In Cloud, Can Scientific Communities Benefit from the Economies of Scale?

Lei Wang, Jianfeng Zhan, Weisong Shi, *Senior Member, IEEE*, Yi Liang

Abstract—The basic idea behind cloud computing is that resource providers offer elastic resources to end users. In this paper, we intend to answer one key question to the success of cloud computing: in cloud, can small-to-medium scale scientific communities benefit from the economies of scale? Our research contributions are three-fold: first, we propose an innovative *public cloud* usage model for small-to-medium scale scientific communities to utilize elastic resources on a public cloud site while *maintaining their flexible system controls*, i.e., create, activate, suspend, resume, deactivate, and destroy their high-level management entities—service management layers without knowing the details of management. Second, we design and implement an innovative system—*DawningCloud*, at the core of which are lightweight service management layers running on top of a common management service framework. The common management service framework of DawningCloud not only facilitates building lightweight service management layers for heterogeneous workloads, but also makes their management tasks simple. Third, we evaluate the systems comprehensively using both emulation and real experiments. We found that for four traces of two typical scientific workloads: high throughput computing (HTC) and many task computing (MTC), DawningCloud saves the resource consumption maximally by 59.5% and 72.6% for HTC and MTC service providers, respectively, and saves the total resource consumption maximally by 54% for the resource provider with respect to the previous two public cloud solutions. To this end, we conclude that small-to-medium scale scientific communities indeed can benefit from the economies of scale of public clouds with the support of the enabling system.

Index Terms—Cloud, Scientific Communities, Economies of Scale, Many-Task Computing, and High Throughput Computing.

1 Introduction

Cloud computing has attracted a lot of attention in the last few years [2] [4] [13] [20] [28]. From the perspective of resource providers, such as Amazon and Google Apps, cloud computing introduces a new computing paradigm: *infrastructure-as-a-service (IaaS)* or *platform-as-a-service (PaaS)*. Usually, clouds are classified into three categories [4] [28]: *public clouds* offer a publicly accessible remote interface for masses' creating and managing resources, e.g., virtual machine instances; *private clouds* give local users a flexible and agile private infrastructure to manage workloads on their own cloud sites; *a hybrid cloud model* enables supplementing local infrastructures with the computing capacity from an external public cloud.

In scientific communities, more and more research groups show great interests in utilizing open source cloud computing tools to build private clouds [19] [28] [34], or proposing hybrid cloud models [22] [25] [26] to augment their local computing resources with external public clouds. In this paper, we take a different perspective to focus on public clouds, and intend to answer one key question: *in public cloud, can small-to-medium scale scientific communities benefit from the economies of scale?* "Economies of scale" refers to reductions in unit cost as the size of a facility increases [40]. If the answer is yes, we can provide an optional cloud solution for scientific communities, which is complementary to state-of-the-art and state-of-the-practice private or hybrid cloud solutions, and hence many small-to-medium scale scientific computing organizations can benefit from public clouds. According to [36], small-to-medium size dedicated clusters constitute a substantial portion (more than 50%) of the total number of servers installed in the US.

Answering this question has two major challenges: first, cloud research communities need to propose innovative cloud usage models, and build enabling systems that support scientific communities to benefit from the economies of scale of public clouds; second, we need to present an innovative evaluation methodology to guide the experiments design to answer our concerned question, since the trace data of consolidating several scientific communities' workloads are not publicly available; moreover, large-scale experiments are forbiddingly costly.

Previous efforts fail to resolve the above issue in several ways. First, no one answers this question from the perspective of scientific communities. Armbrust *et al.* [2] in theory show Web service workloads can benefit from the economies of scale on a cloud site. However scientific workloads are distinguishedly different from Web services in terms of workload characteristics, resource consumption and performance goals. Second, in scientific communities, most of work proposes private or hybrid cloud solutions with focuses on managing virtual infrastructures [28], consolidating heterogeneous workloads [11], creating and managing runtime environ-

- *Lei Wang and Jianfeng Zhan are with Institute of Computing Technology, Chinese Academy of Sciences. Weisong Shi is with the Department of Computer Science, Wayne State University. Yi Liang is with the Department of Computer Science, Beijing University of Technology. Jianfeng Zhan is the corresponding author, whose E-mail is jfzhan@ncic.ac.cn.*

Accepted by IEEE Transactions on Parallel and Distributed Systems, March 2011.

ments [21], sharing commodity clusters among different computing frameworks [41], and facilitating building diverse data-parallel programming models [35], respectively. Wentzlaff *et al.* propose a single system image operating system across both multicore and Infrastructure as a IaaS cloud systems [47]. However, these research efforts can not be directly used to provide platforms for answering our question, since concerns of private or hybrid clouds mainly revolve around activities (or workloads) of a single research institute or group.

Third, state-of-the-art and state-of-the-practice public cloud solutions [3] [5] provide limited supports for scientific communities as service providers in terms of system controls[1]. For example, Deelman *et al.* [5] propose that each staff of an organization (as end users) directly leases virtual machine resources from a public cloud provider—-EC2 in a specified period for running applications. Evangelinos *et al.* [3] propose that an organization as a whole rents resources with the fixed size from EC2 to create a virtual cluster system that is deployed with a queuing system, like OpenPBS. In the rest of this paper, we call these two models *Deelman's model* and *Evangelinos's model*, respectively. We also call two systems incarnating Deelman's model and Evangelinos's model *Deelman's system* and *Evangelinos's system*, respectively. In the context of public clouds, if the backbone system can not provide enhanced supports for service providers in terms of system controls, the paradigm changes from dedicated systems to public clouds will not go smoothly, since a dedicated system is definitely worthwhile [33] as such a system is under the complete control of the principal investigators.

On the Dawning 5000 cluster system, ranked as top 10 of Top 500 super computers in November 2008 (http://www.top500.org/lists/2008/11), we design and implement an innovative system: DawningCloud, which provides the enabling platform for answering our concerned issue. We take a bottom-up approach to building DawningCloud, and present a layered architecture: the lower one is the *common management service framework* (in short, *CSF*) for the resource provider, and the upper one is *a lightweight service management layer that is responsible for managing resources and workloads*, which we call *thin runtime environment software* (in short, *TRE*) in this paper. CSF facilitates building TRE for heterogeneous workloads, and we have built two TRE for two typical scientific workloads: *high throughput computing* (*HTC*), and *many task computing* (*MTC*) [1]. We are also integrating other data-parallel programming models built in our previous work [35], including MapReduce [48], Dryad-like data flow [49], and All-Pairs [50].

When a resource provider adopts DawningCloud, the CSF is predeployed and running on a cloud site before *any service providers' workloads are consolidated*. On the behalf of each service provider, a high-level management entity—lightweight service management layer (or TRE) is created on demand with the support of the CSF. At the contract period, DawningCloud allows each service provider to flexibly control its TRE, i.e., create, destroy, activate, deactivate, suspend, and resume a TRE without knowing the details of management. For two typical scientific workloads: HTC and MTC, we propose an emulation methodology to conduct a comprehensive evaluation of DawningCloud, and two previous public cloud solutions: *Deelman's system* and *Evangelinos's system*. Our emulated systems are based on an *UltraSim* emulation framework, which we will release as an open source code. With respect to the CloudSim system [43], UltraSim has the following differences: (a): most of modules run real codes, and communicate with each other through real interconnections; (b): it can simulate the separation of concerns between the resource provider and the service providers; (c): it can replay workload traces. Meanwhile, we deploy real systems to evaluate the accuracies of the emulated systems.

The contributions of our work are three-fold: First, we propose an innovative cloud usage model, called an enhanced scientific public cloud model (ESP), for small-to-medium scale scientific communities to utilize elastic resources on a public cloud site while maintaining their flexible system controls. Second, on a basis of the ESP model, we design and implement an innovative DawningCloud system, at the core of which are lightweight service management layers running on top of the CSF. Third, for four traces of HTC and MTC workloads, our experiments show that: a) in comparison with Deelman's system, DawningCloud saves the resource consumption maximally by 59.5% (HTC) and 72.6% (MTC) for the service providers, and saves the total resource consumption by 54% for the resource provider; b) in comparison with Evangelinos's system and the dedicated cluster system, DawningCloud saves the resource consumption maximally by 25.6% (HTC) and 67.5% (MTC) for the service providers, and saves the total resource consumption by 29.5% for the resource provider.

The organization of this paper is as follows: Section 2 presents the proposed ESP model; Section 3 presents the DawningCloud design and implementation; Section 4 proposes the evaluation methodologies, and answers our concerned question in experiments; Section 5 summarizes the related work; Section 6 concludes the paper.

2 The Enhanced Scientific Public Cloud (ESP) Model

We propose the ESP model for small-to-medium scientific communities as service providers to utilize elastic resources on a public cloud site, while maintaining their flexible system controls.

With respect to our previous work [11] and the Reservoir project [13] [18] [28] [39], the distinguished dif-

1. We give some examples of system controls. When a university laboratory in summer vacation stops the computing service in a public cloud, it does not want to migrate user data and programs; when the lab resumes work, it hopes clicking a button makes the stopped system up again.

ferences of our ESP model are three-fold. First, in our model, developing a new service management layer (TRE) for another different workload is lightweight, since many common functions, e.g., the monitors, which are responsible for monitoring and reporting resources status, are required by each service management layer, have been implemented in the common service management framework (CSF). Moreover, we can implement functions for fault-tolerance and scalability support in the CSF according to our past experience [30], which makes developing large-scale service management layers simpler. Second, the service management tasks become simpler, e.g., for a TRE, we only need to deploy fewer modules, since other ones are delegated to the CSF, which is predeployed on the cloud site; in addition, creating a TRE on demand is lightweight, since the CSF is ready and running before any TRE is created. Third, our model provides flexible system controls, and a service provider can activate, safe-deactivate, deactivate, suspend, and resume TRE at its own need. Instead, the recent work in [39] mainly focuses on how to enable automatic deployment and scaling for service managements.

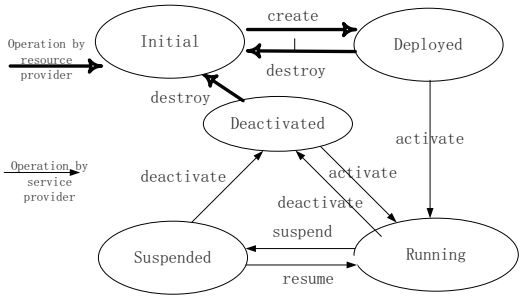


Fig. 1: The lifecycles of a high-level management entity: TRE.

