H-0244 (H0611-003) November 8, 2006
Computer Science

IBM Research Report

Improving Resource Matching through Estimation of Actual Job Requirements

Elad Yom-Tov, Yariv Aridor
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

Improving Resource Matching Through Estimation of Actual Job Requirements

Elad Yom-Tov, Yariv Aridor
IBM Haifa Research Laboratory
Haifa 31905, Israel
{yomtov,yariv}@il.ibm.com

Abstract

Heterogeneous clusters and grid infrastructures are becoming increasingly popular. In these computing infrastructures, machines have different resources (e.g., 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. The problem of over-provisioning has received very little attention so far. We discuss different approaches for applying machine learning methods to estimate the actual resource capacities used by jobs. These approaches are independent of the scheduling policies and the dynamic resource-matching schemes used. Our simulations show that these methods can yield an improvement of over 50% in utilization (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.). Also, 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, prerequisite software packages. Upon scheduling a job, 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 [6]. 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. We do not deal with the problem of over-provisioning of 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). The over-provisioning problem is the focus of this paper.

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 which is 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.

Research of the over-provisioning problem is difficult partly because there are few workload files that contain information on requested versus actual used resources per job. One workload file we found useful is the LANL CM5 [20] workload file. It contains a record of 122,055 jobs submit-

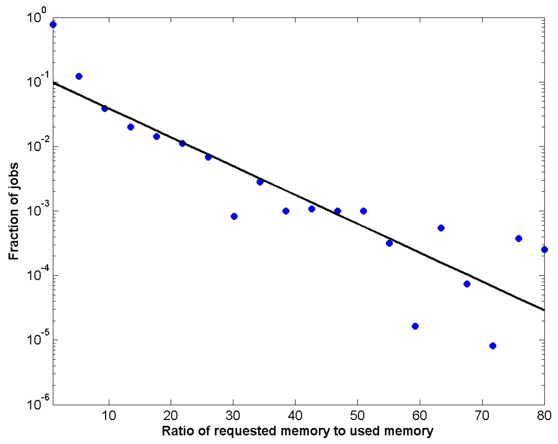

Figure 1. An histogram of the ratio between requested memory size and actual memory used, per job, in the LANL CM5 workload file. The vertical axis is logarithmically scaled

ted 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).

The over-provisioning problem is demonstrated in Figure 1. This figure shows a histogram of the ratio of requested to used memory in the LANL CM5 log [20]. As this figure demonstrates, there are approximately 32.8% of jobs for which there is a mismatch of twice 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 fitness 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.

Throughout the rest of the paper, we use the terms ’resource estimation’ and ’estimation of resource capacity’ interchangeability. The same holds for the pair of terms ’actual resources’ and ’actual job requirements’, and ’memory used’ and ’size of memory used’.

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 [11], Condor [13], PBS [9], and LSF [21]) for mostly homogeneous clusters. Condor [2] suggest 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 [16].

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 [14]. 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 [17].

A linear programming approach for the resource-matching problem in a grid is described in [15]. 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 [14], 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 [12]. 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, in a completely different approach from ours.

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 [18]. 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 sug-

[1]R^2 is a measure of fitness between the points on the graph and the regression line [19]. 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.

gested in this paper.

1.3 Our 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, which 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 with 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.

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 requested resources are insufficient for successful execution of the job. Also, the proposed estimator is independent and can be integrated with different 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 toward heterogeneous cluster environments in which high throughput (and derived measurements such as slowdown) are the primary goals.

To the best of our knowledge, this is the first work to discuss the over-provisioning problem and suggest the integration of machine learning techniques as part of the solution. We found this interdisciplinary problem a real research challenge. We initially decided to experiment with simple resource estimation approaches to understand different tradeoffs (e.g., offline versus online modes of operations) and requirements (e.g., feedback mechanisms) that

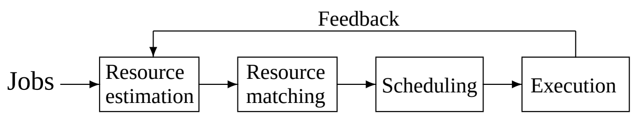

Figure 2. Schematic diagram of the scheduling process with estimation of job requirements

are imposed on scheduling systems. This paper reports the results of our work and provides a roadmap for future research.

