RC24901 (W0911-062) November 9, 2009
Computer Science

IBM Research Report

Workload Management in Cloud Computing Using Meta-Schedulers

Yanbin Liu, Norman Bobroff, Liana Fong, Seetharami Seelam
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Workload Management in Cloud Computing using Meta-Schedulers

Yanbin Liu Norman Bobroff Liana Fong Seetharami Seelam

IBM T. J. Watson Research Center, Hawthorne, NY 10532, USA

email {ygliu, bobroff, llfong, sseelam}@us.ibm.com

Abstract

Cloud computing is an emerging paradigm in the area of distributed systems. While numerous information technology vendors aggressively position themselves in offering various cloud services, many computer technologists earnestly provide their definitions of cloud computing, and compare this paradigm to its predecessors such as grid computing and utility computing. In this paper, we argue and demonstrate that we can extend our meta-scheduling design from grid computing to achieve adaptive usage of resources provided by cloud computing providers. Moreover, our meta-scheduling design has the advantage in managing the cloud resources as a collection instead of individual entities such that the organizations consuming the cloud resources can determine its cloud-using policies and delegate the controlling of the workloads to be executed to meta-schedulers. Evaluations of two workloads from real traces, in terms of user-perspective delay time and system-perspective resource utilization, are presented using our meta-scheduling algorithms and pools of cloud resources in different sizes. An interesting result is that the interaction between grid workload, backfill scheduling, and job routing from an enterprise cluster to a cloud can achieve better performance than a dedicated cluster of the combined size of the enterprise and cloud systems.

1 Introduction

Cloud computing is an emerging paradigm in the area of distributed systems. While numerous information technology (IT) vendors [1, 2, 4] aggressively position themselves in offering various cloud services such as software as a services(SaaS), infrastructure as a service (IaaS), and platform as a services(PaaS), many computer technologists [8, 22, 13, 31, 10] also earnestly provide the definitions of cloud computing from their perspective.

As cloud computing is in the process of being formed and shaped in every aspect (e.g. terminology, architecture design, usage models, what current technologies can be leveraged, and what new technologies need to be developed), researchers [19, 23, 8] compare cloud computing to its predecessors such as grid computing [20] and utility computing [9, 14]. Vaquero et. al [27] summarize the definitions of cloud computing and the comparison with grids from the material published thus far by the computing community.

Since cloud computing can be seen as a next step of grid and utility computing, we expect that grid and utility computing technologies have much applicability in cloud computing. In this paper, we argue and demonstrate that our meta-scheduling technologies originally designed for grid can be extended to address two particular challenges that are present in cloud computing: 1) how much resources would be beneficial to the specific workloads and 2) how to manage and efficiently use the cloud resources.

The researchers from Berkeley [10] examine the types of applications that can be enabled by and benefit from cloud computing. They identified three out of five application types that are parallel batch processing, business analysis, and compute-intensive desktop applications. All three application types are compute-intensive. Since our meta-scheduling design supports parallel applications and has a focus on the performance of compute-intensive applications, we would see our effort can extend to work in cloud computing environment. In this paper, we also use jobs from two real traces of scientific applications to collect experimental results.

We can classify the users of cloud services into two major categories: individual clients and organization users. Individual clients would acquire cloud resources for their own specific usage and in general, would know their resource requirement well. Organization users would acquire and/or release cloud resources as results of the aggregated need from internal users in the organizations, which is generally hard to model. Our study focuses on the organizational need and makes decisions about 1)whether and how much we need cloud resources to extend the existing capacity of resource pools owned by the organizations, and 2) how much workload we should send to the clouds such that we can improve both the performance recognized by the internal users and the resource utilization at the cloud. In this paper, we set up a base resource cluster and study if a cloud cluster, which is available for a period of time, can improve the performance measured as delay time and the resource utilization of cloud cluster. We found that based on the workload characteristics, we can get different results.

In summary, the contributions of our work are:

- Presenting the meta-scheduling design to manage cloud resources,
- Presenting an easy-to-extend meta-scheduling algorithm
- Demonstrating the use of meta-scheduling in cloud computing in a close to real environment and using jobs from real traces

- Studying the effectiveness of using different sizes of cloud resource.

In the following sections, we describe the architectural design and the implementation for managing cloud resources through meta-schedulers in Section 2 and 3; we set up an real experimental environment and study the effectiveness of using cloud resources for scientific workloads in Section 4; we discuss related work in Section 5 and conclude in Section 6.

