A self-optimized job scheduler for heterogeneous server clusters

Elad Yom-Tov
IBM Haifa Research Lab
Haifa 31905, Israel
yomtov@il.ibm.com

Yariv Aridor
IBM Haifa Research Lab
Haifa 31905, Israel
yariv.aridor@gmail.com

Abstract

Heterogeneous clusters and grid infrastructures are becoming increasingly popular. In these computing infrastructures, machines have different resources, including memory sizes, disk space, and installed software packages. These differences give rise to a problem of over-provisioning, that is, sub-optimal utilization of a cluster due to users requesting resource capacities greater than what their jobs actually need. Our analysis of a real workload file (LANL CM5) revealed differences of up to two orders of magnitude between requested memory capacity and actual memory usage. This paper presents an algorithm to estimate actual resource capacities used by batch jobs. Such an algorithm reduces the need for users to correctly predict the resources required by their jobs, while at the same time managing the scheduling system to obtain superior utilization of available hardware. The algorithm is based on the Reinforcement Learning paradigm; it learns its estimation policy on-line and dynamically modifies it according to the overall cluster load. The paper includes simulation results which indicate that our algorithm can yield an improvement of over 30% in utilization (overall throughput) of heterogeneous clusters.

1. Introduction

1.1. Background

Heterogeneous clusters and grid infrastructures are becoming increasingly popular. In these computing infrastructures, the machines have different computing power and resources (memory, networking, etc.). Additionally, machines can dynamically join and leave the systems at any time. Job schedulers provide a means of sending jobs for execution on these computing clusters. A job is defined as a set of processes that run, in parallel, on a single computer or on multiple computers. Dynamic approaches to resource management play a significant role in the management and utilization of these infrastructures. With these approaches, the job is submitted together with a specification of the type and capacity of resources required for successful execution e.g., amount of memory and disk space, and prerequisite software packages. When a job is scheduled, its job request is matched with the available resources. If all the required resources are found, they are allocated and the job is launched for execution.

Dynamic resource matching between jobs and resources has been extensively researched over the years, initially for homogeneous clusters and more recently for heterogeneous and grid computing environments [4]. However, one problem that has rarely been examined is over-provisioning. That is, jobs are allocated more resources than what they actually need due to users overestimating the job requirements. With over-provisioning, we specifically refer to resources in a given computing machine that can affect the completion of the job execution. That is, if the capacity of these resources falls below a certain level, the job cannot complete successfully. Examples of such resources are memory size, disk space, and even prerequisite software packages. This paper focuses on the over-provisioning problem. We do not deal with the problem of over-provisioning the number of machines requested for parallel jobs. This is a complicated problem, which is heavily dependent on the programming model used (i.e., whether the number of machines is hard-coded in the job source program).

Over-provisioning affects machine utilization as best explained by the following scenario. Assume two machines, M1 and M2, and two jobs, J1 and J2. Assume M1 has a larger memory size than M2. Initially, J1 can run on either M1 or M2. However, the resource allocation matches it with machine M1 because the user requests a memory size larger than that of M2, but possible for M1. Later, J2 arrives. Due to its memory size request, the only machine it can use is M1. Now J2 is blocked until J1 completes or a new node with at least the same memory size as M1 is added to the cluster.

The over-provisioning problem is demonstrated in Fig-

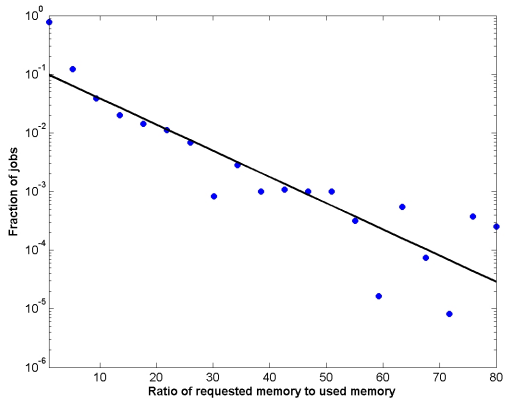

Figure 1. A histogram of the ratio between requested memory size and actual memory used, per job, in the LANL CM5 workload file. The vertical axis is logarithmically scaled

ure 1. This figure shows a histogram of the ratio of requested to used memory in the LANL CM5 log [19]. As this figure demonstrates, less than 70% of jobs correctly estimate their required memory. In over 30% of jobs there is a mismatch by a factor of two or more between requested memory and used memory. The regression line in the figure shows the fit of the over-provisioning ratio to the percentage of jobs. The R^2 coefficient[1] for this regression line is 0.69. This fitting shows that it is possible to estimate, with high accuracy, the fraction of jobs with a given over-provisioning ratio in future log files from similar systems. This is an important design consideration in some learning algorithms.

Unfortunately, it is frequently difficult for most users to correctly specify the needs of their jobs. Therefore, this is a persistent problem which causes waste of computing resources. Ideally, the scheduler in a parallel system should independently overcome this problem by estimating actual user needs. This article attempts to show how such an autonomic scheduler could be used.