Fig.1 shows the major control operations and the state transition diagram of a TRE. As a high-level management entity, a TRE has five different states: *initial*, *deployed*, *running*, *deactivated*, and *suspended*. The state changes are triggered by five control operations: creating and destroying operations are only performed by the resource provider, while activating, deactivating, suspending, and resuming operations are performed by each service provider.

The *initial* state indicates that the service provider and the resource provider are in the process of planning the TRE. At the request of a service provider, the resource provider can *create* a TRE. The creating operation turns the state of TRE from *initial* to *deployed*. The *deployed* state indicates that the TRE is configured and deployed. When the service provider *activates* the TRE that is *deployed*, the latter turns to the *running* state. The *running* state has two-fold implications: resources have been provisioned to the TRE, and the TRE is providing services to end users. When the TRE is providing services, each affiliated end user uses its accounts to submit and manage jobs. At the same time, a TRE can automatically negotiate resources with the proxy of a resource provider to resize resources by leasing more resources or releasing idle resources according to current workload status.

After the TRE is created, the service provider can perform control operations under different conditions.

- When a university laboratory is in the summer vacation and no one will submit jobs, it can *deactivate* the TRE. A *deactivation operation* has two-fold effects: the running jobs will be killed; the user data and programs will be *permanently* kept in the home directories unless the service provider explicitly destroys its TRE, which brings the benefit of flexible system controls to the service provider. The *deactivated* TRE only consumes more storage resources utilized by user data and programs than the *deployed* TRE. When a TRE is deployed, not running, it has no user data and programs.
- When the university laboratory resumes the work after the summer vacation, it can activate the TRE again, and then the TRE is running. With the help of the checkpointing facility, the killed jobs can be resumed.
- When many users submit too many jobs, the service provider can *suspend* the TRE, which turns the TRE into a *suspended* state. The suspended state indicates that the TRE will reject the submissions of new jobs, however it will finish the submitted jobs. Of course, the service provider can *resume* the suspended TRE to the running state.
- The model also has a *safe-deactivation* operation, which will firstly *suspends* the TRE and then deactivates the TRE after confirming that the submitted jobs have finished up. The safe-deactivation operation can prevent the system from killing the unfinished jobs.

When both the service provider and the resource provider agree to stop the negotiation, the TRE can be *destroyed*. Please note that the TRE only can be destroy on condition that it is either *deployed* or *deactivated*. When a service provider wants to destroy its TRE, it will inform its affiliated end users to backup data. Each end user can backup its data to storage servers provided by a resource provider. And then a service provider will destroy accounts of each end user in the TRE. After a service provider confirms that the TRE is ready for destroying, the resource provider will destroy the specified TRE, including user programs and data.

The differences of the ESP model from other models [3] [4] [5] [22] [25] [26] can be found at Appendix A.

3 DawningCloud Design and Implementation

Based on our previous work [11] [30] [31], we design and implement an enabling system, *DawningCloud*. In the rest of this section, we introduce the DawningCloud architecture, the components of the CSF, TRE for two typical

scientific workloads, the basic management mechanisms of TRE, and the resource management and provisioning policies for MTC or HTC workloads.

3.1 DawningCloud Architecture

We present a layered architecture for DawningCloud: the lower one is the CSF for the resource provider, and the upper one is the TRE. Fig.2a and Fig.2b show two architectural differences between DawningCloud and the peer of the Reservoir project as follows:

First, with respect to Reservoir, the CSF of DawningCloud facilitates building lightweight service management layers for heterogeneous workloads. We take a bottom-up approach to building DawningCloud. The common sets of functions for different runtime environment software are delegated to the CSF. CSF facilitates building thin service management layers—TRE for heterogeneous workloads, and a TRE only implements core functions for a specific workload. Instead, the Reservoir project integrates software packages from different partners, including Haizea—a resource lease manager, OpenNebula—a virtual infrastructure manager, and Claudia—a service management layer.

Second, the CSF is running on the cloud site before any TRE is created, and the CSF enables a service provider to flexibly control its TRE, i.e., create, activate, deactivate, suspend, resume, and destroy a TRE. Moreover, different from Reservoir, the implementation of TRE is not bound to virtual machines, e.g., XEN or VMware. Though there are various research efforts to create efficient mechanisms, such as bypass paths, to enhance the I/O performance in virtualized environments [44] [45], virtual machine technologies still bring high overheads to some HPC applications [44] [45]. Taking into account that case, DawningCloud supports both physical and virtual resources provisioning.

3.2 The Main Components of CSF

The major functions of the CSF are responsible for creating, destroying TRE, and provisioning resources to TRE in terms of nodes or virtual machines. The main services of the CSF are as follows:

- The resource provision service is responsible for providing resources to different TRE.
- The lifecycle management service is responsible for managing lifecycles of TRE.
- The deployment services are a collection of services for deploying and booting the operating system, the common service framework and TRE. The major deployment services include DHCP, TFTP, FTP, and SystemImager (http://wiki.systemimager.org). Integrating state-of-the-practice operating system deployment tool— SystemImager, the deployment services can automate operating system updates or even changes according to users' personalized requirements.
- The virtual machine provision service is responsible for managing lifecycles of virtual machines, i.e., XEN.
- The agent is responsible for downloading required software packages, starting or stopping service daemons.
- The monitor. There are two types of monitors: resource monitor and application monitor. *The resource monitor* on each node monitors usages of physical resources, e.g. CPU, memory, swap, disk I/O and network I/O; *The application monitor* monitors application status.
- The configuration service is responsible for managing, updating, and storing the cloud-wide configuration information.

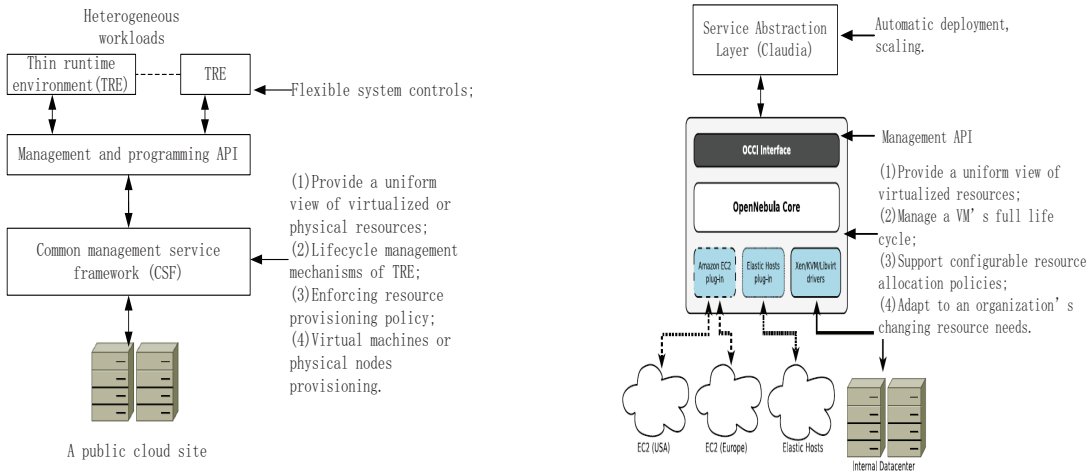
3.3 TRE for Two Typical Scientific Workloads: HTC and MTC

In scientific communities, there are two typical workloads: high throughput computing (HTC) delivers large amounts of processing capacity over long period of time [1], and many task computing (MTC) delivers much large numbers of computing resources over a short period of time to accomplish many computational tasks [1]. With respect to web service workloads, scientific workloads have three distinguished differences:

1) Workload characteristics are different. Parallel batch job workloads are composed of a series of submitted jobs, and each job is a parallel or serial application. while Web service workloads are often composed of a series of requests.
2) Resource consumptions are different. Running a parallel application needs a group of exclusive resources. While for Web services, requests will be serviced simultaneously and interleavedly through multiplex use of resources.
3) Performance goals are different. From perspectives of end users, for parallel batch jobs, in general submitted jobs can be queued when resources are not available. However, for Web services like Web servers or search engines, each individual request needs an immediate response.

In this section, we present two types of TRE for HTC and MTC, respectively. In the design, we distinguish three requirement differences between MTC and HTC TRE as follows:

1) Usage scenes: HTC TRE is designed for running parallel/sequential batch jobs; MTC TRE is designed for running scientific workflows, like Montage workflow [1].
2) Application characteristics: MTC applications [1] can be decomposed to a set of small jobs with dependencies, whose running time is short; while batch jobs in HTC are independent and running times of jobs are varying.
3) Evaluation metrics: A HTC service provider concerns the throughput in terms of the number of



(a) DawningCloud architecture.

(b) Reservoir architecture [28] [39].

Fig. 2: The architectural differences between DawningCloud and Reservoir.

completed jobs over a long period of time; while a MTC service provider concerns the throughput over a short period of time.

In DawningCloud, on a basis of the CSF, we implement two types of TRE: *MTC TRE* and *HTC TRE*. In a HTC TRE, we only need to implement three services: *the HTC scheduler*, *the HTC server*, and *the HTC web portal*. The HTC scheduler is responsible for scheduling users' jobs through a *scheduling policy*. The HTC server is responsible for dealing with users' requests, managing resources, loading jobs. The HTC web portal is a GUI, through which end users submit and monitor HTC applications.

In MTC TRE, we implement four services: *the MTC scheduler*, *the MTC server*, *the trigger monitor*, and *the MTC web portal*. The function of the MTC scheduler is similar to that of the HTC scheduler. Different from the HTC server, the MTC server needs to parse a workflow description model, which the user inputs on the MTC web portal, and then submit a set of jobs/tasks with dependencies to the MTC scheduler for scheduling. Besides, a new service—the trigger monitor, is responsible for monitoring trigger conditions of a workflow, such as database's record or file changes, and notifying changes to the MTC server to drive running of jobs in different stages of a workflow. The MTC web portal is also much more complex than that of HTC, since it needs to provide a visual editing tool for end users to edit different workflows. Fig.3 shows a typical DawningCloud system, of which a MTC TRE and a HTC TRE reuse the CSF.

