Future Generation Computer Systems 28 (2012) 58–65

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka

Christian Vecchiola[a], Rodrigo N. Calheiros[a], Dileban Karunamoorthy[a], Rajkumar Buyya[a,b,*]

[a] *Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia*
[b] *Manjrasoft Private Limited, Melbourne, Australia*

ARTICLE INFO

Article history:
Received 23 December 2010
Received in revised form
2 May 2011
Accepted 7 May 2011
Available online 14 May 2011

Keywords:
Resource provisioning
Hybrid cloud
QoS
E-Science
Aneka

ABSTRACT

Scientific applications require large computing power, traditionally exceeding the amount that is available within the premises of a single institution. Therefore, clouds can be used to provide extra resources whenever required. For this vision to be achieved, however, requires both policies defining when and how cloud resources are allocated to applications and a platform implementing not only these policies but also the whole software stack supporting management of applications and resources. Aneka is a cloud application platform capable of provisioning resources obtained from a variety of sources, including private and public clouds, clusters, grids, and desktops grids. In this paper, we present Aneka's deadline-driven provisioning mechanism, which is responsible for supporting quality of service (QoS)-aware execution of scientific applications in hybrid clouds composed of resources obtained from a variety of sources. Experimental results evaluating such a mechanism show that Aneka is able to efficiently allocate resources from different sources in order to reduce application execution times.

1. Introduction

Cloud computing [1] platforms are rapidly emerging as the preferred option for hosting applications in many business contexts. Start-up companies are relying on public cloud infrastructures to deploy their applications, which helps in reducing their initial costs. Larger companies are also adopting clouds, either public clouds for expanding their existing infrastructures or rapid deployment of test environments, or private clouds for dynamic on-demand provisioning of virtual resources among their internal divisions.

The same widespread adoption of cloud platforms for application deployment is not yet observed in the case of scientific computing applications. Scientific computing, or computational science, is the field of study concerned with devising mathematical models and numerical techniques aiming to address problems in science and engineering. These problems typically involve long-term computer simulations, huge dataset processing, and other types of large-scale computations which, in many cases, require the availability of large IT infrastructures. Traditionally, these needs have been addressed on the internal computing infrastructure of research institutions, which are typically composed of clusters or resources from local area networks (desktop grids).

Later, with the advent of grid computing [2], larger infrastructures were made available to scientists. The availability of both software and infrastructure for grid computing significantly influenced the growth of scientific computing research. As a result, scientific applications became the major user of grid facilities, and many public grids have been deployed for supporting these applications. However, high utilization rates observed in grids, along with technical and bureaucratic issues, limit their utilization.

Moreover, these resources may be insufficient in certain periods of time. For example, peak demand for scientific resources may be seen in some parts of the year, which can lead to long waiting times for utilization of these resources, or the available resources for one application may be insufficient to complete the application before its deadline. In these cases, scientific resources may be complemented by cloud resources. Moreover, by leasing cloud computing services on a pay-per-use basis, even minor institutions can easily access a large number of resources, which are utilized and paid for only for the time they are actually utilized.

For this vision to be achieved, however, middleware supporting provisioning of resources from both local infrastructures and public clouds (known as hybrid clouds) is required, so that applications can transparently migrate to public virtual infrastructures [3].

Aneka [4] is a software platform for building and managing a wide range of distributed systems, allowing applications to receive

* Corresponding author at: Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia.
E-mail address: rbuyya@unimelb.edu.au (R. Buyya).

resources provisioned from different sources, such as desktop grids, scientific grids, clusters, private clouds, and public clouds. Such a hybrid cloud built with resources from a variety of sources is managed transparently by Aneka. Therefore, this platform enables the execution of scientific applications in hybrid clouds.

The contribution of this paper is twofold. First, it describes Aneka's features that are responsible for supporting the quality of service (QoS)-aware execution of scientific applications in hybrid clouds composed of resources obtained from a variety of sources. Second, it presents experimental results of the utilization of these features in an actual hybrid cloud. Results show that Aneka is able to efficiently allocate resources from different sources in order to reduce application execution times.

The rest of this paper is organized as follows. Section 2 presents an overview of scientific applications and current support for them in the cloud. Section 3 presents a general overview of the Aneka framework. Section 4 describes the provisioning mechanism in Aneka and how it supports different resources. Section 5 presents the algorithms for deadline-driven resource provisioning in Aneka. Section 6 presents the results of using Aneka's capabilities for the execution of a scientific application in a hybrid cloud. Section 7 presents related works, and Section 8 concludes the paper.

2. Scientific computing in the cloud

With the advent of grid computing, large computing infrastructures became available to academic and research institutions for deploying and executing scientific applications. These infrastructures are either contained within the boundaries of an institution, such as a University or a high-performance computing (HPC) facility, or span across several organizations and spread worldwide as in the case of the Enabling Grid for E-sciencE (EGEE) [5], TeraGrid [6], and Open Science Grid (OSG) [7].

Because grid resources are shared worldwide, it is necessary to provide users with controlled access to these resources, which leads to competitive use of these facilities that favors large research institutions and more expensive projects. In addition, technical issues related to the specific nature of the runtime environment provided by grids limit the widespread use of these facilities: in many cases, applications have to be written either according to a specific programming model (e.g., Message Passing Interface [8], Bag of Tasks, or workflows) or for a specific operating system, mostly Unix-like.

