Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using Aneka

Adel Nadjaran Toosi, Richard O. Sinnott, Rajkumar Buyya

The Cloud Computing and Distributed Systems (CLOUDS) Laboratory,
School of Computing and Information Systems,
The University of Melbourne, Australia

Abstract

Cloud computing has emerged as a mainstream paradigm for hosting various types of applications by supporting easy-to-use computing services. Among the many different forms of cloud computing, hybrid clouds, which mix on-premises private cloud and third-party public cloud services to deploy applications, have gained broad acceptance. They are particularly relevant for applications requiring large volumes of computing power exceeding the computational capacity within the premises of a single organization. However, the use of hybrid clouds introduces the challenge of how much and when public cloud resources should be added to the pool of resources – and especially when it is necessary to support quality of service requirements of applications with deadline constraints. These resource provisioning decisions are far from trivial if scheduling involves data-intensive applications using voluminous amounts of data. Issues such as the impact of network latency, bandwidth constraints, and location of data must be taken into account in order to minimize the execution cost while meeting the deadline for such applications. In this paper, we propose a new resource provisioning algorithm to support the deadline requirements of data-intensive applications in hybrid cloud environments. To evaluate our proposed algorithm, we implement it in Aneka, a platform for developing scalable applications on the Cloud. Experimental results using a real case study executing a data-intensive application to measure the walkability index on a hybrid cloud platform consisting of dynamic resources from the Microsoft Azure cloud show that our proposed provisioning algorithm is able to more efficiently allocate resources compared to existing methods.

Key words: Dynamic Provisioning, Hybrid Cloud, Aneka Cloud Application Platform, Deadline-driven Scheduling, Data locality, Network Bandwidth, Data-intensive Applications

1. Introduction

Data-intensive applications involving the analysis of large datasets have become increasingly important as many areas of science and business are facing thousand-fold increases in data volumes [1]. The explosive growth of data is mainly driven by the rapid expansion of the Internet, smart cities, social networks, e-commerce, and widespread usage of high-throughput instruments, sensor networks, Internet of Things (IoT) devices, accelerators, and supercomputers. This expansion forms a voluminous amount of structured and unstructured data, known as big data, that needs to be processed to be useful [1]. The ability to analyze and process such large quantities of data has become an important and challenging mission for many fields.

Cloud computing [2] platforms are becoming one of the most preferred ways of hosting data-intensive applications. Challenges posed by big data can be overcome with the aid of cloud computing services offering the illusion of an infinite pool of highly reliable, scalable, and flexible computing, storage, and network resources. However, in many cases, data is available in local IT infrastructure with limited processing capacity, for example, a small cluster or resources from local area networks (desktop grids). In many cases, it is not time or cost effective to transfer the whole dataset to clouds to be processed. To tackle this issue, the *cloud bursting* model can be used in which an application runs in a private infrastructure and bursts onto a public cloud when more resources are required. This model has found broad acceptance due to its benefits such as cost reduction and dealing with issues related to the location of sensitive data [3].

To achieve the vision of cloud bursting, hybrid cloud *middleware* is required to acquire and release resources from both local infrastructures and external cloud providers in a seamless fashion [4]. It is essential for such hybrid cloud middleware to make efficient decisions regarding the workloads that must be outsourced to the public cloud based on the timing and number of externally provisioned resources to meet deadline constraints of applications. In such a setting, however, building a middleware that jointly minimizes cost and meets the deadline for applications is far from trivial [5].

There is a large body of literature aimed at cost and execution time minimization of running computational tasks in hybrid cloud environments. These studies mostly overlook aspects such as data locality, the impact of network bandwidth constraints, and data transfer time which significantly affect the time and cost performance of the scheduling. This is exacerbated for data-intensive applications where the data transfer time to the external cloud is often comparable to the computational time. In this paper, one of our main goals is to take these aspects into consideration for scheduling and resource provisioning of deadline-driven data-intensive applications in hybrid cloud environments.

In this context, *Platform-as-a-Service* (PaaS) solutions offer various tools to implement scheduling and resource provisioning policies in hybrid clouds. We exploit the Aneka platform [6] to implement the proposed solution in this paper. Aneka is a PaaS solution providing a middleware for the development and deployment of applications in hybrid and multi-clouds. Aneka provides application developers with Application Programming Interfaces (APIs) for transparently harnessing and exploiting the physical and virtual computing resources in heterogeneous networks of workstations, clusters, servers, and data centers. Earlier version of Aneka had many features supporting multi-cloud and hybrid computing. However, to provide wider support for scheduling and resource provisioning of data-intensive applications, we incorporate additional functionalities into Aneka.

In this paper, we make the following **key contributions**:

- We propose a new data-aware provisioning algorithm meeting the deadline requirements of applications executing in hybrid cloud environments. The proposed algorithm makes provisioning decisions by taking into account the data transfer time of scheduling tasks onto public cloud resources. This significantly affects data-intensive applications requiring large amount of data transfer. The main novelty of our approach is that while the solution takes into account bandwidth and data locality, it continually and dynamically updates scaling decisions based on the changes in the average runtime of the tasks and data transfer rates.

- The proposed algorithm is plugged into the Aneka platform that allows dynamically adding and removing resources from public clouds into the Aneka resource pool to meet user-defined application deadline requirements.

- Aneka is extended to support dynamic resource provisioning capabilities based on the Microsoft Azure Resource Manager (ARM) deployment service model.

- In an actual hybrid cloud environment built using local resources (desktop machines) and Azure virtual machines, we compared our method with existing approaches and demonstrated Aneka and its new provisioning algorithm's ability to meet deadlines for data-intensive applications. As a case study, a data-intensive application in the smart cities context is employed to measure the walkability index for different neighborhoods of the city of Melbourne utilizing spatial analysis of a large dataset [7].

The rest of the paper is organized as follows: Section 2 presents the motivation for this work and defines the problem domain. Section 3 outlines a general overview of the Aneka framework and describes the dynamic provisioning mechanisms of Aneka. Our proposed algorithm for deadline-driven data-aware resource provisioning is described in Section 4 and its realization in Aneka is discussed in Section 5. Section 6 is dedicated to a performance evaluation of the proposed algorithm. It discusses the hybrid cloud testbed built on top of computing resources from desktop grids and the Microsoft Azure cloud. Then, it describes the data-intensive case study application focused on measuring the walkability index and the assorted experimental results. Section 7 presents related work. Finally, Section 8 presents conclusions and offers future directions.

2. Motivation and Problem Domain

One of the main challenges for efficient scaling of applications is the location of the data relative to the available computational resources [8]. Colocating data and computation is evidently ideal in terms of performance especially for data-intensive applications. However, this is not always feasible for various reasons. For example, data might be located in the storage nodes of the user's local organizational infrastructure (e.g., a cluster or desktop grid) with limited or overloaded computational resources and the user facing deadline constraints may prefer to leverage on-demand computing resources from a public cloud provider to reduce the execution time of the application.