The rest of this paper is structured as follows. In Section 2 we describe how resource capacity estimation can be achieved using similarity of jobs. We discuss how to measure such similarity and how it can be used to estimate resource capacities. In Section 3 we demonstrate the use of one resource capacity estimation algorithm on a real job workload file. Finally, Section 4 discusses future directions for this work.

2 Estimation of Actual Job Requirements

2.1 Prerequisites for Resource Estimation

In this section we present a resource estimation approach that is dependent upon requested resource capacities for the jobs and on the experience gathered with similar jobs previously submitted to the cluster. We discuss other approaches in Section 4. The similar jobs are **disjoint** groups of jobs submissions that use similar amount of resource capacities (hence similarity groups). Resources, in this context, are all the system resources handled by the estimator and are available to running jobs. If a job does not use a specific resource, we consider it to consume zero capacity of this resource.

By "similar amount of resource capacity" we refer to capacity values that are all within a specific range (hence similarity range) for example, 10%. This range value is a qualitative measurement for 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 as described in Section 2.3.

The estimator maintains experience information for each similarity group to estimate the resources for future job submissions that belong to (i.e., are associated with) this similarity group. The experience maintains the current resource estimation determined so far for the similarity group. In addition, depending on the resource estimation algorithm used, it might also include the requested resource capacities of the previously submitted jobs and the actual resource capacities that were available (allocated) to these jobs during execution.

The experience information is updated by feedback gathered per job execution. 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. It 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. Hence the larger the similarity group, the more feedback is collected and closer approximation can be determined. Also, the larger the group, the more jobs that can benefit from accurate approximation values for their actual resources.

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 so it is 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 confuse the estimator to assume that the job failed due to too low (insufficient) estimated resources. In the case of explicit feedback, however, such confusions can be avoided by comparing the resource capacities allocated to the job and the actual resource capacities used.

2.2 Job Similarity

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 [20] do not include jobs IDs. Also, jobs 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 parameters of job requests by which similarity groups can be identified. In this case, every similarity group has job submissions with the same value for all these parameters. As an example, we experimented with the LANL CM5 workload file. Since it does not have job IDs, we decided to identify similar jobs for the LANL CM5 by finding jobs with the same user ID, application number, and requested memory size. This resulted in 9885 disjoint sets of similar jobs from a total of 122,055 jobs.

There is no formal method to determine the best set of job request parameters for job similarity. In practice, it is made through trial-and-error search and measurements (see below). By default, this process will be done offline (i.e., not as part of the resource matching process itself), using traces of explicit feedback from previous job submissions, as part of the training (customization) phase of the estimator.

Two main measurements that can qualitatively indicate a successful selection of job request parameters for similarity groups are shown in Figures 3 and 4. Each point of the histogram in Figure 3 represents all similarity groups of the same size. The horizontal axis shows sizes of similarity groups. The vertical axis shows the fraction of jobs from groups with this size and the total number of jobs in the workload file. As mentioned earlier, the larger the group, the more job submissions that would benefit from estimation of the actual resources required (see also Section 2.3). Thus, ideally, we would expect few larger similar groups which span across most of the jobs. This is not the case at hand. As shown by the histogram, there are many similarity groups and in general, the larger the group, the smaller fraction of jobs spanned. From this perspective, the set of job request parameters we choose for similar jobs may not be the best one possible.

Another measurement is shown by the graph in Figure 4. This graph indicates the quality of the job request parameters for similarity groups in the LANL CM5 workload file and the potential effectiveness of the resource estimator. For each group containing ten or more jobs (19.4% of sets, or 83% of the total jobs in the workload file)[2] we plotted the ratio between the requested memory and the maximum used memory in each group (the vertical axis) as a function of the ratio between the maximum used memory and the minimum used memory in the group (in the horizontal axis). The latter are actually the similarity ranges. As shown, a large fraction of the similarity groups are at the lower end of the similarity range values. This indicates the quality of our criterion for similar jobs.

This graph shows another interesting observation. Naturally, the largest gain in applying estimation would likely be obtained for jobs where the ratio of requested memory to maximal used memory is the highest. These are the op-

[2]The limit of ten jobs was set so as not to clutter the figure and since the largest gain in estimation is obtained from the largest groups.