Cloud computing provides an alternative approach that solves some of these problems. Such technology delivers IT infrastructure and services on demand on pay-per-use basis. The use of such a billing scheme makes clouds accessible to everyone, from large academic institutions to minor research groups. Moreover, clouds are able to provide distributed systems that grow and shrink dynamically according to the requirements of users, which can identify convenient trade-offs between cost and performance for experiments.

Moreover, virtualization technologies remove many of the technical issues previously mentioned. For example, Infrastructure as a Service (IaaS) solutions allow scientists to prepackage and configure the basic building blocks required for carrying out their experiments. This allows a higher degree of customization that helps cover a wider range of scenarios for scientific computing applications. Finally, provisioning on demand of cloud resources simplifies their integration into existing infrastructures. Academic institutions already have their computing facilities, and these infrastructures can be extended by adding virtual resources or services leased from the cloud, which creates a hybrid infrastructure that serves the institution for the time needed to perform huge computations or large-scale experiments.

Although the use of cloud computing for scientific applications is still limited, the first steps towards this goal have been already made. One of the first cloud-based infrastructures for computational science, Science Cloud [9], has been deployed by the joint efforts of the University of Chicago, the University of Illinois, Purdue University, and Masaryk University.

From a research point of view, studies have been conducted on the feasibility of using clouds for scientific computing. A study by Evangelinos and Hill [10] shows that HPC scientific applications deployed in regular Amazon instances have performance similar to the performance achieved with low-cost clusters, whereas a study by Deelman et al. [11] shows that the cost of cloud for HPC applications is acceptable. From a public cloud provider's perspective, there are initiatives such as Amazon EC2's cluster computing instances,[1] which offer HPC resources connected via a high-throughput network. Such an initiative from Amazon provides a platform with better performance for HPC applications than the regular platform evaluated by Evangelinos and Hill, making the adoption of clouds more compelling to researchers.

The aim of using hybrid public/private clouds is a core feature of the Aneka platform, which enables not only utilization of such clouds, but also utilization of virtually any kind of computational resource available for applications, including idle desktops from local networks, clusters, and grids. The Aneka platform is presented in the next section, and the deadline-driven provisioning of resources for scientific applications is presented in Section 4.

3. Aneka: an extensible cloud application platform

Aneka [4] is a software platform and a framework for the development of distributed applications in the cloud. It harnesses the computing resources of a heterogeneous network of workstations, clusters, grids, servers, and data centers, on demand. It implements a Platform as a Service model, providing developers with Application Programming Interfaces (APIs) for transparently exploiting physical and virtual resources. In Aneka, application logic is expressed with a variety of programming abstractions and a runtime environment on top of which applications are deployed and executed. System administrators leverage a collection of tools to monitor and control the cloud, which can be a public virtual infrastructure available through the Internet, a network of computing nodes in the premises of an enterprise, or a combination of them.

The core feature of the framework is its service-oriented architecture that allows customization of each cloud according to the requirements of users and applications. Services are also the extension point of the infrastructure: by means of services, it is possible to integrate new functionalities and to replace existing ones with different implementations. In this section, we briefly describe the architecture and categorize the fundamental services that compose the infrastructure.

3.1. Framework overview

Fig. 1 provides a layered view of the framework. Aneka provides a runtime environment for executing applications by leveraging the underlying infrastructure of the cloud. Developers express distributed applications by using the API contained in the Software Development Kit (SDK) or by porting existing legacy applications to the cloud. Such applications are executed on the Aneka cloud, represented by a collection of internetworked nodes hosting the Aneka Container. One of the nodes runs the *Aneka master*, which provides resource management and application scheduling capabilities, and the other nodes run *Aneka workers* that process tasks that compose the application.

[1] http://aws.amazon.com/ec2/.

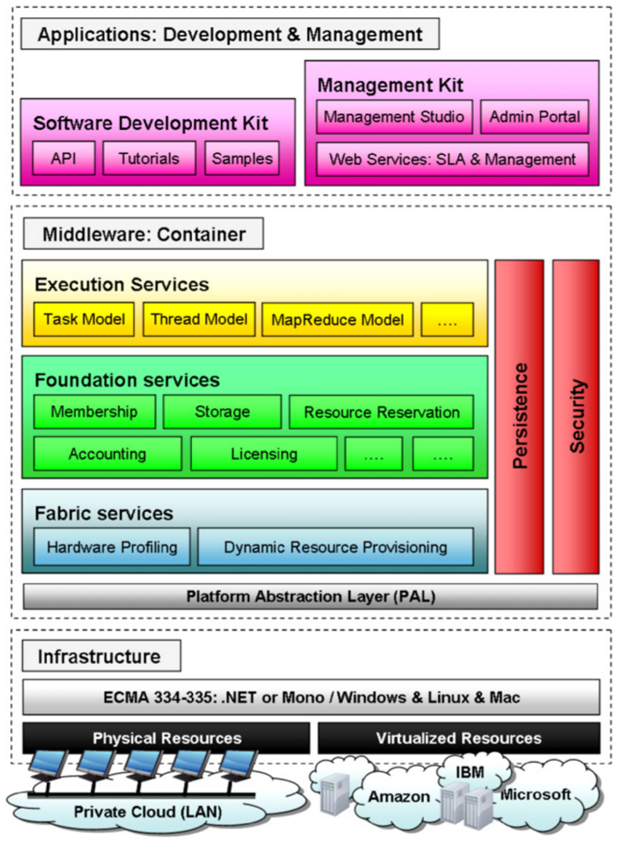

Fig. 1. Aneka framework.

