{j_e}) as the average of sample set tasks runtimes higher than τ_j, the time elapsed since its submission:

$$\tau_{j_e} = \frac{\sum_k^n \tau_k}{n-k}$$

This estimate is used when calculating the new T_i, representing task j as one of those from the tail of the sample's distribution.

The average estimated task execution time for cluster i represents the mapping between the bag-of-tasks and a machine of type i. Therefore any such machine can execute $\frac{1}{T_i}$ tasks from the bag per minute and this quantity is the theoretical average estimated speed of a machine of type i. The T_i values provide information about the quality of the machines in cluster i with respect to the bag-of-tasks currently under execution.

T_i is initialized when the sample set tasks sent to a cluster i finish. After the initialization step we update T_i at given monitoring intervals. We chose the monitoring interval small enough to enable timely detections of possible constraint violations.

We use the average estimated speeds of the machine types $C_i, i \in \{1, ..., C_{nc}\}$ participating in the bag-of-tasks execution to compute estimates of the makespan (T_e) and budget(B_e) needed for the bag-of-tasks execution:

$$T_e = \frac{N}{\sum_{i=1}^{C_{nc}} \frac{a_i}{T_i}} \quad ; \quad B_e = \left\lceil \frac{T_e}{ATU} \right\rceil * \sum_{i=1}^{C_{nc}} a_i * c_i,$$

where N is the number of tasks in the bag, $c_i, i \in \{1, ..., C_{nc}\}$ is the cost per ATU for a machine of type C_i and $a_i, i \in \{1, ..., C_{nc}\}$ are the numbers of machines allocated from each cluster. Our current approach finds the best makespan affordable given the user-specified budget (B): We minimize T_e by maximizing the number of executed tasks per minute, and therefore per ATU, while the cost of executing all N tasks at this speed stays within the user specified budget:

$$\begin{aligned} &\mathit{maximize} && \sum_{i=1}^{C_{nc}} a_i * \frac{1}{T_i} \\ &\mathit{subject\ to} && \left\lceil \frac{N}{ATU * \sum_{i=1}^{C_{nc}} \frac{a_i}{T_i}} \right\rceil * \sum_{i=1}^{C_{nc}} a_i * c_i \le B \end{aligned}$$

where $A_i, i \in \{1, ..., C_{nc}\}$ is the maximum number of machines of type i.

We solve the above non-linear integer programming problem by modifying the Bounded Knapsack Problem (BKP). In general, the BKP takes a set of item types, where each type j is characterized by a profit p_j, a weight w_j and a maximum number of items b_j, and determines the number of items (x_j) from each type to be packed such that it maximizes the total profit while the total weight is less than a certain limit (W).

$$\begin{aligned} &\mathit{maximize} && \sum_{j=1}^{m} p_j * x_j \\ &\mathit{subject\ to} && \sum_{j=1}^{m} w_j * x_j \le W, \quad x_j \in \{0, 1, ..., b_j\} \end{aligned}$$

We reformulate the Bounded Knapsack Problem (BKP) in the following way: maximize the total speed per ATU (profit), while maintaining the total cost within the budget. A cluster becomes an item type with profit $\frac{1}{T_i}$ and cost c_i and the bounds for each item type are given by the maximum number (A_i) of available machines (including the ones already acquired by BaTS) in the respective cluster. The solution to this problem represents a machine configuration, where the number of machines from a cluster i is the number of items of the corresponding type. Intuitively, our modified BKP looks for the fastest combination of machines whose total cost per ATU is within a given limit.

Though BKP is NP-complete, it can be solved in pseudo-polynomial time either by expressing it as a 0-1 Knapsack Problem [11] or by using a specialized algorithm [12]. Since both the number of machine types as well as the number of machines of each type are small (compared to the number of jobs), the input of the reformulated BKP can be considered small in its *length*, not only in its *value*, which greatly reduces the time needed to find an exact solution. We chose to solve our modified BKP as a 0-1 Knapsack Problem, using dynamic programming. We define recursively P(i,w):

