Shared Data-Aware Dynamic Resource Provisioning and Task Scheduling for Data Intensive Applications on Hybrid Clouds using Aneka

Shreshth Tuli[1,2], Rajinder Sandhu[3], Rajkumar Buyya[1]

Abstract

In the recent years, data-intensive applications have been growing at an increasing rate and there is a critical need to solve the high-performance and scalability issues. Hybrid Cloud Computing paradigm provides a promising solution to harness local infrastructure and remote resources and provide high Quality of Service (QoS) for time sensitive and data-intensive applications. Generally, hybrid cloud deployments have a heterogeneous pool of resources and it becomes a challenging task to efficiently utilize resources to provide optimum results. In modern data hungry applications, it is crucial to optimize bandwidth consumption, latency and networking overheads. Moreover, most of them have large extent of file sharing capability. The existing algorithms do not explicitly consider file sharing scenarios that leads large data transmission times and has severe effects on latency. In this direction, this paper focuses on building upon existing dynamic resource provisioning and task scheduling algorithms to provide better QoS in hybrid cloud environments for data intensive applications in a shared file task environment. The efficiency of proposed algorithms is demonstrated by deploying them on Microsoft Azure using Aneka, a platform for developing scalable applications on the Cloud. Experiments using real-world applications and datasets show that proposed algorithms are able to allocate tasks and extend to public cloud resources more efficiently, reducing deadline violations and improving response times to give response time reduction of upto 40.12% for a sample local alignment search application on genome sequences.

Keywords: Shared file aware, Dynamic Provisioning, Task Scheduling, Hybrid Cloud, Aneka Platform as a Service, Data-intensive applications

1. Introduction

Cloud computing has emerged as one of the most effective computing paradigms for hosting and executing data-intensive applications. Cloud computing can serve most of the requirements of data intensive applications such as large amount of computation, storage, dynamic provisioning, pricing, scalability, and reliability [1]. Cloud computing paradigm also guarantees to fulfill the Quality of Service (QoS) requirement of the application based on the signed Service Level Agreement (SLA). When private cloud resources are not able to fulfill the demand of data intensive applications, it can be easily offloaded to public cloud for more computation power or storage, this type of setup is commonly known as hybrid cloud computing.

With the advancements in network technologies and computing paradigms, data intensive applications have increased multiple folds. Applications in different domains are generating large amount of structured and unstructured data which need to be analyzed in timely and effective manner. Many scientific applications like genome sequencing, drug testing and design have large scope for exploiting file sharing capabilities. Other domains like nuclear physics, bio-informatics and real-time mapping softwares like Google Maps have huge amounts of shared data [2, 3]. Internet of Things (IoT) based architectures further increases the data-intensive applications domain with real-time data generation and strict deadlines for analysis. Applications from multiple domains such as smart cities, smart transport, smart healthcare etc. require data to be collected and analyzed with high quality also within given deadlines to be useful [4]. Even though modern cloud platforms support such data intensive applications, they do fail to perform optimally when large number of heavy files are shared. Primary challenge of deadline sensitive data intensive application running on cloud computing infrastructure is to complete all tasks in given deadline to achieve desired QoS.

For example, in smart healthcare applications, past health data of a critical patient needs to be analyzed in a strict given deadline so that treatment can be started at the earliest. To achieve the vision of deploying such deadline sensitive application on Cloud computing resources, efficient schedulers needs to be developed which can utilize both private and public cloud resources effectively. Schedulers should be able to add or release public cloud resources based on the deadline and cost requirement of the application [5]. Key parameter in data intensive applications running on public cloud is the data which

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

[2]Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Delhi, India

[3]Department of Computer Science and Engineering, Jaypee University of Information Technology, Solan, Himachal Pradesh, India

E-mail addresses: shreshthtuli@gmail.com (S. Tuli), rajsandhu1989@gmail.com (R. Sandhu), rbuyya@unimelb.edu.au (R. Buyya)

need to be transferred along with the tasks. Without the data; tasks can not perform the computation and it will further dent the chances of meeting the desired deadline [6]. Data in these kinds of applications can be classified as input data, output data and shared data.

- *Input Data:* This data is unique to each task running on cloud computing infrastructure. Each task is bundled with its input data. For example, in any map service user provides the coordinates of source and destination to find the best route.
- *Output Data:* This is the output generated by the application after completing its execution on cloud computing infrastructure. Output data is communicated to end user or to another component of the application. In map services, the route to follow based on the coordinates given by the user is the final output.
- *Shared Data:* This is the data which is shared by all the tasks generated by the application. For example, spatial data containing roads, buildings, and constructions in any location. Depending on the input coordinates, route can be different or same for every users, but underling common spatial information is shared by all the users using map service application.