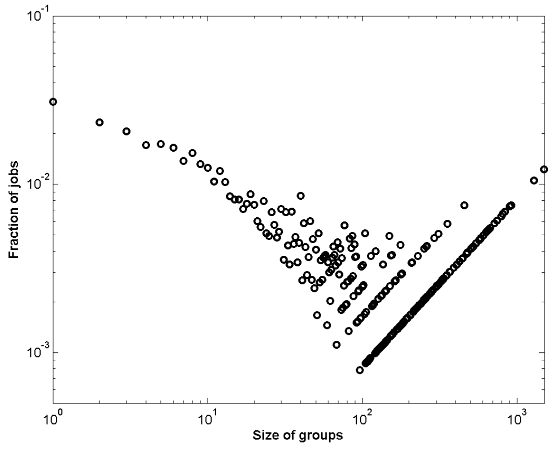

Figure 3. The distribution of jobs according to group size for the LANL CM5 workload file. The vertical axis is logarithmically scaled

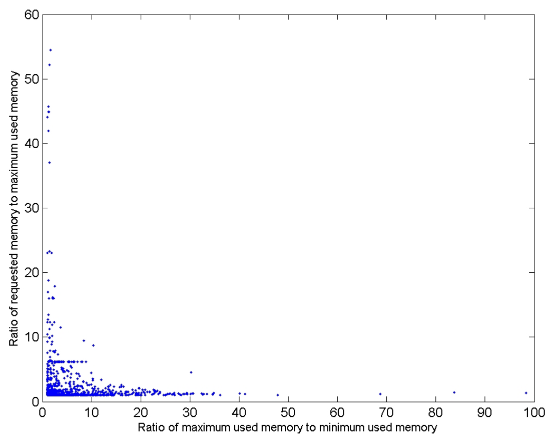

Figure 4. Measuring the possible gain from resource estimation versus group similarity in the LANL CM5 workload file. The possible gain is measured by the ratio of requested memory to maximal used memory. Similarity is computed by the ratio of maximum used memory to the minimum used memory. Each point in this graph represents a similarity group

portunities where the largest unused resource capacity can be saved. The graph in Figure 4 indicates that there are jobs with a very high (above one order of magnitude) ratio between requested memory and maximal used memory and that these jobs are also very similar. Qualitatively, this is a good starting point for effective resource estimation.

2.3 The Resource Estimation Algorithm

In this section, we present an algorithm for resource estimation based on the notion of similar groups. If explicit feedback is available, the resource estimation can be performed by simply using the actual resources used by the previous job submission as the estimated resources for the next job submission in the same similarity group.

The more interesting case is with implicit feedback. Here we propose using a successive approximation algorithm [8]. For reasons of clarity, we present an algorithm to deal with only one type of resource.

The pseudo-code of the algorithm is shown in Algorithm 1. Following is a description of the algorithm steps. The numbers in parentheses refer to line numbers of the pseudo-code. For every new job submission (2), the algorithm attempts to find its similarity group. If none exists, a new group is defined. Every group i maintains the current estimated resource capacity (E_i), initialized with the value of the requested resource capacity of the first job in the group (R), and a learning rate α_i initialized with the default learning rate of α (4). The cluster may not have nodes with the exact resource capacity E_i. Thus, the estimated resource capacity for the job (E') is rounded (denoted by $\lceil \cdot \rceil$) to the lowest resource capacity within the cluster, greater than E_i (6). Then, the resource allocator is called with the estimated resource capacity as the required resource capacity (7) and the algorithm waits for implicit feedback. If the job terminated successfully, the estimated resource of the similarity group, E_i, decreases by the corresponding learning parameter α_i (9). This is to get closer to the actual resources for future job submissions, if any (9). Otherwise, the algorithm assumes that the job did not have sufficient resources to complete successfully. Consequently, it restores the similarity group, E_i, to its previous value (11) and decreases the learning factor, α_i using a global factor β (12). We take care not to lower α_i below the value of one. Otherwise, the estimated resource in (9) would increase instead of decrease. Finally, a new value of E_i is defined for subsequent job submissions (13). These last three steps enable more fine-grained approximation of the actual resource through smaller reductions of the estimation resource value.