$$\begin{aligned}
P(0,w) &= 0\\
P(i,0) &= 0\\
P(i,w) &= max\{P(i-1,w), p_i + P(i-1, w-c_i)\},\\
&\quad if\ c_i \le w\\
P(i,w) &= P(i-1,w),\ otherwise
\end{aligned}$$

where $P(i,w)$ is the profit obtained by using i items with an average cost per ATU of w. We compute $P(\sum_{i=1}^{C_{nc}} A_i, \sum_{i=1}^{C_{nc}} c_i * A_i)$ to find the candidate solutions. We filter the solution set using the constraint

$$price * \left\lceil \frac{N}{ATU * P_{cand}(m, price)} \right\rceil \le B\ ,$$

where N is the number of tasks to be executed and P_{cand} is the candidate solution representing m machines that cost $price$ per ATU. The final solution is processed to obtain the number of machines from each type.

The complexity of our modified BKP is dominated by $(\sum_{i=1}^{C_{nc}} A_i) * (\sum_{i=1}^{C_{nc}} c_i * A_i)$, while an algorithm that searches exhaustively for all possible solutions is dominated by $\prod_{i=1}^{C_{nc}} A_i$.

2.2 Monitoring the plan's execution

As described so far, BaTS estimates task runtimes and the related costs based on the tasks that have been completed during the initial sampling phase, resulting in an initial machine allocation. At regular monitoring intervals, this initial plan is revisited to accomodate the actual progress of the bag of tasks. BaTS uses a monitoring interval equal to a (small) fraction of the ATU, but at least equal to 5 minutes.

There are two reasons why the initial plan needs continuous refinement. First, the average taks completion time gets refined with each completed task. Second, the allocated worker machines are not running in lock-step, such that each machine has its own phase of ATU starting time, and its own ATU utilization given the actual tasks it gets to execute that might leave unused time intervals. If, at a given monitoring interval, the new information indicates a possible budget violation, BaTS has to find another machine allocation, possibly marking certain (expensive) machines for being preempted at the end of their ATU, and/or other (cheaper) machines to be added instead.

For evaluating the current machine allocation, BaTS checks for a possible discrepancy between the remaining budget and the remaining size of the bag. The estimated number of tasks left in the bag, N_e describes the utilization of the paid ATUs for all workers. The remaining budget and its current distribution among the active workers is reflected in the potential number of executed tasks, N_p. The comparison between N_e and N_p provides feedback on whether the current scheduling plan fits the (new) information about the tasks of the bag.

For every cluster $C_i, i \in \{1, ..., C_{nc}\}$ we maintain a list of all machines $m_j, j \in \{1, ..., m_{max_i}\}$ that participated at some point in the computation. For every machine m_j we remember the number of executed tasks (nt_{m_j}), the time spent executing tasks (rt_{m_j}) and the total uptime (up_{m_j}).

According to our economical model, the current ATU of each active machine has been paid for once the machine enters it. However, the machine did not run yet for the whole corresponding time. This means that tasks which will be executed by the end of the machine's current ATU are still in the bag. Follows that the current size of the bag is not an accurate indicator of the necessary amount of money to finish the computation. Therefore, we monitor the actual progress of the bag-of-tasks execution by computing an estimate for the number of tasks, N_e, left in the bag after the time for which we already paid elapses on each machine, at regular intervals. The estimate is based on how many tasks each active machine is likely to execute during the remaining span of their respective current ATU. The remaining span does not include the expected runtime of the task currently executed by the machine (i.e. if the machine is not marked for preemption, the current task could take the machine to the next ATU). If a machine is not marked for preemption we use the estimate of its uptime ($up_{m_{j_e}}$) and its current speed (v_{m_j}) to compute the expected number of tasks executed by it. We compute $up_{m_{j_e}}$ based on the current uptime up_{m_j}, elapsed runtime of task (τ) the task currently executed on m_j and its estimated runtime (τ_e).