2 Workload Management using Cloud Resources

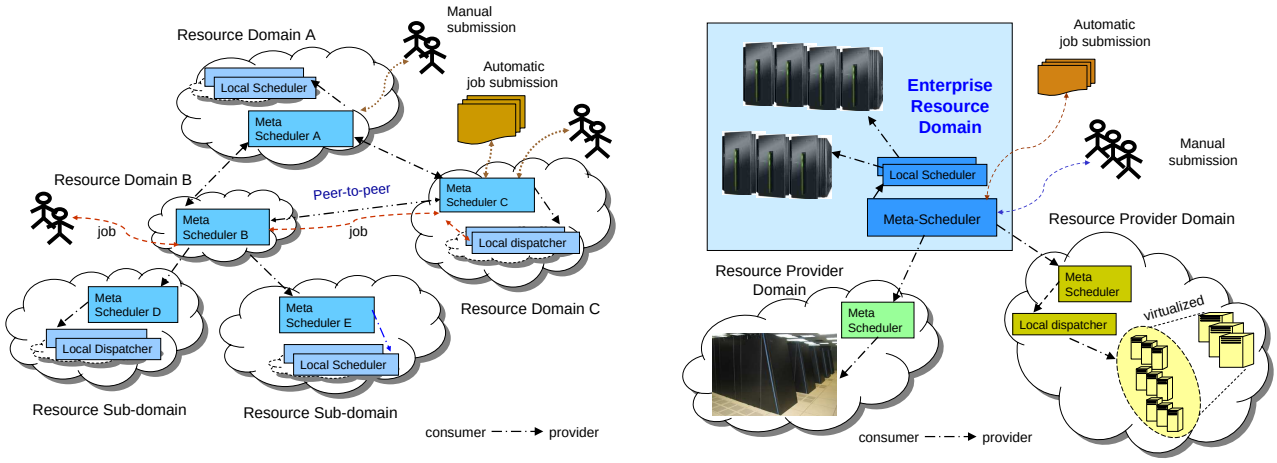
2.1 Motivation Scenarios

When we consider the cloud services, we should be aware that the management goals of cloud resources are vastly different between the provider and consumer of the cloud services. On the provider side, the major issues can be the efficient resource provision that primes the resources to meet the requestors' requirement, the optimal assignment of physical or virtual resources to incoming requests, and appropriate accounting models. On the consumer side, some of the major issues include the selection of cloud providers depending on their capacity and cost model, and the decision of the type, amount and period of time of resources required for specific workloads.

The users of cloud services are of two major categories: individual clients and organization users. Individual clients would acquire cloud resources for their own specific usage and release the resources after usage. Organization users of clouds would acquire and release resources as results of the aggregated need of the individual clients within their organizations. In other words, organization users are likely to use clouds as providers of adjunct resources such that the organization can be benefit from cloud services in term of elasticity and transference of risk in under-utilization and under-provisioning of resources [10].

With the emerging of cloud computing, many researchers study its applicability to specific applications or workloads. In the view of the authors of [10], five applications in consideration are mobile interaction, parallel batch processing, business analytics, extension of compute-intensive desktop and earthbound applications. The Carmen project in e-Science [29] examines the use of clouds for both data sharing and dynamic compute resources. Deelman, et. al. [17] study the cost and performance of the Montage workflow. Curry et. al. [16] experiment with cloud resources to meet the growing demand for social network applications. With or without data storage, most of these applications and workloads are compute-intensive with keen demand in compute resources. This is strong incentive to continue our focus on compute resources in addition to our original technical interests in supporting parallel scientific batch applications in grids using meta-scheduling.

In the following, we discuss in details the ***consumer***-side management of acquired ***compute*** resources from cloud providers to meet the aggregated workload needs of ***organization users***.



(a) Collaborating meta-schedulers (b) Meta-scheduling of Clouds

Figure 1. Meta-Scheduling Scenarios

2.2 Meta-scheduling of Cloud Resources

We assert that utility and grid computing technologies have much applicability in cloud computing. Utility computing explores the low-touch provisioning technologies for various resources [18] and application services [15]. Much has advanced in the virtualization technologies for provisioning compute resources in recent years. From grid computing, the insights gained in workload management with dynamic resources will be applicable to cloud computing with elastic resources.

Here, we demonstrate our grid meta-scheduling [11] as design to address two particular technical issues that are present in cloud compute resources: how to manage and use the acquire cloud resources and how much acquired resources would be beneficially for specific workloads.

In Figure 1(a), a resource domain is described as a functional or administrative entity of an organization to be managed by a meta-scheduler (MS) with one or more local schedulers (LSs). This aspect of domain intends to reflect the reality of many organizations having multiple job schedulers for different clusters of resources dedicated to individual lines of business operation. Moreover, the aspect of peer-to-peer relation between domains can be viewed as an organization with multiple collaborative institution partners with goals in sharing their resources. In our design, peering relationship can be further defined with agreement on the role of a peer. The role may be a consumer, provider, or both. A consumer will use provider's resources to execute its workloads such as batch jobs. A consumer can request providers to supply their resource information periodically or on request while a provider has no such privilege or necessity in obtaining resource information from the consumer organizations.

Figure 1(b) shows a specific instantiation of the collaborating meta-scheduling architecture for a cloud computing scenario. In this particular case, MS of an enterprise domain will be the consumer of two other cloud domains

with MSs as resource providers. The cloud domain on the right-hand has virtualized resources while the domain on the left-hand has physical compute systems as the physical cloud resources. The rationales for having MSs and LSs to manage the availability of cloud resources to an organization are many. Some of the top reasons include:

- Cloud resources are managed as a group rather than individual resources, which has advantage in management scalability;
- Consumer organization can manage cloud resources by capability/capacity instead of resource details;
- Flexibility in managing the dynamical addition and removal of cloud resources through the provider-side meta-scheduler;
- Flexibility in managing and selecting provider domains;
- Consumer organization can control policies on when, how and who can use cloud resources; for example,
  - Only test batch workload can be routed to the cloud resources;
  - Only jobs shorter than 3 hours will be sent to the cloud resources as the lease is good until then
  - Certain groups of users can use the cloud resources
- Outsourcing workload execution to cloud resources instead of importing resources to enterprise resource domain; this partially answer the security concern.

In order to use meta-schedulers in cloud computing, one important design expansion needed would be the instantiation of MSs or LSs that shall manage acquired cloud resources. There are two approaches: i) statically establishing MSs or LSs as part of persistent setup between consumer organizations and cloud providers; only the connections between the consumer and provider domains are created on demand; ii) dynamically establishing the instances of MSs and LSs as part of the provisioning activities when setting up the cloud services. In either approach, the activities and time required for provisioning are outside the scope of this paper. As the provisioning technologies are developing rapidly nowadays, the magnitude of complexity and time and the associated impact to system performance are interest research topics.