Research of the over-provisioning problem is difficult, in part because there are few workload files that contain information on requested resources versus actual used resources per job. One workload file we found useful is the LANL CM5 [19] workload file. It contains a record of 122,055 jobs submitted to a Thinking Machines CM-5 cluster at the Los Alamos National Lab (LANL) over approximately two years. We used this workload file in our simulations for estimation of memory capacity per job (see Section 3).

[1] R^2 is a measure of fitness between the points on the graph and the regression line [17]. It represents the percentage of the data variance explained by the regression line. A high R^2 (i.e., closer to 1) represents a better fit.

1.2. Related Work

Resource management, including monitoring, matching, and allocation, is a well documented area of research. Basic dynamic resource matching is already implemented by all common scheduler systems (e.g., LoadLeveler [7], Condor [9], PBS [5], and LSF [20]) for mostly homogeneous clusters. Condor [1] suggests a declarative language (called ClassAd) and system infrastructure to match job requests for resources with resource owners. Jobs and resources declare their capabilities, constraints, and preferences using ClassAds. Each job is matched with a single machine to run the job; that is, two ClassAds are matched against each other. The basic match-making process deals only with a single resource, hence, one-to-one matching. Also, successful matching occurs when the available resource capacity is equal to or greater than the job request [13].

Several works already extend and optimize dynamic resource allocation specifically for heterogeneous computing environments. An extension for optimal one-to-many matching between a single job and multiple heterogeneous resources is described in [10]. The optimal co-matching of resources is based on application-specific global and aggregation constraints (e.g., total memory size, running all application tasks in the same grid domain). Still, in its essence, it follows the basic matching where on every machine, the amount of resources is equal to or greater than the job request. A similar problem is also solved by [14].

A linear programming approach for the resource-matching problem in a grid is described in [12]. This approach deals with sharing (not necessarily dedicated) resources and many-to-many matching between all jobs in the queue and available resources. Using linear programming instead of a user-specified mechanism as in [10], matching is optimized for different global objectives such as load balancing, throughput (matching as many jobs as possible), and minimizing the number of grid resources used.

A fuzzy resource management framework is proposed in [8]. In this framework, resource allocation is based on a quantitative model as opposed to a binary model. Every resource (e.g., machine) is given a fuzzy value between 0 and 1, which indicates its capabilities (e.g., high/low memory, high/low MFLOPS). Every job is also assigned a fuzzy value, which indicates its nature (e.g., IO intensive, CPU-bound). The matching process tries to maximize the matching of different resource capabilities with the job's nature. Depending on the categorization of job and resource capabilities, this approach can solve the under-utilization scenario, described in Section 1.1, using a completely different approach from the one proposed in this paper.

Another approach from a different perspective replaces the user's runtime estimate with automatic learning of the job runtimes needed to optimize the backfilling scheduling

algorithms [16]. While this is not a resource-matching problem *per se*, it is an example of using a learning method to optimize over-estimation of the user's input in scheduling systems, which is very similar in spirit to the approach suggested in this paper.

Previously, we addressed the problem of resource estimation in cases where similar jobs can be identified [21]. If there exists a metric which identifies two jobs that use similar resource capacities based on the job parameters, a simple learning algorithm can be used to estimate required resources. This is explained briefly below.

2. Learning to Estimate Job Requirements

2.1. The General Approach

All known approaches for dynamic matching between jobs and resources select resources whose available capacity is greater than or equal to the users' specifications. We propose an approach that can select resources whose capacity might also be lower than the job request.

Our approach is based on using automatic learning techniques to estimate the actual job requirements. These requirements assist the job scheduler in matching the jobs to computers with lower resource capacity than that specified by the job requests. These jobs have a high probability of successful termination even though they are assigned fewer resources (e.g., memory capacity) or even no resources at all (e.g., ignore some software packages that are defined as prerequisites), based on experience learned from earlier submitted jobs (e.g., how many actual resources they used). As such, our approach deals efficiently with the basic scenario described in Section 1.1.

In principle, we envision a resource estimation phase prior to resource allocation (see Figure 2). When a job is submitted to the scheduler, its actual job requirements are estimated, based on past experience with previously submitted jobs. Then, the resource allocator matches these estimated job requirements with available resources instead of matching them based on the original job requirements. Once a job completes (either successfully or unsuccessfully), the estimator gathers feedback information to improve its resource approximation for future job submissions (e.g., actual resources used). If jobs terminate unsuccessfully due to insufficient resource capacities they will be resubmitted for execution either by the users or automatically by the scheduling system (see below).

In this work we assume that job requirements are always equal to or greater than the actual used resources. We do not attempt to approximate actual job requirements in cases where the original job resources requested are insufficient for successful execution of the job. Also, the proposed estimator is independent and can be integrated with different

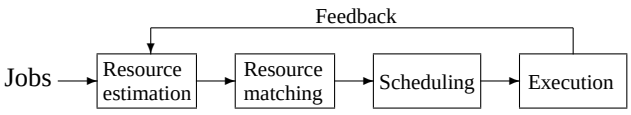

Figure 2. Schematic diagram of the scheduling process with estimation of job requirements

scheduling policies (e.g., FCFS, shortest-first-job), backfilling and different resource allocation schemes. Finally, the primary goal of the estimator is to free unused resources, which otherwise would have been allocated to jobs. As such, it is oriented towards heterogeneous cluster environments in which high throughput (and derived measurements such as slowdown) are the primary goals.