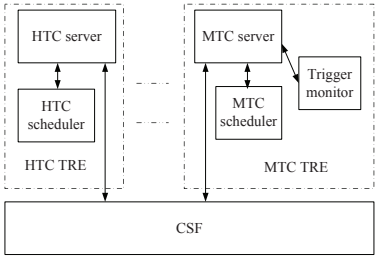


Fig. 3: MTC and HTC TRE on the basis of the CSF.

3.4 Basic Management Mechanisms

Automatic deployment: We take a two-phase solution to automatic deployment: firstly automatically deploy the CSF, and then create a TRE on demand on a basis of the CSF. Based on our previous experience [32], we propose an agent-based solution to self-configuration, and present a role-based self-deployment mechanism for the CSF. The basic idea is as follows [32]:

Deployed on each nodes or the Domain0 of a virtual machine, e.g., XEN, the agent is responsible for detecting the resources in the local environment, including hardware, host operating system, and network configuration. Then the agent will report the information to the configuration service. After a node joins the cloud site, its corresponding agent will accept a role assignment, which indicates what services should be running in this node, from the decision-making module. According to its role, the agent will download a service description file and corresponding program packages from the deployment service. Then the agent will install programs according to the service description file, the detail of which can be found at our previous work [32].

We also propose a booting protocol that is aware of the dependencies of each service in the CSF, which turns the CSF into a self-healing one step by step. Our self-healing mechanism is based on a scalable group membership mechanism. The details can be found in [32] [30] [46]. Please note that services in the CSF binds well-know access addresses, and we use virtual IP (http://en.wikipedia.org/wiki/Virtual_IP_address) to guarantee that the services can be accessed though they may be migrated to other nodes in case of node failures.

Creating TRE: A service provider uses the web portal of the CSF to specify its requirement for TRE, including types of workloads: MTC or HTC, the size of resources, types of operating system, and then requests the resource provider for creating the customized TRE.

If the lifecycle management service validates the requesting information is valid, it marks the state of the new TRE as *initial*. To create a TRE, the web portal of the CSF sends the requesting information to the lifecycle management service. The lifecycle management service sends the message of deploying TRE to the agents on the related nodes, which request the deployment service to download the required software packages of the TRE. After the new TRE is deployed, the lifecycle management service marks its state as *deployed*.

Activating TRE: When the service provider activates its own TRE, the lifecycle management service sends the configuration information of the new TRE to the resource provision service. The lifecycle management service sends the message to the agents to start the components of the new TRE. When the HTC or MTC server is started, the command parameters will tell it what configuration parameters should be read. Then the lifecycle management service marks the state of the new TRE as *running*. The new TRE begins providing the service to end users. Each end user uses the web portal to submit its applications. To save space, destroying, deactivating, suspending, resuming, and safe-deactivating processes are omitted.

Dynamic resource negotiation mechanism: A service provider specifies its requirement for resource management in *a resource management policy*. A resource management policy defines the behavior specification of *the HTC or MTC server*, which is one of the modules of a MTC TRE or HTC TRE, in that the server resizes resources to what an extent according to what criterion. According to a resource management policy, the MTC or HTC server decides whether and to what an extent resizes resources according to the current workload status, and then sends requests of obtaining or releasing resources to *the resource provision service*, which is responsible for provisioning resources to different TRE.

A resource provider specifies its requirement for resource provisioning in *a resource provision policy*, which determines when the resource provision service provisions how many resources to different TRE in what priority. According to a resource provision policy, the resource provision service decides to assign or reclaim how many resources to or from a TRE.

A setup policy determines when and how to do the setup work, such as wiping off the operating system or doing nothing. For each time of node assignment or reclaiming, a setup policy is triggered, and *the lifecycle management service*, which is responsible for managing lifecycles of TRE, is in charge of performing the setup work. Fig.4 shows the dynamic resource negotiation mechanism in DawningCloud.

Because of the page limit, the more detail of DawningCloud can be found in our publicly available technical report [42].

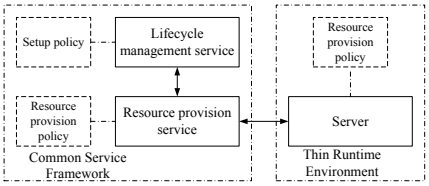


Fig. 4: The dynamic resource negotiation mechanism.

3.5 Resource Management and Provisioning Policies

In DawningCloud, we distinguish two types of resources provisioned to a TRE: *initial resources* and *dynamic resources*. Once allocated to a TRE, initial resources will not be reclaimed by the resource provider until the TRE is destroyed. On the contrary, dynamic resources assigned to a TRE may be reclaimed by the resource provider.

In DawningCloud, a service provider and a resource provider needs to set four parameters: a) *the size of initial resources*; b) *the time unit of leasing resources*. *A lease term of dynamic resources* must be the time unit of leasing resources times an integer. For example, in EC2, the time unit of leasing resources is one hour; c) *the checking resource cycle*. It is a periodical timer that the HTC or MTC server checks jobs in the queue; d) *the threshold ratio of obtaining dynamic resources*.

We propose a resource management policy for a HTC or MTC service provider as follows:

1) At the startup of a TRE, a service provider will request initial resources with the specified size.

2) We define *the ratio of obtaining dynamic resources* as the ratio of *the accumulated resource demands of all jobs in the queue* to *the current resources owned by a TRE*. The HTC or MTC server scans jobs in queues per checking resource cycle. If the ratio of obtaining dynamic resources exceeds the threshold ratio, or *the ratio of the resource demand of the present biggest job in queue to the current resources owned by a TRE* is greater than one (which indicates that if the server does not request more resources, the present biggest job may not have enough resources for running), the server will request dynamic resources with the size of DR as follows:

DR=the accumulated resources demand of all jobs in the queue - the current resources owned by the TRE.

3) After obtaining dynamic resources from the resource provision service, the server registers *a new periodical timer* and checks idle dynamic resources per time unit of leasing resources. If there are idle dynamic resources with the size that is less than the value of *DR*, the server will release resources with the size of *DR= (DR- idle dynamic resources)*; else if there are idle dynamic resources with the size that is *equal to or more than the value of DR*, the server will release resources with the size of *DR* and deregister the timer.

There is only one difference in two resource management policies proposed for a MTC or a HTC service provider, respectively: we need to set the checking resource cycle of MTC as a smaller value than that of HTC, this is because MTC tasks often run over in seconds, while HTC jobs often run over in a longer period.

Since our aim is to consolidate workloads of small-to-medium scale organizations on a cloud site, we presume that in public clouds, a resource provider owns enough resources that can satisfy resource requests of N HTC and MTC service providers *(N>>2)*. So we propose a simple resource provisioning policy for a service provider: the resource provision service provisions the requested initial resources to a TRE at its startup; when the server of a TRE requests dynamic resources, the resource provision service assigns enough resources to the server. When the server of a TRE releases dynamic resources, the resource provision service will passively reclaim resources released by the server.

4 Evaluation Methodologies and Experiments

In this section, first, we report our chosen workloads; second, we present the evaluation methodologies; third, we give out the experiment configurations, and finally we will compare DawningCloud with the other three systems.

4.1 Workloads

We choose three typical HTC workload traces from the Parallel Workloads Archive: http://www.cs.huji.ac.il/labs/parallel/workload/. We choose one trace with lower load—NASA iPSC trace (46.6% utilization), two traces with higher loads—SDSC BLUE trace (76.2% utilization) and LLNL Thunder trace (86.5% utilization). For MTC, we choose a typical workload—the Montage workflow (http://montage.ipac.caltech.edu). The details of the workload traces can be found at Appendix B.1.

4.2 Evaluation Methodologies

In our experiments, we mainly concern the public cloud solutions, and hence a resource provider chooses *DawningCloud*, *Deelman's system*, *Evangelinos's system*, and the dedicated system to provide computing services, respectively. For DawningCloud, Evangelinos's system and Deelman's systems, one resource provider owns a cloud platform.

In the rest of this section, most of experiments are done with the emulation methodology. At the same time, *we deploy the real systems on the testbed to validate the accuracies of the emulated systems*. We also obtain the overhead of adjusting a node on the real system. Choosing the emulation methodology is based the two observations: first, to evaluate a system, many key factors have effects on experiment results, and we need to perform many times of time-consuming experiments, since durations of workload traces are from several weeks to even several months. Through using an emulation method, we can speedup experiments and complete large amount of experiments within shorter periods of time; second, with the real systems, consolidating several scientific communities' workloads needs hundreds of nodes, which results in mass resource requirements. While through using the emulation method, we can eliminate this resources limitation.

In this paper, all of the emulation systems are deployed on a test bed composed of nodes with the configuration of two 1.6GHz AMD Opteron processors, 2G memories and CentOS 5.0 operating system. For each emulated system, the job simulator is used to emulate the process of submitting jobs. For the HTC workloads, the job simulator generates each job by extracting its submission time, real run time, and requested number of nodes from the workload trace file; For the MTC workload, the job simulator reads the workflow file, which includes submission time, real run time, requested number of nodes, and dependencies between each job, and then submits jobs according to the dependency constraints. We speed up the submission and completion of jobs by a factor of 100.

In the rest of this section, we introduce how to emulate systems for different models.

The emulated dedicated cluster systems: For each dedicated cluster system, we deploy the *simplified* DawningCloud with two simulation modules: *the resource simulator*, and *the job simulator* on the testbed. The resource simulator defines the configurations of the dedicated cluster system. Since the workload files are obtained from platforms with different configurations. For example, the NASA trace is obtained from a cluster system with each node composed of one CPU; and the SDSC trace is obtained from a cluster system with each node composed of eight CPUs. In the rest of this paper, we presume that each node in our simulated cluster is composed of one CPU. And then, we scale workload traces with different constant values to the same configuration of the simulated cluster. Besides, the resource simulator not only simulates requesting and releasing resources, but also managing jobs such as loading or killing jobs and so on. The resource limit is enforced by the resource simulator. For the HTC TRE, the job simulator reads the job information from workload trace file and submits the job to the HTC server; for the MTC TRE, the job simulator is used to replace the trigger monitor to read the job information from the workload trace file and analyze control-flow dependencies among jobs to decide submitting the right job to the MTC server.

The emulated DawningCloud system: As shown in Fig.6, we deploy the *simplified* DawningCloud system, which keeps the resource provision service, one server, and one scheduler for each TRE while removing other services. Resource requesting and releasing are simulated by the interactions between the resource provision service and the resource simulator.