The initial setting of the estimator parameters, α and β, can affect the effectiveness of the algorithm (i.e., gain in cluster throughput due to more free resources). Setting α to a large value will require more jobs from each similarity group to reach good estimation accuracy, since such values cause inferior initial approximation of actual resources (before β reduces α and achieves finer approximation). This will reduce the amount of unused resources that can be saved and affect the number of jobs that can be co-matched

Algorithm 1 The successive approximation algorithm for resource estimation. J denotes a job, E_i denotes the estimated resource capacity of the similarity group i, R is the value requested by the user, α (α_i) and β are the algorithm parameters

```
1: Initialize $\alpha > 1$ , $0 < \beta < 1$
2: for each submitted job $J$ do
3:     if a similarity group for the job $J$ is not found: then
4:         Initialize a new group i, and for this group set $E_i \leftarrow R$; $\alpha_i = \alpha$
5:     end if
6:     $E' \leftarrow \lceil E_i \rceil$
7:     Submit job to scheduler using $E'$ as required resource capacity
8:     if J terminated successfully then
9:         $E_i \leftarrow E'/\alpha_i$
10:    else
11:        $E_i \leftarrow E' \cdot \alpha_i$
12:        $\alpha_i \leftarrow \max\left(\alpha_i^{\beta}, 1\right)$
13:        $E_i \leftarrow E'/\alpha_i$
14:    end if
15: end for
```

successfully.

On the other hand, setting α to a value which is too low will cause the algorithm to be too conservative in trying to reduce estimated resource capacities (see Section 3.2 for a demonstration of this phenomenon). For example, consider a similarity class where jobs request 32MB memory while using 4MB of memory, submitted to a cluster where machines have either 32MB, 24MB, or 4MB of memory. Assume $\alpha = 2$ and $\beta = 0$. When the first job of this class is submitted, it will be run on the 32MB machines (with an estimated memory of 32MB). The next submission might run on the 24MB machines with an estimated memory of 16MB. However, a third iteration will not take place because the next step in estimation is 8MB, which is greater than the 4MB of the smallest memory machines. If, however, α is set to a higher value, for example, $\alpha = 10$, the iteration steps would be execution on the 32MB machines and then on the 4MB machines. This, however, will be problematic if the actual memory used was 5MB instead of 4MB, because the estimation will revert back to 32MB, instead of 24MB, as in the previous example.

Finally, the expected variance of the similarity ranges i.e., the difference between the minimal and maximal resource capacity used inside a similarity group affects the selection of the value for the estimator parameters. This variance can be determined during the process of determining similarity groups (see Section 2.2). Thus, when this difference is large, α should be set to a low value to reach fine-grained approximation.

The setting of β requires a balance between two goals. Setting β to a large value (closer to 1) will cause a very gradual decrease in the number of steps required for the resource capacity estimation. Thus, a more accurate estimation can be attained. However, this can also result in many jobs failing repeatedly until their correct resources are estimated.

The algorithm above has a few interesting characteristics to note. First it is very memory space efficient. It only saves two parameters, E_i and α_i, in memory, per every similarity group. Second, as noted earlier, the larger the similarity group, the more attempts will be made to reduce and reach a closer estimation of the actual resources used by the jobs. Third, this algorithm implicitly assumes that all jobs in a given similarity group use the same actual resource capacities. This works fine for small similarity ranges. However, for larger ranges, it impacts the approximation of the actual resources. Assume, for example, two jobs J1 and J2, within the same similarity group, with actual resources of 12MB and 18MB, respectively. Also, assume 64MB as requested memory for both jobs, α=2 and β=0, and that machines in the cluster have memory capacities of 64MB, 32MB, 16MB, and 8MB. If J2 is submitted after J1, the final estimated resources would be 32MB since an attempt to match 16MB for J2 will fail. However, 16MB would be a better estimate for J2. This problem can be solved using a class of robust line search algorithms [1]. This extension is outside the scope of this paper. Finally as mentioned earlier, this algorithm is designed for a single resource instance. If one would attempt to use this algorithm for simultaneous estimation of several resources, modifying several of them at each step, it would be difficult to know which of these resources causes the algorithm to terminate. The algorithm can be generalized for multiple resources using methods of multidimensional optimization [3].

3 Experiments and Results

3.1 The Simulation Environment

In this section we used the LANL CM5 [20] as a real workload file to simulate a scheduling process with estimation of memory capacity per job. The CM-5 cluster had 1024 nodes, each with 32MB physical memory. For our experiments, we needed to run this workload file on an 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 (32MB memory) and another 512 machines with lower memory sizes.

We assumed implicit feedback, which is the general case for the estimator. We also used the algorithm in Section 2 and similarity of jobs based on the user ID, application number, and requested memory[3] (as discussed in Section 2) for estimation of actual memory capacity for jobs. The algorithm parameters were set to $\alpha = 2$, $\beta = 0$. Based on our experiments, these values represent best the tradeoff between larger and lower values of α and β as discussed in Section 2.3.