Aneka Container is the building block of the middleware, and it represents the runtime environment for executing applications; it contains the core functionalities of the system and is constituted of an extensible collection of services that allow administrators to customize the Aneka cloud. There are three classes of services that characterize the Container.

- *Execution services*: these services are responsible for scheduling and executing applications. Each programming model supported by Aneka defines specialized implementations of these services for managing the execution of a work unit defined in the model.
- *Foundation services*: these services are the core management services of the Aneka Container. They are in charge of metering applications, allocating resources, managing the collection of available nodes, and keeping the services registry updated.
- *Fabric services*: these services constitute the lowest level of the services stack of Aneka and provide access to the resources managed by the cloud. An important service in this layer is the Resource Provisioning Service, which enables horizontal scaling (e.g., increase and decrease in the number of resources) in the cloud. Resource provisioning makes Aneka elastic and allows it to grow and shrink dynamically to meet the QoS requirements of applications.

The Container relies on a Platform Abstraction Layer that interfaces with the underlying host, whether this is a physical or a virtualized resource. This makes the Container portable over different runtime environments that feature an implementation of the ECMA 335 (Common Language Infrastructure) specification [12]. Two well-known environments that implement such a standard are the Microsoft.NET framework[2] and the Mono open source.NET framework.[3] Therefore, Aneka is fully supported in both Windows-based and Linux-based systems.

Security in Aneka is carried out by specific Aneka services. The same pluggable interface that allows new services to be added to the container also allows different security mechanisms to be attached to the Aneka container, so that the security requirements of specific installations can be served. For example, if secure and encrypted file transferring is required between the master and the workers, a Secure File Transfer Protocol (SFTP) module can be plugged into Aneka and activated on demand. Regarding communication among nodes, Aneka provides a built-in security based on 256-bit symmetric cryptography. Moreover, the framework implements its own security system, allowing users to selectively access services.

Scalability in Aneka is handled by elastically increasing the installed base of nodes when required: the framework was successfully used to harness and manage up to 200 nodes without significant delays or bottlenecks. Regarding management of applications, Aneka was successfully used for hosting the execution of multiple applications at the same time and to handle the execution of massive applications composed by tens of thousands (between 10,000 and 20,000) tasks. In a worst-case scenario, some bottlenecks were identified in the data transfer facilities, which can be avoided with replication of data sources in order to distribute requests for data required by tasks.

3.2. Aneka application programming models

Developers express the logic of applications by using *programming models*. A programming model is represented by a collection of abstractions that are used by a developer to define the application components, and a set of services that constitute the runtime environment for these abstractions. Generally, the implementation of a programming model includes execution and scheduling services whose coordinated activity provide runtime support for the model. The programming models currently supported by Aneka are as follows.

- *Bag of Tasks*: allows expressing a distributed application as a collection of independent tasks that can be executed in any order.
- *Distributed Threads*: provides an implementation of concurrent threads with shared memory and locks.
- *MapReduce*: implements the programming model proposed by Google [13] for data-intensive applications based on the *map* and *reduce* functions borrowed from functional programming.
- *Actors*: supports the SALSA [14] programming language for the execution of Actors-oriented programs on top of Aneka clouds.
- *Workflow*: allows expressing a distributed application as a collection of interrelated tasks whose dependencies are expressed by a directed acyclic graph.

Additional programming models can be integrated into the infrastructure if the required implementations of the abstractions are provided and the runtime services required for their execution are added to the Container.

4. Resource provisioning in Aneka

Aneka provides seamless access to different computational resources such as Linux desktops, Windows desktops, grids, clusters, private clouds, and public clouds. The Resource Provisioning Service, which is part of the Fabric Services of Aneka, enables dynamic provisioning of various types of resources [3]. Next, we present how each type of resource is supported in Aneka.

2 http://www.microsoft.com/net/.

3 http://www.mono-project.com.

4.1. Desktop grids

Desktop grid resources are examples of Aneka static resources. In the context of Aneka, a static resource is a resource that receives a static configuration and thus can be promptly accessed by Aneka and become part of the Aneka private cloud [3]. This allows them to be provisioned to applications without further interaction with other Aneka architectural elements. The only requirements for this type of resource is being accessible via a network, having a configuration that allows remote access from, and running, the Aneka container.

This type of resource can be used by Aneka either opportunistically, i.e., they are made available when they are not in use by local users, or in a dedicated way, i.e., they are accessed only via Aneka and are available as part of the Aneka cloud at all times. These methods allow Aneka to access resources from both Linux and Windows desktops.

4.2. Public and private clouds

Public and private cloud resources are provisioned dynamically by Aneka. Dynamic resources, in the context of Aneka, are resources that do not receive a static configuration and thus join the Aneka cloud on demand [3]. These resources are the result of dynamic provisioning, which is managed by the Resource Pool Manager. The Resource Pool Manager is responsible for managing resource pools, which are composed of resources obtained from the same source. Because different resources have different management systems and are accessed and used in different ways, the operation and management of Resource Pools vary depending on the specific source of resources.