3 Implementation

Experiments are performed using the configuration of Figure 2. There are two resource domains which interface to external job submitters and each other via a meta-scheduler (MS). Internally, a domain consists of a cluster of compute nodes managed by a local scheduler. The resource domain headed by MS-A contains a static set of

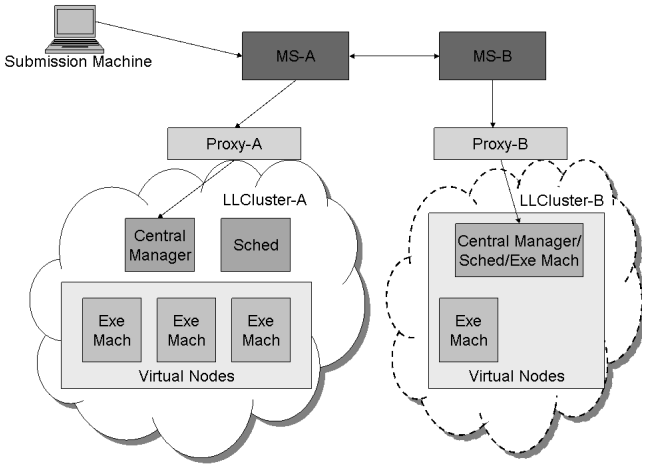


Figure 2. Experiment setup

compute resources and represents the enterprise domain. In our experiments all jobs are initially submitted to this domain. The second domain is the cloud provider domain with a variable pool of resources. The MS of the enterprise domain decides whether to send the jobs to the local scheduler or forward them to the cloud domain as described in 3.2.

3.1 Domain components

The key domain components here are the meta-scheduler (MS) and local-scheduler. The meta-schedulers are based on the commercial product IBM Tivoli Dynamic Workload broker(ITDWB) [6]. ITDWB is a local job scheduler designed with an emphasis on adapting to dynamic changes in the compute node environment. The MS function is added to the ITDWB product code base and provides support for protocols to submit jobs and exchange compute resource information with other MS. The detailed protocols are described elsewhere [12] and not necessary for this discussion. The relevant capability added in MS-ITDWB is the ability to make decisions to forward a job to another MS-ITDWB for scheduling and dispatching, rather than execute the job locally. Forwarding decisions are based on matching job requirements to compute cluster resource and performance information exchanged between MS-ITDWB. Scheduling decisions are not made by the MS, rather the MS chooses to forward a job if based on current resource information it expects the job can be started more quickly at a cloud cluster.

The local scheduler is IBM LoadLeveler [5], an IBM commercial scheduler product for high performance computing (HPC) environment. Because the LoadLeveler product does not support the MS-ITDWB job and resource exchange protocols, a proxy is placed between the MS and local scheduler to mediate the communications. Through the proxy, LoadLeveler sends its compute resources and performance data to MS-ITDWB every 15 seconds. Metrics relevant here include number of available and occupied compute nodes, queue length, and wait time.

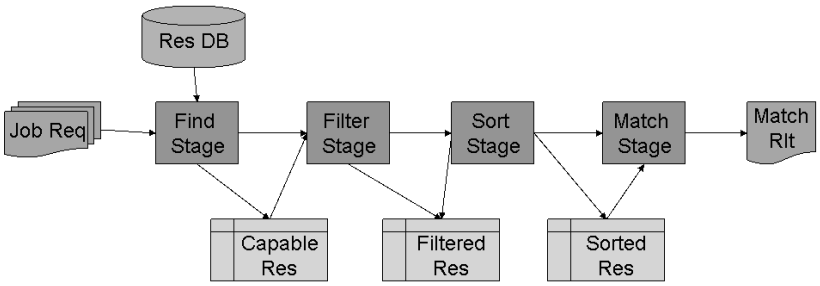


Figure 3. Multistage Matching Algorithm

LoadLeveler uses the backfill scheduling algorithm. Backfill is a well known algorithm that avoids starvation of large jobs while maintaining high system utilization by "backfilling" short jobs on nodes which are idle waiting for sufficient nodes to free up to launch a large job. This interaction of the workload mix of large and small jobs together with the backfill scheduler leads to very interesting results presented in 4.

The LoadLeveler used here has one important extension from the commercial version which is not necessary but greatly facilitates our experiments. It supports a lightweight virtual compute node process. The process reports node resources to the LoadLeveler central manager, initiates jobs, and appears to LoadLeveler as a fully functional unique physical node. Virtual nodes allows us to run many LoadLeveler 'nodes' on a single physical machine. Our experiments typically run up to 256 of these virtual compute nodes on a single large machine. Because our job executable is to sleep for the duration specified in for each job in a trace, lightweight virtualization is a simple and inexpensive method of creating very large clusters.

3.2 Matching Algorithm

Meta-scheduler tries to match the incoming job requests with the resource information it stored and forward the job to the matched resource for execution. In our system, a meta-scheduler can have resource information directly from execution nodes, from local schedulers, or from other meta-schedulers. Thus, the matching is a complex procedure and we adopt a four stages algorithm as shown in Figure 3.

In the first "find" stage, the algorithm finds all the capable resources that meet the job's requirement, whether the resources represent individual resources or a set of resources, whether the resources are local or remote, whether the resources accept parallel jobs, and etc.

In the second "filter" stage, the algorithm filters out the resources that is not qualified for the job requests in consideration. In our system, the filters are implemented as plug ins. Taking MS-A in Figure 2 as an example, it is configured to use following filters in sequence.