Multiple frameworks, architectures, middlewares have been proposed in the literature for using public cloud resources along with private cloud for achieving deadlines. Large volume of literature is focused on cost and execution time minimization of tasks running in hybrid cloud setup [7, 8, 9, 10, 11]. Cloud datacentres being at multiple hops from data intensive applications, data transfer time is one of the key elements in meeting the deadline of the application. However, existing works have not considered type of data, data transfer time, network latency and data locality collectively for migrating tasks to public Cloud infrastructure. Although, the issue of data locality and their role in successful deployment of application in hybrid Cloud computing has been discussed by Toosi et al. [6]. However, they have ignored the shared data which also need to be transferred to the public cloud for computation of tasks. Specially, considering deadline sensitive parameter sweep applications containing multiple Bag of Tasks (BoTs), each task performs analysis on large shared data file (also known as common file) associated with it. If application has shared data file, it is preferable to colocate tasks in multi-core machines on public cloud so that it takes transfer time of only one shared file. As shared files are of large size they consume more time to transfer as compared to the input files which are generally of smaller size.

Public cloud contains multiple configuration of virtual machines with different number of cores and storage. It is also a challenge to provision perfect combination of different configurations to offload the tasks on public cloud infrastructure. The main aim of this paper is to consider type of data file, data transfer time, network bandwidth, and data locality while scheduling and dynamic provisioning public cloud resources of different configuration for parameter sweep based data intensive applications.

To achieve this goal, algorithms have been proposed in this paper which take various parameters related to application, task and data into account and decide how many instances of different configuration need to be provisioned to satisfy the deadline of the application. Aneka platform is used to implement the proposed algorithm. Aneka [12] is a Platform as a Service (PaaS) middleware which provides Software Development Kit (SDK) for rapid development and deployment of cloud applications on hybrid cloud infrastructure. Aneka was introduced in 2009 and since then there have been various advancements which has still kept it the most robust and reliable platform for Cloud deployments. In recent years, Aneka has extended its support to large number of public cloud providers like Amazon AWS and Microsoft Azure. It has also been allowed to support private clouds using Open Stack or GoGrid frameworks. Aneka 5.0, a recent vesion released in 2019, has been further extended to work with Virtual Private Networks (VPNs) for creating secure cloud computing environments. Moreover, new applications that support Ensemble Deep Learning and Computer Vision have been developed using Aneka. Recent works on Aneka have also focused on developing enhanced task scheduling algorithms for data intensive tasks [6]. This work, however, focuses on further enhancing the scheduling algorithms for modern applications which require large amount of shared data across different tasks and physical machines, thereby leveraging the computation resources with highest efficiency and improving performance of such cloud systems. Aneka has pre-built Application Program Interfaces (APIs) from which, end user can effectively deploy and mange cloud applications. Aneka 5.0 has multiple scheduling and dynamic provisioning algorithms for hybrid Cloud computing infrastructure. However, implementing proposed algorithms in Aneka further enhances the capability and functionality to handle shared data intensive application in efficient manner. The key contributions of this paper are:

- Shared-file aware task scheduling and dynamic resource provisioning algorithms for meeting the deadline constraints and provide optimal QoS parameters to the Cloud applications.
- Aneka platform functionality has been extended by using more than one resource pools of different configurations by a single application.
- The proposed algorithms are implemented in Aneka using the Dynamic Provisioning Service which interact with Azure Resource manager (ARM) and extend the Scheduling class.
- The algorithms were tested on real private-public (Microsoft Azure) Cloud testbed to demonstrate their efficiency and improvement compared to prior works.

The rest of the paper is organized as follows. Section 2 provides the related work to the proposed work from available literature. Section 3 discussed Aneka middleware for development and deployment of applications. Section 4 provides explanation of proposed algorithms for shared file aware task schedul-

Table 1: Comparison of relevant works with our proposed approach

Paper	Deadline	Cost	Bandwidth	Input File Transfer time	Shared File Transfer Time	Dynamic Provisioning	Different Configuration Public Machines	Software for Implementation
[13] & [14]	✓	✓						Open Nebula
[15]	✓	✓						Open Nebula
[16]	✓	✓	Partial					Hadoop
[17]	✓	✓						Simulations
[7]	✓	✓	✓	✓		✓	✓	CPLEX optimization software
[10]	✓	✓	✓	✓		✓		Java based simulator
[8]	✓	✓	✓	✓		✓		Simulator
[9]	✓	✓	✓	✓		✓		Simulator
[18]	✓	✓						CloudSim Simulator
[5]	✓	✓				✓		Aneka
[19]	✓	✓				✓		Aneka
[6]	✓	✓	✓	✓		✓		Aneka
This work	✓	✓	✓	✓	✓	✓	✓	Aneka

ing and dynamic provisioning with implementation details in Section 5. Section 6 has performance evaluation of proposed algorithms on Aneka platform. Section 7 concludes the paper along with discussion on future directions.