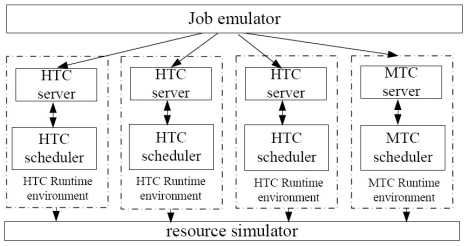


Fig. 5: Four emulated dedicated cluster systems for HTC and MTC.

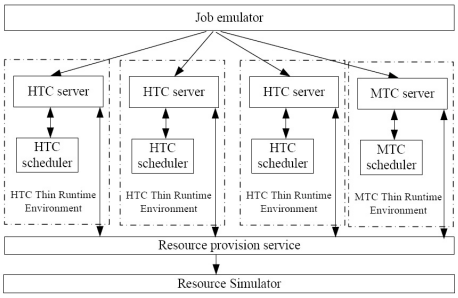


Fig. 6: The emulated DawningCloud.

The emulated Deelman's system: Since in Deelman's model, end users in scientific communities directly use EC2 for scientific computing. Based on the framework of DawningCloud, we implement and deploy an EC2-like system as shown in Fig.7 on the testbed with two simulation modules: the job simulator and the resource simulator. With respect to the real DawningCloud system, we only keep the resource provision service and the VM provision service. Resource requesting and releasing are enforced by the interactions between the resource provision service and the resource simulators. VM creating and destroying are enforced by the interactions between the VM provision service and the resource simulators. The job simulator reads the number of nodes which each job requests in the trace file, and sends requests to the resource provision service, which assigns corresponding resources for each job. When each job runs over, the job simulator will release resources to the resource provision service.

The emulated Evangelinos's system: We implement Evangelinos's system based on the framework of DawningCloud. Since a service provider in Evangelinos's system leases resources with the fixed size from a resource provider at a time, so the emulated Evangelinos's system is closely similar to that of the dedicated cluster system, shown in Fig.5.

4.3 Evaluation Metrics

We choose *the number of completed jobs* [8] and *the number of tasks per second* [1] to evaluate the performance metrics of HTC and MTC service providers, respectively. For a service provider, we choose *the resource consumption in terms of node*hour* to evaluate its cost. That is to say, we sum the product of the consumed resources in terms of nodes and their corresponding consumed hours as the cost of a service provider. In the Deelman's system, there is no role of a service provider, so we calculate the accumulated resource consumption of all end users, which amounts to the cost of a service provider in other models. For the dedicated cluster system, since a service provider owns resources, we calculate the resource consumption of a service provider as the product of the configuration size of a dedicated cluster system and the duration of a certain period. Please note that for different service provider, the above metrics are obtained in different durations to preserve the completeness of the workload trace.

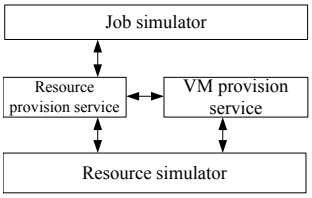


Fig. 7: The emulated Deelman's system.

For a resource provider, we choose *the total resource consumption* in terms of *node*hour* to evaluate the cost, which is the sum of all service providers' resource consumptions. Especially, we care about *the peak resource consumption* in terms of *nodes*. For the same workload, if the peak resource consumption of a system is higher, the capacity planning of a system is more difficult. In DawningCloud, Deelman's and Evangelinos's systems, since allocating or reclaiming resources will trigger setup actions, and we use the *accumulated times of adjusting nodes*, which is the sum of all service providers' *times of adjusting nodes*, to evaluate the management overhead of a resource provider. Since four workload traces have different durations, we choose the duration of Montage workload trace (two weeks) as the baseline period, and hence for NASA, SDSC, and LLNL, we extract their first two weeks trace to compare the resource provider's metrics in different systems.

4.4 Emulation Experiment Configurations

We emulate a public cloud scenario in which there are only one resource provider, three organizations providing HTC services and one organization providing MTC service. Of course, on a basis of the UltraSim emulation framework, we can easily extend to the case that one resource provider provisions resources to more service providers.

Resource configurations: According to the HTC workload trace's information, we set the configuration sizes of the dedicated cluster systems for the NASA, SDSC and LLNL traces as 128 nodes, 144 nodes and 1002 nodes in the emulation experiments, respectively. For the Montage workload, since in most of the running

time the accumulated resource demand of all jobs in the queue is 166 nodes, we set the configuration size of the dedicated cluster system in the experiment as 166 nodes to improve the throughput in terms of tasks per second. In the emulated Evangelinos's system, the fixed lease term of resources is the time duration of the trace; meanwhile, the sizes of leased resources are 128, 144, 1002 and 166 nodes for NASA, SDSC, LLNL and Montage, respectively. DawningCloud and Deelman's system request elastic resources according to varying workload traces.

Scheduling policies: A scheduling policy is needed by the schedulers in DawningCloud, Evangelinos's system, and the dedicated cluster system. In this paper, *we do not investigate the effect of different scheduling policies*, so we simply choose *the first fit scheduling policy* for HTC. The first-fit scheduling algorithm scans all the queued jobs in the order of job arrival and chooses the first job whose resources requirement can be met by the system to execute. For the MTC workload, firstly we generate the job flow according to dependency constraints, and then we choose the FCFS (First Come First Served) scheduling policy in DawningCloud, Evangelinos's system and the dedicated cluster system, respectively. According to [1], we set the scheduling cycle of the HTC and MTC schedulers as 60 seconds and 1 second, respectively. Deelman's system uses no scheduling policy, since all jobs run immediately without queuing.

Resource management and provisioning policies: DawningCloud, Deelman's and Evangelinos's systems adopt the same resource provisioning policy stated in Section 3.5. The dedicated cluster system owns static resources. DawningCloud adopts the resource management policy proposed in Section 3.5, while the dedicated cluster system and Evangelinos' system adopt the static resource management policy. Just like EC2, Deelman's system relies on the manual resource management, and we presume that a user only releases resources at the end of each time unit of leasing resources if a job runs over. This is because: first, EC2 charges the resource usage in terms of a time unit of leasing resources (an hour); second, it is difficult for end users to predict the completed time of jobs, and hence releasing resources to resource provider on time is almost impossible.

4.5 System-level Evaluation

In DawningCloud, we need to set the following parameters for the service provider:

a) The time unit of leasing resources, which is represented as *C minutes*. The time unit of leasing resources has effect on both DawningCloud and Deelman's system. When the time unit of leasing resources is shorter, resources will be adjusted more frequently, which brings higher management overhead.

b) The size of initial resources, which is represented as *B*.

c) The checking resource cycle, which is represented as *S seconds*. We set *S* as the same value of the scheduling cycle in the scheduling policy.

d) The threshold ratio of obtaining dynamic resources, which is represented as *R*.

Before reporting experiment results, we pick the following parameters as the baseline for comparisons, and detailed parameter analysis can be found at Appendix B.2.

Through comparisons with large amount of experiments, we set the baseline configurations in DawningCloud: [60C/40B/1.5R/60S] for the NASA and SDSC workloads, [60C/300B/1.5R/60S] for the LLNL workload and [60C/20B/8R/1S] for the MTC workload, where *[xY]* indicates that *Y* is *x*.

In the following experiments, each experiment is performed six times. We report the mean values across six times experiments. Because we obtain data sets with different units instead of a single data, we use *the coefficient of variation* instead of the standard deviation to measure data variation. The coefficient of variation is defined as the standard deviation to the mean. After calculation, we find that the coefficient of variations are not more than 0.08%.

TABLE 1: THE METRICS OF THE SERVICE PROVIDER USING FOUR SYSTEMS FOR THE NASA TRACE.

Configuration	number of completed jobs	resource consumption (node*hour)	saved resources
dedicated cluster system	18237	282752	/
Evangelinos's system	18237	282752	0
Deelman's system	18237	386235	-36.6%
DawningCloud	18237	210507	25.6%

TABLE 2: THE METRICS OF THE SERVICE PROVIDER RUNNING FOUR SYSTEMS FOR THE SDSC TRACE.

Configuration	number of completed jobs	resource consumption (node*hour)	saved resources
dedicated cluster system	223389	3383712	/
Evangelinos's system	223389	3383712	0
Deelman's system	223391	6849090	-102%
DawningCloud	223391	2911383	14%

For the dedicated cluster system and Evangelinos's system, they have the same configurations with the only one difference in that a service provider in the dedicated cluster system owns resources while a service provider in Evangelinos's system leases resources, so they gain the same performance.

Table 2-4 summarize the experiment results of two HTC service providers and one MTC service providers who run DawningCloud, Evangelinos's system, the dedicated cluster system, and Deelman's systems, respec-

TABLE 3: THE METRICS OF THE SERVICE PROVIDER RUNNING FOUR SYSTEMS FOR THE LLNL TRACE.

Configuration	number of completed jobs	resource consumption (node*hour)	saved resources
dedicated cluster system	118791	3612210	/
Evangelinos's system	118791	3612210	0
Deelman's system	118791	8205050	-127%
DawningCloud	118791	3326085	7.9%

tively. *The percents of the saved resources* are obtained against the resource consumption of the dedicated cluster system.

TABLE 4: THE METRICS OF THE SERVICE PROVIDER RUNNING FOUR SYSTEMS FOR THE MONTAGE WORKFLOW.

Configuration	tasks per second	resource consumption (node*hour)	saved resources
dedicated cluster system	2.46	55776	/
Evangelinos's system	2.46	55776	0
Deelman's system	2.68	66200	-18.7%
DawningCloud	2.46	18108	67.5%

For the NASA, SDSC, and LLNL traces, in comparison with the dedicated cluster system or Evangelinos's system, service providers in DawningCloud save the resource consumption maximally by 25.6% and minimally by 7.9%, and at the same time gain the same or higher throughputs. This is because service providers in DawningCloud can resize resources according to varying workload status, while service providers in the dedicated cluster system or Evangelinos's system owns or leases resources with the fixed size; we also find that the the saved resource consumption is inversely proportional to the resource utilization reported in the traces, this is because that high utilization implies less space for saving resources.