Parallel filter: if the job request is a parallel job, the resource should have the capability to deal with it.

Failed resource filter: if the job has been allocated resources before and has failed to finish its execution successfully on those resources, we do not want to try to the same set of resources again.

Visited schedulers filter: if the job request is forwarded by other schedulers, we do not want to forward the job back to the schedulers that it already visited.

Cloud filter: if we know the job is a very short job whose expected execution time is less than 3 seconds [1], we do not want to forward the job to a cloud cluster since the overhead is too big compared to its execution time.

In the third "sort" stage, the algorithm sorts the filtered capable resources. This is the stage when we use the scheduling related information collected by proxy. We compute a priority of each filtered resource and then sort them. The freenode algorithm used in the experiments of this paper first calculate the priority of a resource as $freeNode - procLen$, where freeNode is the number of idle nodes, and procLen is the number of requested processors of the waiting jobs. If the queue length, which is the number of waiting jobs, of a resource is 0, then we adjust its priority to be bigger than the priority of the resources whose queue length is bigger than 0. Then, we sort the resource set first on priority, 2nd on queue length, 3rd on the distance to the meta-scheduler.

In the fourth "match" stage, we simply retrieve the first resource set of the sorted resource list from "sort" stage and return it as the match result.

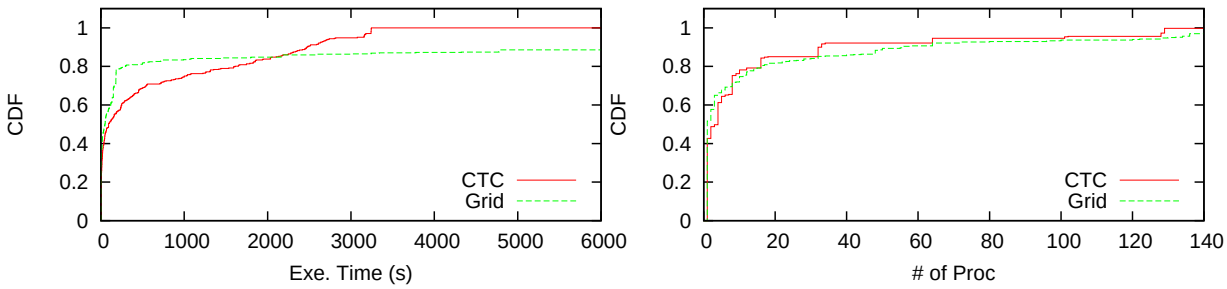
4 Experiment

Our experiments on the experimental platform described in Section 3 is driven by job traces representative of high performance computing workload and a grid workload. After characterizing these workloads, experimental results are presented that differentiate the effectiveness of cloud managed clusters for the workloads.

4.1 Jobs from Traces

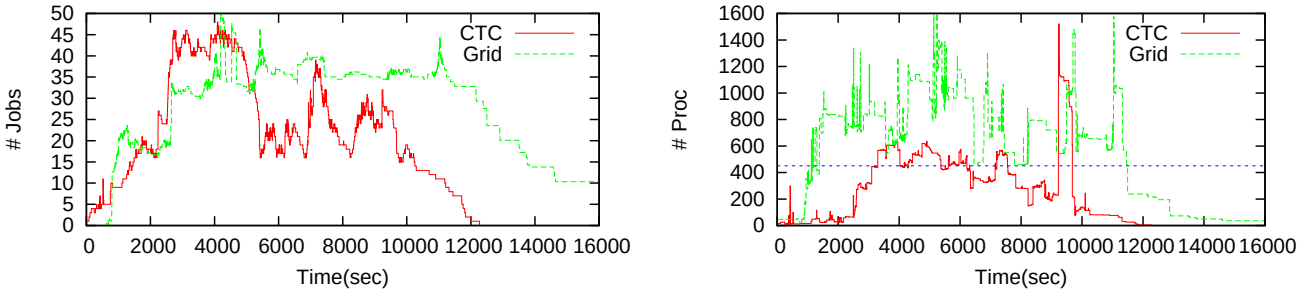
We drive our experiments using jobs extracted from traces. We choose our experiment traces from traces representing real usage scenarios. In this paper, we demonstrate our results with two traces. One trace is from the Parallel Workloads Archive [7] and contains 11 months of execution at the Cornell Theory Center (CTC) on a cluster of about 450 single processor IBM SP2 nodes. The other trace is from the Grid Workloads Archive [3] and records the Grid5000 experiment, which comprises 9 different locations and 15 computing clusters. Both traces are primarily scientific workload and contain parallel jobs.

[1] Scheduler does not exactly the execution time of a job. However, a job can be categorized as a very short job, short job, long job, or very long job.



(a) Execution time (b) Number of requested processors

Figure 4. CDF of execution time and number of requested processors of traces



(a) Number of active jobs (b) Number of requested processors

Figure 5. Trace execution in the ideal environment (zero overhead, unlimited capacity)

Since we conduct real experiments, in order to finish experiments in a reasonable period of time, we have to compress the original traces. Using a heuristic approach [24] that retain the workload pattern of the original trace, we reduce the part of a trace that represents the workload of a whole week to about 3 hours. As a result, we shall submit 408 jobs from reduced CTC trace in 2.8 hours and 533 jobs from reduced Grid trace in 3.1 hours.