$up_{m_{j_e}} = up_{m_j} + (\tau_e - \tau)$

We compute the current speed of this machine using the number of executed tasks (nt_{m_j}), the total runtime of executed tasks (rt_{m_j}) and the estimate of the currently running task:

$v_{m_j} = \frac{nt_{m_j}+1}{rt_{m_j}+\tau_e}$

To compute the expected future number of tasks (ft_{m_j}) executed by m_j during this ATU we learn the remaining span (δ_{m_j}) by using $up_{m_{j_e}}$:

$ft_{m_j} = \lfloor \delta_{m_j} * v_{m_j} \rfloor$,

where $\delta_{m_j} = ATU - up_{m_{j_e}} \bmod ATU$.

We can now express N_e as

$$N_e = \sum_{i}^{C_{nc}} \sum_{j=1}^{m_{max_i}} ft_{m_j}$$

Workers are not synchronized with each other. Therefore, each worker is at a different stage in the current execution plan. At regular intervals we need to check that the remaining time for each worker according to the current execution plan covers the estimated number of tasks left in the bag. For this purpose, we learn how many tasks each worker is likely to execute in their remaining ATUs of the execution plan, nr_{m_j}. The sum of these tasks is the potential number of executed tasks, N_p:

$$N_p = \sum_{i}^{C_{nc}} \sum_{j=1}^{m_{max_i}} \lfloor (nr_{m_j} * ATU + \eta_{m_j}) * v_{m_j} \rfloor$$

where $\eta_{m_j} = \left\lceil \frac{up_{m_{j_e}}}{ATU} \right\rceil - \frac{ft_{m_j}}{v_{m_j}}$ represents the time left of the previous ATU which could not accommodate the execution of a task, but becomes useful since the machine is not preempted.

We also keep track of the money spent so far by accumulating the current cost of each machine used, including machines no longer active. To estimate the cost of a task k still running we use again its τ_{k_e} value. Since the remaining budget must accommodate the execution of N_e we check at each monitoring interval that $N_p >= N_e$ holds to avoid possible budget violations. If not, BaTS invokes BKP with the actual remaining budget and the current estimate of the remaining problem size to find a new machine configuration that satisfies the new budget constraint.

2.3 The BaTS algorithm

Based on the mechanisms developed so far, we can formulate the BaTS scheduling algorithm, as shown in Fig. 3. BaTS takes as input a bag-of-tasks with a known size N, the description (cost and maximum number of machines) of a set of available clusters($(c_i, A_i), i \in \{1, .., C_{nc}\}$) and a user-specified budget (B). Based on N it computes the sample size n (see Section 2.1) and acquires a number iw of machines, the *initial workers* on each participating cluster (currently, $iw = \min\{\frac{1}{10} * N, n\}$). iw ideally equals n, but is limited to 10 % of N in order to keep sufficiently many unprocessed tasks for finding a proper configuration. This set of machines becomes the initial configuration. Currently, bags which are too small to accommodate iw workers for each cluster are not considered.

BaTS acts as a master, while the acquired machines act as workers. As workers join the computation, BaTS dispatches randomly selected tasks from the bag in a first-come first-served manner, thus avoiding any bias from the task order within the bag. When a worker running on a machine M from cluster c_m reports back with a task T's result, BaTS updates the worker-related information (runtime - time spent executing tasks, and the number of tasks executed by M), as well as the total execution time (rt_{done}) for cluster c_m with T's execution time.

There are two types of conditions which trigger the search for a new configuration: there is enough information to derive the first stochastic properties for all clusters and/or

```
BaTS Algorithm
1: compute $n$ = sample size
2: construct initial configuration $C$
3: acquire machines according to $C$
4: while bag has tasks do
5:     wait for any machine $M$ to ask for work
6:     if $M$ returned result of task $T$ then
7:         update statistics for machine $M$
8:         update the $rt_{done}$ for $M$'s type $Mt$
9:     end if
10:    if sample set tasks for $Mt$ finished then
11:        update cluster stats for $Mt$
12:    end if
13:    if (monitoring time) || (first clusters stats) then
14:        compute estimates
15:        if constraint violation then
16:            call BKP to compute a new configuration $C'$
17:            acquire the extra machines required by $C'$
18:            save $C'$ in $C$
19:        end if
20:    end if
21:    if number of machines of $Mt$ satisfies $C$ then
22:        send $M$ a randomly selected task $T'$
23:        remove $T'$ from bag and place it in pending
24:    end if
25:    if number of machines of $Mt$ should decrease then
26:        release $M$
27:    end if
28: end while
```

Figure 3. Summary of BaTS Algorithm

estimates computed at the end of a monitoring interval indicate a budget constraint violation. If this is the case, the new configuration C' replaces C. BaTS decides whether M should continue to be part of the computation or it should be released, based on the current configuration. Note that the BaTS performs no reconfigurations until it can derive stochastic properties for all participating clusters. If there are no tasks finished during a monitoring interval, BaTS updates both the profiling and cluster utilization estimates as described in previous sections. If a budget constraint violation is signaled, BaTS tries to compute a new configuration.