In the above particular scenario, it may not be ideal for the user to move the entire data set to the cloud as the data transfer time, due to the data size and network bandwidth, might dominate over the performance gain resulting from utilizing external CPUs. Moving data to distant computational resources, in particular for big data and data-intensive applications, to get access to more CPUs is often inefficient and can become the bottleneck in many cases. Therefore, any scheduling and resource provisioning algorithm aimed at improving data-intensive application performance by dynamic acquisition of cloud resources must take into account the time and amount of data movement. In other words, attempts to address the scheduling problem of data-intensive application would not be successful if they take into account computation separately from data movement. Accordingly, we focus on data-aware scheduling of data-intensive application considering data locality as well as monetary and performance costs of transferring data that has been neglected by many other scheduling methods in the literature.

Specifically, we focus on characteristics and requirements of hybrid cloud schedulers for executing deadline-constrained Bag-of-Tasks applications having large volumes of data. We assume that the application workload consists of a number of parallel tasks that each can run on an independent computing node. In addition, each task is associated with a data set residing within the local infrastructure of the user that has to be fully transferred to the public cloud for those tasks to be executed on externally provisioned computational resources. Finally, the application has a deadline by which all its tasks must finish their execution.

In a hybrid cloud setting, the execution of an application happens through cloud bursting deployment models. Cloud bursting allows an application to run in a private data center and burst into a public cloud when more resources are required to meet a given deadline. In this paper, the application is primarily scheduled on private resources allocated from organizational infrastructure based on a best-effort algorithm. The scheduling algorithm needs to compute the time left for the deadline based on the average runtime of tasks. Extra resources from public clouds are dynamically allocated if the scheduling algorithm determines the number of resources (locally) acquired by the application is insufficient to meet the deadline. This process must take place repeatedly to continually update the average runtime of tasks. Note that in the remaining part of the paper, we use the term *runtime* to refer to the time period required to execute a task and the term *execution time* to refer to the total execution time of the application.

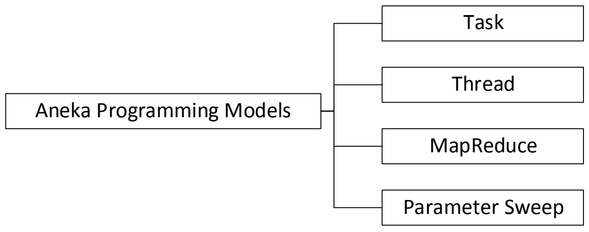


Figure 1: Aneka Programming Models.

The main new added feature compared to our previous work is that the scheduler explicitly takes into account the size and transfer time of input/output data for the estimation of the required resources. We employ the Aneka middleware as the basis for supporting the proposed scheduling and provisioning algorithms to transparently execute the application in a hybrid cloud setting.

3. Aneka and Dynamic Resource Provisioning

Aneka [6] is a software platform and framework facilitating the development and deployment of distributed applications onto clouds. It offers a collection of tools to build, control, and monitor cloud environment. The Aneka cloud built up this way can be composed of a collection of heterogeneous resources from a public cloud virtual infrastructure available through the Internet, a network of computing nodes in the premises of an enterprise, or a combination of both. Aneka provides developers with Application Programming Interfaces (APIs) for transparently exploiting physical and virtual resources in the Aneka cloud. Developers express the logic of applications using programming models and define runtime environments on top of which applications are deployed and executed. As shown in Figure 1, Aneka currently supports four different programming models [6]: *Bag of tasks model*, *Distributed threads model*, *MapReduce model*, and *Parameter sweep model*.

3.1. Aneka Architecture

The core components of the Aneka framework are designed and implemented in a service-oriented fashion.We briefly describe the architecture and the fundamental services that comprise the Aneka platform. Following this,

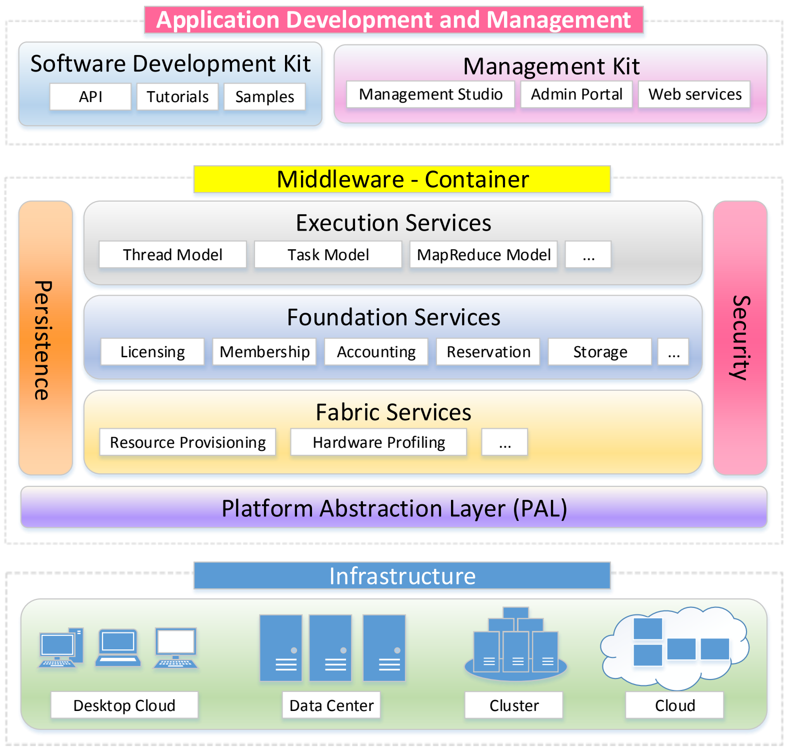


Figure 2: Aneka Framework Overview [6].

we focus on the scheduler and provisioning services that are central to this paper.

Figure 2 provides a layered view of the Aneka components. Aneka provides a runtime environment for executing applications by leveraging heterogeneous resources on the underlying *infrastructure* built on the top of computing nodes employed from network of workstations, clusters, grids, and data centers. In other words, the infrastructure layer is a collection of nodes hosting components of Aneka middleware.

The *middleware* provides a collection of services for interactions with the Aneka cloud. The container represents the unit of deployment of Aneka clouds and the runtime environment for services. The core functionalities residing in the *Platform Abstraction Layer (PAL)* constitute the basic services that are used to control the infrastructure of Aneka clouds. It provides a uniform interface for management and configuration of nodes and the containers instances deployed on them in the infrastructure layer. Middleware is composed of two major components representing the building blocks of Aneka clouds: the *Aneka Daemon* and *Aneka Container*. Each node hosts the Aneka daemon and one or more Aneka container instances. The daemon is a management component controlling the container instances installed on the particular node. A node forms the infrastructure layer running the Aneka master container which plays the role of resource manager and application scheduler. Nodes running Aneka worker containers are responsible for processing and executing work units of the applications. In addition, each container provides a messaging channel for accessing features of different services provided by the container. There are three classes of services characterizing the container:

1. *Execution services*: are responsible for scheduling and executing applications. Specialized implementations of these services are defined for execution of work units of each programming model supported by Aneka.
2. *Foundation services*: are in-charge of metering applications, allocating resources, managing the collection of available nodes, and keeping the services registry updated.
3. *Fabric services*: provide access to the physical and virtualized resources managed by the Aneka cloud. The *Resource Provisioning Service (RPS)* enables horizontal scaling out and allows for elastic and dynamic growth and shrinkage of the Aneka cloud to meet Quality of Service (QoS) requirements of applications.