In the simulation we used first-come-first-served (FCFS) as the scheduling policy. We expect that the results of cluster utilization with more aggressive scheduling policies like backfilling will be correlated with those for FCFS. However, these experiments are left for future work. We assumed 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. Once it fails, the job returns to the head of the queue.

3.2 Simulation Results

In our first experiment, we measured the effect of resource estimation on the cluster utilization [5]. We experimented with a cluster of 512 machines each with 32MB memory, and an additional 512 machines each with 24MB memory. Figure 5 shows a comparison of the utilization [5] with and without resource estimation. With the latter, the resource matching used the resources specified in the user requests. As shown, utilization with resource estimation improved by 58%[4].

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 resources that they don't actually need.

Figure 6 shows the effect of resource estimation on slowdown [5][5]. It shows the ratio between the slowdown without resource estimation and the slowdown with resource estimation for several loads. As shown, resource estimation never causes slowdown to increase. Moreover, slowdown decreases dramatically at a load of 60%. The reason for

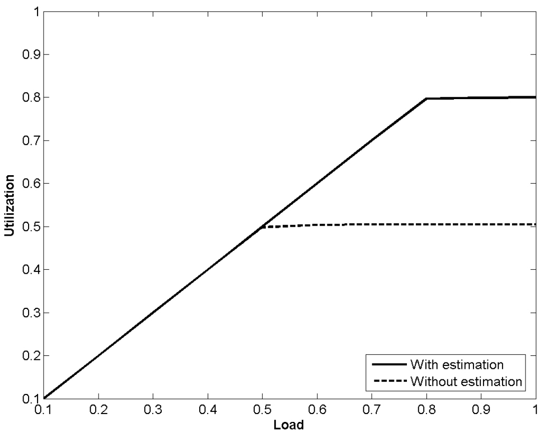

Figure 5. The effect of resource estimation on cluster utilization for the LANL CM5 workload and an heterogeneous cluster of 512 nodes with 32MB memory and an additional 512 machines with 24MB memory each

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 60% load is a point at which the job queue is still not extremely long and resource estimation is already useful for reducing the wait-time of jobs in the queue.

The advance of memory estimation per job is shown in Figure 7. It refers to a particular set of similar jobs whose requested memory was 32MB and the actual memory was slightly more than 5MB. As shown, the estimated memory decreased by a factor of two until it dropped below the actual memory used. This caused the job to terminate abnormally. Consequently, the final estimated memory was 8MB. For these jobs, a four-fold reduction in memory resources was found.

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 in which we used 512 machines with 32MB of memory and an additional 512 machines with different memory sizes between 1MB and 32MB. All other simulation parameters remained as in the previous experiments.

Figure 8 shows the ratio of utilization when using memory estimation, compared to using user requirements. The greatest improvement in utilization was obtained for clusters with the 512 machines whose memory size was modified to between 16MB and 28MB. There is no improvement for clusters where machine had memory below 15MB and

[3]The reader should not be confused. In the previous section we used the LANL CM5 file just to measure the quality of our general criterion for similar jobs, not to determine the best criterion possible for this workload file. Thus, this does not interfere with using this workload file for simulation of job scheduling in this section.

[4]In each case, we used the utilization values at the saturation points where the linear growth of utilization stops [7].

[5]The average of the job's wait time in the queue and its execution time divided by the execution time. One possible analogy of slowdown is latency in a network.

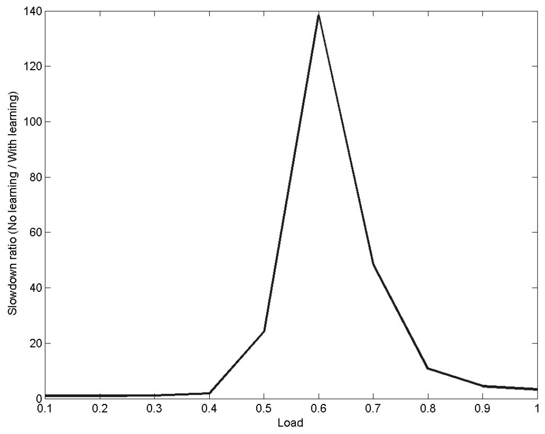

Figure 6. The effect of resource estimation on slowdown for the LANL CM5 workload and an heterogeneous cluster of 512 nodes with 32MB memory and an additional 512 machines with 24MB memory each