2. Related Work

Many works have explored Multi Criteria Decision Making (MCDM) and meta-heurtics based scheduling algorithms in hybrid cloud which are compiled by Kalra and Singh in [20]. Nayak and Tripathy proposed an AHP [13] and MCDM [14] based scheduling algorithm for deadline sensitive applications in Cloud computing environment. They used Open Nebula for implementation of proposed scheduler and did not consider dynamic provisioning on public Clouds. Nayak et al. [15] again used Open Nebula to devise a backfilling based scheduling algorithm for deadline constraint tasks on cloud computing environment. Proposed algorithm resulted in better results as compared with default FCFS algorithm. Fan et al. [16] used agent based framework for selection of appropriate VM instances from multiple Cloud providers based on the deadline of the job. They consider network bandwidth partially while selection of appropriate public Cloud provider but do not consider data transfer time in their algorithm. Moschakis and Karatza [17] proposed a simulated annealing meta-heuristic based scheduler for multi criteria deadline sensitive BoT applications. Although, approach used deadline sensitive BoT applications which are similar to application used in this paper but failed to consider transfer time and different configurations available in public Cloud. Zuo et al. [11] proposed an ant colony based scheduling algorithm for deadline sensitive application in Cloud computing. They did not consider dynamic provisioning and data transfer time as compared to proposed algorithms in this paper.

Other related works have considered data transfer time of input file and discussed benefit of using different configuration and providers. Some of the relevant work has been cited and discussed however no relevant literature was found which consider shared file and actual implementation of scheduler in hybrid Cloud setup. Abdi et al. [7] consider the deadline sensitive BoT applications to schedule on multiple clouds. However, they consider data transfer time and different configuration of cloud resources, but they failed to separate the shared file and input file. Their proposed model is mathematical and numerical results were generated using optimization software rather than actual implementation. Bossche et al. [10] proposed a model named HICCAM similar to [7] model and consider data transfer of input data file while making scheduling decision. They used backfilling similar to [15] for taking shortest jobs infront of the queue but failed to experimentally test in actual cloud computing environment. Malawski et al. [8] tried to minimize the cost of using public cloud for deadline sensitive application by formulating it as mixed integer nonlinear programming problem. Like other models, they failed to take shared file transfer time into account and used a simulator to test the proposed method. Malawski et al. [9] proposed a model which schedule as well as dynamically provisioned resources for deadline sensitive scientific workflow applications. They simulated the proposed model and did not consider the shared file transfer time for workflow tasks. Xiong et al. [18] considered the deployment of deadline sensitive data intensive applications on Cloud computing infrastructure. They designed a framework for the scheduling purpose but did not consider dynamic provisioning, different configuration of public resources, and data transfer time.

The algorithms proposed in this paper are part of novel work extending previously designed algorithms in Aneka platform for scheduling data intensive applications on hybrid cloud. Vecchiola et al. [5] implemented deadline sensitive scheduling algorithm in Aneka for data intensive BoT task applications. However, they did not consider the booting time of public machines, data transfer of files and different configuration requirement which resulted in bad performance when data size is large. Calheiros et al. [19] used the reservation service in Aneka and developed an algorithm which considers the spot market and reserved resources price of public clouds. Toosi et al. [6] mentioned that in data intensive application; data plays an important role and not considering data transfer time will further decrease the performance of scheduling algorithm. They devised the algorithm after considering input data file size and dynamically decided how many public cloud resources will be required. Taking data transfer time into account enhanced the