In the case of private clouds, local data centers, and other organizational resources allocated in the form of Virtual Machines (VMs), there is one Resource Pool able to support each specific data center/cluster. For example, if the company has one virtualized cluster running Xen, another one running VMware, and a data center managed via Eucalyptus software, three Resource Pools have to be instantiated, and each one is managed according to the particularities of each resource type.

In the case of private clouds, it is also necessary to have one Resource Pool per provider, because typically each provider deploys its own API and access tools, and they are incompatible with other provider's tools and APIs.

In both cases, resources are provisioned in the form of virtual machines that once started activate an Aneka container previously configured and stored in the VM image. The Aneka container in the initiated VM triggers the process of joining the Aneka cloud.

4.3. Grids and clusters

Aneka is also able to provision resources from typical scientific computing infrastructures, such as grids and clusters. How these resources are made available to Aneka depends on whether virtualization technology is applied in the grid/cluster or not.

If the grid or cluster is also virtualized, its utilization is similar to the utilization of clouds: a Resource Pool able to make allocations in the specific infrastructure must be available. Also similar to the approach for clouds, a VM image with a preconfigured Aneka container is required, so VMs join the Aneka cloud once they are initiated in the grid/cluster. Once the resource is available to Aneka, it is used similarly to any other type of resource.

If the grid or cluster does not support virtualization technology, its utilization in Aneka is slightly different. Even though VMs are not created in the resource, these resources still require allocation, deallocation, and management, and thus a Resource Pool having ability to interact with the Resource Management System of the grid/cluster is required. Once resources are allocated by the Resource Pool, an Aneka container is loaded in each allocated node so these resources join Aneka cloud like desktop grid resources.

One key difference between clouds and grids (and clusters) is that the latter requires definition of an allocation time during resource acquisition. This is because in grids and clusters the period of time in which the resources are required is typically decided before allocation. On clouds, on the other hand, allocation time is not previously decided, and thus Resource Pools release resources only when they are not required. Poor decisions of providing allocation time in clusters and grids may cause applications to be canceled during their execution if they do not finish before the allocation time expires. Therefore, Resource Pools for these resources have to consider this extra issue during resource allocation.

5. Deadline-driven resource provisioning algorithm

Fig. 2 presents an overview of Aneka's provision mechanism. Besides aspects related to the management of resources from different sources, other important aspects to be considered are (i) when such a process of resource allocation takes place and how many resources are requested, and (ii) which of the available resource sources are used in a particular provisioning request.

A decision about the former is made by the Scheduler Service. This decision is driven by the application QoS, which is expressed in terms of the deadline for application completion. The deadline-driven policy is a best-effort algorithm that considers the time left for the deadline and the average execution time of tasks that compose an application to determine the number of resources required by it, as shown in Algorithm 1. For each request with QoS constraints, the Scheduler Service considers its deadline, number of tasks, and task runtime estimation to determine if the deadline can be met (Line 4). If the Service detects that the deadline cannot be met, it determines the number of extra resources required and submits a request, containing the request and the number of resources, to the Provisioner Service (Lines 5–7).

Notice that this process takes place either to scale a single application across multiple sites or to provide dynamic resources to multiple applications in execution in the Aneka cloud, and the process is repeated every time a new request is received by Aneka and every time a task completes or fails. On the other hand, if the Scheduling Service detects that the request does not require all the resources currently allocated to it, it indicates to the Provisioner Service that some of these resources can be released to be used by other requests (Lines 8–21).

For a decision about the specific source of resources to be used for each resource request, Algorithm 2 presents the algorithm applied by the Provisioning Service, which is called in Algorithm 1, Line 7. By default, resources are allocated first from local static and dynamic resources (e.g., desktop grids and private clouds), then from "free" resource pools (such as grids), and finally from public clouds. Inside each of these classes of resource sources (e.g., different public Cloud providers in the case of paid external resources), the source of resources and the preferred order is defined by the Aneka administrator in an input configuration file. Notice that this algorithm operates in a best effort manner: if the provisioner cannot allocate the exact number of resources requested by the scheduler, it returns as much as it can get from all the available sources (Line 16). Moreover, the time for preparation of the system (e.g., start up of VMs) is not taken into account by the provisioning mechanism, and thus there may be small delays in task processing.

Other policies, which consider data locality and monetary and performance costs of file transferring, have currently to be defined