2.2. Usability Issues

Our approach for automatic resource estimation in job schedulers has several implicit assumptions regarding jobs and user experience:

1. Potential side-effects of job failures due to insufficient resources, for example, temporary opened files and allocated storage, are no different than side-effects due to other type of failures, e.g., user aborts or segmentation violation. Moreover, they are handled by similar methods and techniques such as postmortem reset and cleanup processes.

2. Users will be made aware of automatic resource allocation and the possibility of job failures due to insufficient resources. They can disable or enable automatic resource estimation for various reasons, on a per job basis, for example by adding a corresponding Boolean flag in the job request files.

3. Users have means (e.g., error logs and traces) to identify job failures due to insufficient resources. Otherwise, they may decide to disable automatic resource allocation for specific jobs. Also, if the system needs to identify these failures and resubmit jobs in an automatic fashion, some policies need to be identified. For example, many jobs allocate their memory at the initialization phase. Thus, if job execution lasts for more than one minute, it indicates that the job will not fail due to low memory resources [11].

4. Jobs with insufficient resources will eventually fail. For example, jobs which may have different behavior because of insufficient memory, i.e., intensive swapping operations, are outside the scope of our paper.

		Feedback type	
		Implicit	Explicit
Identification of similar jobs	Yes	Successive approximation	Last instance identification
	No	RL with classification	RL with regression

Table 1. Algorithms for resource estimation

2.3. Methods for Estimation of Job Requirements

Estimating job requirements depends on two main factors: the ability to infer similarity of jobs and the type of feedback offered by the scheduling system. Table 1 suggests four possible algorithms for estimating resource capacity, based on these two factors.

Similar jobs are defined as jobs that require similar amounts of resource capacity, where similar refers to capacity values that are all within a specific range (hence similarity range), for example, 10%. This range value is a qualitative measurement for the similarity of jobs within a group (i.e., there isn't a criterion for non-similar jobs). The lower the value, the more similar the jobs. It is beneficial to identify similarity groups with very low ranges. This improves the effectiveness of the resource estimator.

In this context, the resources are all system resources handled by the estimator and available for running jobs. If a job does not use a specific resource, we consider it to consume zero capacity of this resource. The similar jobs are **disjoint** groups of job submissions that use similar amounts of resource capacities.

The most simple case of similar jobs is repeated job submissions. Assume every job is assigned a unique identifier (ID), which can be used to recognize repeated submissions of the exact same job (i.e., same program, input data, and input parameters). In this case, a similarity group would include all the repeated submissions of a specific job and the resource estimator would use the experience gathered from previous submissions of that job to estimate the actual job requirements. Unfortunately, in many cases, such job IDs are not available. For example, most of the workload files in [19] do not include job IDs. Also, job IDs (assigned by users) may be a restrictive approach, narrowing the job space in which similar jobs are detected.

A more general method is to determine a set of job request parameters that can be used to identify similarity groups. This can be done by observation, where we find parameters that indicate jobs using similar resources. Alternatively, one can partition a set of jobs into groups that use similar resources and train a classifier to classify new jobs into their similarity group. By default, this process will be done offline (i.e., not as part of the resource matching process itself), using traces of feedback from previous job submissions, as part of the training (customization) phase of the estimator.

As noted above the second factor which affects estimation of job requirements is the type of feedback gathered after each job is executed. Feedback can range from implicit to explicit. **Implicit feedback** refers to a Boolean value indicating whether the job completed successfully or not. This is a basic indication supported by every cluster and scheduling system. **Explicit feedback** also includes the actual amount of resources used by a job upon its termination. Explicit feedback depends on the availability of a cluster infrastructure to gather and report this information. The feedback information is used to refine the approximation and get closer to the actual job requirements.

In practice, some balance between explicit and implicit feedback can probably be expected. That is, explicit feedback will be available for some resources, but not all of them. Explicit feedback is more informative and it is therefore expected that resource estimation will achieve better performance compared to cases where only implicit feedback is given. An additional drawback of resource estimation using implicit feedback is that it is more prone to false positive cases. These cases are, for example, job failures due to faulty programming (e.g., a job generating an exception) or faulty machines. These failures might mislead the estimator into assuming that the job failed due to insufficient estimated resources. In the case of explicit feedback, however, such confusion can be avoided by comparing the resource capacities allocated to the job and the actual resource capacities used.

The following paragraphs provide a description of methods for resource estimation that do not assume similar jobs can be identified. Methods that require the detection of similar jobs were described in detail in [21] and are not further elaborated on for lack of space. In this paper we focus here on the practical on-line algorithms (that do not require an off-line training phase for their operation).

The estimation of actual job requirements without job similarity is best approached using Reinforcement Learning (RL) [6]. RL is a class of learning algorithms where an agent learns a behavior policy by exploring a state-space. The agent can take actions, where it receives rewards for good actions or penalties for poor actions. The goal of the agent is to maximize its cumulative reward by modifying its behavior policy. RL has been applied to autonomous systems before, for example for improving load balancing by applications in server farms [15].