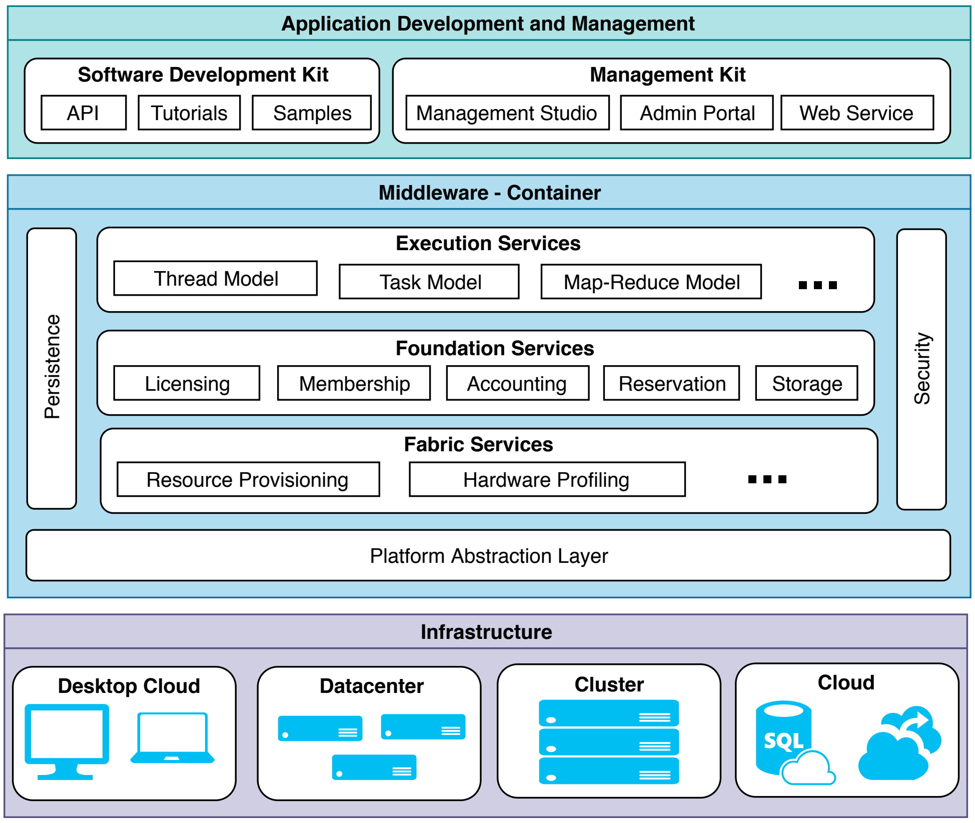


Figure 1: Aneka framework overview

performance of scheduler to large extent. However, all these proposed algorithms failed to discuss the role of shared file in BoT data intensive applications. They also used only single type of public cloud resource and did not provision machines with multiple configurations. Work done in this paper further enhanced the performance and functionalities of Aneka platform by adding multiple resource pools with different configuration and providers which also consider shared file while scheduling.

3. Background - Aneka

Aneka [12] is a Platform as a Service (PaaS) that provides users and developers to deploy distributed computing applications on Grid or Cloud computing platforms. Aneka provides many services, tools and APIs that enable developers to build, control and integrate new capabilities to test and realize their optimized algorithms. Aneka allows developers to make use of private and public cloud resources. These features differentiate Aneka from other Cloud management software and characterize it as a platform for development and deployment of applications. It provides four different programming models: Bag-of-Tasks model, Distributed Threads model, Parameter sweep Model and Map-Reduce model. A summary of the Aneka framework with architecture and services is provided in Figure 1.

3.1. Architecture

In this section, the Aneka architecture and the fundamental services are discussed along with scheduling and dynamic provisioning services which are central to this work. Aneka is built in a multi-layer modular fashion where the lower most layer provides the computation resources which comprises the infrastructure including the desktop clouds, datacenters, clusters and cloud environments. This layer is the collection of computing nodes that interact with the Middleware containers and provide themselves as the hosts for Aneka containers.

The Middleware containers provide a layer to glue the infrastructure to the Aneka distributed applications and management. The Middleware containers contain diverse services and tools that act as the backbone of the Aneka platform. The Platform Abstraction Layer (PAL) provides the core services to manage and control the cloud resources. The PAL also provides an interface to manage the containers and configure their deployment on the underlying infrastructure. The Middleware consists of two major components: Aneka Daemon and Aneka Container. The former has only one instance in each host and controls the containers instances installed in a host. The latter can have many instances in a host and controld the scheduling and execution of the Aneka applications. In a cloud environment, there exist one or more Aneka Master containers that manage worker containers, provision resources and schedule tasks among them. The hosts on which the Aneka worker containers are installed perform the computation necessary for execution of tasks and provide results to Aneka master. The services are divided into three classes:

- *Execution Services* that perform scheduling and execution of tasks.
- *Foundation Services* that perform monitoring, accounting and storage management.
- *Fabric services* that perform resource provisioning and profiling for dynamic growth of shrinkage of resources to meet the QoS requirements and deadlines of applications.

To allow file sharing without compromising the security of the tasks, Aneka support encryption in each of its layers. It allows users to deploy their custom security measures for encryption and firewalls but also provides base security schemes in its default configuration. A collection of transversal services operate at all the levels of the container and provide persistence and security for the run-time 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. Apart from this all communication across Aneka Broker and worker nodes is encrypted via public-private key management. Furthermore, modules supported by the public cloud providers can also be integrated to further secure the system.