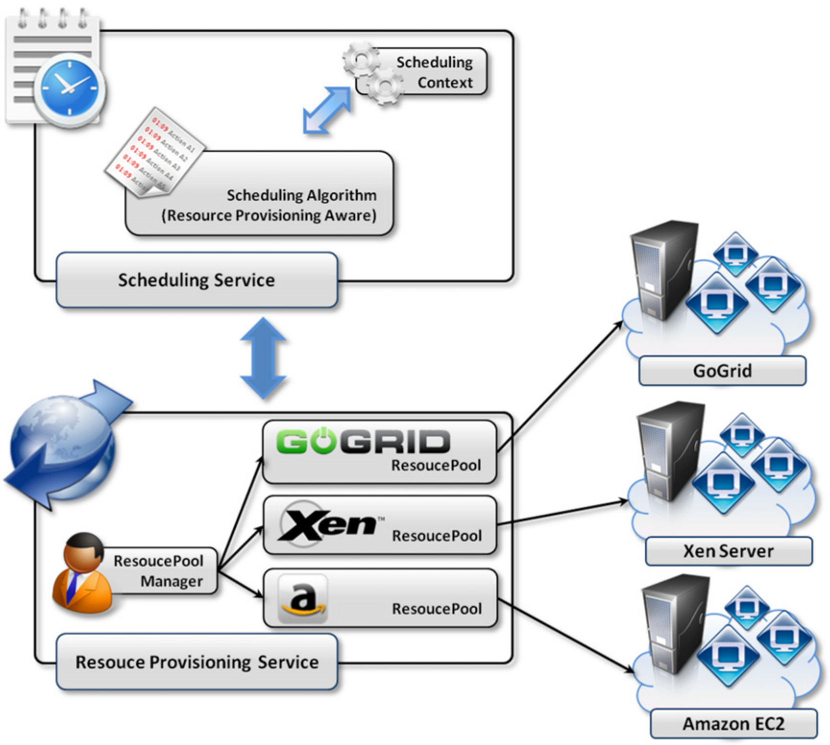

Fig. 2. An overview of Aneka's dynamic provisioning infrastructure.

Algorithm 1: Deadline-driven provisioning in Aneka.

```
1  foreach request with QoS constraints do
2      resources ← available resources for the application;
3      pendingTasks ← number of tasks in the queue;
4      $eft \leftarrow \frac{pendingTasks}{resources} \times averageTaskRuntime$;
5      if $eft > applicationTimeRemaining$ then
6          $extraResources \leftarrow \frac{pendingTasks \times averageTaskRuntime}{applicationTimeRemaining}$;
           // Invokes Algorithm 2 for resource
              provisioning
7          Provisioner.selectResources(applicationId,
           extraResources);
8      else
9          toRelease ← 0;
10         if pendingTasks<resources then
11             toRelease ← pendingTasks − resources;
12         end
13         else
14             pendingTasks ← pendingTasks + runningTasks;
15             $eft \leftarrow \frac{pendingTasks}{resources} \times averageTaskRuntime$;
16             if $eft < applicationTimeRemaining$ then
17                 toRelease ←
                   $resources - \lceil \frac{pendingTasks \times averageTaskRuntime}{applicationTimeRemaining} \rceil$;
18             end
19         end
20         Provisioner.releaseResources(applicationId,
           toRelease);
21     end
22 end
```

Algorithm 2: Resource pool selection in Aneka.

```
1  Provisioner.selectResources(applicationId,
   extraResources)
2      providerList ← list of local static resource pools;
3      providerList ← providerList ∪ list of local dynamic
       resource pools;
4      providerList ← providerList ∪ list of remote free resource
       pools;
5      providerList ← providerList ∪ list of remote paid
       resource pools;
6      req ← extraResources;
7      resourceList ← ∅;
8      foreach resourcePool rp in providerList do
9          av ← available resources from rp;
10         resourceList ←
           resourceList ∪ rp.allocate(min(req, av));
11         if |resourceList| = extraResources then
12             break;
13         end
14         req ← req − av;
15     end
16     Scheduler.addResources(applicationId, resourceList);
17 end
```

by the system administrator, though these policies are planned to be automatically provided by Aneka in the future.

Once dynamic resources join the Aneka cloud, they must be properly managed, since these resources are typically subject to a usage cost. In particular, current practices for billing by use of cloud resources consider their usage in terms of time blocks whose granularity varies among providers. For example, in Amazon EC2, the value of the time block is set to one hour. In this case, there is no advantage in shutting down the virtual resources before the time block expiration time. Therefore, even if the application that first required the external resources already released it, they are kept in the pool, and thus they are made available for other requests that need extra resources, until the current time block expires.

Other issues that are relevant to the hybrid provisioning mechanism, such as cross-boundary security, load balance, and data transfer, are managed by different components of the Aneka container. Cross-boundary security is enforced by specific security modules that execute the Aneka container in confined sandboxes on remote resources, so the application is protected against unauthorized access by other applications. The cross-boundary load balance is handled by the Scheduler Service, which ensures that workers receive new tasks to run whenever they become idle, regardless of whether the corresponding resources are locally or remotely provisioned. Cross-boundary data transfer is secured via security modules defined by users, which might be in this case either secure copy (SCP) or SFTP.

6. Performance evaluation

In this section, we present an experiment to demonstrate Aneka's features supporting the execution of scientific applications in hybrid Cloud environments. The hybrid cloud used in the experiment is shown in Fig. 3.

Static resources are part of a private cloud containing five Linux machines (one master node and four worker nodes). Dynamic resources are composed of EC2 resources provisioned when local resources are not able to meet the application deadline. The EC2 resource pool policy in use allocates only Large Standard Amazon EC2 instances, which are dual core machines with 1.7 GB of RAM memory costing US$0.17 per hour. These instances have similar capacity, regarding the number of cores and processor speed, to that of the local machines used in this experiment.

Because local resources are part of the local infrastructure and are supplied in a best-effort basis (i.e., they can leave the Aneka cloud at any time if there is a local user accessing the machine) and thus they offer limited QoS for the task's execution, their usage cost is not considered in this experiment.