To characterize the workload of the selected traces, we plot their respective execution time and compute processor demand in the form of cumulative distribution functions in Figure 4. From Figure 4(a), we can see that the distribution of the execution time of CTC and Grid jobs is different. Grid trace has a larger percentage of short jobs (80%), whose execution time is less than 30 minutes, than CTC trace has(62%). Then, while CTC does not have any job longer than 1 hour, 10% of Grid jobs are very long jobs whose execution time is longer than 3 hours. Combined together, grid jobs are more diversified than CTC jobs in execution time. From Figure 4(b), we see that CTC and Grid traces have a similar distribution of number of requested processors though Grid trace is still more diversified than CTC trace. Grid trace has more single jobs(52%) than CTC trace has(43%) and Grid trace has more big jobs(8%), which requests more than 64 processors, than CTC trace has(6%).

However, analyzing the execution time and the number of requested processors separately can not give us the whole picture of the workload. For example, the sequence of jobs is decisive to the workload patterns. Additional

insight into the traces is provided by computing the resource "demand". A simple analytic evaluation is performed on each trace to compute the number of concurrent jobs and their occupied processors in the system as a function of time. The analysis is done assuming the ideal environment that has unlimited resources and has no overhead. In other words, each job is scheduled immediately as it is submitted; there is no latency and no waiting time for free nodes. The number of active jobs and their requested processors of CTC and Grid traces are shown in Figure 5.

From Figure 5(a), we can see that at any time, there are less than 50 active jobs in the system. However, since around half of the jobs are parallel jobs, the number of active jobs is not very informative about the workload. We should give more attention to the number of requested processors, which is shown in Figure 5(b). From Figure 5(b), we can see that CTC and Grid traces are different. Grid trace has a heavier workload than CTC trace most of the time and the Grid curve has a long tail due to the very long jobs. Both CTC and Grid curves have peaks that are caused by the arrival of big jobs in a short period of time. For example, CTC's peak (> 1400 processors) around 9000 seconds is due to the sequential arrival of 10 big jobs, each of which requests 129 processors, during 5 seconds.

By the curve of requested processors, we can not only see the workload pattern but also roughly predict the performance given a resource size. For example, if we are given a resource set of 450 nodes. We can see from Figure 5(b) that it shall satisfy the resource requirement of CTC trace most of the time and should give us a good performance. However, the same resource size can not satisfy the requirement of Grid most of the time and we may not expect good performance.

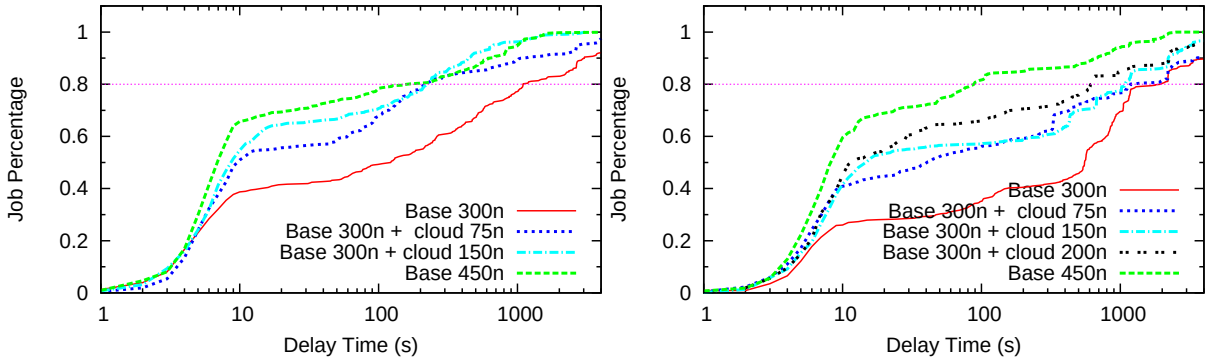
To simplify the experiment, we use sleep jobs in our experiment runs; each job executes exclusively on the allocated nodes; and each node has one processor. Next, we shall use node and processor interchangeably.

4.2 Experimental Results

Now, we present the results of our experiments. We first present the result on a single cluster to gain experience of the system and the overhead. Then, we evaluate the performance improvement by introducing a cloud cluster.

Different from the ideal environment we used to analyze the traces, in a real environment, we can not avoid the latency from different components of the system and the queuing time caused by system load. For example, in our system, the latency accounts for 1) communication latency including the communication time between MS and MS, between MS and LS, and between LS and node; 2) software latency including software computation time, scheduling interval and etc. We combine latency and queuing time together to indicate the system performance. That is, we calculate a delay time as

$$delayTime = finishTime - arrivalTime - executionTime,$$



(a) CTC (b) Grid

Figure 6. Delay time

where arrivalTime is when a job request arrives at the system and finishTime is when the receiver of the job request receives the successful done notification, and executionTime is the real running time of the job.

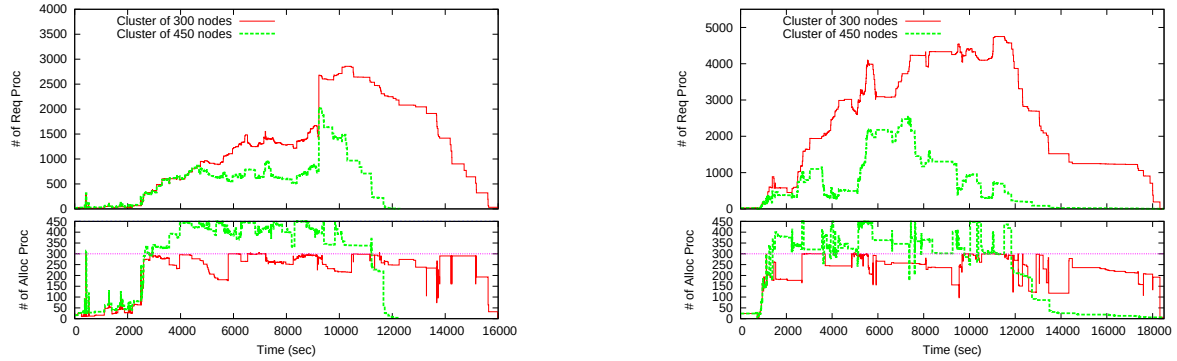
While delay time gives us an user perceived indicator of the system performance, we are also concerned about the system side performance, such as the utilization of resources. Thus, we demonstrate the number of processors occupied by the running jobs and the number of processors requested by all the submitted jobs including both the running jobs and the waiting jobs. The former one demonstrates the resource utilization along the time; the latter one demonstrates the dynamic workload along the time. Combined two together, we can have a better understand of the system and pinpoint the situation that we would like to improve.