The services of the middleware are accessible through a set of interfaces and tools in the *development and management* layer. The *Software Development Kit (SDK)* embodies a collection of abstractions and APIs for definition of applications and leveraging existing programming models. 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*, a comprehensive graphical environment providing a global view of the cloud for administrators.

3.2. Aneka Scheduling and Dynamic Resource Provisioning

Dynamic provisioning is the ability to dynamically acquire resources and integrate them into existing infrastructures and software systems. In the most common case, resources are *Virtual Machines* (VMs) acquired from an *Infrastructure-as-a-Service* (IaaS) cloud provider. Dynamic provisioning in Aneka happens as part of the Fabric Services by offering provisioning services for allocating virtual nodes from public cloud providers to complement local resources. This is mainly achieved as a result of the interaction between two services: the *Scheduling Service* and the *Resource Provisioning Service*. The former triggers on-demand provisioning requests based on the system status and the requirements of applications, while the latter is responsible for interacting with IaaS providers to instantiate VMs and deploy Aneka containers to meet the requests.

Execution of applications in Aneka happens through allocating tasks to the available set of resources in a dynamic fashion using the existing scheduling algorithms. Scheduling algorithms might be designed to leverage dynamic provisioning to cope with the application or system requirements. The scheduling algorithm makes decisions regarding when and how many resource allocations must take place to meet the application QoS requirements.

Aneka supports interactions with different resource providers, e.g., Amazon Elastic Computer Cloud (EC2), Microsoft Azure, XenServer, and GoGrid, using its dedicated provider-specific *resource pool* component. The main operations performed by this component are the translation of provisioning requests into provider specific requests, controlling the life cycle of VMs, and shutting them down when they are no longer needed. The life cycle of resource pools and redirecting provisioning requests, their release, or directing queries to the appropriate pool is the responsibility of the *pool manager* component. The pool manager also notifies the provisioning service when a dynamic resource is activated and terminated. Figure 3 illustrates a schematic overview of Aneka's dynamic provisioning mechanism.

Aneka features several provisioning algorithms that are designed to support dynamic provisioning of virtual resources. Among these, the algorithms proposed in [4] and [6] are designed to leverage dynamic resources to meet the deadline requirements of Bag-of-Tasks applications. The algorithm proposed by Veccholia et al. [6], which we call `Default` here, makes an estimation of the expected completion time of the application with currently available resources and if the expected completion time is later than the deadline defined in the Quality of Service parameters of the application, it requests extra resources from the public cloud to complete the application within given deadlines. This algorithm provides a best effort strategy for meeting the required deadlines based on the average task runtime estimation. Since the Default

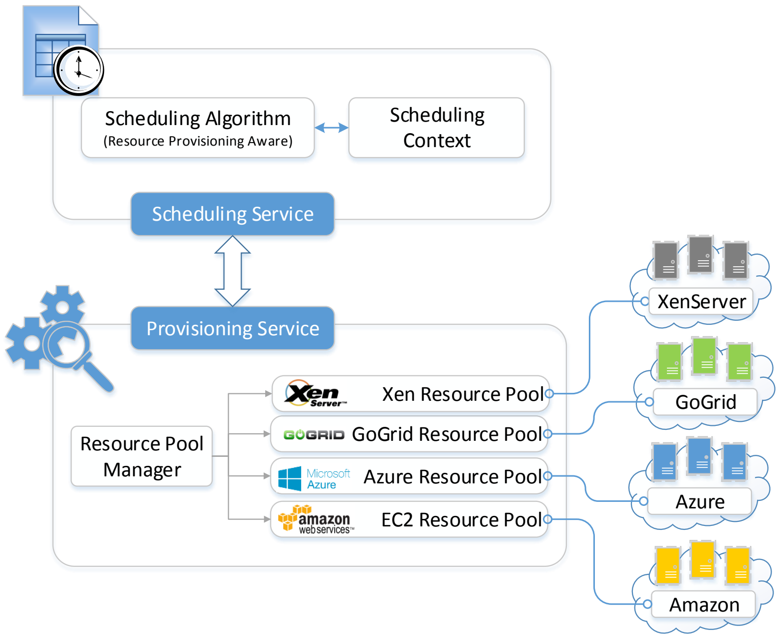


Figure 3: A schematic overview of Aneka's dynamic provisioning.

algorithm ignores the deployment time of resources (e.g., VM startup time) during the calculation of number of extra required resources, Calheiros et al. [4] proposed an improved provisioning algorithm, which we call `Enhanced` here, that dynamically leases resources to meet deadlines while it takes deployment time of resources into account. The Enhanced algorithm is designed to utilize Amazon EC2 Spot Instance resources with on average higher deployment time but lower budget than the Default algorithm. One of the key differences between our proposed provisioning algorithm in the next Section and these existing algorithms is that we consider the network bandwidth and data transfer time during calculation of extra resources. In Section 6, we compare our proposed algorithm with the Default and Enhanced algorithms based on different parameters such as application execution time, the number of resources launched on the public cloud, and the cost of resources.

4. Deadline-driven Data-aware Resource Provisioning Algorithm

Suppose that a private cloud with a limited number of resources in the local infrastructure is available for execution of a data-intensive Bag-of-Tasks application. Since the number of tasks that can be running concurrently on the private cloud is limited, to meet the deadline requirement of the application, extra resources from the public cloud need to be acquired to scale out available resources. We assume that the public cloud provider is able to fulfill all requests and thus has an infinite number of resources available from the user's perspective. Note that the network *bandwidth* available between the private and public cloud is limited and can impose a significant amount of data transfer time for each task running on the public cloud resource.

We assume that the application's workload consists of a number of trivially parallel tasks, each requiring specific input data files located in the local infrastructure. Bag-of-Tasks applications with independent tasks are used in a variety of scenarios, especially when the same piece of computation logic must be executed over a large volume of data, e.g., Monte Carlo simulations, data mining algorithms, parameter sweep applications [9]. This makes them suitable for hybrid cloud scenarios as tasks running on two separate clouds do not need to communicate with each other.

Algorithm 1 presents the newly proposed dynamic resource provisioning approach for Aneka, called *Data-aware* provisioning. The Data-aware algorithm takes into account the available bandwidth and the data size associated with each task and calculates the number of extra resources required to meet the deadline constraints of the application. The algorithm is executed when any of the following conditions are observed: (i) a task from the application is queued, and (ii) execution of a task completes. The variable `toGrow` at line 16 is `True` if the former condition happens and is `False` otherwise. If completion of a task triggers the algorithm and the computed number of extra resources by the algorithm is negative, a release request for the task allocated resource is submitted to the *Resource Provisioning Service*; otherwise, extra resources computed by the algorithm are requested to be added to the pool. Moreover, in order to reduce unnecessary calls of the algorithm, we only execute the algorithm in the growing mode if the average runtime of tasks is increased in comparison to the previous round of algorithm execution. Similarly, we call it in the shrinking mode if the average runtime of the task is decreased in comparison to the previous round of algorithm execution. For the sake of brevity, these conditions are not shown in the algorithm.