3.2. Scheduling and dynamic resource provisioning in Aneka

Aneka provides the ability to dynamically extend resources to public cloud and integrate them with the local infrastructure/resources through the *Aneka Dynamic Provisioning Service*. This service is part of the Fabric Services in the Middleware container and it is achieved by allocating virtual machines from public cloud resources in addition to the local compute nodes. This feature is enabled by the interaction of three main services: scheduling, monitoring and resource provisioning services. The Scheduling service triggers on-demand resource extension requests based on the status of the private infrastructure provided by the monitoring service. If applications are data and compute intensive, the Monitoring service detects this and forwards the resource utilization characteristics to the Scheduling service which then decides the number and types of public resources to be acquired and send this request to the Resource Provisioning service. This then interacts with Infrastructure as a Service (IaaS) to meet the requirements of the Scheduling service. The Scheduling service then gets to know the current number and status of resources and dynamically allocates tasks to the current pool of resources. The scheduling algorithm decides how many resource allocations must take place to meet the application deadline constraints and QoS requirements.

Aneka currently allows dynamic resource provisioning with Microsoft Azure, Amazon Elastic Computer Cloud (EC2), XenServer and GoGrid. The management of dynamic provisioning requests and notification of the provisioning service when dynamic resources are activated or terminated is the job of the pool manager. The Resource Provisioning service of Aneka interacts with this pool manager to acquire and destroy resources as requested by the Scheduling service as shown in Figure 2.

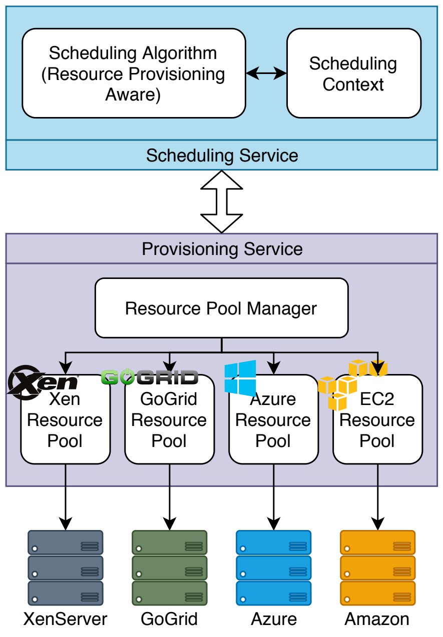


Figure 2: Schematic Overview of Aneka Dynamic Provisioning

4. Proposed Algorithm

In this section, we propose and discuss algorithms for task scheduling and dynamic resource provisioning. As noted in Sections 2, the earlier algorithms do not consider that data intensive applications have high scope of sharing files and data among various parallel tasks. This fact can be exploited to improve the task placement in a hybrid cloud environment and provide better QoS characteristics by reducing the number of shared-file transfers across nodes. This can be achieved by provisioning cloud resources and scheduling tasks so that shared files can be shared to reduce cost and enhance QoS.

4.1. Shared-File Aware Resource Provisioning Algorithm

In typical hybrid cloud deployments, local compute nodes are available for execution of data-intensive and time sensitive applications. Most of these tasks are generated as Bag-of-Tasks which consists of multiple tasks that usually are independent and can be executed in parallel. Furthermore, many tasks come with deadlines under which they must be completed. This is usually common in healthcare applications, traffic management and other time sensitive applications [4]. To execute such applications in their deadlines, dynamic provision becomes necessary in cloud environments to extend the private cloud resources to public cloud to be able to complete tasks in the required deadlines [19, 6, 5]. However, existing dynamic resource provisioning algorithms do not consider the possibility of shared files to allow better QoS in terms of response time and reduce the network bandwidth consumption.

Algorithm 1 also called Shared-File aware resource Provisioning, provides a seamless solution to extend the private cloud resources and allows use of shared file across tasks to a high extent. Like the Data-aware resource provisioning algorithm provided by Toosi et al. [6], proposed algorithm in this paper considers start-up time of public cloud VMs, file transfer times and available cores in private and public clouds. Additionally, it considers the transfer time of shared files. The variable $coreEstimated$ estimates the number of public cores required without considering the transfer time of input and shared files. This estimate is initially calculated as the quotient of total remaining task run-time (product of average task runtime and remaining time) and number of remaining tasks. This is because each task is run on a single core and for the remaining time if divided equally among all tasks gives us an average value of cores required for the remaining tasks. If number of estimated cores is finite then the total machines for each resource pool is calculated as quotient of estimated core count and number of cores in that resource pool.