In the context of resource estimation, at each step the RL agent (the resource estimator) attempts to determine a policy of whether a job can be submitted for execution. The policy is learned on-line, based on the system state, (i.e.,

the status of each node whether idle or busy, and if busy, for how long), the resources of each machine, and the requested resource capacities of the jobs in the queue. A reward would be an improvement in utilization or slowdown, whereas a penalty would be a decrease in these parameters. The RL policy is initially random, but converges to a stable policy over time, via a process of trial and error. RL algorithms can also adjust the policy over time if system behavior changes. The main differences with methods that use similarity groups is that in RL, the policy is global and applied to all jobs, and that learning is performed on-line (i.e., with no training phase).

RL is general enough to be applied with either explicit or implicit feedback. Explicit feedback will help reach a more fine-grained policy, with a better estimation of the average actual resource capacities through learning of a regression model. This is done by estimating the actual resource requirements. If implicit feedback is available, a classification model will be employed which, at each time step, would decide whether or not the job can be submitted using currently available resources.

In this work, we unified the two methods (classification and regression) by using a non-linear Perceptron for estimating the job requirements. In the case of explicit feedback, the Perceptron attempts to estimate the actual resources required, after which it can be decided whether the job can be submitted using currently available resources. If only implicit feedback is available, the Perceptron only decides whether or not the job can be submitted for execution.

Pseudo-code of the reinforcement learning algorithm is shown in Algorithm 1. This algorithm is based on the Soft-Max strategy [18]. The algorithm is initiated (line 1) with a separate weight vector $\mathbf{w}$ for each resource capacity. For simplicity, in the following we assume only a single resource capacity should be estimated. In our simulations we attempted to estimate the required memory capacity for each job.

When a job is submitted, its required resources are estimated by the Perceptron (line 3). The Perceptron uses as input a feature vector derived from parameters of the job request file (for example, the requested resource capacities) and from system measurements such as the number of free nodes, the load on the input queue, etc.

If the estimated resources can be satisfied using currently available resources, the job is executed. However, even if current resource capacities cannot satisfy the estimated resources, the job might still executed. A random number is generated (line 7) and if it is larger than a threshold, the job will be sent for execution with zero estimated resources. The threshold is the probability determined by a Gibbs distribution of the parameter p, which is the fraction of successful executions so far. This is the exploration stage of the RL paradigm.

If explicit feedback is available (lines 12-13) a successfully executed job (i.e., one that was allocated sufficient resource capacities) reports the utilized capacity. If it failed, it reports the maximal allocated capacity. This means that learning in the case of failure is hindered by the fact that the estimator only knows that the allocated resource capacity was insufficient, but not what the job would have required for a successful execution. If implicit feedback is available (lines 15-16), only a Boolean indicator of success or failure is given.

Finally, the Perceptron weights and the exploration threshold p are updated according to the execution results (lines 18-21). The Perceptron is updated with the object of minimizing the error between the estimator and the actual results, where both estimation and actual results are dependent on feedback type, as defined in lines 12-16.

3. Simulations

3.1. Metrics for Evaluation

The rate at which jobs are submitted for execution on a computing cluster is measured by the **offered load**. The higher the rate, the higher the offered load. Given a set of jobs arriving at time a_i, each requiring M_i nodes for a runtime of T_i seconds ($i = 1, 2, \ldots, N$), the offered load is computed as:

$$Offered\ Load\ = \sum_{i=1}^{N} M_i \cdot T_i / \left(M_T \cdot (a_N + T_N)\right) \quad (1)$$

where M_T is the total number of nodes in the cluster.

We used two main metrics, **slowdown** and **utilization**, to evaluate the proposed algorithm. Slowdown [3] is a measure of the time users' jobs wait for execution, calculated using the following formula:

$$Slowdown\ = \frac{1}{N} \sum_{i=1}^{N} \left(W_i + M_i\right) / M_i \quad (2)$$

where W_i is the time that the i-th job waited in the input queue. One possible analogy of slowdown is latency in a network.

Utilization [3] is a measure of the clusters' activity. It is computed as:

$$Utilization\ = \sum_{i=1}^{N} M_i \cdot T_i / \left(M_T \cdot T_T\right) \quad (3)$$

where T_T the total simulation time.

In our experiments we changed the rate of job submissions within the workload files and measured the slowdown and utilization as a function of the offer load. Utilization

Algorithm 1 Reinforcement learning algorithm for estimating job requirements. J denotes a job, E' denotes the estimated resource capacity, E the available resource capacity, and U the used resource capacity. $\mathbf{w}$ is the weight vector of the Perceptron operating on a job feature vector $\mathbf{x}$, $f(\cdot)$ the activation function of the Perceptron and $f'(\cdot)$ the derivative of this function with respect to w. p is the fraction of successful executions so far. $rand$ denotes a random number chosen uniformly from $[0, 1]$. η is a learning constant and τ the decay constant for the RL algorithm.