Algorithm 1 Data-aware Provisioning Algorithm.

```
1: $privateCores \leftarrow$ private cores available for the application;
2: $avgTaskRuntime \leftarrow$ Average task runtime on a prviate core;
3: $timeRemaining \leftarrow$ Time to application deadline;
4: $totalTasks \leftarrow$ Total number of tasks in the application;
5: $tasksCompeleted \leftarrow$ Total number of tasks compeleted so far;
6: $startupTime \leftarrow$ Startup time of a resource (VM);
7: $tasksInPrivate \leftarrow \lfloor \frac{timeRemaining}{avgTaskRuntime} \times privateCores \rfloor$;
8: $tasksRemaining = (totalTasks - tasksCompeleted - tasksInPrivate)^+$;
9: $totalTransferTime \leftarrow tasksRemaining \times \frac{taskInputDataSize}{upBandwidth}$;
10: $actualTimeRemaining \leftarrow (timeRemaining - startupTime - totalTransferTime)^+$;

11: $avgTaskRuntimeOnPublic \leftarrow$ Average task runtime on public core;
12: $provisionedCores \leftarrow$ Current daynamically prorvisioned cores;
13: $totalExecutionTime \leftarrow tasksRemaining \times avgTaskRuntimeOnPublic$;
14: $tasksPerCore \leftarrow \lfloor \frac{actualTimeRemaining}{avgTaskRuntimeOnPublic} \rfloor$;
15: if $tasksPerCore < 1$ then
16:     if $toGrow$ then
17:         $totalCoresRequired \leftarrow provisionedCores$;
18:     else
19:         $totalCoresRequired \leftarrow provisionedCores - 1$
20:     end if
21: else
22:     $totalCoresRequired \leftarrow \lceil \frac{totalExecutionTime}{tasksPerCore \times avgTaskRuntimeOnPublic} \rceil$;
23: end if
24: $extraResources \leftarrow \lceil \frac{totalCoresRequired - provisionedCores}{numberofCoresPerResource} \rceil$;
```

The Data-aware algorithm checks if the currently available resources are sufficient for the completion of the application tasks within the given deadline based on estimation of the average runtime of tasks on the private resources (`avgTaskRuntime`). Note that `avgTaskRuntime` includes the data transfer time in the calculation as we assume that the data transfer time within the private cloud is insignificant compared to the task execution time. Therefore, we do not consider a separate variable to capture that in the algorithm. The algorithm first updates `timeRemaining` based on the left time to the deadline. Then it computes the number of tasks that can be completed on the private resources within the left time to the deadline (Line 4). Then, it calculates the number of remaining tasks (Line 8). Here $(x)^+$ means $max(0, x)$. These remaining tasks must be scheduled on the dynamic resources. Since the execution of tasks on resources from the public cloud requires transferring data from the local storage to the allocated VMs, the algorithm first calculates the total transfer time for the tasks that should be executed on the public cloud resources (Line 9). In Line 10, the actual remaining time which can be effectively used for the execution of the tasks on dynamic resources is computed by subtracting the total transfer time and the start-up time of resources from the remaining time to the deadline. Note that `avgTaskRuntimeOnPublic` in Line 11, contrary to `avgTaskRuntime`, is only calculated based on the actual time tasks are being executed on the public cloud resources and does not include any associated data transfer time, i.e., it is the time period from when the task starts execution on the compute node (after all input data is available) up to the moment its execution is over. In line 13, the total time required to execute all remaining task on a single public CPU core is calculated and stored in the `totalExecutionTime` variable.

By dividing the actual time remaining by the average runtime of tasks on public cloud resources, the number of tasks each CPU core on the public resources can execute is calculated. If the number of tasks can be executed on each core in the public resources is at least one, then the total number of required cores can be estimated by dividing `totalExecutionTime` by the result of tasks per core multiplied by the average runtime of the task (Line 22). Otherwise, there is not enough time for allocating new resources and the algorithm sets the number of required CPU cores to that of the already provisioned cores in the pool (Line 17). If this is a shrinking mode, the task allocated cores will be nominated for release (Line 19). Line 24, calculates extra resources that must be added to or removed from the pool by the ratio of extra required cores to the number of cores per resource.

5. Realization of Data-aware Provisioning Algorithm in Aneka

To support execution of applications in hybrid clouds, it is important for Aneka to have access to public cloud resources. The Aneka resource provisioning service currently supports provisioning for providers such as Amazon EC2, GoGrid, and Microsoft Azure. Microsoft Azure has undergone a significant transformation in recent years and as a result, there are two different sets of APIs for resource management and deployment in Azure: Azure Resource Manager (ARM) and Classic. Aneka originally supported the Classic deployment model in which each resource (e.g., storage disk, VM, Public IP address, etc.) existed independently and there was no way to group related resources together. In 2014, Azure introduced ARM, which added the concept of a resource group as a container for resources that share a common lifecycle. As part of implementing Data-aware provisioning algorithm, we added an Azure-specific resource pool based on ARM APIs into Aneka which handles provisioning requests for Azure.

Since in the hybrid cloud environment, computational nodes are scattered throughout multiple networks (i.e., private and public cloud networks), public IP addresses are required to make the communication between nodes possible. Providing public IP addresses for all nodes, in particular for private resources in the organizational infrastructure, is not always feasible. Therefore, we used Azure Virtual Private Network (VPN) to solve this issue. We configured a Point-to-Site (P2S) Virtual Network in Azure which allows a secure connection from an individual client computer to the virtual network. Point-to-Site is a VPN connection over SSTP (Secure Socket Tunneling Protocol) and does not require a VPN device or a public-facing IP address to work. Figure 4 shows a sample Azure Point-to-Site virtual network configuration.

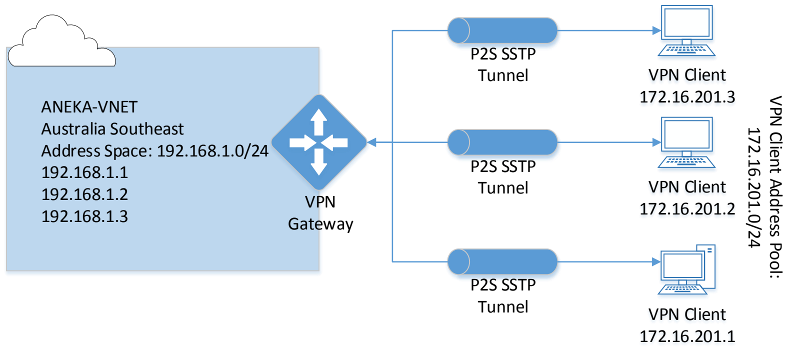


Figure 4: A sample Azure Point-to-Site virtual network.

Aneka provides interfaces for installation of Aneka containers on static resources (i.e., private cloud resources) via its Aneka Management Studio. However, in dynamic resource provisioning when machines are added on-demand by instantiating VMs, the installation must be done automatically without any human intervention. Therefore, we create virtual machine images on Azure for Aneka Workers with an initial configuration for containers. To complete the installation process, instantiated VMs are further configured using PowerShell scripts to set the Master container IP and Port, configure IP addresses in configuration files and to start the container service.