The variable $coreEstimated$ is then used to allocate resources to the public cloud. For each resoure pool we obtain the number of machines required in terms of core count. This core requirement is stored in *required* list and is subtracted from $coreEstimated$. The total cloud machines required (saved in variable $totalMachinesRequired$) is the sum of the *required* list. The total transfer time of all files (shared and input files, and saved as $totalTransferTime$) is then estimated as sum of transfer times of shared files and input file. The input file's transfer time is the quotient of input file size and uplink bandwidth. This is multiplied by the number of remaining tasks to estimate total transfer time for all input files. For each shared file, the transfer time is quotient of shared-file size and

Algorithm 1 Shared File Aware Resource Provisioning

```
    Inputs:
1:  $availablePrivateCore \leftarrow$ Available cores in private cloud
2:  $upBandwidth \leftarrow$ Bandwidth of uplink
3:  $startupTime \leftarrow$ Startup time of a public resource (VM)
4:  $publicResourcePool \leftarrow$ Total public cloud resource pools available with increasing number of cores
5:  $publicResourcePool[i].core \leftarrow$ Number of cores in $i^{th}$ resource pool
6:  $publicResource \leftarrow$ Total public cloud resource available from all pools
7:  $actualTaskRuntimePublic \leftarrow$ Average run-time of tasks on public resource
8:  $timeRemaining \leftarrow$ Time to application deadline
9:  $avgTaskRuntime \leftarrow$ Average task runtime on private core
10: $totalTasks \leftarrow$ Total number of tasks in the application
11: $taskCompleted \leftarrow$ Total number of tasks completed
12: $provisionedCores \leftarrow$ Total resources already provisioned to application on public and private cloud
13: $tasksSubmitted \leftarrow \left\lfloor \frac{timeRemaining}{avgTaskRuntime} \times provisionedCores \right\rfloor$
    Output:
14: $tasksToSend$
    Begin
15: $tasksRemaining \leftarrow$ (totalTasks - tasksCompleted - tasksSubmitted)$^{+}$
16: $coreEstimated \leftarrow$ Requirement of public core estimated before taking transfer time into account
17: do
18:     $coreEstimated \leftarrow \frac{avgTaskRuntime \times timeRemaining}{tasksRemaining}$
19:     if $coreEstimated > 0$ then
20:         for $i = publicResourcePool$ to $i > 0$ do
21:             $required[i] \leftarrow$ Total number of machines required for $i^{th}$ resource pool
22:             $j \leftarrow \frac{coreEstimated}{publicResourcePool[i].core}$
23:             if $j \geq 1$ then
24:                 $required[i] \leftarrow j$
25:                 $coreEstimated \leftarrow coreEstimated - j \times publicResourcePool[i].core$
26:             end if
27:         end for
28:     end if
29:     $totalMachinesRequired \leftarrow \sum_i required[i]$
30:     $inputDataSize \leftarrow$ Total size of input of each task
31:     $sharedFileSize \leftarrow$ Size of shared-file used by tasks
32:     $transferTimeShared \leftarrow totalMachinesRequired \times \frac{sharedFileSize}{upBandwidth}$
33:     $transferTimeData \leftarrow tasksRemaining \times \frac{inputDataSize}{upBandwidth}$
34:     $totalTransferTime \leftarrow transferTimeShared + tranferTimeData$
35:     $actualTimeRemaining \leftarrow (timeRemaining - startupTime - totalTransferTime)$
36:     $tasksToSend \leftarrow \left\lfloor \frac{actualTimeRemaining}{avgTaskRuntimePublic} \times coreEstimated \right\rfloor$
37:     if $tasksToSend < tasksRemaining$ then
38:         $timeRemaining \leftarrow actualTimeRemaining$
39:     end if
40: while $tasksToSend < tasksRemaining$
41: if $tasksToSend < 1$ then
42:     Do not provision resources from public cloud
43: else
44:     Provision $tasksToSend$ resources from public cloud
45: end if
    End
```

uplink bandwidth. This is used because this quotient gives a rough estimate of the transfer time on an average for all files. However, this is multiplied by $totalMachinesRequired$ instead of remaining tasks as shared file is shared across all tasks on the same machine. Hence, actual remaining time becomes $actualTimeRemaining = (timeRemaining - startupTime - totalTransferTime)$. The number of tasks to send to public cloud is then estimated as product of $coreEstimated$ and estimate of remaining tasks in public cloud ($actualTimeReaming/avgTaskRuntimePublic$). This gives a new estimate of remaining time as $actualTimeRemaining$. The steps described above are repeated till convergence. The variable containing actual remaining time subtracts start-up time and transfer time from total remaining time, to get a better estimate of the execution time left. This helps in estimating the number of tasks to be sent to public cloud. The tasks that are sent to public cloud are those which if executed on public cloud can allow execution of all tasks within the remaining time and hence not violating the agreed deadline limit Finally, if estimated number of tasks to send to cloud is < 1 then public cloud resources are not used and all tasks are provisioned on private cloud.