The scientific application executed in the cloud is a multi-objective evolutionary optimizer called EMO [15]. EMO is based on Genetic Algorithms [16] and its initial implementation is single threaded [15]. Subsequently, a Bag of Tasks version of EMO has been developed [17]. Such a version addresses the issue of reducing the computation time for non-trivial problems running in grids and other HPC platforms. The application suits the purpose of the experiment since it is a CPU-intensive task that generates a small amount of data in terms of output files (less than 1 MB).

To evaluate Aneka's deadline-driven provisioning mechanism, we first submitted distributed EMO to execution in Aneka without setting a deadline for the application. Later, we repeated execution of the same scenario with different deadlines, showing how the provisioning infrastructure behaves. By running the first experiment, we can estimate the expected execution time of the application, which will allow us to impose a deadline triggering the resource provisioning in the other experiments.

One single execution of EMO in our experiment generates a job composed of 40 tasks. The first test identified that the execution of a single task (the evaluation of the DLTZ6 benchmark function with a population of 300 individuals and an evolutionary process of 150 generations) takes on average 6 min and 49 s. This means that a possible upper bound for the execution by only leveraging static resources is 70 min (when each of the four worker nodes executes ten tasks sequentially).

The test was then repeated considering different deadlines. The results of various execution scenarios are presented in Table 1. Aneka's provisioning algorithm is able to provision resources and schedule the execution of the application in time for larger deadlines, with a small budget spent to allocate resources from EC2. As the deadline becomes stricter, the estimation of execution time is more important for successfully executing the application, because underestimation makes the provisioning allocate fewer resources than are really necessary. Moreover, the time required by the virtual resources to join the Aneka cloud also affects the results. In particular, our experiments revealed that the delay for initialization of VMs provisioned from EC2 is on average 90 s, and it varies according to the number of resources requested.

Therefore, our conclusion is that both the deployment time in external resources and delays caused by particularities of the environments have to be considered when providing the Aneka scheduler with an estimation of the execution time of tasks in the cloud. Historical information about previous executions of the application, even if in a different infrastructure, together with the size of the transferred files and network details can help in providing a better estimation of the application runtime.

Nevertheless, even without considering such a deployment delay, the Aneka scheduler is able to reduce the application execution time by delivering resources from different sources, which enables execution of the application with small violation of the deadline. With mechanisms for better estimation of the execution time of tasks, it is expected that there will be an improvement in the performance of scientific applications in hybrid clouds.

7. Related work

The idea of using public clouds to enhance the capability of grid resources has been explored theoretically in different works. Assunção et al. [18] present a simulation-based analysis of different algorithms for provisioning of resources both in a local cluster and in the cloud. Such an analysis is based on common grid and cluster workloads. Kondo et al. [19] present a cost-analysis study of mixed cloud and desktop grid environments for high-throughput, CPU-intensive applications. Such a study shows that hybrid approaches where servers for the desktop grid are hosted in the cloud enable savings in infrastructure costs.

Regarding actual implementations of systems supporting hybrid clouds for scientific applications, CometCloud [20] is an autonomic engine for hybrid grids and cloud systems, which supports the execution of workflow applications. Aneka, on the other hand, provides support for different programming models such as workflow, MapReduce, threads, and Actors-oriented programming. Moreover, it can also exploit resources from idle desktop machines, including those running the Windows operating system.

The ASKALON grid environment has been extended [21] to support the execution of workflow applications in both grids and clouds (either public or private). The CaGrid Workflow Toolkit [22] performs discovery, data access, service invocation, and execution of workflows in multiple types of resource. Both systems support only workflow applications and limited types of resource, whereas Aneka supports different programming models and computing environments.

GridWay [23] supports the execution of applications both in local grids and in different cloud providers with the help of Globus Nimbus.[4] It supports any type of local resource that can be managed by the Globus middleware, and also supports programming models supported by the latter. Therefore, both GridWay and Aneka are able to provision any type of resource to applications, even though Aneka supports more application models than GridWay.

Finally, Elastic Site Manager [24] is a resource manager that is able to dynamically provision resources from private and public clouds to scientific applications. OpenNebula combined with Haizea [25] supports the dynamic provision of virtualized resources from private and public Clouds. Resources managed by these systems are virtualized resources only, whereas Aneka is able to leverage applications with both virtualized and non-virtualized resources simultaneously, due to its provisioning capabilities.

[4] http://workspace.globus.org.

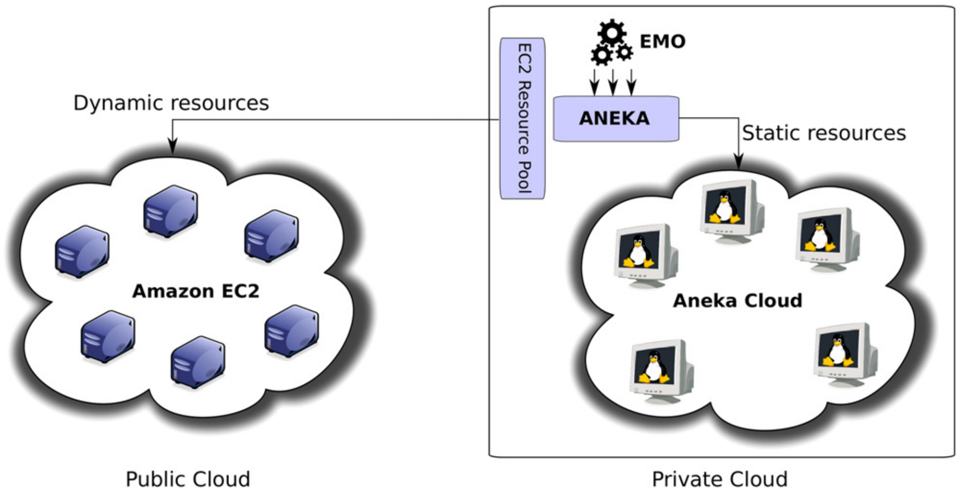

Fig. 3. Hybrid cloud used in the experiments.

