Future Generation Computer Systems 28 (2012) 861–870

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

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

# The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds

Rodrigo N. Calheiros[a], Christian Vecchiola[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 1 June 2010
Received in revised form
4 July 2011
Accepted 21 July 2011
Available online 29 July 2011

*Keywords:*
Cloud computing
Resource provisioning
Hybrid Clouds
Spot market
Aneka

ABSTRACT

Cloud computing alters the way traditional software systems are built and run by introducing a utility-based model for delivering IT infrastructure, platforms, applications, and services. The consolidation of this new paradigm in both enterprises and academia demanded reconsideration in the way IT resources are used, so Cloud computing can be used together with available resources. A case for the utilization of Clouds for increasing the capacity of computing infrastructures is Desktop Grids: these infrastructures typically provide best effort execution of high throughput jobs and other workloads that fit the model of the platform. By enhancing Desktop Grid infrastructures with Cloud resources, it is possible to offer QoS to users, motivating the adoption of Desktop Grids as a viable platform for application execution. In this paper, we describe how Aneka, a platform for developing scalable applications on the Cloud, supports such a vision by provisioning resources from different sources and supporting different application models. We highlight the key concepts and features of Aneka that support the integration between Desktop Grids and Clouds and present an experiment showing the performance of this integration.

## 1. Introduction

Cloud computing [1] led to an innovative approach in the way in which IT infrastructures, applications, and services are designed, developed, and delivered. It fosters the vision of any IT asset as a utility, which can be consumed on a pay-per-use basis like water, power, and gas. This vision opens new opportunities that significantly change the relationship that enterprises, academia, and end-users have with software and technology. Cloud computing promotes an on-demand model for IT resource provisioning where a resource can be a virtual server, a service, or an application platform.

Three major service offerings contribute to defining Cloud computing: *Infrastructure-as-a-Service (IaaS)*, *Platform-as-a-Service (PaaS)*, and *Software-as-a-Service (SaaS)*. Infrastructure-as-a-Service providers deliver on-demand components for building IT infrastructure such as storage, bandwidth, and most commonly virtual servers, which can be further customized with the required software stack for hosting applications. Platform-as-a-Service providers deliver development and runtime environments for applications that are hosted on the Cloud. They allow abstraction of the physical aspects of a distributed system by providing a scalable middleware for the management of application execution and dynamic resource provisioning. Software-as-a-Service providers offer applications and services on-demand, which are accessible through the Web. SaaS applications are multi-tenant and are composed by the integration of different components available over the Internet.

The offer of different models on which computing resources can be rented creates new perspectives on the way IT infrastructures are used, because Cloud offers the means for increasing IT resource availability whenever necessary, by the time these resources are required, reducing costs related to resource acquisition and maintenance.

A case for exploring such a feature of Clouds is in Desktop Grids, which are platforms that use idle cycles from desktop machines to achieve high-throughput computing [2]. Typically, applications are executed in such platforms on a best-effort basis, as no guarantees can be given about the availability of individual machines that are part of the platform. If Desktop Grid resources are combined with Cloud resources, a better level of confidence about resource availability can be given to users, and so it is possible to offer some QoS guarantees related to the execution time of applications at a small financial cost.

Aneka [3] is a Platform-as-a-Service implementation providing a middleware for the development and deployment of applications in private or public Clouds. Unlike earlier frameworks, such as

* Corresponding author at: Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia. Tel.: +61 3 8344 1335; fax: +61 3 9348 1184.
*E-mail address:* raj@csse.unimelb.edu.au (R. Buyya).

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2011.07.005

Condor [4] and ProActive [5], Aneka is the first middleware technology to provide a service-oriented container framework that allows realization of different programming models and enables seamless integration of enterprise computing resources (e.g., Desktop Grids and servers) with public Cloud resources. The Aneka framework has been used to implement various programming models including Thread, Task, MapReduce, and deployment of their applications on private and public Clouds seamlessly. It is important to note that Aneka is the first Cloud application platform supporting market-oriented resource provisioning for optimal leasing of resources from public Clouds to minimize the cost for consumers of Aneka applications.

In a previous work [6], we discussed how Aneka supports scientific applications by allowing provisioning of resources from typical scientific IT infrastructures including Grids and Clouds. This paper extends such a discussion in the context of Desktop Grids, providing a thoroughly description of how Aneka goes beyond what current Desktop Grid platforms offer to users by supporting hybrid Desktop Grids and Clouds and supporting different application models. In summary, *the key contributions of this paper* are:

- It presents the key components of Aneka, its service-oriented container architecture that allows realization of different programming models and enables seamless integration of enterprise computing resources (e.g., Desktop Grids and servers) with public Cloud resources;
- It proposes a new provisioning algorithm that combines Desktop Grids and public Cloud resources in order to execute distributed applications within a user-defined deadline;
- It demonstrates experimentally Aneka and its new provisioning algorithm's ability to dynamically lease resources to meet deadlines with fewer public Cloud machines and with less budget expenditure than existing approaches by utilizing Amazon EC2 Spot Instance resources.

The rest of the paper is organized as follows: Section 2 presents systems that are related to Aneka; Section 3 describes the general architecture of Aneka and its major components; Section 4 describes how Aneka supports Desktop Grids and how Aneka scales these platforms with Cloud resources with the use of Spot Instances; Section 5 presents experiments aiming at assessing the performance of the middleware when running applications. Finally, Section 6 presents conclusion and further works.

## 2. Related work

Condor [4] is a Desktop Grid system which was later expanded to support Grid applications. It allows formation of local pools of resources that can exchange resources with other pools, likewise typical Grid middleware. XTremWeb [7] is a Desktop Grid system that allows execution of unmodified applications in idle desktops. Aneka, on the other hand, can manage resources from multiple sources such as desktops, clusters, Grids, and Clouds, both physical and virtual to execute applications with SLA. Moreover, because Aneka was designed to support Cloud environments, management of budget related to execution of applications is performed by the platform itself, while the other approaches cannot manage this aspect because they were designed for Grid environments, that are typically based on cooperation rather economical benefit.