4.2.1 Base Cluster

We begin with a single cluster. We choose the number of nodes in the cluster to be 450. This number is close to the original environment, which has 451 processors, where the CTC traces were collected. From Figure 5(b), we know that we have enough nodes most of the time for CTC trace, while we can not satisfy Grid jobs' requirement most of the time. Then, we reduce the cluster size to 300 and compare the results.

Given a delay value, we show the percentage of jobs that can finish with a delay less than or equal to the value. We show the result of CTC trace in Figure 6(a) and that of Grid trace in Figure 6(b). Not surprisingly, using 450 nodes has a better performance than using 300 nodes. Taking 80 percentile delay time as an example, for CTC trace, it is 200 seconds using 450 nodes compared with 1100 seconds using 300 nodes; for Grid trace, it is 90 seconds using 450 nodes compared with 2000 seconds using 300 nodes. Decreasing the resource size has a more severe influence on Grid trace that has a heavier workload. We also list the 50, 80, 90, and 95 percentile delay time in Table 1.

Then, we show the number of allocated and requested processors of jobs in the system along time in Figure 7.



(a) CTC (b) Grid

Figure 7. Allocated and requested processors of active jobs with one cluster

We show the results of CTC trace in Figure 7(a). In both 300-nodes and 450-nodes cases, there is a peak of requested processors around 9000. The bigger the resource size, the faster the system can "digest" the requirement. Another interesting observation of the 300-nodes case is that during the period from 4500 to 6000, though the requirement(> 500) is higher than the capacity (300) all the time, the system does not use its full capacity, but use as low as only 200 nodes. The reason is that big jobs, which are already scheduled to LoadLeveler, block smaller jobs, which comes later. Thus, it causes the low utilization of resources while the demand is high. The same situation happens for other periods too. For example, the period of time around time 2000 for Grid trace in Figure 7(b). In fact, we observe that the situation is more common with Grid, where the workload is heavier.

4.2.2 Cloud Cluster

Cloud is a novel addition to grid systems. In this paper, through a set of experiments, we demonstrate that cloud can be incorporate into grid system using meta-schedulers and improve the performance of the system.

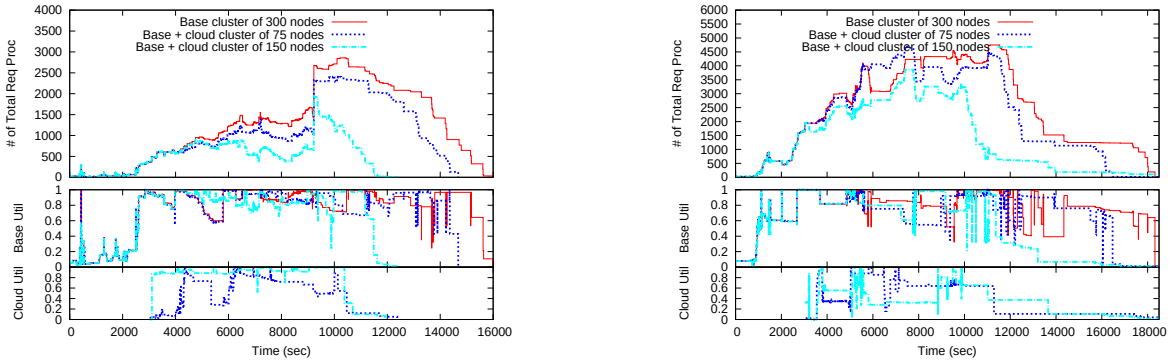
In general, cloud provision is time consuming and depends heavily on the system settings. Thus, we would like to leave cloud provision out of this paper. Instead of providing cloud cluster dynamically, we pre-fix the period of time when a cloud cluster is present and study its influence to the system performance.

We experimentally determine the period of time when to provide cloud cluster. Consider CTC trace. Given the analysis of it in the ideal environment as shown in Figure 5(b), we set the begin time to 3000 since after it the number of requested processors is above 300 most of the time until around 9700. We set the end time to 10000. We can verify our choice of the period by the curve of requested processors of 300-nodes case in Figure 7.

After determining the period of time when to provide cloud cluster, we start the virtual nodes on cloud cluster

		50 percentile	80 percentile	90 percentile	95 percentile
CTC	450 nodes	7s	200s	690s	1000s
	300+150 nodes	9s	220s	450s	700s
	300+75 nodes	10s	220s	1200s	2900s
	300	120s	1100s	3100s	4500s
Grid	450 nodes	8s	90s	690s	1160s
	300+150 nodes	15s	1090s	2200s	3300s
	300+75 nodes	40s	1200s	4000s	5400s
	300	550s	2000s	4200s	6300s

Table 1. Delay Time Percentile



(a) CTC (b) Grid

Figure 8. Allocated and requested processors of active jobs with cloud MB

at the beginning time. The resource information is collected by proxy and is reported to MS-B and then to MS-A. Thus, MS-A can match the cloud resource to job and forward job to MS-B for execution. At the ending time, we "drain" the virtual nodes. This has two effects: the jobs already allocated can continue to finish; the nodes number is reported as 0. Thus, MS-A stops to forward job to the cloud cluster. Those jobs that do not execute yet shall be reported as FAILED to MS-A and be re-scheduled by MS-A to the base cluster.