For the Montage workload, DawningCloud has the same performance as that of the dedicated cluster system or Evangelinos's system for the service provider. This is because driven by the resource management policy stated in Section 3.5, the MTC server will adjust dynamic resources to *the size of the accumulated resource demand of jobs in queue*, which is same as the chosen configurations of the dedicated cluster system or Evangelinos's system (166 nodes). In comparison with the dedicated cluster system or Evangelinos's system, the service provider in the DawningCloud saves the resource consumption by 67.5%, this is because the service provider in the dedicated cluster system or Evangelinos's system owns or leases resources with the fixed size, while the service provider in DawningCloud owns *initial resources* with the smaller size, and resizes dynamic resources driven by the change of workload status.

For the NASA, SDSC, and LLNL traces, with respect to Deelman's system, DawningCloud saves the resource consumption maximally by 59.5% for the service providers with the same performance. On one hand, this is because the dynamic resource negotiation and queuing based resource sharing mechanisms in DawningCloud lead to the decrease of resource consumption. On the other hand, in Deelman's system, each end user directly obtains resources from the resource provider, which results in that Deelman's system consumes more resources than that of DawningCloud.

For the Montage workload, DawningCloud saves the resource consumption by 72.6% with respect to that of Deelman's system for the same service provider. This is because the required resources of end users will be provisioned immediately in Deelman's system and the peak resource demand of the MTC workload is high. At the same time, Deelman's system gains higher throughput than that of DawningCloud.

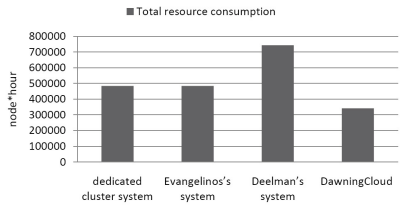


Fig. 8: The total resource consumptions of the resource provider using four different systems.

Fig.8 and Fig.9 show experiment results for the resource provider who uses four different systems: DawningCloud, Evangelinos's system, Deelman's system, and the dedicated cluster system, respectively.

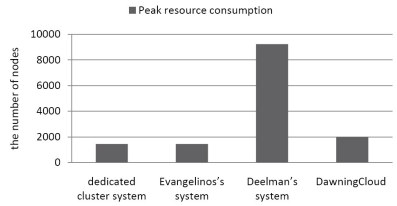


Fig. 9: The peak resource consumptions of the resource provider using four different systems.

Using the dedicated cluster system and Evangelinos's system, the resource provider has the same total resource consumption and the same peak resource consumption, since they have only one difference in that the former owns resources while the latter leases resources.

Using DawningCloud, the total resource consumption of the resource provider is 341175 node*hour, which saves the total resource consumption by 29.5% with respect to that of the dedicated cluster system or Evangelinos's system. In the dedicated cluster system or Evangelinos's system, the service providers lease or

purchase resources with the fixed size that is decided by the peak resource demand of the largest job. In contrast, in DawningCloud, a service provider can start with the small-sized initial resources and resize dynamic resources according to varying resource demand. Hence, the total resource consumption of DawningCloud is less than that of the dedicated cluster system or Evangelinos's system when workloads of four service providers are consolidated. At the same time, With DawningCloud, the peak resource consumption of the resource provider is 2000 nodes, which is only 1.39 times of that of dedicated cluster system or Evangelinos's system.

Using DawningCloud, the resource provider saves the total resource consumption by 54% with respect to that of the Deelman's system, and the peak resource consumption of DawningCloud is only 0.22 times of that of the Deelman's system. Because the required resources of each job will be provisioned immediately in Deelman's system, its peak resource consumption is larger than that of DawningCloud.

Fig.10 shows the management overhead of the resource provider using Evangelinos's model, Deelman's model and DawningCloud. For the dedicated cluster system, since the resource provider owns resource, it has no management overhead in terms of obtaining dynamic resources. From Fig.10, we can observe that Evangelinos's system has the lowest management overhead, since it leases resources with the fixed duration. DawningCloud has smaller management overhead than that of Deelman's system, since the initial resources will not be reclaimed until a TRE is destroyed.

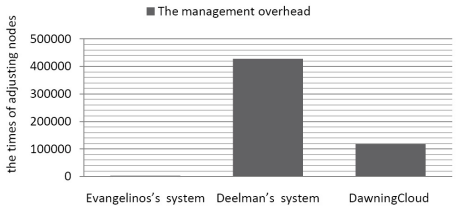


Fig. 10: The management overhead of the resource provider.

In our real test, excluding wiping off the operating system, the total cost of assigning and reclaiming one node is 15.743 seconds, which includes the operation of stopping and uninstalling previous TRE packages, installing and starting new TRE packages. That is to say that the resource consumption of adjusting nodes in DawningCloud is approximately 259 node*hour when the total resource consumption is 341175 node*hour. This overhead is acceptable (only 0.076%).

4.6 The Accuracies of Two Emulated Systems

In order to verify the accuracies of our emulated systems, we deploy two real systems: the dedicated cluster system and DawningCloud on the testbed. The testbed is composed of 40 X86-64 nodes, of which the configuration is 2 Quad-Core 2 GHz Intel Xeon processors with 8G Memories, and the operating system on each node is CentOS 5.2.

We synthesize a HTC workload, which includes 100 jobs with the size from 8 to 64 cores. 100 jobs are submitted to the dedicated cluster system and DawningCloud for HTC within 10 hours, respectively, and the average interval of submitting jobs is 300 seconds. For the synthesized workload trace, the distribution of resource demands of jobs in terms of processors and average execution time vs. different resource demands of jobs in terms of cores are shown in Fig.11.

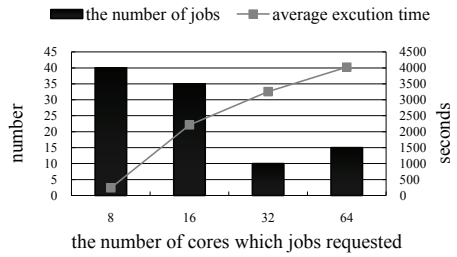


Fig. 11: The real workload trace.

In two-step experiments, first, we submit the synthesized workload to the real dedicated cluster system and the real DawningCloud system for HTC, and then collect the workload trace, including: job size, job submission time, job execution time. Second, we submit the workload trace to the emulated dedicated cluster system and DawningCloud for HTC. We compare the metrics of real systems and emulated systems to evaluate the accuracies of emulated systems.

In two real systems, the scheduling,resource management and resource provision policies are same as those in the above emulated systems, respectively. we set the configuration sizes of the dedicated cluster systems as 20 nodes, which is decided by the the utilization rate. When the size is 20 nodes, the utilization rate is 93.2% for the above synthesized workload. The baseline setting for the HTC TRE in DawningCloud is [60C/8B/1.5R/60S].

After we obtain the workload trace through experiments of the real systems, we submit the same workload trace to two simulated systems again. Each experiment is performed six times, of which the coefficient of variation for experiment results are not more than 0.4%. Table 5 shows the mean results of both real and emulated systems. Please note that in the real and emulated systems, the duration between the submission time of the first job and the time when the last job runs over are different. In Table 5, we adopt the right duration which ensures the last job has run over in each system, respectively.

In the rest of this section, we report the mean values of experiments. We find that: first, for both the dedicated cluster system and DawningCloud, the ratio of the resource consumptions of the real systems to that

TABLE 5: THE METRICS BETWEEN REAL and EMULATED SYSTEMS.

Configuration	number of completed jobs	resource consumption (node*hour)
Real dedicated cluster system	100	280
Emulated dedicated cluster system	100	240
Real DawningCloud system	100	266
Emulated DawningCloud system	100	234

of the emulated systems are larger than one, and very close (1.17 and 1.14). In Table 6, we use this factor to re-evaluate the results of the emulated systems. Table 6 shows all percentages of the saved resources are larger than that of the original value after the re-evaluation. Second, in comparison with the dedicated cluster system, it is credible that service providers in DawningCloud can save the resource consumption while gaining the same throughput in both the real and emulated systems.

TABLE 6: THE ORIGINAL AND RE-EVALUATED RESOURCE CONSUMPTIONS OF EMULATION SYSTEMS.

Configuration	dedicated cluster system	DawningCloud	Saved resources
NASA original value	282752	210507	25.6%
NASA re-evaluated value	330820	239978	27.5%
SDSC original value	3383712	2911383	14%
SDSC re-evaluated value	3958943	3318977	16.2%
LLNL original value	3612210	3326085	7.9%
LLNL re-evaluated value	4226286	3791737	10.3%

From the above analysis, we can make the conclusion that our two emulated systems are enough accurate with respect to the real systems.

5 Related Work

We summarize the related work from three perspectives: evaluation of cloud systems, infrastructure for scientific communities, and resource management issues.

5.1 Evaluation of Cloud systems:

Armbrust *et al.* [2] in theory show the workloads of Web service applications can benefit from the economies of scale of cloud computing systems, however, no one answers this question from the perspective of scientific communities. In the context of hybrid cloud, de Assuncao *et al.* [26] investigate whether an organization operating its local cluster can benefit from using cloud providers to improve the performance of its users' requests; Marshall *et al.*'s evaluation of elastic site [25] consists primarily of a comparison of the three different policies (on demand, steady stream, and bursts) in an attempt to maximize job turnaround time while minimizing thrashing and idle VMs. Palankar *et al.* [29] evaluates S3 as a black box and reasons whether S3 is an appropriate service for science grids.

5.2 infrastructure for scientific communities

Public Cloud solutions: Amazon's EC2 directly provides resources to end users, and relies upon end user's manual management of resources. EC2 extended services: RightScale (http://www.rightscale.com/) provides automated Cloud computing management systems that helps you create and deploy only *Web service applications* running on EC2 platform. There are two proposed usage models for EC2-like public clouds in scientific communities. Deelman *et al.* [5] propose each staff of an organization to directly lease virtual machine resources from EC2 for running applications in a specified period. Evangelinos *et al.* [3] propose that an organization as a whole rents resources with the fixed size from EC2 to create a leased cluster system that is deployed with a queuing system, like OpenPBS, for HTC workloads. With respect to our previous work [21], our new contributions are three-fold: first, we propose an enhanced scientific public cloud model (ESP) that encourages small or medium scale research organizations rent elastic resources from a public cloud provider. Second, we improve the resource management policy for both HTC and MTC workloads, and investigate effects of different configuration parameters on the experiment results. Third, we propose an emulation methodology to answer our concerned economies of scale issue. Moreover, we selectively perform real experiments to validate the accuracies of the emulated systems.