ProActive [5] is a middleware for parallel, distributed, and multi-core programming. It provides a set of APIs for development of distributed implementations of different applications, including embarrassingly parallel applications, Monte-Carlo simulations, SPMD, and workflow applications. A distinctive feature of ProActive with respect to the previous frameworks is its ability to harness virtual resources from Cloud computing infrastructures in addition to desktop machines, clusters, and Grids. Even though ProActive supports different programming models and resources, it is based on Java architecture and features (such as RMI and classes) for supporting distributed execution of applications, while Aneka provides a service-oriented container model, which allows creation of multiple programming models using its service architecture. As a result of this, Aneka framework has been used to implement various programming models including Thread, Task, MapReduce, and deployment of applications created using these models on private and public Clouds. In addition, Aneka is the first Cloud application platform to support market-oriented resource provisioning for optimal leasing of resources from public Clouds to minimize the cost for consumers of Aneka applications.

BOINC [8] is a framework for volunteer and Grid computing. It allows turning desktop machines into volunteer computing nodes that are leveraged to run jobs when such machines become inactive. Because it targets volunteer desktops, it strongly relies on tasks replication to achieve reliability in the results. Moreover, job execution is based in best-effort, and few guarantees are given about deadlines for application execution. The later can be achieved by Aneka, because it can provision resources from reliable sources to complement desktop resources if deadlines are not being achieved.

Falkon [9] is a task execution framework. Together with Swift, a parallel scripting language for scientific applications, it supports the execution of many-task computing applications. It has been mostly designed to support the execution of scientific workflows and parameter sweeping applications and has peta-scale scientific environments as its target infrastructure. Aneka, on the other hand, has been designed to support both scientific and enterprise applications, and it targets private Clouds as the primary infrastructure. Therefore, requirements from both systems are different, which led to different implementation strategies and different capabilities of each system. For example, Falkon does not address the problem of efficient provisioning of Spot Instances resources to applications, as does Aneka.

H20 [10] is a component-based, service-oriented framework for Grid computing. Similarly to Aneka, it provides a runtime middleware supporting the deployment and discovery of services. Based on a container model similar to the applied on Aneka, H20 aims to be a general framework for integration of services spanning across different administrative boundaries, while Aneka focuses on a single administrative domain with extra resources provisioned from other sources such as public Clouds.

Ibis [11] is both a programming system and a deployment system for distributed applications. It is composed of a set of APIs allowing users to express applications with different abstractions, such as MPI, master–worker, and workflow. For what concerns the deployment system, Ibis leverages the Grid Application Toolkit [12], which constitutes a uniform interface to several Grid middleware implementations. Aneka supports not only Grids, but also Clouds to supply computing resources for applications. Moreover, Aneka also supports different operating systems and both physical and virtual machines as compute resources, and it is able to efficiently leverage resources from various sources including Spot Instances.

Regarding the problem of using Spot Instances for execution of applications in the Cloud, Yi et al. [13] propose and compare different checkpointing strategies. In our approach, we apply high bids and hybrid resources rather than checkpointing strategies. Thus, even in case of termination of Spot Instances, tasks running in local resources will complete, and such resources are kept available for running other jobs. Chohan et al. [14] present an analysis on utilization of Spot Instances to speed up execution of MapReduce applications. Such a work focuses on defining costs and benefits of the subject, and no provisioning algorithm to explore such capacity is supplied, while our work presents a provisioning algorithm for combining Desktop Grids and Spot Instances regardless the application model.

Finally, Mattess et al. [15] presents a provisioning algorithm for extending cluster capacity with Spot Instances. The goal of such algorithm is reducing waiting time of jobs in cluster queues, where users have reservations that define amount of time required by the job and number of resources required by the job. Our approach, on the other hand, is elastic in the sense that amount of resources allocated to a job can change during its execution if this is required for meeting application deadline. Moreover, our provisioning algorithm does not require resources reservation, as does the described one.

## 3. Aneka: a cloud application platform

Aneka [3] is a framework for development, deployment, and management of Cloud applications. It consists of a scalable Cloud middleware that is deployed on top of heterogeneous computing resources and an extensible collection of services coordinating the execution of applications, monitoring the status of the Cloud, and providing integration with existing Cloud technologies. One of the key advantages of Aneka is its extensible API for development of distributed applications, integration of new capabilities into the Cloud, and support of different types of Clouds: public, private, and hybrid. These features differentiate Aneka from typical infrastructure management software and actually characterize it as a platform for development and deployment of applications.

In this section we illustrate the architecture of Aneka and describe the fundamental components that constitute the framework by first looking at the structure of the Aneka container and then discussing its various services.

### *3.1. Architecture*

Fig. 1 provides an overview of the components of the framework. The core infrastructure of the system provides a layer allowing the framework to be deployed over different platforms and operating systems. The physical and virtual resources representing the bare metal of the Cloud are managed by the Aneka container, which abstracts the peculiarities of the underlying hardware and hosting operating system to provide a homogeneous runtime environment for applications and services, which are shared among users from the platform.

The container is installed on each node and constitutes the basic building block of the middleware. A collection of interconnected containers forms the Aneka Cloud: a single domain in which services are made available to users and developers. The container is also the unit of deployment in Aneka Clouds: the middleware is able to scale on demand by dynamically provisioning additional resources and automatically deploying the container on them. Services are the fundamental abstraction by which the features available in the container are implemented. The container features three different classes of services: *Fabric Services*, *Foundation Services*, and *Application Services*. They respectively take care of infrastructure management, cloud middleware, and application management and execution. These services are made available to developers and administrators by the means of the application management and development layer, which includes interfaces and APIs for developing Cloud applications and the management tools and interfaces for controlling Aneka Clouds.

Aneka implements a service-oriented architecture (SOA) and services are the fundamental components of an Aneka Cloud. They operate at container level and provide developers, users, and administrators with features offered by the framework. Services also constitute the extension and customization point of Aneka Clouds: the platform allows the integration of new services or replacement of the existing ones with a different implementation. The framework includes the basic services for infrastructure and node management, application execution, accounting, and system monitoring.