4.2. Shared-File Aware Task Scheduling Algorithm

Consider a Bag-of-Tasks (BoT) being sent to a hybrid cloud environment for computation. Generally, many of the tasks that belong to this BoT share files required for computation. This can be observed in many real-life applications such as healthcare, traffic management, and other distributed computation tasks [4]. It is assumed that the workload of applications consist of many tasks that are trivially parallelizable, each requiring specific input data files, some of which may be shared, located in the local infrastructure. For the simplicity it is assumed that the size of tasks that is the number of cores they require is of the form 2^n so that the tasks (and sub-tasks) can be partitioned into two equal sub-tasks always.

Exploiting the fact of file sharing across different tasks, a new task scheduling algorithm has been proposed, shown in Algorithm 2. This aims to assign tasks to VMs in such a manner that minimizes the number of shared-file transfers by having maximum extent of file sharing among the tasks. Algorithm 2, also called Shared-File aware scheduling takes into account different tasks which may be subdivided into different cores as per their size. This group of sub-tasks divided from a single task can share files and hence the algorithm prefers to schedule such sub-tasks to the same Aneka container. To achieve this, the algorithm considers the tasks in decreasing order of their sizes as *taskList*, and tries to schedule them to the container with the highest number of cores possible to maximize the extent of file sharing. The algorithm maintains *containerCapacity* of each Aneka container to prevent overloading of containers. If a task can not be completely assigned to any of the available containers then it is partitioned into two equal size sub-tasks and these are inserted into the *taskList*. If size of remaining tasks is 1 and they can not be assigned to any container then they are assigned in the next scheduling interval. Now, when considering the tasks/sub-tasks in decreasing order of size, the tasks/subtasks that have not been partitioned are considered first. This is because it is more likely that the partitioned sub-tasks of a task may be scheduled in different intervals and hence the whole task would complete only after the latest sub-task is complete. This helps in reducing the completion time of tasks on an average and is achieved by the ModifiedSort procedure. Overall, this allows tasks that have largest task/file size to be allocated to VMs with largest capacity and hence allowing highest possible size of shared file with reduction in the number of instances of shared files being resent for execution across different task running on different cores. As number of instances of shared file is lower across cores of different VMs and higher across cores on same VMs, it reduces the number of times the file must be uploaded to public cloud, thus reducing file transfer time and total response time.

Algorithm 2 Shared File Aware Task Scheduling

```
    Inputs:
 1: taskList ← List of tasks $[x_1, x_2, ..., x_n]$
 2: sizeOfTask(x) ← Number of cores required by $x = 2^n$
 3: partioned(x) ← Whether task x has been partitioned (false
    by default)
 4: containerList(x) ← List of containers $[c_1, c_2, ..., c_n]$
 5: containerCapacity(c) ← Capacity of container c
    Output:
 6: containerAssigned(x) ← Container assigned to task $x \in$
    containerList
    Procedure ModifiedSort(list)
 7: Sort list based on task size in decreasing order
 8: Break ties for tasks $x_1, x_2$, the one with partioned(x) =
    false comes first
    End Procedure
    Begin
 9: Sort taskList in decreasing order of task size
10: Sort containerList in decreasing order of number of cores
11: for task x in taskList do
12:     for container c in containerList do
13:         if complete task x can be assigned on c then
14:             containerAssigned(x) ← c
15:             containerCapacity(c)− = sizeOfTask(x)
16:             Break
17:         end if
18:     end for
19:     if x is unassigned then
20:         if sizeOfTask(x) == 1 then
21:             Push remaining tasks in taskList to queue for next
    scheduling interval
22:             Break
23:         end if
24:         containerList.remove(x)
25:         y, z ← partition x into equal subtasks
26:         partioned(y) ← true
27:         partioned(z) ← true
28:         containerList.append([y, z])
29:         modifiedSort(containerList)
30:     end if
31: end for
    End
```

In Section 5 we provide the details of how the two proposed algorithms are implemented for resource provisioning and task scheduling in Aneka.

5. Implementation in Aneka

Aneka provides support for dynamically extending resources to public cloud using Aneka Dynamic Resource Provisioning service. This service currently supports provisioning resources from providers including Microsoft Azure, Amazon EC2 and

GoGrid. For provisioning resources from Microsoft Azure, two sets of APIs exist: Azure Resource Manager (ARM) and Classic. Among these, ARM uses the definition of resource group as a container for resources that have a common lifecycle. To implement the Shared-File aware dynamic resource provisioning algorithm 1 we used Microsoft Azure resource pools and Aneka's ARM APIs which dynamically serve provisioning requests.