Finally, to support execution of data-intensive applications in hybrid cloud environments, we incorporate the Data-aware algorithm in Aneka. Aneka offers abstract interfaces that can be used to implement new resource provisioning algorithms. Our algorithm is implemented as a new scheduling algorithm by implementing the `ISchedulingAlgorithm` interface. The algorithm invokes the provisioning service to add extra resources to the pool of available resources based on the QoS requirement of the application.

6. Performance Evaluation

6.1. Hybrid Cloud Setup

The experimental testbed used to evaluate the performance of the proposed resource provisioning and scheduling algorithms is a hybrid cloud environment constituting of two desktop machines (one master and one slave) residing at `The University of Melbourne` and dynamic resources provisioned from `Microsoft Azure`. A schematic view of the hybrid cloud in which the experiments are carried out is depicted in Figure 5 and the configurations of machines used in the experiment are shown in Table 1.

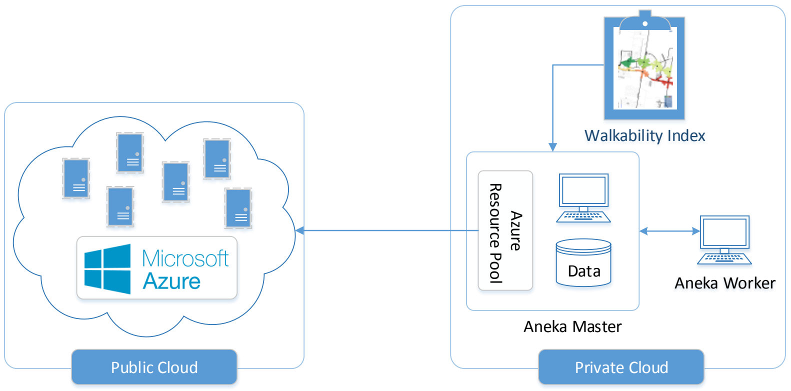


Figure 5: Hybrid cloud testbed.

Public cloud resources are dynamically provisioned from Microsoft Azure cloud when local resources are not able to meet application deadlines. Public cloud resources are deployed on `Azure Australia Southeast` region using `Standard DS1` VM instances with the configuration shown in Table 1. The initial estimated start-up time of Azure VMs used by the proposed algorithm

Table 1: Configuration of machines used in the experiments.

Machine	Type	CPU	Cores	Memory	OS
Master	Intel Core i7-4790	3.60 GHz	8	16GB	Windows 7
Worker	Intel Core i7-2600	3.40 GHz	8	8GB	Windows 7
Azure Instances	Standard DS1	2.4 GHz	1	3.5GB	Windows Server 2012

is 250 seconds. However, this is updated by a new value each time a VM is added to the resource pool.

The master and worker machines are connected through a high bandwidth LAN connection that imposes a negligible data transfer time between the local machines. On the other hand, the data transfer from the data repository on the master node to dynamic resources on the cloud is performed over the Internet. Since, the Internet connection on the master is relatively high for the intended experiments; we used `NetLimiter`[1] Version 4 to limit both the up-link and down-link bandwidth of the master node to 2MB/sec.

6.2. Walkability Application and Dataset

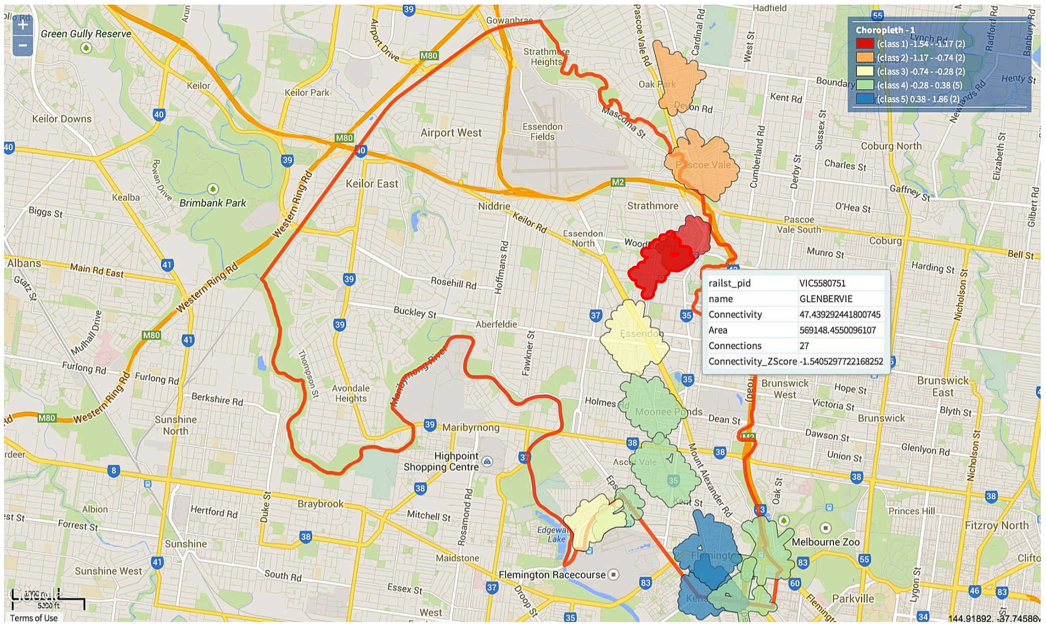
The data-intensive application used for the experiments is a Bag-of-Tasks for measuring a walkability index [7]. A walkability index is used to assess how walkable a given neighborhood is based on factors such as road connectivity, gross dwelling density and the land use mix in the area. The information can be used for better understanding the growing urban challenges such as obesity and increasing dependence on cars. The walkability index application is part of the AURIN project which supports nationwide urban and built environment research across Australia [10].

In our experiments, we use the walkability application to provide walkability indexes for 220 different neighborhoods in the city of Melbourne. The walkability application suits the purpose of our experiments since it is data-intensive and it can be broken into independent tasks, each computing a walkability index for a neighborhood. The test application contains 55 tasks, each calculating walkability indexes for four different neighborhoods The input data for each task contains geospatial datasets for road networks provided by the *Public Sector Mapping Agent (PSMA)* Australia[2] and points representing the center of each neighborhood. The size of input data for each task

[1]NetLimiter, `https://www.netlimiter.com/download`.

[2]PSMA, `https://www.psma.com.au/`

is 130MB including the Java executable JAR file. The input dataset must be transferred to the computing machine from the Aneka data repository residing in the master node. The output is the computed walkability score for each neighborhood and its size for each task is negligible. Figure 6 displays a sample snapshot of output data for several neighborhoods visualized on the map according to the computed walkability indexes. Each area is colored based on the walkability score with blue marked area noted as the most walkable and red representing the least walkable.


Choropleth - 1
🟥 (class 1) -1.54 - -1.17 (2)
🟧 (class 2) -1.17 - -0.74 (2)
🟨 (class 3) -0.74 - -0.28 (2)
🟩 (class 4) -0.28 - 0.38 (5)
🟦 (class 5) 0.38 - 1.86 (2)

railst_pid	VIC5580751
name	GLENBERVIE
Connectivity	47.439292441800745
Area	569148.4550096107
Connections	27
Connectivity_ZScore	-1.5405297722168252

Figure 6: A snapshot of visualized Walkability indexes computed for a set of neighborhoods around Melbourne.