#### *3.1.1. Infrastructure*

The infrastructure of the system is based on the .NET technology and allows the Aneka container to be portable over different platforms and operating systems. Any platform featuring an ECMA-335 (Common Language Infrastructure) [16] compatible environment can host and run an instance of the Aneka container, which is implemented by both Microsoft .NET framework[1] (for Windows-based systems) and the Mono open source implementation of the .NET framework[2] (for Linux-based systems).

The Common Language Infrastructure (CLI), which is the specification introduced in the ECMA-335 standard, defines a common runtime environment and application model for execution of programs, but does not provide any interface to access the hardware or to collect performance data from the hosting operating system. Moreover, each operating system has a different organization of the file system and stores that information differently. Aneka's *Platform Abstraction Layer* (PAL) addresses this heterogeneity and provides the container with a uniform interface for accessing the relevant hardware and operating system information, thus allowing the rest of the container to run unmodified on any supported platform. The PAL is a small layer of software comprising a detection engine which automatically configures the container at boot time with the platform specific component to access information about the hosting platform.

Another important function of the PAL is to provide a platform-independent interface for managing the Aneka Cloud infrastructure. The Aneka Cloud is characterized by a network of interacting containers that are deployed on top of both physical and virtual nodes. Each node contains an Aneka Daemon that is responsible for managing multiple instances of the container.

The architecture of the system is completed by a repository server that makes the Aneka codebase available to the other nodes for updates via different protocols such as HTTP, file share, or other solutions that can be implemented by the administrator. Each Aneka Daemon can be contacted with proper credentials and controlled remotely by means of the management interfaces. All the nodes are also accessible via a management console from which it is possible to deploy new instances of the container and remotely control them through the management API offered by the PAL.

#### *3.1.2. Middleware*

The middleware represents the distributed infrastructure constituting Aneka Clouds. It provides a collection of services for interaction with the Cloud. These include monitoring, execution, management, and all the other functions implemented in the framework.

The middleware is composed of two major components representing the building blocks of Aneka Clouds: the *container* and the *Aneka Daemon*. The container represents the unit of deployment of Aneka Clouds and the runtime environment for services and applications. The middleware scales elastically by dynamically adding or removing container instances. The daemon is a management component controlling the container instances installed on a single node (physical or virtual). Its responsibilities are: installing or removing container instances, managing software updates, and dynamically configuring, starting,

[1] http://www.microsoft.com/net/.
[2] http://www.mono-project.com.

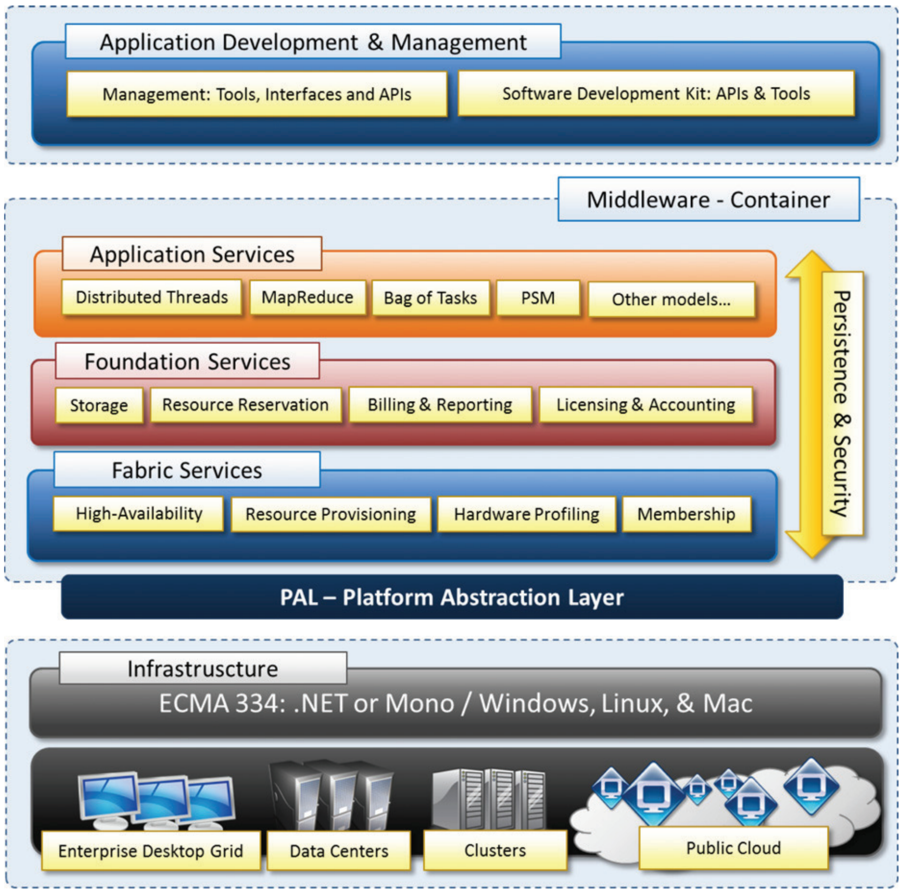


**Fig. 1.** Aneka framework overview.

and stopping container instances. A standard deployment of Aneka is characterized by a single daemon on each node and one or more container instances controlled by the daemon. Multiple installations of the daemon on a single node is useful for deployment of isolated Aneka Clouds on the same physical infrastructure. This is generally not required because Aneka provides ways for partitioning the containers within a Cloud by putting them into separate groups.

The Aneka container performs service management tasks, while all the other functions are implemented by using services. Services are self-contained components used to develop a feature that is accessible through the messaging channel made available by the container. There are three major classes of services, and they are:

| | |
|---|---|
| Fabric services: | Provide access to the resource provisioning subsystem and to the hardware of the hosting machine. Services in this class include resource provisioning service, hardware profiling services, and reservation service. Reservation is available for users that, for some reason, want to postpone execution of their jobs. |
| Foundation services: | Constitute the pillars of the Aneka middleware and are mostly concerned with providing runtime support for execution of services and applications, providing services such as directory and membership, resource reservation, monitoring, storage management, licensing, accounting, reporting, and billing. |
| Application services: | Consist of services that are directly involved in the execution of distributed applications. Services in this class include scheduling service and execution service. |

Finally a collection of transversal services operate at all the levels of the container and provide persistence and security for the runtime environment. Persistence provides support for recording the status of the Cloud. The persistence infrastructure is composed of a collection of storage facilities that can be configured separately for tuning the performance and the quality of service of the Cloud. Aneka provides two different implementations for each storage: in memory and relational database. The former is used for testing environments and provides better performance in terms of speed. The latter provides a more solid and reliable support and can be used over different database management systems.