Private and hybrid cloud solutions: two open source projects, OpenNebula (www.opennebula.org/) and Haizea (http://haizea.cs.uchicago.edu/), are complementary and can be used to manage Virtual infrastructures in private/hybrid clouds [28]. In the context of hybrid cloud, recently, Sun Microsystems has added support for Amazon EC2 into Sun Grid Engine (SGE); Moreno-Vozmediano *et al.* [22] analyze the deployment of generic clustered services on top of a virtualized infrastructure layer that combines a VM manager (on a local cluster) and a cloud resource provider (external cloud provider: Amazon EC2). Marshall *et al.* [25] have implemented a resource manager, built on the Nimbus toolkit to dynamically and securely extend existing physical clusters into the cloud. Rodero-Merino *et al.* [39] proposes a new abstraction layer that allows for their automatic deployment and escalation depending on the service status. Our previous Transformer [35] programming framework aims to facilitate the building of diverse data-parallel programming models: Dryad-like data flow, MapReduce, and All-Pairs. Hindman *et al.* [41] presents Mesos, a platform for sharing commodity clusters between multiple diverse cluster computing

frameworks, such as Hadoop and MPI. With respect to Reservoir, DawningCloud has two-fold differences as follows: first, the CSF of DawningCloud not only facilitates building lightweight service management layers for heterogeneous workloads, but also makes their management tasks simpler. Second, the CSF of DawningCloud allows a service provider to flexibly control its TRE. Moreover, the implementation of TRE is not bound to virtual machine technologies. DawningCloud supports both physical and virtual resources provisioning.

Virtual execution environments: Irwin *et al.* [7] propose a prototype of service oriented architecture for resource providers and consumers to negotiate access to resources over time. On a basis of virtualization technologies, previous systems provide virtual execution environments either for grid computing [16] [18] [23] [24] or data center [14] [15] [17]. Walker *et al.* [27] presents a system for creating personal clusters in user-space to support the submission and management of thousands of compute-intensive *serial jobs*, which allows the expansion of local resources on-demand during busy computation periods. Lim *et al.* [37] addresses elastic control of the storage tier in multi-tier application services.

Cloud operating system: Wentzlaff *et al.* propose a single system image operating system, named *fos*, across both multicore and Infrastructure as a Service (IaaS) cloud systems [47]. Though DawingCloud shares the idea of factoring some functions into management services, it sits on top of fos, providing an infrastructure for typical scientific workloads, i.e., HTC and MTC.

5.3 Resource management issues:

Resource management issues are widely investigated in the context of cloud computing and grid computing. In the context of private cloud, Sotomayor *et al.* [34] present the design of lease management architecture, Haizea, which implements leases as virtual machines (VMs) to provide leased resources with customized application environments. In the context of hybrid cloud, de Assuncao *et al.* [26] evaluate the cost of six scheduling strategies used by an organization that operates a cluster managed by virtual machine technology. Dan *et al.* [12] propose the algorithm for scheduling mixed workloads in multi-grid environments.

6 Conclusion

In this paper, we have answered one key question to the success of cloud computing: In scientific communities, can small-to-medium scale research organizations benefit from the economies of scale? Our contributions are threefold: first, we proposed the ESP model for small-to-medium scale scientific communities to utilize elastic resources on a public cloud site while maintaining their flexible system controls. Second, on a basis of the ESP model, we designed and implemented an innovative system, DawningCloud, at the core of which are lightweight service management layers running on top of a common management service framework. Third, we evaluated the systems comprehensively using both emulation and real experiments. For four traces of two typical workloads: HTC and MTC, we concluded that small-to-medium scale scientific communities indeed can benefit from the economies of scale of public clouds with the support of the enabling system—DawningCloud.

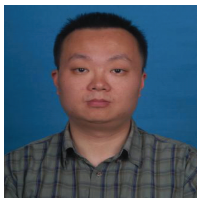
Acknowledgment

We are very grateful to anonymous reviewers. This work is supported by the Chinese 863 project (Grant No.2009AA01Z128), the Chinese 973 project (Grant No.2011CB302500 and Grant No.2007CB311100) and the NSFC projects (Grant No.60703020 and Grant No.60933003). Weisong Shi is in part supported by US NSF CAREER grant CCF-0643512.

References

[1] Raicu, I. *et al.* 2008. Many-task computing for grids and supercomputers, In Proc. of MTAGS 2008.
[2] Armbrust, M. *et al.* 2009. Above the clouds: A berkeley view of cloud computing, Tech. Rep. UCB/EECS-2009-28.
[3] Evangelinos, C. *et al.* 2008. Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazons EC2, In Proc. of CCA 08.
[4] Garfinkel, S. L.. 2007. Commodity grid computing with amazon's S3 and EC2, Login: The USENIX Magazine.
[5] Deelman, E. *et al.* 2008. The cost of doing science on the cloud: the montage example, In Proc. of SC 08.
[6] Kondo, D. *et al.* 2009. Cost-benefit analysis of Cloud Computing versus desktop grids, 18th International Heterogeneity in Computing Workshop (HCW09).
[7] Irwin, D. *et al.* 2006. Sharing networked resources with brokered leases, In Proc. of USENIX ATC 2006.
[8] Gaj, K. *et. al.* 2002. Performance evaluation of selected job management systems, In Proc. of 16th IPDPS.
[9] Livny, M. *et al.* 1997. Mechanisms for high throughput computing, SPEEDUP journal.
[10] Grit, L.E. *et al.* 2008. Weighted fair sharing for dynamic virtual clusters, Proceedings of SIGMETRICS 08, pp.461–462.
[11] Zhan, J. *et al.* 2008. Phoenix Cloud: Consolidating Different Computing Loads on Shared Cluster System for Large Organization, In Proc. of CCA08. arXiv:0906.1346v6 [cs.DC].
[12] Dan, M. *et al.* 2006. Scheduling mixed workloads in multi-grids: the grid execution hierarchy, In Proc. of 15th HPDC, pp.291–302.
[13] Rochwerger, B. *et al.* 2009. The reservoir model and architecture for open federated cloud computing, IBM Systems Journal, Vol.53(4).
[14] Kallahalla, M. *et al.* 2004. SoftUDC: A software-based data center for utility computing, Computer, Vol.37(11), pp.38-46.
[15] Ruth, P. *et al.* 2005. Viocluster: Virtualization for dynamic computational domains, In Proc. of Cluster 05.
[16] Keahey, K. *et al.* 2005, Virtual workspaces in the grid, In Proc. of Euro-Par 2005, pp.421–431.
[17] Jiang, X. *et al.* Soda: A service-on-demand architecture for application service hosting utility platforms, In Proc. of 12th HPDC.
[18] Montero, R.S. *et al.* 2008. Dynamic deployment of custom execution environments in Grids, In Proc. of ADVCOMP'08, pp.33–38.
[19] Sotomayor, B. *et al.* 2008. Combining batch execution and leasing using virtual machines, In Proc. of 17th HPDC, pp.87–96.
[20] Buyya, R. *et al.* 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems, Vol.25(6), pp.599–616.
[21] Wang, L. *et al.*. 2009. In cloud, do MTC or HTC service providers benefit from the economies of scale? In Proc. of MTAGS' 09.
[22] Moreno-Vozmediano, R. *et al.* 2009. Elastic management of cluster-based services in the cloud, In Proc. of ACDC' 09, pp.19–24.

[23] Chase, J.S. *et al.* 2003. Dynamic virtual clusters in a grid site manager, In Proc. of HPDC 03.
[24] Montero, R.S. *et al.* 2009. Dynamic Provisioning of Virtual Clusters for Grid Computing, Euro-Par 2008 Workshops-Parallel Processing.
[25] Marshall, P. *et al.* 2010. Using Clouds to Elastically Extend Site Resources, In Proc. of CCGrid 2010.
[26] de Assunção, M. D. *et al.* 2009. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters, In Proc. of the 18th HPDC, pp.141–150.
[27] Walker, E. *et al.* 2007. Personal adaptive clusters as containers for scientific jobs, Cluster Computing, Vol.10(3), pp.339–350.
[28] Sotomayor, B. *et al.* 2009. Virtual infrastructure management in private and hybrid clouds, IEEE Internet Computing, Vol.13(5), pp.14–22.
[29] Palankar, M. R. *et al.* 2008. Amazon S3 for science grids: a viable solution?, In Proc. of DADC' 08, pp.55–64.
[30] Zhan, J. and Sun, N. 2005. Fire phoenix cluster operating system kernel and its evaluation, In Proc. of Cluster 05.
[31] Zhan, J. *et al.* 2007. The design methodology of Phoenix cluster system software stack. In Proc. of CHINA HPC '07. ACM, New York, NY, 174-182.
[32] Zhang Z. *et al.* 2006. Easy and reliable cluster management: the self-management experience of Fire Phoenix, In Proc. of IPDPS 2006.
[33] Evangelinos, C. *et al.* 2009. Many task computing for multidisciplinary ocean sciences: real-time uncertainty prediction and data assimilation, In Proc. of MTAGS' 09.
[34] Sotomayor, B. *et al.* 2008. F. Capacity leasing in cloud systems using the opennebula engine, In Proc. of CCA' 08.
[35] Wang, P. *et al.* 2010. Transformer: A New Paradigm for Building Data-Parallel Programming Models. IEEE Micro 30, 4 (Jul. 2010), 55-64.
[36] Walker, E. 2009. The Real Cost of a CPU Hour. Computer 42, 4 (Apr. 2009), 35-41.
[37] Lim, H. C. *et al.* 2010. Automated control for elastic storage. In Proc. of ICAC'10. ACM, New York, NY, 1-10.
[38] Galn F. *et al.* 2009. Service specification in cloud environments based on extensions to open standards. In Proceedings of COMSWARE '09, pp.1-12.
[39] Rodero-Merino, L. *et al.* 2010. From infrastructure delivery to service management in clouds. Future Gener. Comput. Syst. 26, 8 (Oct. 2010), 1226-1240.
[40] Sullivan, A. *et al.* 2003. Economics: Principles in action. Upper Saddle River, New Jersey 07458: Pearson Prentice Hall. pp. 157. ISBN 0-13-063085-3.
[41] Hindman, B., *et al.* 2010. Mesos: A platform for fine-grained resource sharing in the data center, UCBerkeley, Tech. Rep. UCB/EECS-2010-87. [Online].
[42] Zhan, J., *et al.* 2010. PhoenixCloud: Provisioning Resources for Heterogeneous Workloads in Cloud Computing. Technical Report. arXiv:1006.1401v2 [cs.DC].
[43] Calheiros, R. *et al.* CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. In Proc. ICPP' 09.
[44] Verma, A., *et al.* 2008. Power-aware dynamic placement of HPC applications. In Proc. ICS '08.
[45] Huang, W. *et al.* 2006. A case for high performance computing with virtual machines. In Proc. of ICS' 06.
[46] Zhou, W. *et al.* 2011. Scalable Group Management in Large-Scale Virtualized Clusters, To appear in the Journal of High Technology Letters. Available at http://arxiv.org/abs/1003.5794.
[47] Wentzlaff, D. *et al.* 2010. An operating system for multicore and clouds: mechanisms and implementation. In Proc. SoCC '10, pp.3-14.
[48] Dean, J. *et al.* MapReduce: Simplified Data Processing on Large Clusters. In Proc. SOSP' 04, pp. 137-150.
[49] Isard, M. *et al.* Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks, ACM SIGOPS Operating Systems Rev., vol. 41, no. 3, 2007, pp. 59-72.
[50] Moretti, C. *et al.* All-Pairs: An Abstraction for Data-Intensive Cloud Computing. In Proc. IPDPS' 08, pp. 1-11.