6.3. Experimental Results

In order to evaluate the performance of the proposed data-aware algorithm, we first submit the walkability application to Aneka for execution without setting a deadline for the application. Later, we repeat the execution with different deadlines, showing how the proposed algorithm behaves. All experiments are repeated for the Default [6] and Enhanced [4] algorithms for the sake of comparison. By running the preliminary experiment, we are able to estimate the expected execution time of the application, which allows us to impose a deadline triggering the resource provisioning in other experiments. We identified that the execution time of the application without setting a deadline which only uses private (local) resources takes 45.4 minutes. Therefore, we considered test scenarios with four different deadline values of 35, 40, 45 and 50 mins. Note that the deadline value of 35 mins is the tightest deadline we trial in our experiments since all algorithms failed to meet the deadline within 30 mins.

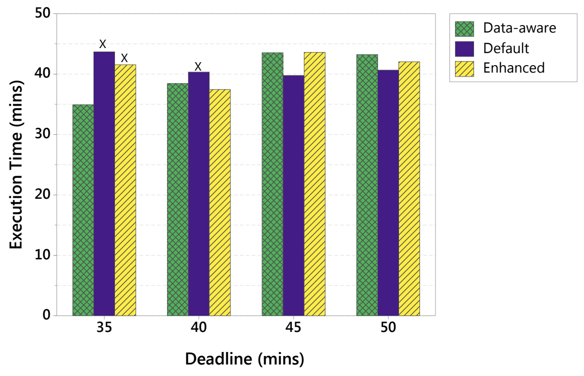
Figure 7 and Table 2 show the results of the application execution under different deadlines in two different forms. As shown by Figure 7(a), the proposed data-aware algorithm meets the deadline in all scenarios. However, the Default and Enhanced algorithms miss the deadline under tight deadline constraints (Bars marked by "x"), i.e., both algorithms violate the deadline constraint when the deadline is set to 35 minutes whilst the Default algorithm misses the deadline when it is set to 40 minutes. The key reason is that these algorithms rely on only a single variable for measuring average runtime of tasks (the value includes the data transfer time) to allocate dynamic resources that is significantly different in our scenario for private and public cloud resources. This difference is due to the dissimilarity of the data transfer time for the tasks executed on the private and public cloud resources which in is dependent on the difference in the available bandwidth for each case. We conclude that our new algorithm is able to meet strict application deadlines by taking into account the start-up time of the VMs and data transfer time, while the Default and Enhanced algorithms fail to do the same as they underestimate the overall runtime by neglecting significantly different intra and inter data transfer time in the hybrid cloud setting.

The experimental results also show that Data-aware algorithm instantiates a lower number of VMs in comparison to the two other algorithms when deemed necessary (e.g., at deadline values of 40, 45, and 50 minutes). This can be justified by the fact that the Data-aware algorithm takes into account the data transfer time and is less affected by the fluctuation of the average runtime of tasks. The fluctuation of the average task runtime leads to more hasty decisions by the Default and Enhanced algorithms to terminate VMs that may subsequently need to be instantiated again shortly after the termination. It is noted that the only case that the Data-aware algorithm instantiates a higher number of VMs than other algorithms is when the deadline is 35 minutes (i.e., 19 VMs compared to 10 and 11 for Default and Enhanced, respectively) however as noted both other competing algorithms fail to meet the deadline in this case.

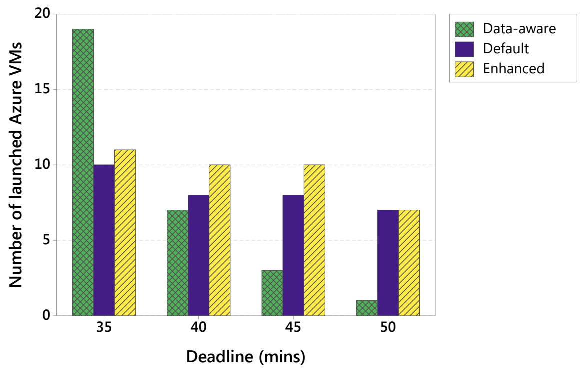
As shown by Figure 7(a) and (c) and Table 2, the proposed Data-aware algorithm not only guarantees the deadline but also optimizes the (monetary) cost spent on allocation of public cloud resources from Azure. Since Azure VMs are billed on a *per-minute* basis, we report the total number of minutes that dynamically added resources are kept running in the resource pool by each algorithm as an indication of cost. Compared to Data-aware, the Default and Enhanced algorithms run dynamically provisioned VM resources for a higher number of minutes in all cases where they meet the deadline. This happens because of both the higher expenditure on total start-up time of VMs due to more frequent instantiation and termination of VMs and the over-provisioning of resources due to late or wrong decisions regarding the addition of dynamic resources.

As stated before, the estimated execution time of the application under no deadline constraints is roughly 45 minutes. Therefore, we expect that none of the algorithms provision any dynamic resources when the deadline is set to a value higher than 45 minutes. However, all algorithms witness some degree of over-provisioning when the deadline is set to 50 minutes or even 45 minutes. The reason behind this is the primary estimate provided for the tasks runtime which is higher than the actual average runtime in our case can lead to wrong initial decisions made by algorithms and adding dynamic resources to the pool of resources. In the Aneka platform, whenever a resource is allocated to a task the mapping is irreversible unless a task or resource failure occurs. As a result, all algorithms use dynamically added VMs for the execution of at least one task. As can be seen in both Figure 7 and Table 2, the Data-aware algorithm incurs a lower amount of over-provisioning compared to other algorithms in case of over-estimation of task runtime in the initial setup of the scheduling algorithms.

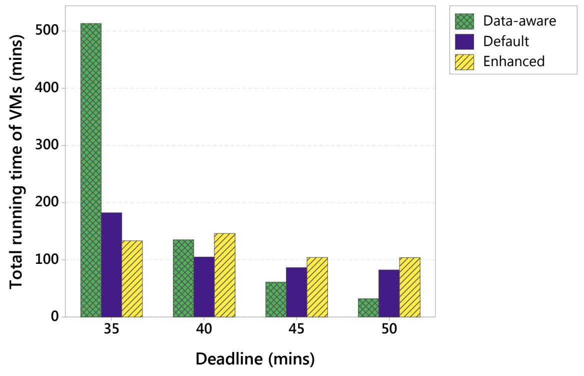
Our conclusion is that providing the Aneka scheduler with more accurate initial estimates of the runtime of tasks helps to minimize these over-provisioning issues. Moreover, considering the start-up of VMs that are particular to different infrastructures together with the size of the transferred data and network bandwidth are necessary in providing an efficient scheduling in the Aneka platform and this is particularly acute for data-intensive applications. In line with this, our experimental results show that our proposed algorithm is able to meet strict application deadlines with minimal budget expenditure by taking into account such factors.



(a)



(b)



(c)

Figure 7: (a) Execution time (b) Number of launched Azure VMs and (c) Total running time of VMs for Default, Enhanced, and Data-aware algorithms considering different application deadlines. The X symbol shows a violated deadline.

Table 2: Experimental results for Default, Enhanced, and Data-aware algorithms considering different application deadlines.