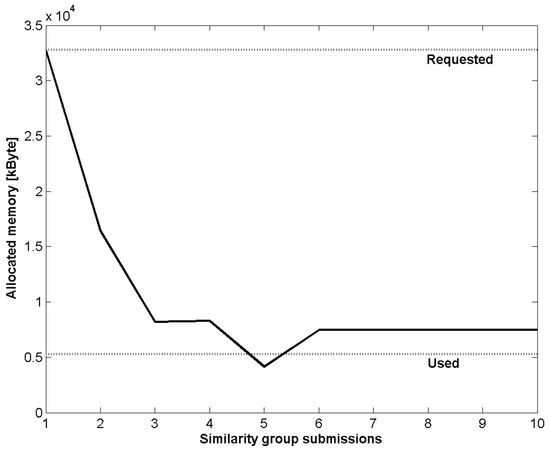

Figure 7. Estimated memory for a single similarity group across several estimation cycles. The requested memory is 32MB and the job's actual memory usage is slightly more than 5MB

for the cluster in which all machines had 32MB memory.

Our explanation of this behavior is that the improvement in utilization through resource estimation is dependent on only a fraction of jobs for whom estimation was beneficial and the number of machines they requested. Such jobs have two relevant characteristics.

First is a cluster issue. That is, the job has more candidate machines to run on using the actual resource capacities compared to when using the requested resource capacities. One such case is when the requested memory is greater than the minimal memory size of the cluster machines, while its actual used memory is lower than this minimal memory size. Consider, for example, a job which requests 20MB of memory, while requiring only 10MB to run. If the cluster has machines with 30MB and 15MB of memory, this job could only be executed on the 30MB machines, if resource estimation is not used. However, if the estimation is successful (for example, if $\alpha = 2$), this job could also be run on the machines with the 15MB memory (see also the scenario in Section 1.1). In this example, jobs with actual used memory above 15MB will not benefit from resource estimation.

Second is an estimation algorithm issue. The estimated resource capacities should enable running the job on machines with lower resource capacities which would otherwise be impossible using the requested resource capacities. In the case of Algorithm 1, the factor α should be such that the requested capacity of these jobs will be reduced so as to enable utilizing cluster machines with lower resource capacities than originally required. For example, consider the job described above again, but assume $\alpha = 1.2$. The job will not be sent for execution on the lower capacity machines that have 15MB available, because the requested memory (20MB) divided by α is equal to 16.7MB, which is larger than 15MB. This forces the allocation of machines with 30MB of memory (See Algorithm 1, step 6). However, this is not the case if $\alpha = 2$, as described above.

Thus, only jobs with the above-mentioned characteristics would benefit from resource estimation. This explains the two regions in Figure 8 for which there is no improvement in utilization. First, when all the machines in the cluster have a memory of 32MB there are no jobs which answer the first condition above. When the cluster nodes have either 32MB memory or a memory capacity in the the 1-15MB range there are few jobs for which estimation was effective due to the second condition. The small benefit to utilization was offset by job failures due to under-estimation of the Algorithm 1.

Moreover, when the number of requested nodes of the jobs for which estimation is effective are counted and compared to the ratio of utilization with and without resource estimation (as shown in Figure 8), an even more interesting observation emerges. In the range of 16-28MB there is a linear fit ($R^2 = 0.991$) between the node count of the jobs described above and the improvement in utilization.

This realization of which jobs benefit from resource estimation and the almost perfect fit between the node count of these jobs and the improvement in utilization has one very advantageous outcome. Given the distribution of requested and actual resource capacities, possibly derived

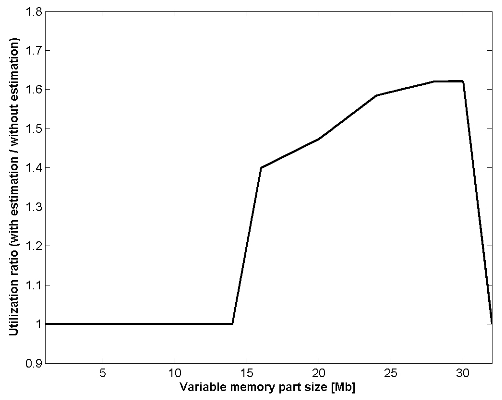

Figure 8. Ratio of utilization with estimation to utilization without estimation

from a scheduler log, and a resource estimation algorithm, it is possible to design a cluster (that is, to choose the machines constituting it) so as to increase the cluster utilization. This can be done by choosing the resource capacities of the cluster machines to maximize the number of jobs for which estimation is advantageous, as described above.