3 Performance Evaluation

We have implemented a Java-based BaTS prototype using our Ibis platform [2]. The prototype consists of two parts: the master, which can run on any desktop-like machine, and the worker which is deployed on cloud machines. The master component implements the core of BaTS, while the worker is a lightweight wrapper for task execution. All communication between the master and the workers uses Ibis's communication layer, IPL.

We have emulated an environment of two (cloud) clusters in the DAS-3 multi cluster system. The physical machines are 2.4 GHz AMD Opteron DP, each has 4 GB of memory and 250 GB of local disk space, running Scientific Linux. Both emulated clusters have 32 machines.

Requests for machines that will run the worker component are sent by the master component to the local cluster scheduler (SGE), thus incurring realistic, significant startup times as in real clouds. However, we do not allow queueing delays due to competing requests.

We evaluated BaTS using a workload of relatively medium size, where the assumptions made by the statistical device behind BaTS hold. Recent work [6] on the properties of bags-of-tasks has shown that in many cases the intra-BoT distribution of execution times follows a normal distribution. Accordingly, we have generated a workload of 1000 tasks whose runtimes (expressed in minutes) are drawn from the normal distribution $N(15, \sigma^2), \sigma = \sqrt{5}$. We enforce these runtimes by executing the *sleep* command accordingly.

We compare BaTS to a simple, budget-oblivious Round Robin (RR) algorithm that schedules randomly selected tasks in a first-come first-served manner. For all runs of BaTS and RR, the random generator was using an identical seed, presenting the tasks to all algorithms in the same order.

One emulated cluster ($\texttt{cluster}_0$) charges \$3 per machine per ATU and executes tasks according to the runtimes drawn from $N(15, \sigma^2), \sigma = \sqrt{5}$. We create 5 different scenarios w.r.t. the price and speed of the other emulated cluster $\texttt{cluster}_1$ compared to $\texttt{cluster}_0$:

S_{1-1}: $\texttt{cluster}_1$ charges the same price for a machine and has the same speed;

S_{1-4}: $\texttt{cluster}_1$ charges the same price for a machine, but is 4 times as fast;

S_{4-1}: $\texttt{cluster}_1$ charges 4 times as much (\$12 for a machine) but has the same speed;

S_{3-4}: $\texttt{cluster}_1$ is 3 times as expensive (charges \$9 for a machine), and is 4 times as fast;

S_{4-3}: $\texttt{cluster}_1$ is 4 times as expensive (charges \$12 for a machine), and is 3 times as fast.

For emulating variable speeds of $\texttt{cluster}_1$, we modify the parameter to *sleep* accordingly. All prices are per accountable time unit, which we set to 60 minutes (without loss of generality).

We ran RR, BaTS with a relatively small budget and BaTS having as budget the cost incurred by the respective RR on each of the scenarios above.

RR experiments use all machines (32+32) from the beginning of the computation. We refer them as RR_S, S the respective scenario name.

BaTS experiments were conducted using a monitoring interval of 5 min and the following values for the parameters of the sample size n formula: $z_{\alpha=0.95} = 1.96$, $\Delta = 0.25$, which lead to n_{1000}=30. The sample size preserves the constraints implied by the statistical assumptions, as $n \leq 0.05 * N$ and $n \geq 30$ [7]. According to the formula presented in Section 2.3 the initial number of machines acquired on each cluster is 30.

As a first set of experiments, we ran BaTS for each scenario with a budget constraint equal to the cost incurred by RR for the respective scenario S, resulting in 5 experiments: $BaTS_{\mathrm{RR}_S}$.