The Aneka Cloud is also resilient to failure by providing a multiple-master failover mechanism. This allows the master container to be replicated on multiple nodes in the network, ensuring that an active master is available at all times for monitoring and managing the execution of applications on the Cloud. The failover mechanism is implemented using an election algorithm.

The security system of Aneka is composed of a central security service that manages users, groups, and resources available in the Cloud and a pair of providers—authentication and authorization providers—that operate at the container level to identify the user and authorize the processing of each message. Authorization and authentication providers can be dynamically configured, thus providing a more flexible infrastructure. The containers in a Cloud also authenticate messages exchanged between each other using a shared secret key that is unique to the Aneka cloud.

Billing is calculated considering the time that a task occupies a container. Each container (running on a machine) has an associated cost. The bill then is the sum of the cost of the task in the container for all tasks run by the user. Each application in Aneka is associated to a user, thus Aneka tracks usage of resources per user. Moreover, the scheduling algorithm is able to specifically provision resources for a given application, and then bill the user that owns the application.

### 3.1.3. Application development and management

The services of the middleware are accessible through a set of tools and interfaces that together define the Software Development Kit (SDK) and the Management Kit. The SDK represents the collection of end user abstractions and APIs for definition of applications by leveraging the existing programming models or for implementation of new models and services. The Management Kit contains a collection of tools for management, monitoring, and administration of Aneka Clouds.

All the management functions of the Aneka Cloud are made accessible through the Management Studio, which is a comprehensive graphical environment where administrators have a global view of the Cloud. Since Clouds are constituted of hundreds or even thousands of machines, both physical and virtual, it is not possible to reach and set up each single machine by hand. The Management Studio provides remote access to each node belonging to the Cloud and advanced interfaces for deployment of the container and customization of its behavior.

## 3.2. Programming models

One of the original aspects of Aneka is its ability to provide different abstractions for programming distributed applications. These abstractions are translated into applications and executed in the Cloud by means of *programming models*. A programming model comprises both the end-user abstractions used to define the logic of applications and the runtime support in terms of services and client components required to execute the resulting applications. All the programming models supported by Aneka base their design on a common root that is the *Aneka Application Model*. The Application Model defines the common properties of a distributed application despite the specific programming model it belongs to, and provides an extensible object model from which developers can create new programming models.

Fundamental to the model is the concept of *application*, which represents the context in which a self-contained interaction between the user and the middleware happens.

By using the concept of application, different paradigms can be expressed. For example in the case of bag of tasks and workflow applications, tasks are directly created and submitted by the user; in the case of MapReduce or Parameter Sweep applications, the units of work are dynamically generated by the middleware according to the user input and the implementation logic of the model. These units generally are the containers of the executable code, have a life cycle determined by states whose transitions are determined by the middleware, and require to be unequivocally identified.

The application and the application manager instances constitute the client components that manage the execution of distributed applications for a specific programming model. The other components reside in the middleware and are implemented as services. In general, two major operations are required to provide execution support for a specific application: scheduling and execution. As a result, all supported programming models feature a scheduling and an execution service. In general, only one scheduling service is required while there are multiple execution services for scaling application execution. For what concerns the management of files, each programming model relies on the storage service that provides a staging facility for applications. Fig. 2 provides an overview of the execution of distributed applications in Aneka.

Currently, Aneka supports four different programming models, which are briefly described here. A deeper description of each model was presented in our previous work [3] and is out of the scope of this paper.

| | |
|---|---|
| Task model. | The Task Programming Model provides developers with the ability of expressing bag of tasks and workflow applications. |
| Thread model. | This model allows quickly porting of applications composed of independent threads (i.e, threads that do not share data) into a distributed environment. Developers familiar with the threading API exposed by the .NET framework or Java can easily take advantage of the set of compute resources available with Aneka in order to improve the performance of their applications. |
| MapReduce model. | This model is an implementation of Google's MapReduce [17] that leverages Aneka for processing or generating large data sets. |
| Parameter sweep model. | This model represent applications that iterate the execution of the same task over different value ranges from a given parameter set. |

These programming models, together with the support for hybrid Desktop Grids and Clouds in Aneka, provide infrastructure users with a rich set of options related to both type of applications that can be developed or ported to run in Aneka and type of resources that can be used to support application execution. In the next section, we describe this process in details.

## 4. Desktop Grids and Aneka

In this section, we discuss Desktop Grid support in Aneka and the role of different Aneka services on such a support.

### 4.1. General overview

Aneka is a portable middleware that can be deployed over different operating systems and hardware, including both physical and virtual. The core functionalities that support this feature reside in the Platform Abstraction Layer, which provides a uniform interface for management and configuration of nodes and the containers instances deployed on them. Together with basic management operations of the PAL, the dynamic resource provisioning module constitute the basic services used to control the infrastructure of Aneka Clouds.

The typical deployment scenario of Aneka consists of one master node that contains all the management services and represents the access point to the Cloud and one or more worker nodes that share the workload of the system. Worker nodes may include resources from several sources, including clusters, Grids, and private and public Clouds [6]. However, the most common type of resource to be included in an Aneka cloud is desktop machines from the local Aneka site, whose idle cycles are explored opportunistically by Aneka. Alternatively, it is also possible to add dedicated machines to Aneka, so they will be used whenever resources are required, even in the presence of local users.