Finally, based on our simulations, we conclude that the algorithm was extremely conservative in its action. For all the different cluster configuration we tried, at most only 0.01% of job executions resulted in failure due to insufficient resources, while 15%-40% of jobs were successfully submitted for execution with lower estimated resources than the job requests.

4 Summary and Future Work

Heterogeneous clusters and grid infrastructures are becoming increasingly popular. One of the differences between these computing infrastructures and more traditional homogeneous clusters is that machines have different resource capacities, e.g., memory sizes, disk space and installed software packages. In this article, we demonstrate that these differences, together with the difficulty users encounter when trying to assess job requirements, result in a problem of over-provisioning of resources. To the best of our knowledge, this paper is the first to highlight this problem and to attempt to solve it.

We suggest the idea of estimation of actual resource capacity to cope with over-provisioning resources. We focus on resource estimation using the concept of job similarity. However, this is by no means the only method for resource estimation. Table 1 suggests four possible algorithms for estimation of resource capacity, based on the type of feedback available and whether it is possible to identify similar jobs

		Feedback type	
		Implicit	Explicit
Identification of similar jobs	Yes	Successive approximation	Last instance identification
	No	Reinforcement learning	Regression modeling

Table 1. Algorithms for resource estimation

(i.e., similarity groups with small similarity ranges). Note that the algorithms in the first row of this table are those discussed in this paper.

The estimation of actual job requirements without job similarity is best approached using reinforcement learning (RL) [10], which is a class of learning algorithms where an agent learns a behavior policy by exploring a state-space. The agent can take actions for which 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. In the context of resource estimation, the policy that the RL agent (the resource estimator) would attempt to find, is whether at each time step, 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 (idle or busy, and if busy, for how long) and the resources of each machine as well as 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. The main difference with methods that use similarity groups is that in RL, the policy is global and applied to all jobs. For example, if all users over-estimated their resource capacities by 100%, the global policy to which RL will converge is that it is sufficient to send jobs for execution with only 50% of their requested resources. RL is general enough to be applied with either explicit or implicit feedback. Explicit feedback will help to reach a more fine-grained policy i.e., better estimation of the average actual resource capacities.

If explicit feedback is available, it is also possible to use regression models [4] to estimate required resources. Regression models (either linear or non-linear) can be used to learn a mapping from the request file parameters to the actual resource capacities used (which is why explicit feedback is required for regression models). For example, in the case of linear models, the regression model estimates the actual resources as a weighted sum of the requested resources. Unlike the RL approach, in regression models a mapping is determined by training using workload files of previously submitted jobs, each with its requested resource capacities and the actual resource capacities used.

This mapping is then used to estimate resource capacities upon job scheduling. Considering the example above, regression models would learn a mapping which would be to divide each requested resource capacity by 2. The end result of regression models might be similar to that of RL with explicit feedback, but the methods by which these results are reached is very different. While regression models learn a mapping from input parameters (the request file) to output parameters (the used resource capacities), RL learns by trial and error.

The above is by no means a detailed description of alternative approaches. It is our research roadmap and plans for future work. This is in addition to other open research challenges derived by this paper. For example, online identification of similarity groups (unlike the offline approach discussed in this paper), dealing with side-effects of jobs failures due to under-provisioning of resource capacities (because of too low estimation), and more formal ways to initialize the learning algorithm's parameters (e.g., α and β). In this paper, we aimed at defining the foundation and identifying the issues and tradeoff that need to be addressed for further research of over-provisioning with learning methods.

5 Acknowledgements

We thank Miron Livny for the fruitful discussion about the ideas in this paper.

References

[1] E. J. Anderson and M. C. Ferris. A direct search algorithm for optimization with noisy function evaluations. *SIAM J. on Optimization*, 11(3):837–857, 2000.

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

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

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

[5] 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.

[6] 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.

[7] E. Frachtenberg and D. G. Feitelson. Pitfalls in parallel job scheduling evaluation. In *11th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 257–282, New-York, USA, 2005.

[8] J. D. Gibson. *The Communications Handbook*. CRC Press, 2002.

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

[21] 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.