Table 1
Results of the execution of a Bag of Tasks EMO job in a hybrid Aneka cloud.

Deadline	Static resources	EC2 VMs leased	Execution time (min)	Public cloud usage cost (U$)
No deadline	4	0	70	0.00
60 min	4	5	33.3	0.85
40 min	4	6	31	1.02
30 min	4	7	31	1.19
20 min	4	11	25	1.87

8. Conclusions and future directions

Cloud Computing quickly became the platform of choice in many practical scenarios in a business context. Nevertheless, its adoption is limited in the context of computational science. Scientific applications requiring larger amounts of computing power than can be delivered by local resources within a given time frame can utilize clouds, which can deliver this required capacity with minimal effort in terms of the configuration of hardware platforms.

An obstacle for the adoption of clouds for scientific applications is taking advantage of such platforms when legacy systems are still used. Different operating systems, programming languages, and software platforms supported by each system can make this integration hard.

Aneka addresses these issues by supporting seamless integration of resources from a range of sources that include desktop grids, clusters, grids, public clouds, and private clouds to support QoS-aware execution of applications. Aneka's features were demonstrated in experiments that showed that it is able to efficiently allocate resources from different sources in order to reduce application execution times. Improvements in Aneka's dynamic resource provisioning are under development, and once these improvements are available we expect that applications will run more efficiently in hybrid resources.

The experiment presented in this paper addressed the case of applications requiring a small amount of data transfer: the input files, output files, and application together were smaller than 1 MB. Provisioning mechanisms more suitable for data-intensive HPC applications – such as data location-aware provisioning of hybrid resources, which attempts to select providers that contains all or part of the data required by the application – are also the subject of future research.

We are developing support for the integration of multiple Clouds in Aneka according to the InterCloud [26] model. In this model, providers interact via a marketplace where they can either negotiate resources for serving their jobs or they can outsource jobs to other Clouds upon a compensation to the party receiving the job. This will further expand the range of different sources of resources that can be integrated by Aneka, leading to its ultimate goal of supporting QoS-aware execution of applications using any relevant programming model.

References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering IT services as the 5th utility, Future Generation Computer Systems 25 (6) (2009) 599–616.
[2] I. Foster, K. Kesselman (Eds.), The Grid 2: Blueprint for a New Computing Infrastructure, 2nd ed., Morgan Kaufmann Publishers, 2004.
[3] C. Vecchiola, X. Chu, M. Mattess, R. Buyya, Aneka — integration of private and public clouds, in: R. Buyya, J. Broberg, A. Goscinski (Eds.), Cloud Computing: Principles and Paradigms, Wiley Press, 2011.
[4] C. Vecchiola, X. Chu, R. Buyya, Aneka: a software platform for. NET-based cloud computing, in: W. Gentzsch, L. Grandinetti, G. Joubert (Eds.), High Performance and Large Scale Computing, IOS Press, 2009.
[5] F. Gagliardi, M. Begin, EGEE—providing a production quality grid for e-Science, in: Proceedings of the IEEE International Symposium on Mass Storage Systems and Technology, 2005.
[6] C. Catlett, TeraGrid: a foundation for US cyberinfrastructure, in: Proceedings of the International Conference on Network and Parallel Computing, 2005.
[7] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Wurthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, R. Quick, The open science grid, Journal of Physics: Conference Series 78 (1) (2007) 12–57.
[8] W. Gropp, E. Lusk, A. Skjellum, Using MPI—2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation), The MIT Press, 1999.
[9] K. Keahey, T. Freeman, Science clouds: early experiences in cloud computing for scientific applications, in: Proceedings of the Cloud Computing and its Applications Conference, 2008.
[10] C. Evangelinos, C. Hill, Cloud computing for parallel scientific HPC applications: feasibility of running coupled atmosphere–ocean climate models on Amazon's EC2, in: Proceedings of the Cloud Computing and Its Applications Conference, 2008.
[11] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, The cost of doing science on the cloud: the Montage example, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, 2008.
[12] J. Miller, S. Ragsdale, The Common Language Infrastructure Annotated Standard, Addison Wesley, 2004.
[13] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: Proceedings of the 6th Symposium on Operating Systems Design and Implementation, 2004.