Inclusion of new desktop machines in the cloud happens via a management interface. When the new machine is added to the Aneka cloud, required files are copied to the machine, so it is able to communicate with the master node and join the cloud. The requirements of a machine to join Aneka cloud are (i) being accessible via a network and (ii) having a configuration that allows

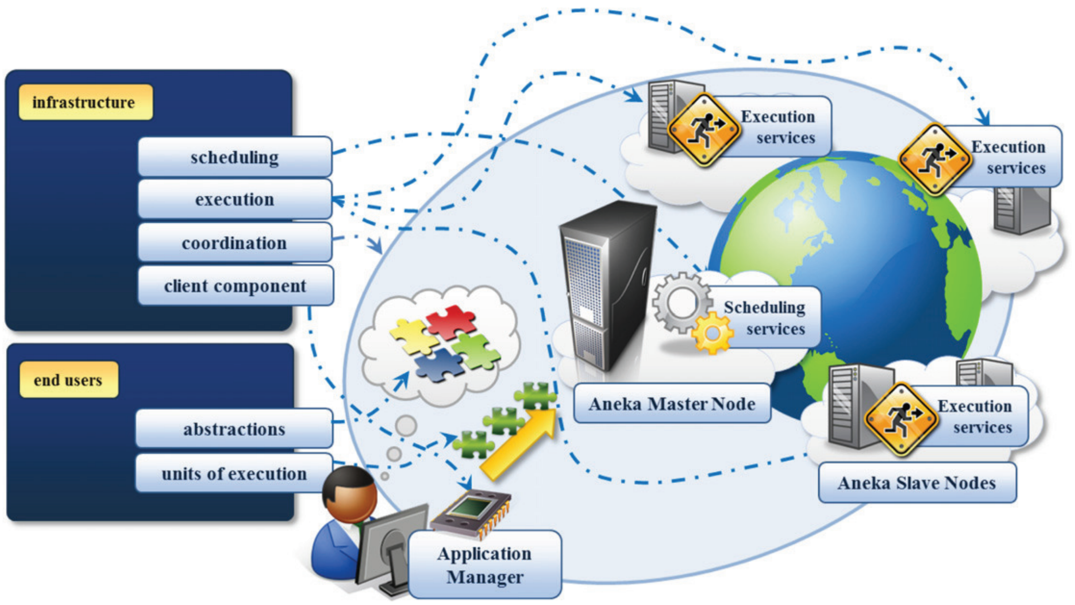


**Fig. 2.** Application execution scenario.

remote access and execution of the Aneka container. For the latter to be possible, the resource must have the runtime environment required by the Container (e.g., .NET, Mono).

Besides the support from PAL, other Aneka services have also role in the support of Desktop Grids. In the following sections we discuss the role of each service separately.

### 4.2. Security and Desktop Grids

Desktop Grids harness commodity machines to create a distributed infrastructure. Differently from a cluster where nodes are dedicated to the Grid and there is a stricter access control, desktop machines are exposed to more security threats, being used by several different users, who in many cases have complete control of the machine. Therefore, it is important to enforce the appropriate security measures to guarantee privacy and confidentiality. Aneka provides a pluggable security model that can be configured with different implementations. Each communication to and from instances of the Aneka Container is secured. Moreover, the hosting environment is protected by a sandbox that isolates effects of execution of Aneka tasks in the system. Finally, user data and execution files are deleted after job execution to avoid unauthorized access.

### 4.3. Reservation and Desktop Grids

Resources reservation is supported by two components: Reservation Service and Allocation Manager. The Reservation Service is a central service that keeps track of the allocation map of all the nodes constituting the Aneka Cloud, while the Allocation Manager provides a view of the allocation map of the local Container. Reserved nodes only accept jobs that belong to the reservation request that is currently active. In case there is no active reservation on the node, any job that matches the security requirements set by Aneka Cloud is executed. The Allocation Manager is responsible for keeping track of the reserved time frames in the local node and for checking, before the start of a job, whether it is admissible or not.

Dynamic Provisioning is a useful complement to the reservation infrastructure since it contributes to provide a better reliability of the system. In case of reserved nodes are unavailable when their reservation should start, and there is no other local resources to replace them, the reservation infrastructure leverages dynamic provisioning, which leases resources from public Clouds, to guarantee the missing computing capacity. This is an important aspect in case of Desktop Grids where nodes are typically more volatile than in other contexts.

### 4.4. Accounting and Desktop Grids

Aneka prices the execution of tasks according to the cost of the machines running the tasks. However, there is no charge for a task execution if it does not complete because of resource unavailability. For example, if an Aneka node leaves Desktop Grid in the middle of the execution of a task, the corresponding execution time is not charged from the user. Instead, the condition is detected by Aneka and the task is automatically rescheduled.

Quality of Service in terms of deadline-based application execution is another issue of concern in Desktop Grids. As discussed next, dynamic provisioning and scheduling services coordinate their efforts in order to meet the deadline with the given budget assigned to the applications. The current implementation of such algorithms still have limitations on meeting QoS in case of sudden and massive shortage of resources when applications are very close to the deadline. Task replication based on historical execution pattern and prediction of failures and reaction before the failure are two approaches currently under investigation for overcoming the aforementioned limitations.

### 4.5. Dynamic provisioning and Desktop Grids

Desktop machines, either dedicated or accessed opportunistically, are managed by the scheduler service. When requests arrive, this service allocates resources to the requests, considering an initial user estimation of task duration. If available resources are not enough to serve requests on time, the scheduler requests the provisioning service to provision dynamic resources to serve jobs.

Therefore, capacity of local Desktop Grid is enhanced with public Cloud resources by the Provisioning Service. This service, via its *Resource Pool Manager*, allocates resources from public cloud providers to complement local resources. Different public Cloud providers, and hence different Resource Pool Managers, are supported by Aneka. System administrators define which Resource Pool Managers can be used and the preferred order of utilization via explicit configuration. Administrators can also define the allocation policy to be adopted for resource provisioning. For example, a straight forward strategy for resource allocation on public Clouds provided by Aneka is to pick the first option set by the administrator; if these resources are not enough, or in case of unavailability of the provider, the next provider in the list is selected and so on. A more interesting resource selection strategy is to use a cost-based approach to resource allocation, where the cheapest provider is selected first.

Access to different providers is made possible via the implementation of different *Resource Pools*. A Resource Pool implements the particular operations that are required for allocating, submitting jobs, and releasing resources from the provider it represents.

Resources allocated from the Cloud are kept in the Aneka pool until the end of the allocation time window where it is no longer required. For example, in the case of Amazon EC2 [18], where resources are charged hourly, EC2 instances are kept for the whole 60 min period even if they are no longer required. Because allocation time windows is different for different providers, this feature is also implemented by the Resource Pool.