Initialize $\mathbf{w} = \mathbf{0}$, $p = 0.5$
for each submitted job J **do**
  The estimated resource capacity $E' = f\left(\mathbf{w} \cdot \mathbf{x}^T\right)$.
  if $E' \leq E$ **then**
    Submit job to scheduler using E' as required resource capacity
  else
    if $rand > e^{p/\tau} / \left(e^{p/\tau} + e^{(1-p)/\tau}\right)$ **then**
      Submit job to scheduler using zero as the required resource capacity
    end if
  end if
  if *explicit feedback* is available **then**
    $t = \begin{cases} U & if\ \ J\ terminated\ successfully \\ E & if\ \ J\ terminated\ unsuccessfully \end{cases}$
    $z = E'$
  else
    $t = \begin{cases} +1 & if\ \ J\ terminated\ successfully \\ -1 & if\ \ J\ terminated\ unsuccessfully \end{cases}$
    $z = \begin{cases} +1 & if\ \ E' \leq E \\ -1 & if\ \ E' > E \end{cases}$
  end if
  $dE = f'\left(\mathbf{w} \cdot \mathbf{x}^T\right)$
  $d\mathbf{w} = \eta \cdot (t - z) \cdot dE \cdot \mathbf{x}^T$
  $\mathbf{w} = \mathbf{w} + d\mathbf{w}$
  Update p such that p is the fraction of successful job executions.
end for

always grows linearly as the offered load increases until the scheduling system becomes saturated. This is when, for the first time, jobs are queued in the input queue of the scheduler awaiting available resources. The higher the offered load at the saturation point, the better the computing cluster is utilized [3].

Utilization is usually computed under the assumption that all jobs performed a useful function. However, in the current setup, some jobs will ultimately fail because insufficient resource capacities are allocated to them, without performing a useful function. We therefore report **effective utilization**, which is the utilization of successfully executed jobs and **total utilization**, which is the utilization associated with both successful and unsuccessful jobs.

3.2. Simulation Setup

We used the LANL CM5 [19] as a real workload file to simulate a scheduling process where we estimated the memory capacity per job. The reason for using this workload file was because to our knowledge it is the only publicly-available workload file which reports both requested and used memory capacities. Figure 3 shows histograms of the distribution of run-time, requested and used memory, and requested processors in the CM5 workload.

In our simulations we dealt with two possibilities: Explicit feedback where if a job successfully completed it reported actual memory usage (if it did not, only allocated memory capacity is known), and implicit feedback where the only feedback was whether a job completed successfully (because it had ample resource capacity) or not.

The CM-5 cluster had 1024 nodes, each with 32 MB physical memory. For our experiments, we needed to run this workload file on a heterogeneous cluster. Thus, we had to change the workload file. We found that the minimum change would be to remove six entries for jobs that required the full 1024 nodes of the original CM5 cluster. This removal enabled us to rerun the workload file for a heterogeneous cluster with 512 original machines (32 MB memory) and another 512 machines with lower memory sizes.

We also used the algorithm in Section 2.3 to estimate the actual memory capacity for jobs. The learning constant η was set to 0.1 and the decay constant τ to 10. We used a hyperbolic tangent activation function for the Perceptron, such that $f(x) = a \cdot M \cdot tanh(b \cdot x)$, where $a = 1.716$ and $b = 2/3$ (set according to the recommendations in [2], pg. 308) and M the largest memory size in the cluster (i.e., 32 MB).

In the simulation we used a scheduling policy of first-come-first-served (FCFS). We expect that the results of cluster utilization with more aggressive scheduling policies such as backfilling will be correlated with those for FCFS. However, these experiments are left for future work. We as-

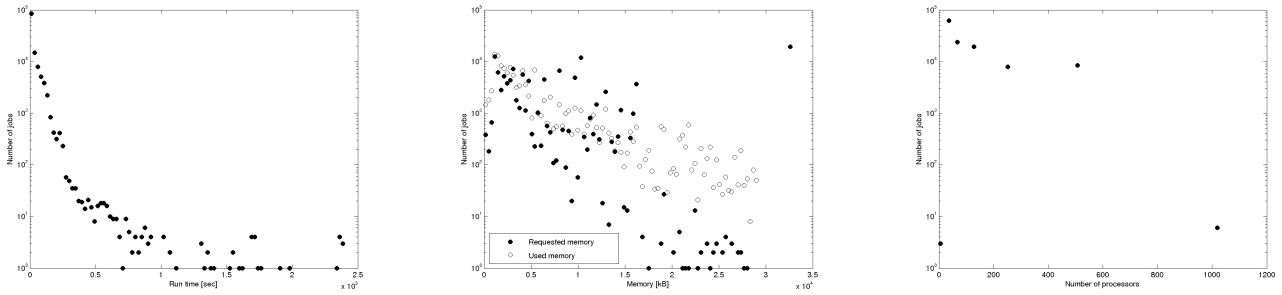

Figure 3. distribution of run-time, requested and used memory, and requested processors for the CM5 workload

sumed no job pre-emption. Moreover, when a job is scheduled for execution, but not enough resources are allocated for it, it fails after a random time, drawn uniformly between zero and the execution run-time of that job. We assume that jobs do not fail except for insufficient resources. Moreover, once failed, a job is automatically resubmitted for execution.

The offered load (see Equation 1) in each run was modified by multiplying the submission time of all jobs by a constant factor so as to achieve a required load.