As a second set of experiments, we again ran BaTS for each scenario with a budget constraint equal to $1.1 * B_{min_S}$ (10% extra), where B_{min_S} is the minimum budget needed to execute the entire bag on scenario S with no regard for makespan minimization. The resulting 5 experiments are referred as $BaTS_{B_{min_S}}$, S the respective scenario name. B_{min_S} is computed as the cost of the sampling phase plus the cost of running the rest of the bag on one machine of the most profitable type, i.e. the machine type offering the best value for money. We define the *profitability* of a machine type i with respect to the cheapest machine type available m as the speed increase compared to the cost increase:

$$\textit{profitability} = \frac{T_m}{T_i} * \frac{c_m}{c_i}$$

where T_m, T_i represent the average theoretical task execution time for clusters m and i, respectively; c_m, c_i represent the cost of using a machine from the respective cluster for one ATU.

We summarize the results of all the experiments in Figure 4. Results are represented by pairs of bars, indicating the makespan (in minutes) and the cost (in dollars) of each run. Results are grouped by their respective scenarios and the scenarios are ordered by the profitability of the faster machine type. We analyze each BaTS experiment in the order they appear in Figure 4.

Scenario S_{4-1} has a sampling cost of \$450. We first look at $BaTS_{\mathrm{RR}_{S_{4-1}}}$ where the budget is equal to the cost $RR_{S_{4-1}}$, \$2034. Compared to $RR_{S_{4-1}}$ makespan of 246m16s, BaTS takes 279m48s. The 33 minutes delay is due to the new configuration BaTS finds affordable after it collected the necessary statistical information about all participating clusters. BaTS starts with a configuration of 30 initial machines on each cluster, whereas RR starts with 32 workers on each cluster. When the statistical properties of both clusters are inferred, the remaining budget (\$1584) and the estimated number of tasks left in the bag (784) lead BaTS to reconfigure to 25 machines from $\texttt{cluster}_1$ and

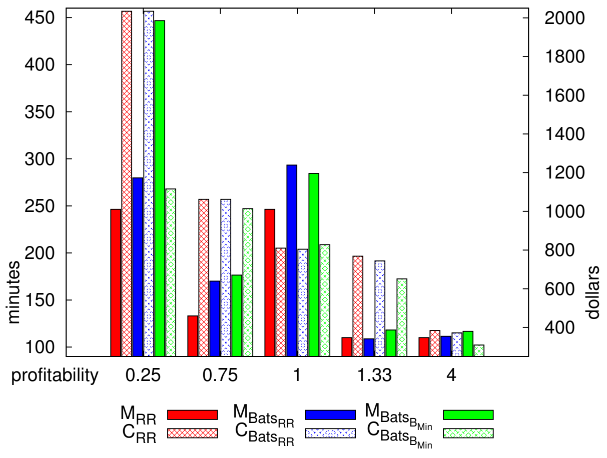

Figure 4. Makespan and cost for $N = 1000$, comparing round-robin (RR) to BaTS with two different budgets.

32 machines from $\texttt{cluster}_0$; this configuration is used for the rest of the computation. A closer look at $RR_{S_{4-1}}$ shows that only 8 machines from $\texttt{cluster}_1$ and 6 machines from $\texttt{cluster}_0$ need a few extra minutes of the 5th ATU to finish the tasks currently executing, i.e. the tail phase. As BaTS does not distinguish between high-throughput and tail phases, it estimates that the time needed to execute the rest of the bag is 4 ATU (after the current one), for which the remaining budget cannot accommodate 32 machines from each cluster. BaTS acquires all most profitable machines and as many as possible from the less profitable type.

In the same S_{4-1} scenario we test BaTS with a budget of \$1128, computed as 10% extra of $B_{min_{S_{4-1}}}$, \$1026. The actual cost of the run is \$1116, but the makespan increases to 446m46s compared to 279m48s from the previous experiment. After sampling, BaTS reconfigures to 1 machine from $\texttt{cluster}_1$ and 32 machines from $\texttt{cluster}_0$, again prefering the most profitable type. Compared to the makespan affordable with $B_{min_{S_{4-1}}}$, 11551m36s, BaTS finds a much smaller makespan for only 10% more money.