	Execution Time (mins)			Launched Azure VMs (#)			Total running time of VMs (mins)		
Deadline(mins)	Default	Enhanced	Data-aware	Default	Enhanced	Data-aware	Default	Enhanced	Data-aware
35	44	42	35	10	11	19	182	133	513
40	40	37	38	8	10	7	105	146	135
45	40	44	44	8	10	3	86	104	61
50	41	42	43	7	7	1	82	104	32
$+\infty$	45	45	45	0	0	0	0	0	0

7. Related Work

Resource provisioning is one of the most challenging problems in the cloud environment [11]. Resources must be allocated dynamically according to the Quality of Service (QoS) requirements and workload changes of the application. Autonomic systems provide solution to this problem by offering the environment in which dynamic resource provisioning for applications can be performed without human intervention [12]. To achieve the goal of autonomic systems, the *Monitor-Analyze-Plan-Execute* (MAPE) reference model proposed by IBM is of the most popular architectures. Ghobaei-Arani et al. [13] proposed a generic resource provisioning framework for cloud applications based on the MAPE architecture. In contrast to our approach, they focus on a single cloud application and do not investigate the impact of network latency, bandwidth constraints, and location of data on the dynamic resource provisioning.

The idea of using public cloud resources to expand the capacity of local infrastructure has been explored by many studies. Mateescu et al. [14] propose an architecture that provides a platform for the execution of High-Performance Computing (HPC) scientific applications. The cornerstone of the proposed architecture is the *Elastic Cluster* which makes an expandable hybrid cloud environment. Their approach differs from ours since we focus specifically on data-intensive applications and take into account the impact of data transfer times. Assunção et al. [15] analyze the trade-off between performance and usage costs of different provisioning algorithms for use of resources from the cloud to expand a cluster capacity. Similar to [14], they neglect the impact of data transfer time. Javadi et al. [16] propose failure-aware resource provisioning policies for hybrid cloud environments which they evaluated using a model-based simulation as opposed to our real case study performance evaluation. Xu and Zhao [3] propose a privacy-aware hybrid cloud framework which supports a tagging mechanism for the location of sensitive data. While they focus on compliance with the location of sensitive data, we focus on the data locality and data transfer time to compute the number of public cloud resources required by the application in order to meet deadlines. Belgacem and Chopard [17] conduct an experimental study of running a large, tightly coupled, distributed application over a hybrid cloud consisting of resources from Amazon EC2 clusters and an existing HPC infrastructure. They evaluated the overhead of using public cloud resources for a tightly coupled, massively parallel MPIs application in which tasks are communicating with each other without considering QoS related factors such as deadlines. However, in this work, we look into the design of dynamic resource provisioning algorithms meeting the deadline constraint of the locally coupled Bag-of-Tasks applications. Mattess et al. [18] present a provisioning algorithm for extending cluster capacity with Amazon EC2 *Spot Instances*. In contrast to our work, they focus on compute-intensive applications without considering network related costs and delays. Yuan et al. [19] propose a profit maximization model for private cloud providers by utilizing the temporal variation of prices in hybrid cloud. While similar to many others, they assume the time and cost related to data and network are negligible. The majority of these works focus largely on theoretical aspects and evaluate their method through simulation, while we focus on practical aspects and execute our case study on a real hybrid cloud environment using the Aneka platform with a real application.

Scheduling and resource provisioning techniques in a hybrid cloud for data-intensive applications where the data transfer time is comparable to computational time, adds new levels of complexities requiring addressing the impact of network latency, bandwidth constraints, as well as economic aspects such as costs and prices. A thorough survey on resource scheduling and provisioning in cloud environments has been conducted by Singh and Chana [20]. Bossche et al. [5] proposed scheduling algorithms to deal with cost optimization problem for deadline-constrained applications while taking into account data constraints, data locality and inaccuracies in task runtime estimates. Similar to our approach, their algorithm considers computational and data transfer costs as well as network bandwidth constraints. However, their work is a simulation-based study and does not consider the dynamic nature of networks and fluctuations in the runtime of tasks. Kailasem et al. [21] propose a Hadoop-based cloud bursting framework for data-intensive workloads. They focus on checkpointing schemes to overlap data download with processing and upload. They consider the performance optimization of interactive jobs while we target Bag-of-Tasks jobs. They also ignore the uncertainty of job execution runtimes. Similar to our work, Bicer et al. [22] propose a resource allocation framework suitable for a hybrid cloud settings to support time and cost sensitive execution of data-intensive applications. Different from our work, their method is designed for Map-Reduce applications and is based on a feedback mechanism in which the compute nodes regularly report their performance. Malawski [23] proposed a mixed integer nonlinear programming model to tackle the problem of resource allocation on multiple heterogeneous clouds taking into account the cost of instances and data transfers. While we focus both on how many and when extra resources must be added in a dynamic fashion, they try to provide a single decision using a mixed integer non-linear programming model to tackle the resource provisioning problem in hybrid cloud environments. In contrast to our work, they have used simulation to evaluate their method.

Scheduling and resource provisioning in hybrid clouds has been researched for other types of application as well. Examples include big data analytics [24], workflows applications [25], online commerce [26], mobile phone applications [27] and compute intensive applications [28].

8. Conclusions and Future Directions

Supplementing on-premises private infrastructure of organizations with dynamically provisioned resources from public cloud providers introduces the problem of cost-efficiently executing applications. This paper presents a provisioning algorithm for scheduling deadline-constrained data-intensive applications while taking into account aspects such as data transfer time, the location of data, and the network bandwidth. This work builds upon previously proposed provisioning algorithms for the Aneka platform for developing and deploying scalable applications on the cloud. In this work, we propose a provisioning algorithm that computes the extra resources needed to complete application tasks within deadlines by considering aspects such as data locality, start-up time of public cloud resources, network bandwidth, and data transfer time. We demonstrate that our proposed algorithm is able to meet strict deadlines for a sample data-intensive application while minimizing cost and the total number launched instances compared to other existing algorithms. Contrary to other algorithms, the proposed algorithm measures the average runtime of tasks on public cloud resources as a separate variable and takes the data transfer time calculated based on the available bandwidth into account.

In this paper, we focused on scheduling and resource provisioning of Bag-of-Tasks applications with a set of trivially parallel tasks which can be executed independently of one another. However, there are other common types of data-intensive applications with a non-trivial workflow structure that includes precedence constraints and data dependencies between tasks. A meaningful future work is to extend our proposed algorithm for workflows with data dependencies. This is a complex problem and the major challenge in the design of such algorithms in addition to consideration of data transfer time and data locality is how to devise the task grouping and task assignment techniques that minimize inter-cloud communications.

Another future work in line with this contribution consists of developing provisioning policies that can support the integration of multiple clouds with different pricing and network latency in Aneka. We also plan to look into the scheduling of Bag-of-Tasks applications with tasks partly sharing input data and other QoS parameters such as budget constraints.

Future directions of data-intensive application scheduling include new algorithms for multi-cloud resource allocation, innovative provisioning algorithms honoring user requirements such as privacy and the location of sensitive data, energy efficient techniques for minimizing cost and carbon footprint. This can be further extended to support resource allocation techniques leveraging software-defined networks. This can also be explored in the context of various other programming paradigms such as Map Reduce model.