In our simulations the estimator used as features the following measurements to estimate the required memory resources:

1. The fraction of available processors in the cluster with a given memory size
2. The fraction of available processors in the cluster required to complete the job
3. Logarithm of the requested memory capacity for the current job
4. Logarithm of the number of jobs in the queue (i.e., the queue length)

These features represent a heuristic of indicators that could be useful and relatively easy to obtain. However, system designers can choose to use other measurements for the estimation process.

3.3. Results

In our first experiment, we measured the effect of resource estimation on effective utilization. We experimented with a cluster of 512 machines, each with 32 MB memory, and an additional 512 machines, each with 24 MB memory. Figures 4 and 5 show a comparison of the effective utilization [3], with and without resource estimation, for both the implicit and explicit feedback scenarios. The difference between utilization with implicit and explicit feedback is small (approximately 1%). This is surprising since explicit feedback provides much additional information for the learning algorithm. We hypothesize that this is related to the naive learning method we used (Perceptron), which failed to utilize the additional information, as well as the fact that when a job fails the feedback only contains the (insufficient) allocated capacity, not the required capacity. We address these issues further in Section 4. Figures 4 and 5 further show that effective utilization with resource estimation improved by approximately 33% for the case of implicit feedback and 31% for the case of implicit feedback.

The reason for the improvement in utilization with resource estimation is as follows. At low effective loads, most of the jobs are likely to have sufficient resources as defined by the corresponding user requests. However, as the load increases, fewer jobs are likely to have available resources that match the job requests. Resource estimation increases the number of these jobs; once scheduled, it enables them to run on the cluster instead of waiting in the queue for more

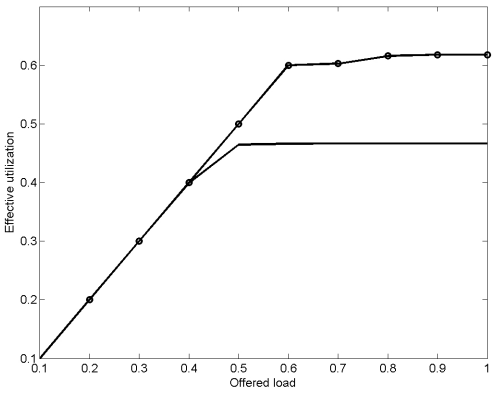

Figure 4. Effective utilization with resource estimation (dotted line) and without resource estimation (solid line) with implicit feedback

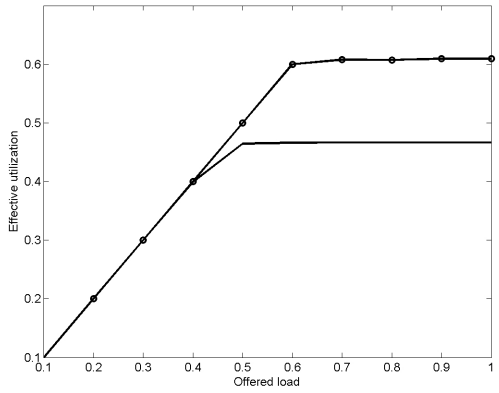

Figure 5. Effective utilization with resource estimation (dotted line) and without resource estimation (solid line) with explicit feedback

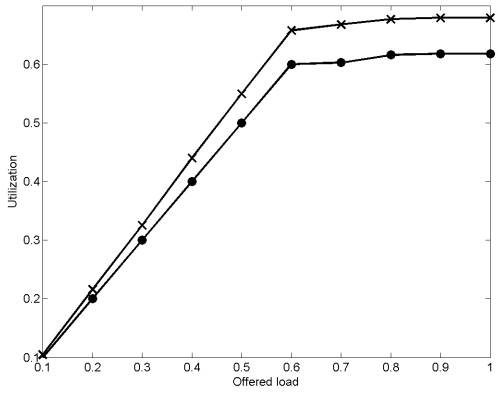

Figure 6. Effective utilization (dotted line) compared to the total utilization (line marked by crosses) in the case of resource estimation with implicit feedback

resources that they don't actually need.

Figure 6 compares the effective utilization to the total utilization in the case of implicit feedback. The difference between the effective utilization and the total utilization is the utilization caused by jobs which terminated unsuccessfully due to insufficient resources. This difference represents the overhead (in terms of computing resources, electricity, etc.) which the system administrator has to pay for the learning process to find a useful policy.

Figures 7 and 8 show the effect of resource estimation on slowdown for both types of feedback. As shown, over most of the range of offered load, resource estimation halves the slowdown compared to the case where no resource estimation is performed, and never causes slowdown to increase beyond the case where no resource estimation is performed. Moreover, slowdown decreases dramatically (by a factor of approximately 10) around loads of 50%. The reason for this peak in performance can be explained by the fact that a FCFS scheduling policy is used. The higher the loads, the longer the job queue, and the relative decrease in slowdown is less prominent. The 50% load is a point at which the job queue is still not extremely long and resource estimation is already useful in reducing the wait-time of jobs in the queue.

Reinforcement Learning finds a policy for submitting jobs. In our case, the policy was indirect in that the RL algorithm was required to estimate the memory requirements rather than the actual scheduling policy. This was done to make the proposed system independent of the scheduling policy. Clearly, the policy should also affect the submission policy. For example, if a good policy would be to try and submit all the jobs regardless of their memory requirements and the available processors, we would expect RL to estimate the required memory as lower than the minimal available memory so the jobs would be submitted with minimal requested resources.