Lei Wang received the master degree in computer engineering from Chinese Academy of Sciences, Beijing, China, in 2006. He is currently a research assistant with Institute of Computing Technology, Chinese Academy of Sciences. His current research interests include resource management of cluster and cloud systems. He was a recipient of the Distinguished Achievement Award of the Chinese Academy of Sciences in 2005.

Jianfeng Zhan received the Ph.D degree in computer engineering from Chinese Academy of Sciences, Beijing, China, in 2002. He is currently an Associate Professor of computer science with Institute of Computing Technology, Chinese Academy of Sciences. His current research interests include distributed and parallel systems. He has authored more than 40 peer-reviewed journal and conference papers in the aforementioned areas. He is one of the core members of the petaflops HPC system and data center computing projects at Institute of Computing Technology, Chinese Academy of Sciences. He was a recipient of the Second-class Chinese National Technology Promotion Prize in 2006, and the Distinguished Achievement Award of the Chinese Academy of Sciences in 2005.

Weisong Shi received the Ph.D degree in computer engineering from Chinese Academy of Sciences, Beijing, China, in 2000. He is currently an Associate Professor of computer science with Wayne State University. His current research interests include computer systems and mobile computing. He has authored more than 100 peer-reviewed journal and conference papers in the aforementioned areas. He is a recipient of the NSF CAREER award.

Yi Liang received the Ph.D degree in computer engineering from Chinese Academy of Sciences, Beijing, China, in 2005. She is currently an Associate Professor of computer science with Beijing University of Technology, Beijing, China. She is a member of Technical Committee of High Performance Computing in China. Her current research interests include computer architecture and distributed computing.

TABLE 7: THE COMPARISONS OF DIFFERENT USAGE MODELS

item[a]	DS	PC	HC	DP	EP	ESP
SP vs. RP[b]	1:1	1:1	n:1	n:1	n:1	n:1
Separation of concerns	No	No	No	No	Yes	Yes
resources	local	local	local + rented	rented	rented	rented
resource provisioning	fixed	fixed	fixed + elastic	elastic	fixed	initial + dynamic
SP's system control	No	No	No	limited	limited	flexible

a. DS: dedicated system; PC: private cloud; HC: hybrid cloud; DP: Deelman's public cloud; EP: Evangelinos's public cloud; ESP : our model.

b. SP: service provider; RP: resource provider

APPENDIX A
THE DIFFERENCES OF THE ESP MODEL FROM OTHER ONES

Table 7 compares the ESP model with other ones. There are three distinguished features of the ESP model.

First, our ESP model allows a resource provider to provide TRE and provision elastic resources to n ($n>>2$) small-to-medium scale scientific communities, and hence it guides the design and implementation of the enabling platform helping us to answer the concerned question. The dedicated systems and private clouds' limited use scopes will prevent service providers from benefiting from the economical of scale. In hybrid clouds, users own local resources, and request elastic resources from external public clouds for workload spike. Hybrid clouds that connect local resources with external public clouds are difficult for some parallel applications that rely heavily on frequent collective communications, since these applications are generally sensitive to network delays [26]. In Deelman's public cloud model, each end user manually requests or releases resources from a resource provider. In Evangelinos's public cloud model, an organization as a whole obtains resources with the fixed size from a resource provider.

Second, the ESP model separates the concerns of the resource provider and service providers. In private clouds, service providers are often affiliated with the resource provider, and the role of service provider is complexly intertwined with the role of resource provider. In the hybrid cloud model, users own local resources as a resource provider, at the same time they also rent resources from the external public cloud as a service provider. Deelman's model does not separate the concern of the service provider.

Third, in the ESP model a service provider does not own resources, and instead automatically requests elastic resources from the resource provider. Besides, we propose a different resource provisioning mode, which distinguishes two types of provisioned resources to a TRE: *initial resources* and *dynamic resources*. Once allocated to a TRE, initial resources will not be reclaimed by the resource provider until the TRE is destroyed. On the contrary, dynamic resources assigned to a TRE may be reclaimed by the resource provider. In Deelman's model, each end user manually requests or releases (dynamic) resources from a resource provider. In Evangelinos's model, an organization as a whole obtains (initial) resources with the fixed size from a resource provider. In the dedicated system and private cloud models, in general they own fixed resources, though for the latter, a specific workload may use elastic resources within the organization for a specific duration. In hybrid clouds, users also rent elastic resources from the external public cloud; however, it is difficult for the others to share idle local resources when local loads are light.

APPENDIX B
DETAILS OF EVALUATION METHODOLOGIES AND EXPERIMENTS

B.1 Workload Details

We choose three typical HTC workload traces from the Parallel Workloads Archive: http://www.cs.huji.ac.il/labs/parallel/workload/. The utilization rate of all traces in the Parallel Workloads Archive varies from 24.4% to 86.5%. We choose one trace with lower load-NASA iPSC trace (46.6% utilization), two traces with higher load-SDSC BLUE trace (76.2% utilization) and LLNL Thunder trace (86.5% utilization).

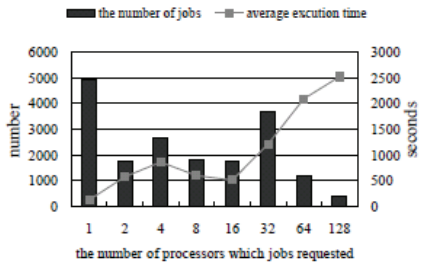


Fig. 12: The NASA workload trace.

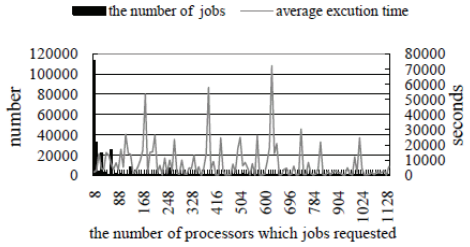


Fig. 13: The SDSC workload trace.

For the NASA trace, the time duration is three months, the average execution time is 764 seconds, and the total number of jobs is 18239. For the SDSC trace, the time duration is 32 months, the average execution time is 4381

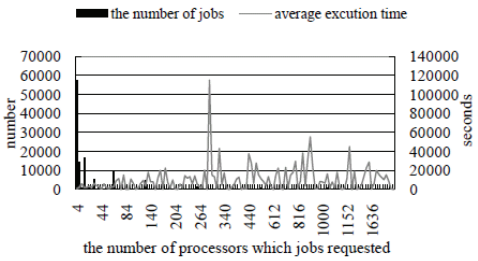


Fig. 14: The LLNL workload trace.

seconds, and the total number of jobs is 223407. For the LLNL trace, the time duration is five months, the average execution time is 2227 seconds, and the total number of jobs is 118791.

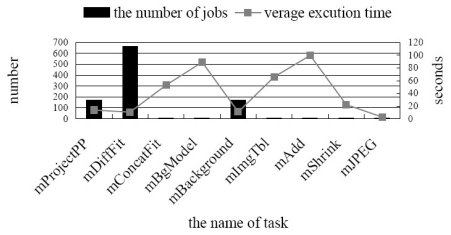


Fig. 15: The Montage workload trace.

For MTC, we choose a typical workload, the Montage workflow (http://montage.ipac.caltech.edu). Montage is an astronomy workflow application, created by NASA/IPAC Infrared Science Archive for gathering multiple input images to create custom mosaics of the sky. The workload generator can be found on the web site http://vtcpc.isi.edu/pegasus/index.php/WorkflowGenerator, and the workload file includes the job name, run time, inputs, outputs and the list of control-flow dependencies of each job. The chosen Montage workload includes 9 types of task with the total amount of 1,000 jobs. Each job requests one node for running, and the average execution time of jobs is 11.38 seconds. The run time of the MTC workload is more shorter than that of HTC workload traces, so we synthesize the MTC workload through repeatedly submitting Montage workload in two weeks.

B.2 Parameter Analysis in the Emulated Systems

Because of space limitation, we are unable to present the data for the effect of all parameters; instead, we constrain most of our discussion to the case that one or two parameters varies while the other parameters keep the same as those of the baseline configuration in Section 4.5, which are representative of the trends that we observe across all cases. We only report the first two weeks' metrics for the NASA, SDSC and Montage traces with different configurations, since they have different time durations.

The effect of the size of initial resources and the threshold ratio of obtaining dynamic resources.

To save space, in DawningCloud we tune the size of initial resources (B) and the threshold ratio of obtaining dynamic resources (R) at the same time, while other parameters are [60C/60S] for HTC workload and [60C/1S] for MTC workload.

We respectively set B as (0,20,40,60,80,100,144) for the SDSC workload and (0,20,40,60,80,100,128) for the NASA workload, and (0,20,40,60,80,100,166) for the Montage workload; at the same time, we tune R as (1,1.2,1.5,2,4,100) for the HTC workloads and (1,2,4,8,16,100) for the MTC workload.

Fig.16 shows the effect of different parameters. In Fig.16, *Bx_Ry* indicates that B is x and R is y.