### 4.6. Provisioning of resources from Desktop Grids and spot instances in Aneka

For the intent of supporting execution of applications in hybrid Desktop Grids and Clouds, a new provisioning algorithm has been developed in Aneka that explores Spot Instances. Spot Instances consist of virtual machines (VMs) for which users produce a bid price that represent the maximum value they are willing to pay for using each VM. Amazon then periodically updates resource value (spot price) and, whenever the spot price is equal or smaller than the bid price, corresponding VMs are executed. When the spot price is bigger than the bid price, VMs are terminated. Users are charged hourly at the spot price, and fractions of hours are ignored in case of preemption. Therefore, if a VM is preempted after 1.5 h, the user is charged for one hour of utilization. However, if the user terminates a VMs after 1.5 h, he or she is charged for two hours of utilization. Spot Instances represent a cheaper type of Cloud resource that can be valuable to speed up applications running in the local Desktop Grid.

Algorithm 1 describes Aneka's Spot Instance-Aware Provisioning Algorithm.[3] The algorithm is executed when any of the following conditions is observed: (i) a new job is received by the system (ii) a task from a job is queued, and (iii) execution of a task completes.

Basically, the algorithm checks if currently available resources are enough for completing jobs within their deadlines (Line 7). The calculation considers only tasks in the waiting queue; therefore, during estimation of remaining execution time (Lines 6 and 22) the algorithm considers that current running tasks have to complete before the waiting ones are scheduled and executed.

One of the key differences between the proposed provisioning algorithm and the previous one [6] is that the former considers the deployment time in public Clouds during calculation of number of extra required resources (Lines 18–16): values $\alpha$ and $\beta$ represent, respectively, the ratio between remaining time and task runtime and deployment time and task runtime. If $\alpha < \beta$, it is possible to complete the job before its deadline. Otherwise, new VMs will not become available before the deadline, and number of extra resources is calculated based on availability of local resources and estimated utilization of external resources. In either case, the number of external resources to be provisioned is limited to the number of waiting tasks (Line 16).

3 For the sake of simplicity, we do not present in the algorithm listing activities related to management of requests and check for failures in the requests processing that are performed by Aneka.

**Algorithm 1:** Spot Instance-Aware Provisioning Algorithm.

```
1  foreach job with QoS contraints do
2      executionTime ← updated estimate of task execution time;
3      timeLeft ← time before job deadline;
4      resources ← available resources for the application;
5      pendingTasks ← number of tasks in the queue;
6      requiredTime ← ($\lceil \frac{tasks}{resources} \rceil + 1$) × executionTime;
7      if requiredTime > timeLeft then
8          $\alpha \leftarrow \lfloor \frac{timeLeft}{executionTime} \rfloor$;
9          $\beta \leftarrow \lfloor \frac{deploymentTime}{executionTime} \rfloor$;
10         totalResources ← resources + requestedResources;
11         if $\alpha - \beta > 0$ then
               // deadline can be met
12             extraResources ← $\lceil \frac{(1-\alpha) \times totalResources + pendingTasks}{\alpha - \beta} \rceil$;
13         else
14             extraResources ← pendingTasks − ($\beta$ + 1) × totalResources;
15         end
16         extraResources ← min(extraResources, pendingTasks);
17         if extraResources > 0 then
18             request extraResources spot instances at on-demand price;
19             requestedResources ← requestedResources + extraResources;
20         end
21     else
           // check if one less resource enables deadline to be achieved
22         requiredTime ← ($\lceil \frac{tasks}{resources-1} \rceil + 1$) × executionTime;
23         if requiredTime < timeLeft then
24             return one resource to the Resource Pool Manager;
25         end
26     end
27 end
```

The pricing strategy used for provisioning (Lines 17–19) of Spot Instances is utilization of on-demand price for the equivalent type of instance. This strategy makes the probability of VM preemption small, especially if external resources are required for a small number of hours [19], at the same time it allows smaller prices than on demand requests. If allocation of spot instances fails (for example, if spot price is higher than the on-demand price), resources can be sought in the on-demand market or in other public Cloud providers, which are controlled by other Aneka Resource Pools.

Finally, each time a task completes the algorithm also checks if the job is able to complete within its deadline with one less resource (Lines 22–25). If so, one resource is returned to the Resource Pool Manager, which can decide between releasing the resource (if the one-hour billing period is about to expire) or allocating it to another running job.

## 5. Performance evaluation

The scenario in which the tests were carried out is a Desktop Grid constituted of eight machines (one master and seven slaves) that were configured to run Bag of Tasks applications with Aneka. This infrastructure is dynamically grown with resources from public Clouds in order to meet jobs' QoS requirements. We evaluated the performance of the original and enhanced scheduling and provisioning services to address peak loads and accomplish application execution within the deadline.

**Table 1**
Experimental set up.

| Desktop grid nodes | 7 |
|---|---|
| Desktop grid nodes operating system | Windows server |
| Amazon instance type | m1.small |
| Amazon instance type operating system | Fedora Linux |
| Tasks per job | 50 |
| Task execution time | 10 min |

Public Cloud resources are deployed on Amazon EC2 (USA East Coast data center) using *m1.small* instances: they have a single core, a computing power of 1 EC2 unit, which is equivalent to 1.0–1.2 *GHz CPU* 2007 *Opteron or* 2007 *Xeon Processor*,[4] and 1.7 GB of memory, at the cost of US$0.085 per instance per hour. The operating system of Desktop Grid nodes is Windows Server while operating system of public Cloud resources is Fedora Linux. The default resource provisioning algorithm of Aneka uses on-demand resources, whereas the new resource provisioning uses Spot Instances. The estimated start up time of new VMs used by the Spot Instance-Aware Provisioning Algorithm for making provisioning decisions is 4 min, even though even longer start up times were observed during the experiments.

The test application is a Bag of Tasks application containing 50 tasks. The task selected for the test is a sleep task that occupies the computing resource for 10 min. The reason for this choice is to have an homogeneous performance across the entire set of slave nodes and to consider indifferently virtual and physical slave nodes in our experiments. Experimental set up is summarized in Table 1.