Figure 9 shows the feature weights versus load. Feature 1 is the fraction of available 32 MB nodes. Feature 2 is the fraction of available 24 MB nodes. Feature 3 is the requested memory and Feature 4 the length of the queue. The weights associated with the fraction of available processors required to complete the job is not shown because it is highly correlated with Features 3 and 4. As Figure 9 shows, all weights tend to lower values as the load increases. This implies that as the load becomes higher, more and more jobs will be estimated as requiring low memory, and will thus be submitted whenever enough processors of *any* capacity are available. This is to be expected because at high loads it makes sense to try and use whatever resources are available.

This effect is visible in Figure 10, which shows the number of unsuccessful job submissions as a function of the load[2]. Clearly, as the load increases, more jobs are submitted with a low memory usage estimate and later fail. However, even at this high failure rate, effective utilization still increases by 30%.

Another interesting trend can be seen in Figure 9. The top row shows almost linear decline in the weights ($R^2 > 0.85$), while the lower row shows a much less linear decline ($R^2 = 0.85$ for Feature 3 and $R^2 = 0.66$ for Feature 4). We interpret this behavior by noting that at low loads the queue length is usually very short and computing resources

[2]We note that although the values on the vertical axis are large, one should bear in mind that the total number of jobs in the workload file is approximately 122,000 and that each job can fail multiple times.

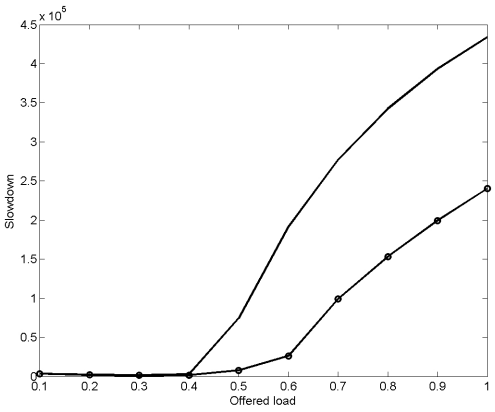

Figure 7. Slowdown with resource estimation (dotted line) and without resource estimation (solid line) with implicit feedback

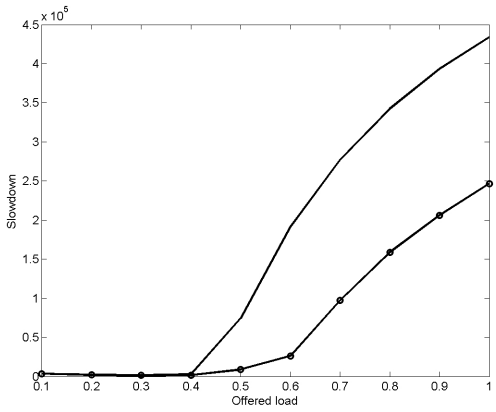

Figure 8. Slowdown with resource estimation (dotted line) and without resource estimation (solid line) with explicit feedback

are usually available. Therefore, these weights have little effect on the memory estimation. These weights only play an important role when the load is high, as shown on the graph.

All the above experiments were done with one particular heterogeneous cluster. In the following experiment, we measured the cluster utilization with and without resource estimation for different clusters. We used 512 machines with 32 MB of memory and an additional 512 machines with different memory sizes between 1 MB and 32 MB. All other simulation parameters remained as in the previous experiments.

Figure 11 shows the ratio of utilization when using memory estimation, compared to using user requirements, as a function of the variable memory part of the cluster. The greatest improvement in utilization was obtained for clusters with the 512 machines whose memory size was modified to between 16 MB and 28 MB. The utilization at the rightmost part of the graph, when all machines have 32 MB of memory, is the same with and without learning because at this point the cluster is homogeneous and any job can be successfully executed on the available machines. There is a slight decrease in utilization for clusters where the machine had memory below 15 MB. For such configurations only a small proportion of jobs could be run with lower memory. Consequently, utilization decreases because of repeated attempts to create a good estimator that is only applicable for a few jobs.

4. Discussion

This paper presents an autonomic module for assessing the actual resources required by jobs submitted to distributed computation systems. Its huge benefits were demonstrated by extremely significant improvements in utilization and slowdown. It is estimated that these improvements would not be easy to achieve through modification of user behavior.

The proposed system can utilize both implicit feedback, which only informs the learning system whether the job submission was successful, and explicit feedback, which provides more information on resource usage. Interestingly, our simulations show almost identical performance for the two types of feedback. This is surprising because it is to be expected that when additional information exists, better performance will ensue. We hypothesize that the reason for this behavior is due to the type of learning algorithm we used. We expect that better algorithms, such as Support Vector Machines (SVM) or neural networks, will show the superiority of explicit feedback over implicit feedback.

Finally, we note that the problem we addressed in this paper is prevalent in many other cases. These cases are characterized by settings where users give (poor) estimates of their needs or requirements, but where it is possible to automatically improve these estimates by learning from larger populations. One such example is setting parameters to achieve given service level objectives (SLOs) in complicated systems and creating policies for storage systems. We are currently applying the algorithms described in this paper to such problems.