Acknowledgments

This work was partially supported by Australian Research Council (ARC) Future Fellowship and the Australia-India Strategic Research Fund (AISRF). We thank Microsoft for providing access to the Azure IaaS infrastructure. We also thank William Voorsluys for his technical support for the execution of the case study application.

References

[1] C. P. Chen, C.-Y. Zhang, Data-intensive applications, challenges, techniques and technologies: A survey on big data, Information Sciences 275 (2014) 314 – 347. doi:10.1016/j.ins.2014.01.015.

[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (6) (2009) 599 – 616. doi:10.1016/j.future.2008.12.001.

[3] X. Xu, X. Zhao, A framework for privacy-aware computing on hybrid clouds with mixed-sensitivity data, in: Proceedings of the IEEE International Symposium on Big Data Security on Cloud, 2015, pp. 1344–1349. doi:10.1109/HPCC-CSS-ICESS.2015.110.

[4] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, R. Buyya, The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds, Future Generation Computer Systems 28 (6) (2012) 861 – 870. doi:10.1016/j.future.2011.07.005.

[5] R. V. den Bossche, K. Vanmechelen, J. Broeckhove, Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds, Future Generation Computer Systems 29 (4) (2013) 973 – 985. doi:10.1016/j.future.2012.12.012.

[6] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, R. Buyya, Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka, Future Generation Computer Systems 28 (1) (2012) 58 – 65. doi:10.1016/j.future.2011.05.008.

[7] R. O. Sinnott, W. Voorsluys, A scalable cloud-based system for data-intensive spatial analysis, International Journal on Software Tools for Technology Transfer 18 (6) (2016) 587–605. doi:10.1007/s10009-015-0398-6.

[8] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: Proceedings of 2008 Grid Computing Environments Workshop, 2008, pp. 1–10. doi:10.1109/GCE.2008.4738445.

[9] F. A. da Silva, H. Senger, Scalability limits of bag-of-tasks applications running on hierarchical platforms, Journal of Parallel and Distributed Computing 71 (6) (2011) 788–801, Special Issue on Cloud Computing. doi:10.1016/j.jpdc.2011.01.002.

[10] R. O. Sinnott, C. Bayliss, A. Bromage, G. Galang, G. Grazioli, P. Greenwood, A. Macaulay, L. Morandini, G. Nogoorani, M. Nino-Ruiz,

M. Tomko, C. Pettit, M. Sarwar, R. Stimson, W. Voorsluys, I. Widjaja, The australia urban research gateway, Concurrency and Computation: Practice and Experience 27 (2) (2015) 358–375, cPE-13-0325.R1. doi:10.1002/cpe.3282.

[11] M. Amiri, L. Mohammad-Khanli, Survey on prediction models of applications for resources provisioning in cloud, Journal of Network and Computer Applications 82 (2017) 93–113. doi:10.1016/j.jnca.2017.01.016.

[12] S. Singh, I. Chana, R. Buyya, STAR: SLA-aware autonomic management of cloud resources, IEEE Transactions on Cloud Computingdoi:10.1109/TCC.2017.2648788.

[13] M. Ghobaei-Arani, S. Jabbehdari, M. A. Pourmina, An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach, Future Generation Computer Systemsdoi:10.1016/j.future.2017.02.022.

[14] G. Mateescu, W. Gentzsch, C. J. Ribbens, Hybrid computing–where HPC meets grid and cloud computing, Future Generation Computer Systems 27 (5) (2011) 440 – 453. doi:10.1016/j.future.2010.11.003.

[15] 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. doi:10.1007/s10586-010-0131-x.

[16] B. Javadi, J. Abawajy, R. Buyya, Failure-aware resource provisioning for hybrid cloud infrastructure, Journal of Parallel and Distributed Computing 72 (10) (2012) 1318 – 1331. doi:10.1016/j.jpdc.2012.06.012.

[17] M. B. Belgacem, B. Chopard, A hybrid hpc/cloud distributed infrastructure: Coupling EC2 cloud resources with HPC clusters to run large tightly coupled multiscale applications, Future Generation Computer Systems 42 (2015) 11 – 21. doi:10.1016/j.future.2014.08.003.

[18] M. Mattess, C. Vecchiola, R. Buyya, Managing peak loads by leasing cloud infrastructure services from a spot market, in: Proceedings of the 12th IEEE International Conference on High Performance Computing and Communications (HPCC), 2010, pp. 180–188. doi:10.1109/HPCC.2010.77.

[19] H. Yuan, J. Bi, W. Tan, B. H. Li, Temporal task scheduling with constrained service delay for profit maximization in hybrid clouds, IEEE Transactions on Automation Science and Engineering 14 (1) (2017) 337–348. doi:10.1109/TASE.2016.2526781.

[20] S. Singh, I. Chana, A survey on resource scheduling in cloud computing: Issues and challenges, Journal of Grid Computing 14 (2) (2016) 217–264. doi:10.1007/s10723-015-9359-2.

[21] S. Kailasam, P. Dhawalia, S. J. Balaji, G. Iyer, J. Dharanipragada, Extending mapreduce across clouds with bstream, IEEE Transactions on Cloud Computing 2 (3) (2014) 362–376. doi:10.1109/TCC.2014.2316810.

[22] T. Bicer, D. Chiu, G. Agrawal, Time and cost sensitive data-intensive computing on hybrid clouds, in: Proceedings of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012), 2012, pp. 636–643. doi:10.1109/CCGrid.2012.95.

[23] M. Malawski, K. Figiela, J. Nabrzyski, Cost minimization for computational applications on hybrid cloud infrastructures, Future Generation Computer Systems 29 (7) (2013) 1786–1794. doi:10.1016/j.future.2013.01.004.

[24] F. J. Clemente-Castell, B. Nicolae, K. Katrinis, M. M. Rafique, R. Mayo, J. C. Fernndez, D. Loreti, Enabling big data analytics in the hybrid cloud using iterative mapreduce, in: Proceedings of the 8th IEEE/ACM International Conference on Utility and Cloud Computing (UCC), 2015, pp. 290–299. doi:10.1109/UCC.2015.47.

[25] M.-A. Vasile, F. Pop, R.-I. Tutueanu, V. Cristea, J. Koodziej, Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing, Future Generation Computer Systems 51 (2015) 61–71. doi:10.1016/j.future.2014.11.019.

[26] G. Lackermair, Hybrid cloud architectures for the online commerce, Procedia Computer Science, World Conference on Information Technology 3 (2011) 550 – 555. doi:10.1016/j.procs.2010.12.091.

[27] H. Flores, S. N. Srirama, C. Paniagua, A generic middleware framework for handling process intensive hybrid cloud services from mobiles, in:

Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia, MoMM '11, ACM, New York, NY, USA, 2011, pp. 87–94. doi:10.1145/2095697.2095715.

[28] M. Brock, A. Goscinski, Execution of compute intensive applications on hybrid clouds (case study with mpiblast), in: Proceedings of the Sixth International Conference on Complex, Intelligent, and Software Intensive Systems, 2012, pp. 995–1000. doi:10.1109/CISIS.2012.109.