Fig. 3 and Table 2 presents results of job execution with different deadlines. The total execution time of the workload in a single resource is 500 min. Considering only the seven Desktop Grid nodes available in the experiments, the workload requires 80 min to complete without any overhead incurred by the system. Therefore, provisioning of public Cloud resources is necessary in all the experiments.

Fig. 3(a) shows that the Spot Instance-Aware Provisioning Algorithm was able to meet all the deadlines. The bigger execution time of the Spot Instance-Aware Provisioning Algorithm is caused by a more conservative budget expenditure for completing the application: the algorithm allocates the minimum amount of resources that enables the deadline to be met. The result is that execution times for all cases are always closer to the deadline, while the default algorithm allocates more resources and meets relaxed deadlines by a larger margin.

The result of the conservative approach for budget by our new algorithm can be seen in Fig. 3(b): in general, our new algorithm provisions less virtual machines for the same scenario. The exception is the strictest deadline: in this case, Spot Instance-Aware Provisioning Algorithm allocates one extra resource than the default algorithm. This is because SPOT considers the delay in instantiation (which was set to 4 min) when deciding number of resources. In some of the observed cases, the start up time of the Spot request was longer than six minutes, against two minutes for on-demand requests. This would made the deadline be missed by a larger margin if Spot Instance-Aware Provisioning Algorithm provisioned the same number of machines as the default approach.

The impact in the budget of the decision of using Spot Instances can be seen in Fig. 3(c). Because in most scenarios SPOT required less virtual machines than DEFAULT, it would lead to savings in budget in any case. However, because Spot Instances are, during most part of time, significantly cheaper than on demand instances, our new algorithm enabled savings in investment for completing execution within the deadline of up to 85.7%. In fact, during the

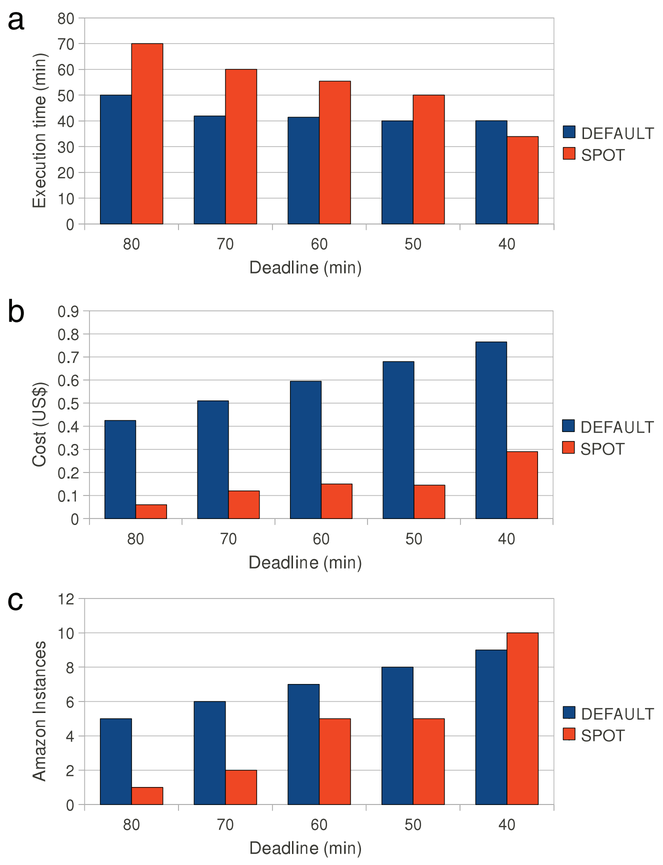


**Fig. 3.** Experimental results for the two provisioning strategies: the default provisioning algorithm (DEFAULT in the graphs) and Spot Instance-Aware (SPOT in the graphs), considering different application deadlines. (a) Execution time (b) Cost (c) Number of Amazon instances deployed.

experiments the spot price for small instances in the US-East data center varied between US$0.029 and US$0.03 per instance per hour. Therefore, we conclude that our new algorithm is able to meet even strict application deadlines with minimal budget expenditure by taking advantage of Spot Instances.

## 6. Conclusions and future directions

In this paper we presented how Aneka, a framework for developing, managing, and deploying Cloud computing applications, can be used to extend capacity of Desktop Grids with Cloud computing resources hired from Amazon EC2 Spot Instances.

The middleware is based on an extensible service-oriented architecture and the core functionalities of the system are exposed by means of the Aneka container, which is also the basic building block of the Cloud. Aneka Clouds scale elastically and on demand by integrating or removing container instances provisioned from Infrastructure-as-a-Service providers. The container is a portable software component hosting different classes of services that coordinate their activity in order to build the required context for applications execution. Fabric services provide an interface with the physical and virtual infrastructure of the middleware. Foundation services provide support for applications such as storage, reservation, accounting, monitoring, and reporting. Application services are directly involved with the execution of applications and provide the runtime support for the different programming models supported by Aneka.

A programming model represents a specific way of expressing distributed applications. It consists of a set of abstractions used by developers and the corresponding runtime support allowing these

[4] As by the time this article was written, according to http://aws.amazon.com/ec2/instance-types/.

**Table 2**
Experimental results for the two provisioning strategies: default provisioning algorithm (DEFAULT in the table) and Spot Instance-Aware (SPOT in the table), considering different application deadlines.

| Deadline (min) | Amazon instances | | Execution time (min) | | Cost (US$) | |
|---|---|---|---|---|---|---|
| | DEFAULT | SPOT | DEFAULT | SPOT | DEFAULT | SPOT |
| 80 | 5 | 1 | 50 | 70 | 0.43 | 0.06 |
| 70 | 6 | 2 | 42 | 60 | 0.51 | 0.12 |
| 60 | 7 | 5 | 42 | 56 | 0.60 | 0.15 |
| 50 | 8 | 5 | 40 | 50 | 0.68 | 0.15 |
| 40 | 9 | 10 | 40 | 34 | 0.77 | 0.29 |

abstractions to leverage Aneka for their execution. The framework provides an extensible application model that constitutes the base on which all the supported programming models are implemented. Currently, Aneka provides support for bag of tasks, distributed threads, parameter sweep, and MapReduce applications. The system can be extended with new models by either leveraging the implementation of the existing ones or by extending the application model directly.