[14] C. Varela, G. Agha, Programming dynamically reconfigurable open systems with SALSA, ACM SIGPLAN Notices 36 (12) (2001) 20–34.
[15] M. Kirley, R. Stewart, Multiobjective evolutionary algorithms on complex networks, in: Proceedings of 4th International Conference Evolutionary Multi-Criterion Optimization, 2007.
[16] K. Deb, Multi-Objective Optimization Using Evolutionary Algorithms, John Wiley & Sons, 2001.
[17] C. Vecchiola, M. Kirley, R. Buyya, Multi-objective problem solving with offspring on enterprise clouds, in: Proceedings of the 10th International Conference on High-Performance Computing in Asia-Pacific Region, 2009.
[18] M.D. de Assunção, A. di Costanzo, R. Buyya, A cost-benefit analysis of using cloud computing to extend the capacity of clusters, Cluster Computing 13 (3) (2010) 335–347.
[19] D. Kondo, B. Javadi, P. Malecot, F. Cappello, D.P. Anderson, Cost-benefit analysis of cloud computing versus desktop grids, in: Proceedings of the 18th International Heterogeneity in Computing Workshop, 2009.
[20] H. Kim, Y. el Khamra, S. Jha, M. Parashar, Exploring application and infrastructure adaptation on hybrid grid–cloud infrastructure, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, 2010.
[21] S. Ostermann, R. Prodan, T. Fahringer, Extending grids with cloud resource management for scientific computing, in: Proceedings of the 10th IEEE/ACM International Conference on Grid Computing, 2009.
[22] W. Tan, R. Madduri, A. Nenadic, S. Soiland-Reyes, D. Sulakhe, I. Foster, C. Goble, CaGrid Workflow Toolkit: a taverna based workflow tool for cancer grid, BMC Bioinformatics 11 (1) (2010) 542.
[23] C. Vázquez, E. Huedo, R.S. Montero, I.M. Llorente, Dynamic provision of computing resources from grid infrastructures and cloud providers, in: Proceedings of the Workshops at the Grid and Pervasive Computing Conference, 2009.
[24] P. Marshall, K. Keahey, T. Freeman, Elastic site: using clouds to elastically extend site resources, in: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010.
[25] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, Virtual infrastructure management in private and hybrid clouds, Internet Computing 13 (5) (2009) 14–22.
[26] R. Buyya, R. Ranjan, R.N. Calheiros, InterCloud: utility-oriented federation of cloud computing environments for scaling of application services, in: Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing, 2010.

Christian Vecchiola is Research Fellow at the Cloud Computing and Distributed Systems Laboratory (CLOUDS Lab) in the Department of Computer Science and Software Engineering, at The University of Melbourne, Australia. His primary research interests include Grid/Cloud Computing, Distributed Evolutionary Computation, and Software Engineering. Since he joined the CLOUDS Lab he has focused his research activities and development efforts on two major topics: middleware support for Cloud/Grid Computing and distributed support for evolutionary algorithms.

Christian completed his Ph.D. in 2007 at the University of Genova, Italy, with a thesis on providing support for evolvable Software Systems by using Agent Oriented Software Engineering. During his Ph.D. studies he worked under the supervision of Prof. Antonio Boccalatte in the Department of Communication Computer and System Sciences, and he was actively involved in the design and the development of the AgentService that is a software framework for developing distributed systems based on Agent Technology. Dr. Vecchiola also investigated the advantages of providing support for agent-based development at a programming language level by extending the object oriented language with abstractions for representing the key elements of the agent computing model. This has gave him the opportunity to cultivate another of his research interests that is represented by Programming Languages and Compiler Technology.

Rodrigo N. Calheiros is a Research Fellow in the Cloud Computing and Distributed Systems Laboratory (CLOUDS Lab) in the Department of Computer Science and Software Engineering, University of Melbourne, Australia. He completed his Ph.D. degree in Computer Science in 2010 at PUCRS, Brazil, and his M.Sc. degree in 2006 at the same University. His research interests include Cloud Computing and the simulation and emulation of distributed systems, with emphasis on grids and clouds.

Dileban Karunamoorthy is a Research Fellow at the Cloud Computing and Distributed Systems Laboratory (CLOUDS Lab) in the Department of Computer Science and Software Engineering, at the University of Melbourne, Australia, where he contributes to the ongoing research and development of Aneka. He designed and developed several key features in Aneka, including master failover, licensing and discovery services, security, management, monitoring and reporting. Mr. Karunamoorthy has an M.Sc. degree in Distributed Computing from the University of Melbourne. His primary area of interest lies in distributed computing, with specific interests in the algorithms and principles for scalable, fault-tolerant, real-time and high-performance systems, covering a broad range of application areas such as cloud and grid computing, high-performance computing, distributed real-time and embedded systems, P2P, mobile and ad hoc computing.

Rajkumar Buyya is Professor of Computer Science and Software Engineering; and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft., a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored 350 publications and four text books. He also edited several books including "Cloud Computing: Principles and Paradigms" recently published by Wiley Press, USA. He is one of the highly cited authors in computer science and software engineering worldwide (h-index = 53, g-index = 114, 15000 + citations).

Software technologies for Grid and Cloud computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. Dr. Buyya has led the establishment and development of key community activities, including serving as foundation Chair of the IEEE Technical Committee on Scalable Computing and five IEEE/ACM conferences. These contributions and international research leadership of Dr. Buyya are recognized through the award of "2009 IEEE Medal for Excellence in Scalable Computing" from the IEEE Computer Society, USA. Manjrasoft's Aneka Cloud technology developed under his leadership has received "2010 Asia Pacific Frost & Sullivan New Product Innovation Award".