References

[1] J. Basney, M. Livny, and T. Tannenbaum. High throughput computing with condor. *HPCU news*, 1(2), 1997.

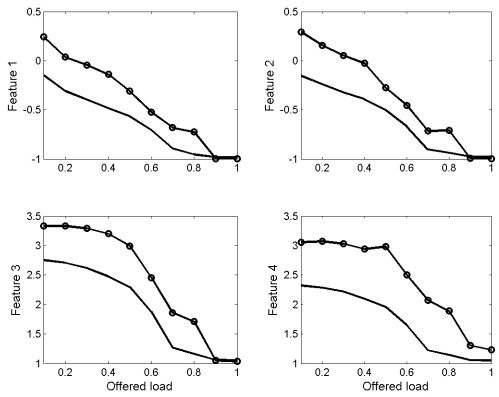

Figure 9. Feature weights versus load. Feature 1 is the fraction of available 32 MB processors. Feature 2 is the fraction of available 24 MB processors. Feature 3 is the requested memory and Feature 4 the length of the queue. Solid lines represent learning with explicit feedback. Lines marked by circles denote learning with implicit feedback.

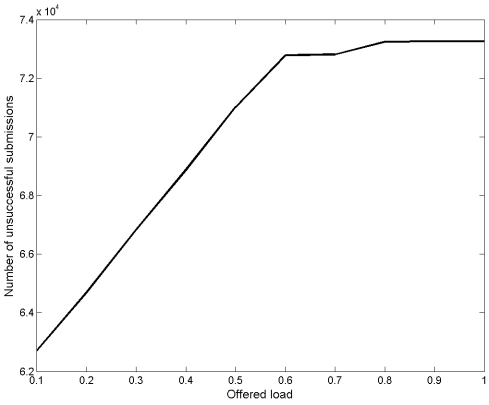

Figure 10. Number of unsuccessful job submissions as a function of the load (Implicit feedback)

[2] R. Duda, P. Hart, and D. Stork. *Pattern classification*. John Wiley and Sons, Inc, New-York, USA, 2001.

[3] D. G. Feitelson. Metrics for parallel job scheduling and their convergence. In *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 188–206, London, UK, 2001. Springer-Verlag.

[4] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling: a status report. In *10th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–16, New-York, USA, 2004.

[5] R. L. Henderson. Job scheduling under the portable batch system. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (IPPS '95)*, pages 279–294, London, UK, 1995. Springer-Verlag.

[6] L. P. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[7] S. Kannan, M. Roberts, P. Mayes, D. Brelsford, and J. F. Skovira. *Workload Management with LoadLeveler*. IBM Press, 2001.

[8] K. Kumar, A. Agarwal, and R. Krishnan. Fuzzy based resource management framework for high throughput computing. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2004)*, pages 555–562. IEEE, 2004.

[9] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[10] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and evaluation of a resource selection framework for grid applications. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02)*, page 63, Washington, DC, USA, 2002. IEEE Computer Society.

[11] M. Livny. Personal communication, 2005.

[12] V. Naik, C. Liu, L. Yang, and J. Wagner. On-line resource matching in a heterogeneous grid environment. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*. IEEE, 2005.

[13] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.

[14] R. Raman, M. Livny, and M. Solomon. Policy driven heterogeneous resource co-allocation with gangmatching. In *12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12 '03)*, pages 80–89, 2003.

[15] G. Tesauro, N. Jong, and R. D. abd M.N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC) 2006*, pages 65–73, Dublin, Ireland, 2006.

[16] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using runtime predictions rather than user estimates. Technical Report TR 2005-5, School of Computer Science and Engineering, Hebrew University of Jerusalem, 2003.

[17] G. Upton and I. Cook. *Oxford Dictionary of Statistics*. Oxford University Press, Oxford, UK, 2002.

[18] J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. In *Proceedings of the 16th European Conference on Machine Learning (ECML 2005). Lecture Notes in Computer Science*, volume 3720, pages 437–448, Porto, Portugal, 2005.

[19] Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload.

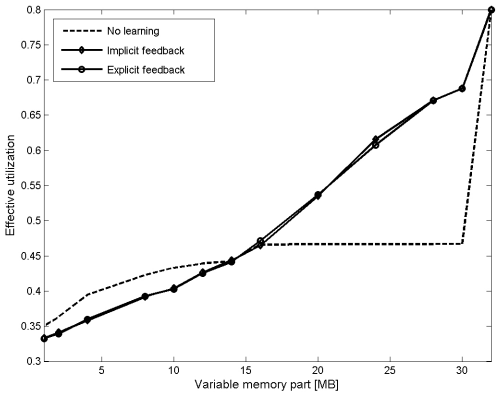

Figure 11. Utilization with and without resource estimation for the implicit and explicit feedback cases

[20] M. Q. Xu. Effective metacomputing using lsf multicluster. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 100, Washington, DC, USA, 2001. IEEE Computer Society.

[21] E. Yom-Tov and Y. Aridor. Improving resource matching through estimation of actual job requirements. In *IBM Research Report H-0244*, 2006.