Together, these services allow Aneka to offer an environment that goes beyond Desktop Grid systems capacities by allowing provisioning of resources from public Clouds to complement opportunistic resources, so the system is able to meet deadlines of applications.

The infrastructure for dynamic provisioning, together with the ability of managing a large number of resources for application execution, was validated with experiments that show that Aneka is able to meet even strict application deadlines with minimal budget expenditure by taking advantage of Spot Instances, which are a cheaper type of Cloud resource that may be preempted in favor of more profitable requests to the Cloud provider.

As future work, we plan to develop new provisioning policies that can take advantage of more information that can be provided by the scheduling service. For example, if the provisioning is aware of the amount of time in which external resources are required, and also have more information about current and predicted utilization of local infrastructures, it can decide to deploy resources from the resource pool whose resources have the optimal cost in the envisioned situation. This will drive for further reduction in budget expenditure for execution of applications in hybrid Clouds, what will further motivate adoption of hybrid Clouds as a platform for execution of applications.

A trial version of the software is available for download at http://www.manjrasoft.com.

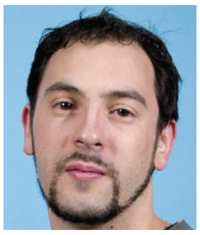
## Acknowledgment

The authors would like to thank Xingchen Chu for his contribution on the research and development of Aneka, which motivated the work presented in this paper.

## 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] D. Kondo, A. Chien, H. Casanova, Scheduling task parallel applications for rapid turnaround on enterprise desktop Grids, Journal of Grid Computing 5 (4) (2007) 379–405.
[3] C. Vecchiola, X. Chu, R. Buyya, Aneka: a software platform for .NET-based cloud computing, in: W. Gentzsch, L. Grandinetti, G. Joubert (Eds.), High Speed and Large Scale Scientific Computing, IOS, 2009, pp. 267–295.
[4] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, J. Pruyne, A worldwide flock of Condors: load sharing among workstation clusters, Future Generation Computer Systems 12 (1) (1996) 53–65.
[5] L. Baduel, F. Baude, D. Caromel, A. Contes, F. Huet, M. Morel, R. Quilici, Programming, composing, deploying for the Grid, in: J.C. Cunha, O.F. Rana (Eds.), Grid Computing: Software Environment and Tools, Springer-Verlag, 2006, pp. 205–229.
[6] C. Vecchiola, R.N. Calheiros, D. Karunamoorthya, R. Buyya, Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka, Future Generation Computer Systems (2011) doi:10.1016/j.future.2011.05.008.
[7] F. Cappello, S. Djilali, G. Fedak, T. Hérault, F. Magniette, V. Néri, O. Lodygensky, Computing on large-scale distributed systems: XtremWeb architecture, programming models security, tests and convergence with Grid, Future Generation Computer Systems 21 (3) (2005) 417–437.
[8] D.P. Anderson, BOINC: a system for public-resource computing and storage, in: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, IEEE Computer Society, 2004, pp. 4–10.
[9] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde, Falkon: a Fast and Light-weight tasK executiON framework, in: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, ACM, 2007.
[10] D. Kurzyniec, T. Wrzosek, D. Drzewiecki, V.S. Sunderam, Towards self-organizing distributed computing frameworks: the H2O approach, Parallel Processing Letters 13 (2) (2003) 273–290.
[11] H.E. Bal, J. Maassen, R.V. van Nieuwpoort, N. Drost, R. Kemp, N. Palmer, G. Wrzesinska, T. Kielmann, F. Seinstra, C. Jacobs, Real-world distributed computing with Ibis, Computer 43 (8) (2010) 54–62.
[12] G. Allen, K. Davis, T. Dramlitsch, T. Goodale, I. Kelley, G. Lanfermann, J. Novotny, T. Radke, K. Rasul, M. Russell, E. Seidel, O. Wehrens, The GridLab Grid application toolkit, in: Proceedings of the 11th International Symposium on High-Performance Distributed Computing, IEEE Computer Society, 2002.
[13] S. Yi, D. Kondo, A. Andrzejak, Reducing costs of spot instances via checkpointing in the Amazon Elastic Compute Cloud, in: Proceedings of the 3rd International Conference on Cloud Computing, IEEE Computer Society, 2010, pp. 236–243.
[14] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, C. Krintz, See spot run: using spot instances for mapreduce workflows, in: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, USENIX, 2010.
[15] M. Mattess, C. Vecchiola, R. Buyya, Managing peak loads by leasing cloud infrastructure services from a spot market, in: Proceedings of the 12th International Conference on High Performance Computing and Communications, IEEE Computer Society, 2010, pp. 180–188.
[16] J. Miller, S. Ragsdale, The Common Language Infrastructure Annotated Standard, Addison Wesley, 2004.
[17] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Communications of ACM 51 (1) (2008) 107–113.
[18] J. Varia, Best practices in architecting cloud applications in the AWS Cloud, in: R. Buyya, J. Broberg, A. Goscinski (Eds.), Cloud Computing: Principles and Paradigms, Wiley Press, 2011, pp. 459–490.
[19] B. Javadi, R. Buyya, Comprehensive statistical analysis and modeling of spot instances in public cloud environments, Technical Report CLOUDS-TR-2011-1, The University of Melbourne, 2011.

**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 PhD degree in Computer Science in 2010 at PUCRS, Brazil, and his MSc degree in 2006 at the same University. His research interests include Cloud Computing and simulation and emulation of distributed systems, with emphasis in Grids and Clouds.



**Christian Vecchiola** 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. His primary research interests include: Grid/Cloud Computing, Distributed Evolutionary Computation, and Software Engineering. Since he joined the CLOUDS Lab he 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 the Ph.D. he worked under the supervision of Prof. Antonio Boccalatte in the Department of Communication Computer and System Sciences and has have been 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 gave him the opportunity to cultivate another research interest that is represented by Programming Languages and Compiler Technology.

**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 a MSc. 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, and 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".