Scenario S_{4-3} has a sampling cost of \$450. The budget for $BaTS_{RR_{S_{4-3}}}$ equals the cost of $RR_{S_{4-3}}$: \$1062. BaTS takes 170m5s, whereas RR took 133m10s. From the initial configuration of 30 workers on each cluster, BaTS reconfigures to 18 machines in $\texttt{cluster}_1$ and 30 machines in $\texttt{cluster}_0$. This BaTS run shares the problem found with $BaTS_{RR_{S_{4-1}}}$. It also indicates another problem raised by the lack of special treatment for the tail phase: since the bag is empty, fast machines are released though the last few tasks have just started execution on slow machines. Given the completion time distributions and the current scenario, this leads to an increase in makespan of about 10 minutes.

The budget given to $BaTS_{B_{min_{S_{4-3}}}}$ is \$1015, computed as 20% extra of $B_{min_{S_{4-3}}}$, \$846. Here, 10% extra proved too little to cover for the tasks left after the sampling phase, leading to BaTS running out of money and quitting the computation before the bag was empty. When on a tighter budget, the problem identified with $BaTS_{RR_{S_{4-3}}}$ becomes the failure cause. However, for the \$1015 budget, BaTS finds a new configuration of 16 machines from $\texttt{cluster}_1$ and 30 machines from $\texttt{cluster}_0$ and executes the entire bag in 176m26s at the cost of \$1014, compared to 7947m24s, the makespan corresponding to the minimum budget.

Scenario S_{1-1} involves a sampling cost of \$180. $BaTS_{RR_{S_{1-1}}}$ is given a budget of \$810, which is the cost $RR_{S_{1-1}}$. The actual cost of BaTS is \$804 for which it delivers a makespan of 293m17s, compared to 246m15s, the $RR_{S_{1-1}}$ makespan. The reconfiguration comprises 32 machines from one cluster and 20 machines from the other. We give $BaTS_{B_{min_{S_{1-1}}}}$ a budget of \$831, which is 10% extra of $B_{min_{S_{1-1}}}$, \$756. The resulting makespan is 284m23s, for a cost of \$828, compared to 11551m36s. The configuration used after sampling consists of 32 machines from one cluster and 22 machines from the other. Both experiments confirm that when all machine types are identical and no a-priori task execution time information is available, RR is the recommendable approach to the bag-of-tasks execution. For instance, the cost of $RR_{S_{1-1}}$ is 7.14% extra of the minimum budget \$756.

Scenario S_{3-4} incurrs a sampling cost of \$360. We give $BaTS_{RR_{S_{3-4}}}$ a budget equal to the cost of $RR_{S_{3-4}}$, \$768. The makespan for this run is 108m47s, compared to the $RR_{S_{3-4}}$ makespan of 110m6s. The actual cost of the BaTS run is \$744. BaTS reconfigures to 32 machines from each cluster. Here, we do not encounter the problem identified with $BaTS_{RR_{S_{4-1}}}$, therefore BaTS takes a comparable makespan as RR. The difference in cost comes from the smaller number of machines used by BaTS in the first ATU, i.e. before reconfiguration. Next, we ran $BaTS_{B_{min_{S_{3-4}}}}$ with a budget of \$653, which is 10% extra of $B_{min_{S_{3-4}}}$, \$594. The resulting makespan is 118m9s at an actual cost of \$651, compared to 6152m32s, the makespan obtained for the minimum budget. BaTS reconfigures to 32 machines from $\texttt{cluster}_1$ and 1 machine from $\texttt{cluster}_0$.