For three workload traces, when *the size of initial resources is the same like that of Evangelinos's system and the threshold ratio of obtaining dynamic resources is so high as to no dynamic resource will be obtained*, DawningCloud has the same performance metrics as that of Evangelinos's system. For example, for the NASA, SDSC, and Montage workloads, the configuration is respectively B128_R100, B144_R100, and B166_R100.

For HTC workloads, we have the following observations:

1) In DawningCloud, the resource consumption is proportional to the size of initial resources; this is because initial resources are statically allocated to a service provider in DawningCloud. For the same workload, the size of initial resources increases, idle resources will also increases. When the size of initial resources is below the configuration size of Evangelinos's model, the size of initial resources has no significant effect on the number of completed jobs, since dynamic resources can be obtained in DawningCloud.

2) The resource consumption is inversely proportional to R when dynamic resources are not zero; this is because that larger threshold ratio can result in less opportunity of obtaining dynamic resources. There is no obvious relationship between R and the number of completed jobs.

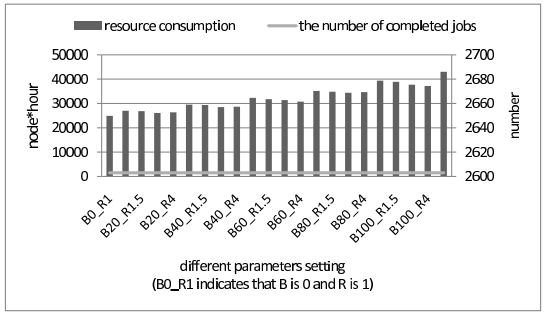
For MTC workloads, we have the following observations:

1) There are several configurations (such as B20_R4, B20_R8 and B40_R4 in Fig.16) that make the service provider consumes less resources. From our observation, we found those configurations satisfying the empirical formulas $(B*R < RA)$ *and* $(RA*R > RM)$, where RA is *the accumulated resource demand of jobs in queue* in most of running time, and RM is the maximal accumulated resource demand of jobs in queue in running time. For Mantage workload trace, RM is 662, and RA is 166.

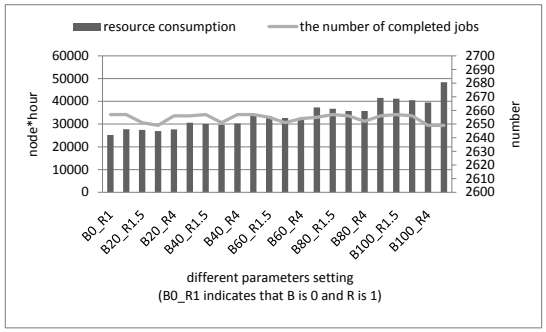
2) There is no obvious relationship between B and the resource consumption or the number of tasks per second.

3) There is no obvious relationship between R and the resource consumption or the number of tasks per second.

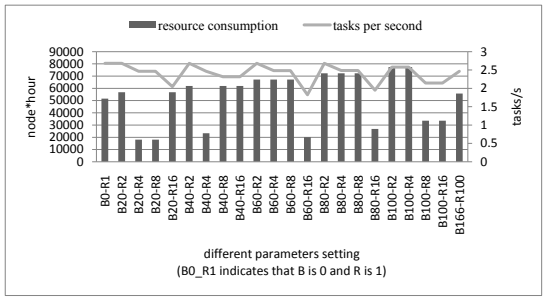
Fig.17 shows the effects of *the size of initial resource* (B)



(a) The resource consumption and the number of completed jobs V.S. different parameters setting for the NASA trace.

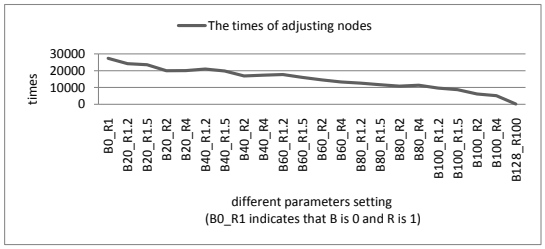


(b) The resource consumption and the number of completed jobs V.S. different parameters setting for the SDSC trace.

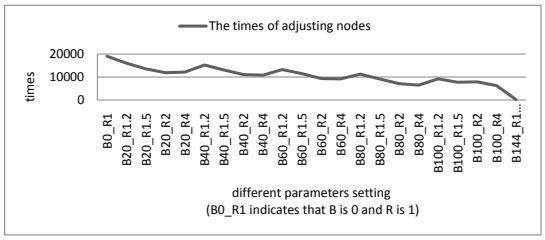


(c) The resource consumption and tasks per second VS. different parameters for the Montage workload.

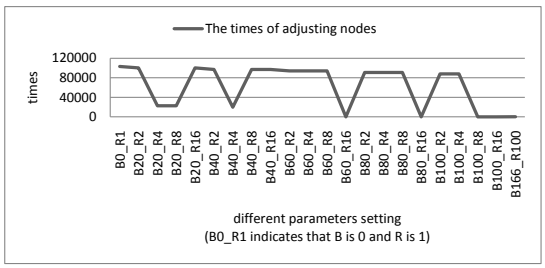
Fig. 16



(a) The service providers' times of adjusting nodes in two weeks VS. different parameters setting for the NASA trace.



(b) The service providers' times of adjusting nodes in two weeks VS. different parameters setting for the SDSC trace.



(c) the service providers' times of adjusting nodes in two weeks VS. different parameters setting for the Montage workload.

Fig. 17

and *the threshold ratio of obtaining dynamic resources* (*R*) on the management overhead of the resource provider, which is the sum of all service providers' times of adjusting nodes.

For HTC workloads, we have the following observations:

1) The management overhead of the service provider is inversely proportional to *B*; this is because more initial resource, less times of obtaining dynamic resources.

2) The management overhead is inversely proportional to *R*; this is because larger threshold ratio, less times of obtaining dynamic resource.

For MTC workloads, we have the following observations:

1) For some configurations satisfying the empirical formulas *(B*R < RA)* and *(RA*R > RM)* or *(B*R > RM)*, the management overhead is less than that of other configurations.

2) There is no obvious relationship between the times of adjusting nodes for the MTC service provider and *B/R*.

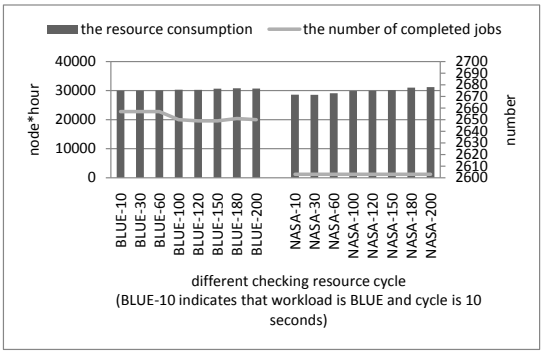
The effect of checking resource cycle.

In DawningCloud, we respectively set the checking resource cycle *S* as 10/30/60/100/120/150/180/200 seconds, while other parameters are [60C/40B/1.5R] for HTC workloads. In the scheduling policy, *the scheduling cycle* is the same amount as *S*.

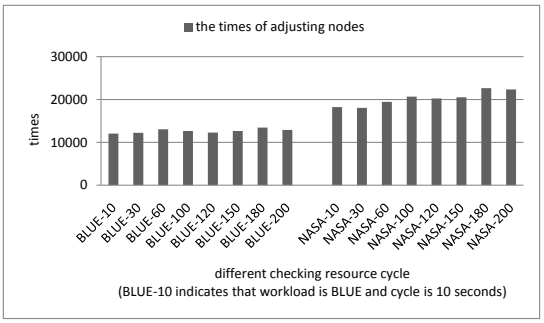
From Fig.18, we can observe that: *S* has small impact on the resource consumption, the number of completed jobs and the service provider's times of adjusting nodes. So we just set S as the same value of the scheduling cycle. Here is 60 seconds for HTC workloads. Taking it into account that the average execution time of tasks in MTC workload is only about 10 seconds, we set *S* as 1 second in MTC.

The effect of time unit of leasing resources.

In DawningCloud, we respectively set *the time unit of leasing resources C* as 10/30/60/90/120 minutes, while other parameters are [40B/1.5R/60S] for HTC workload and [20B/8R/1S] for MTC workload. For the resource



(a) The resource consumption and the number of completed jobs VS. the checking resource cycle for the SDSC BLUE and NASA traces.



(b) The service providers' times of adjusting nodes VS. the checking resource cycle for SDSC BLUE and NASA trace.

Fig. 18

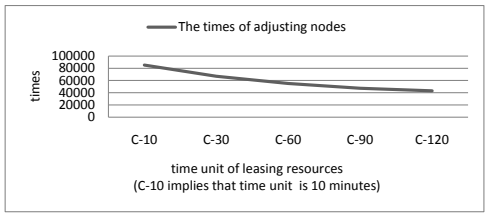


Fig. 19: The management overhead VS. the time unit of leasing resources.

provider, the management overhead in terms of *the accumulated times of adjusting nodes* in DawningCloud are obtained with varying *time units of leasing resources* in Fig.19.

From Fig.19, we have the following observation:

1) The management overhead is inversely proportional to *C*. This is because when the time unit of leasing resources is less, the service provider requests dynamic resources more frequently, which results in larger management overhead that is the sum of all service providers' times of adjusting nodes.

Taking it into account dynamic resources are charged at the granularity of time unit of leasing resources, we make a tradeoff and select *C* as 60 minutes in DawningCloud and Deelman's system. In fact, in EC2 system, resources are also charged at the granularity of one hour.

Implications of Analysis:

At the end of this section, we give some suggestions to the service provider in setting parameters:

1) For HTC workload: the size of initial resources *B* can be set as 1/4 to 1/3 of the configuration size of the dedicated cluster system; the threshold ratio of obtaining dynamic resources *R* can be set as 1.5; the checking resource cycle *S* can be set as 60 seconds.

2) For MTC workload: *B* is set as approximate 1/8 of the configuration size of the dedicated cluster system; *R* need to satisfy the condition of *(B*R < RA)* and *,(RA*R > RM)* where *RA* is *the accumulated resource demand of jobs in queue* in most of the running time and *RM* is *the maximal accumulated resource demand of jobs in queue* in the running time (*RA* and *RM* can be calculated through experiments); *S* can be set as 1 second.