The delay results are shown in Figure 6 and Table 1. For CTC trace, compared with no-cloud case, we decrease 80 percentile delay time from 1100 seconds to 220 seconds by adding a 75-nodes cloud, which is very close to 450-nodes cluster's performance. Adding a bigger size cloud such as 150 nodes does not decrease 80 percentile value. However, an interesting observation is that with 450 nodes in total, adding a 150-nodes cluster outperforms using one single 450-nodes cluster.

We try to gain deeper insights of the delay time results by using the data shown in Figure 8. From Figure 8(a), we can see that while keeping a high "demand", adding a cloud cluster will not decrease the utilization of the base cluster at a specific period of time. Consider the time period from 5000 to 6000 seconds. Both no-cloud case and

	CTC		Grid		
Cloud Size	75	150	75	150	200
Avg Util	0.56	0.57	0.40	0.38	0.52

Table 2. Average Utilization of the Cloud Cluster

75-nodes-cloud case has a low utilization at the base cluster. This is because the big jobs, which block smaller jobs, request more processors than 75 nodes and can not be scheduled by the cloud. And with 150-nodes-cloud case, the big jobs can be sent to the cloud and does not block at the base cluster. Thus, we are able to have a high utilization during the period. This also explains why the case with 300-nodes cluster and 150-nodes-cloud outperforms the case of one single 450-nodes case when the total number of nodes are the same.

Besides the delay time that can be perceived by the user, we also look at the utilization at the cloud cluster. The results are shown in Table 2. Note that the main goal of our scheduling algorithm is to improve the user-perceived delay time. For CTC trace, the bigger 150-nodes cloud has a similar utilization of the 75-nodes cloud. However, with a bigger cloud size, this means 150-nodes cloud has a larger resource idle time than 75-nodes cloud has. Thus, if the cloud is charged by time instead of by usage, it maybe reasonable to choose 75-nodes over 150-nodes cloud considering their similar 80-percentile delay time.

The Grid results are different from that of CTC. The delay data with different settings is shown in Figure 6(b). We can see that adding cloud cluster does not have the big improvement of delay time as experienced by CTC trace. The 150-nodes case does not have a obvious improvement over 75-nodes case; and it is even worse than 75-nodes case for some data. With 450 nodes in total, it is obvious that we should choose one 450-nodes cluster over a 300-nodes cluster plus a 150-nodes cloud. Even with a bigger total number of nodes, we can see that combining a 300-nodes cluster and a 200-nodes cloud cluster still performs worse as measured in delay time than the 450-nodes single cluster case.

The system execution of grid along time is demonstrated in Figure 8(b). Though a cloud helps with the performance by finishing the total workload earlier, the delay time improvement is not as obvious as the CTC trace, even with a bigger size of cloud. Check the cloud utilization of running Grid trace in Table 2. Though Grid workload is much heavier than CTC workload, the cloud utilization is smaller than CTC. Also, a larger cloud of 150 nodes has a smaller utilization than that of 75 nodes. This means that Grid trace can not use cloud efficiently. Considering its heavy workload, it may be a better choice to increase the size of the base cluster instead of adding clouds to improve its performance.

5 Related Work

In the introduction section, we cite the IT vendors that currently position their service offerings in cloud computing [1, 2, 4]. The definitions of cloud computing and related technologies are included in various published papers [8, 22, 13, 31, 10, 19, 23, 8, 27]. Moreover, there are large collaborative projects being built based on the cloud computing paradigms such as the Reservoir project [26], and Virtual Computing Laboratory (VCL)[28]. New applications are being experimented to take advantages of the cloud computing such as [29, 17].

To study the functions of resource management from the consumer perspectives, the Group Manager in the project Eucalyptus [25] provides similar functions as our meta-scheduler. The Reservoir project [26] also has the objective to address the federation of multiple cloud providers. There are similarities in functions between our meta-scheduler and their Virtual Execution Environment Manager (VEEM). However, our meta-scheduler is more batch processing oriented while their VEEM support more generic functions.

To understand the performance of workloads using the cloud resources, Yigitbasi et. al. [30] proposed a framework leveraging and extending BrenchMark. Instead of being a general framework, our meta-scheduling framework is built for batch processing.

In term of workload evaluation, [21] compare the performance of workload on platform Grid5000 and EGEE. They do not use jobs that require multiply processors. Also, our evaluation focuses on the use of cloud resources.

6 Conclusion and Future Work

Many computer analysts claim that cloud computing is the realization of utility computing, a usage model for grid technologies with a service oriented abstraction, and a transformational paradigm for the IT industry. All of us also understand that there are many technology challenges ahead. The effective management and efficient uses of cloud resources is just one of many research topics. In this paper, we purpose the use of meta-scheduling as a technology to manage and use cloud resources to meet the aggregated service need of organization or enterprise users. We show our design in managing the cloud resources as a collection instead of individual entities, and delegating the usage of resources for workload execution to meta-schedulers. We evaluate our design using two workloads from real traces and collect performance data in job execution delay time and system utilization. Not surprisingly, understanding the workload characteristics is important for making decision on when and how capacity of cloud is effective. An interesting result in our experiement is that the interaction between grid workload, backfill scheduling, and job routing from an enterprise cluster to a cloud can achieve better performance than a dedicated cluster of the combined size of the enterprise and cloud systems.