Scenario S_{1-4} has a sampling cost of \$180. First, we run $BaTS_{RR_{S_{1-4}}}$ with a budget equal to the cost of $RR_{S_{1-4}}$, \$384. The resulting makespan is 111m28s, whereas $RR_{S_{1-4}}$ makespan is 110m5s. The actual cost of BaTS is \$372. After sampling, BaTS finds a new configuration consisting of 32 machines in each cluster. Next, we ran $BaTS_{B_{min_{S_{1-4}}}}$ with a budget of \$309, which is 20% extra of $B_{min_{S_{1-4}}}$, \$258. Here, 10% extra proved again too little due to the problem encountered with $BaTS_{B_{min_{S_{4-3}}}}$. For a budget of \$309, BaTS finds a schedule that takes 116m42s to execute the bag, compared to 6152m32s, the makespan obtained for $B_{mins_{1-4}}$. The schedule uses a new configuration comprising 32 machines from $\texttt{cluster}_1$ and 11 machines from $\texttt{cluster}_0$.

From these results, we conclude that BaTS successfully schedules bags of tasks within given, user-defined budget contraints. We identify three distinct ways to improve BaTS' performance: (a) reduce the overhead incurred by sampling on each cluster, (b) a different treatment of the final phase of the computation, both in the reconfiguration and the scheduler modules; and (c) an additional condition to trigger the search for new configurations further minimizing the makespan and/or saving money.

4 Related Work

Recent research efforts have addressed different aspects of bag-of-tasks applications. We compare our present proposal to existing research with respect to the assumptions made on task and resource characteristics, the proposed goals, the scheduling plan characteristics, the performance metrics used to compare against established algorithms and the respective algorithms.

The assumptions on task characteristics involve the existence of prior knowledge on arrival rate, execution time, or deadline for each task in the bag. Work presented in [9, 14, 15, 17] assumes a-priori known task execution times. One relaxation is found in [4] where all tasks are assumed to be in the same complexity class and there is a calibration step to determine execution time estimates per machine type. The assumptions are further relaxed in [13] to relative complexity classes of tasks, though all the classes are supposed to be known in advance. BaTS, in contrast, only assumes that some form of runtime distribution exists, and uses stochastic methods to detect it while executing the bag of tasks.

Work presented in [18] addresses homogenous grid resources and is extended to heterogenous systems in [19]. Completely heterogenous systems are addressed by [3, 13, 15, 17]. Our system is composed of several heterogenous sets of homogenous machines, which fits perfectly a composite of scientific grids and cloud systems.

Makespan minimization is the main focus of research done in [3, 9]. This is accompanied by response time minimization at task level in [17]. These scheduling algorithms are compared against traditional algorithms such as Min-Min, Max-Min [5] and Sufferage [10]. A mixture of robustness optimization, while satisfying makespan and price constraints is presented in [15]. It assumes a fixed, one-time cost per machine type. Robustness optimization is the main focus of research conducted in [13]. In contrast, we use an economic model for resource utilization, that matches the current, elastic cloud system offerings.

The mapping between tasks and resources can be done either off-line [15, 19] or on-line. The on-line techniques can be further categorized using the mapping event granularity: fine granularity implies the mapping is performed as soon as a task arrived/is ready [3]. Coarse granularity makes mapping decisions on a batch of tasks. Hybrid approaches are employed in [13, 17]. With BaTS, we consider all tasks to be available for execution when the application starts. However, mapping events are triggered by an (adjustable) timeout.

5 Conclusions

Elastic computing, as offered by Amazon and its competitors, has changed the way compute resources can be accessed. The elasticity of clouds allows users to allocate computers on the fly, according to the application's needs. While each commercial offering has a defined quality of service, users still lack guidance for deciding how many machines of which type and for how long would be necessary for their application to complete within a given budget, as quickly as possible.

Bags of tasks are an important class of applications that lend themselves well for execution in elastic environments. In this work, we have introduced BaTS, our budget-constrained scheduler for bag-of-tasks applications. BaTS requires no a-priori information about task execution times. It uses statistical methods to execute samples of tasks on all cloud platforms that are available to a user. BaTS monitors the progress of the tasks and dynamically reconfigures the set of machines, based on the expected budget consumption and completion time.

We have evaluated BaTS by emulating different clouds on the DAS-3 multi-cluster system. For each test, we used two clouds with different profitability (price-performance ratio) and let both a cost-oblivious round robin (RR) and BaTS schedule a bag of 1000 tasks. For all our tests, BaTS managed to schedule within the user-defined budget (and stopped early when it noticed that it would exceed the limit). Given the actual cost of RR as its budget, BaTS produced somewhat longer schedules, due to the initial, conservative sampling phase. With smaller budgets, BaTS produced schedules for a guaranteed budget, albeit slower than RR.

Our results are very encouraging, but they also open up new questions. Helping the user estimate a suitable budget is an obvious one. We are currently working on decoupling the initial sampling phase from the main execution in order to produce estimates for several cost-makespan combinations. Improving the tail phase of the schedule seems a promising approach for further minimizing BaTS' makespans, without raising the costs incurred.

References

[1] Amazon Web Services. http://aws.amazon.com.

[2] H. E. Bal, J. Maassen, R. V. van Nieuwpoort, N. Drost, R. Kemp, N. Palmer, T. Kielmann, F. Seinstra, and C. Jacobs. Real-world distributed computing with Ibis. *Computer*, 43(8):54–62, 2010.

[3] W. Cirne, D. P. da Silva, L. Costa, E. Santos-Neto, F. V. Brasileiro, J. P. Sauvé, F. A. B. Silva, C. O. Barros, and C. Silveira. Running bag-of-tasks applications on computational grids: The mygrid approach. In *ICPP*, 2003.

[4] H. González-Vélez. Self-adaptive skeletal task farm for computational grids. *Parallel Comput.*, 32(7):479–490, 2006.

[5] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *J. ACM*, 24(2):280–289, 1977.

[6] A. Iosup, O. Sonmez, S. Anoep, and D. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 97–108, New York, NY, USA, 2008. ACM.

[7] E. S. Keeping. *Introduction to Statistical Inference*. D. Van Nostrand, Princeton, New Jersey, 1962.

[8] J. F. Kenney and E. S. Keeping. *Mathematics of Statistics*. D. Van Nostrand, Princeton, New Jersey, 1962.

[9] Y. C. Lee and A. Y. Zomaya. Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. *IEEE Transactions on Computers*, 56(6):815–825, 2007.

[10] 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 *In Eight Heterogeneous Computing Workshop*, pages 30–44. IEEE Computer Society Press, 1999.

[11] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[12] D. Pisinger. A minimal algorithm for the bounded knapsack problem. Technical report, University of Copenhagen, 1994.

[13] J. Smith, L. D. Briceno, A. A. Maciejewski, H. J. Siegel, T. Renner, V. Shestak, J. Ladd, A. M. Sutton, D. L. Janovy, S. Govindasamy, A. Alqudah, R. Dewri, and P. Prakash. Measuring the robustness of resource allocations in a stochastic dynamic environment. In *IPDPS*, 2007.

[14] P. Sugavanam, H. Siegel, A. Maciejewski, J. Zhang, M. Shestak, M. Raskey, A. Pippin, R. Pichel, M. Oltikar, A. Mehta, P. Lee, Y. Krishnamurthy, A. Horiuchi, K. Guru, M. Aydin, M. Al-Otaibi, and S. Ali. Robust processor allocation for independent tasks when dollar cost for processors is a constraint. In *Cluster Computing, IEEE International Conference on*, 2005.

[15] P. Sugavanam, H. J. Siegel, A. A. Maciejewski, M. Oltikar, A. M. Mehta, R. Pichel, A. Horiuchi, V. Shestak, M. Al-Otaibi, Y. G. Krishnamurthy, S. A. Ali, J. Zhang, M. Aydin, P. Lee, K. Guru, M. Raskey, and A. J. Pippin. Robust static allocation of resources for independent tasks under makespan and dollar cost constraints. *J. Parallel Distrib. Comput.*, 67(4):400–416, 2007.

[16] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[17] C. Weng and X. Lu. Heuristic scheduling for bag-of-tasks applications in combination with qos in the computational grid. *Future Generation Comp. Syst.*, 21(2):271–280, 2005.

[18] Q. Zhu and G. Agrawal. An adaptive middleware for supporting time-critical event response. *Autonomic Computing, International Conference on*, pages 99–108, 2008.

[19] Q. Zhu and G. Agrawal. A resource allocation approach for supporting time-critical applications in grid environments. *Parallel and Distributed Processing Symposium, International*, pages 1–12, 2009.