There are many directions to expand our work, including dynamic selection of cloud providers, use of cost model

in making selection in providers and pool sizes, incorporating impact of provisioning complexity and time, etc.

References

[1] Amazon elastic compute cloud. http://aws.amazon.com/ec2/.

[2] Google app engine. http://code.google.com/appengine.

[3] The grid workloads archive. http://gwa.ewi.tudelft.nl/pmwiki/.

[4] Ibm cloud services initiative. http://www-03.ibm.com/press/us/en/pressrelease/25341.wss.

[5] IBM tivoli workload scheduler loadleveler. http://publib.boulder.ibm.com/-infocenter/clresctr/vxrx/index.jsp.

[6] Ibm's tivoli dynamic workload broker. http://www-306.ibm.com/software/tivoli/products/dynamic-workload-broker/index.html.

[7] Parallel workload archive. http://www.cs.huji.ac.il/labs/paralle/workload/models.html.

[8] Twenty-one experts define cloud computing. http://cloudcomputing.sys-con.com/node/612375, January 2009.

[9] Appleby, K., Fakhouri, S., Fong, L., Goldszmidt, G., Kalantar, M., Krishnakumar, S., Pazel, D. P., Pershing, J., and Rochwerger, B. Oceano - sla based management of a computing utility. In *In Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management* (2001), pp. 855–868.

[10] Armbrush, M., Fox, A., and et. al. Above the clouds: A berkeley view of cloud computing. http://http://www.eecs.berkeley.edu/pubs/techrpts/2009/eecs-2009-28.pdf.

[11] Bobroff, N., Dasgupta, G., Fong, L., Liu, Y., Viswanathan, B., Benedetti, F., and Wagner, J. A distributed job scheduling and flow management system. *SIGOPS Oper. Syst. Rev. 42*, 1 (2008), 63–70.

[12] Bobroff, N., Fong, L., Rodero, I., Sadjadi, S. M., and Villegas, D. Enabling interoperability among meta-schedulers. In *IEEE 8th International Symposium on Cluster Computing and the Grid (ccGrid)* (May 2008), IEEE Computer Society.

[13] Buyya, R., Yeo, C. S., and Venugopal, S. Market-oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC 2008)* (Dalian, China, Sept. 2008), IEEE Computer Society: Los Alamitos, CA, USA, pp. 5–13.

[14] Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., and Doyle, R. P. Managing energy and server resources in hosting centers. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2001), ACM, pp. 103–116.

[15] Clemm, A., Shen, F., and Lee, V. Generic provisioning of heterogeneous services: a close encounter with service profiles. *Comput. Netw. 43*, 1 (2003), 43–57.

[16] Curry, R., Kiddle, C., Markatchev, N., Simmonds, R., Tan, T., and Arlitt, M. Facebook meets the virtualized enterprise. In *12thIEEE International EDOC Conference (EDOC08)* (September 2008).

[17] Deelman, E., Singh, G., Livny, M., Berriman, B., and Good, J. The cost of doing science on the cloud: the montage example. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (Piscataway, NJ, USA, 2008), IEEE Press, pp. 1–12.

[18] Fong, L. L., Kalantar, M. H., Pazel, D. P., Goldszmidt, G. S., Fakhouri, S. A., and Krishnakumar, S. M. Dynamic resource management in an eutility. In *NOMS* (2002), pp. 727–740.

[19] Foster, I., Zhao, Y., Raicu, I., and Lu, S. Cloud computing and grid computing 360-degree compared.

[20] Foster, I. T., Kesselman, C., and Tuecke, S. The anatomy of the grid - enabling scalable virtual organizations. *CoRR cs.AR/0103025* (2001).

[21] Glatard, T., Montagnat, J., and Cnrs, I. An experimental comparison of grid5000 clusters and the egee grid. In *Workshop on Grid* (2006).

[22] Hayes, B. Cloud computing. *Communication of ACM 51*, 7 (2008), 9–11.

[23] Jha, S., Merzky, A., and Fox, G. Using clouds to provide grids higher-levels of abstraction and explicit support for usage modes http://www.ogf.org/ogf-special-issue/cloud-grid-saga.pdf.

[24] Liu, Y., Villegas, D., Bobroff, N., Fong, L., Rodero, I., Sadjadi, S. M., and Seelam, S. An experimental system for grid meta-broker evaluation. In *Workshop on Large-Scale System and Application Performance (LSAP)* (2009).

[25] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. The eucalyptus open-source cloud-computing system.

[26] Rochwerger, B., Breitgand, D., and et. al. The Reservoir model and architecture for open federated cloud computing. IBM System Journal. Submitted for publication. http://www.cs.umu.se/ elmroth/papers/reservoir-ibmsj08.pdf.

[27] Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev. 39*, 1 (2009), 50–55.

[28] Vouk, M. A. Cloud computing - issues, research and implementations. *Journal of COmputing and Information Technology 16*, 4 (2008).

[29] Watson, P., Lord, P., Gibson, F., Periorellis, P., and Pitsilis, G. Cloud computing for e-science with carmen. In *IBERGRID. 2nd Iberian Grid Infrastructure Conference Proceedings* (May 12-14 2008).

[30] Yigitbasi, N., Iosup, A., Epema, D., and Ostermann, S. C-meter: A framework for performance analysis of computing clouds. to appear. In *International Workshop in Cloud Computing in conjunction with CCGRID* (May 2009).

[31] Youseff, L., Butrico, M., and Da Silva, D. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop(GCE)* (November 2008).