Moreover, to allow communication between the Aneka containers, we had to setup Azure Virtual Private Network (VPN) and Azure VPN Gateway. The Point-To-Site configuration of the Azure VPN allows communication of Aneka Master in local infrastructure connected to Azure VPN Gateway, with the Aneka Worker containers installed in Azure VMs connected to the same VPN. The connection is established using Secure Socket Tunneling Protocol (SSTP) which allows connecting to a VPN device without the need of a public-facing IP address. Aneka provides automatic installation of Aneka containers in on-demand VMs using a pre-designed configuration file.

Finally, for Aneka to adopt the proposed algorithms, proposed algorithms are incorporated in Aneka scheduling API [21]. Aneka source code provides template interfaces for integrating custom scheduling and provisioning algorithms. The proposed algorithms are implemented as a new scheduling and provisioning algorithms by utilizing the $ISchedulingAlgorithm$ interface in Aneka source code. The algorithms invoke the provisioning service to add extra resources to the pool of available resources based on the parameters of the shared files, deadline constraints and QoS requirement of the application. Additional field of $sharedFileSize$ are added in the QoS class to allow the algorithms to collect shared file parameters and use them to provide optimum scheduling and provisioning decisions. Further, once a file is declared as "shared" in the Aneka environment it remains shared even if its size changes with time. New files are automatically considered as shared if more than one tasks require those files.

6. Performance Evaluation

We compare the performance of both shared-file aware scheduling and resource provisioning algorithms with prior work to show their efficacy in terms of total execution time and/or file transfer time, VM run-time and number of VMs launched in public cloud. In this regard, this section presents application workload, experimental test-bed and performance results in static and dynamic cloud environments.

6.1. Basic Local Alignment Search Tool (BLAST) and Dataset

To evaluate the efficacy of the proposed algorithm, the data-intensive application with high degree of file sharing capability was used as a Bag-Of-Tasks of input sequences of Basic Local Alignment Search Tool (BLAST) [22] inputs. The set of programs part of the BLAST tool are widely used for searching sequence similarities among proteins and DNA databases. For protein comparisons, a variety of definitional, algorithmic and statistical refinements described here permits the execution time of the BLAST programs to be decreased substantially while enhancing their sensitivity to weak similarities. Different Bag-of-Tasks were tested to demonstrate the robustness and variation of performance improvement compared to related work for different cases. The Dataset comprises of the K-12 MG1655 section of Escherichia coli 1 of 400 of the complete genome of file size 1132 MB and the input sequences varied from 560 to 5600 letters in length. For experiments with the static cloud environment, different Bag-of-Tasks are used summary of which is shown in Table 2.

Table 2: Summary of different Bag-of-Tasks

Bag-of-Tasks	Number of Tasks	Average Input File Size	Shared File Size
BoT-1	10	1 KB	1132 MB
BoT-2	20	2 KB	1132 MB
BoT-3	30	2.5 KB	1132 MB

6.2. Static Cloud Environment

To compare the shared-file aware scheduling algorithm a static test setting is used. A fixed number of resources on the public and private cloud are sent Bag-of-Tasks which differ in the number and size of tasks. For different settings which comprised of private cloud, public cloud in East-US and public cloud in South-East Australia, execution times of different Bag-of-Tasks are compared.

6.2.1. Hybrid Cloud Setup

The experimental test-bed used for evaluating the performance of the proposed shared file aware resource provisioning and task scheduling algorithms consisted of both private and public cloud settings. The private cloud was deployed using multiple machines in the CLOUDS Lab, University of Melbourne, Australia. Microsoft Azure VMs act as the public cloud resources and are provisioned as and when local resources are not able to meet the application deadlines. The public cloud resources are deployed on two Azure geographical locations: Virginia (East US) and Victoria (South-East Australia). The test-bed consisted of the following cloud configuration adapted from other commonly used private cloud systems [6]:

I Private Cloud:
  (a) Dual Core Dell XPS 13 laptops with Intel i5-7200 CPU @ 2.50GHz, 8.00 GB DDR4 RAM, SSD Storage and 64-bit Windows 10. These are in CLOUDS Lab at University of Melbourne, Victoria, Australia. 2 systems of this type were used.
  (b) Quad Core Dell Latitude 5490 laptop with Intel i7-8650U CPU @ 1.9GHZ, 16.00 GB DDR4 RAM, SSD Storage and 64-bit Windows 10. This was also at CLOUDS Lab, University of Melbourne, Victoria, Australia. 3 systems of this type were used, giving a total of 5 laptops in private cloud.

II Azure Public Cloud in Victoria, Australia or Virgina, US:

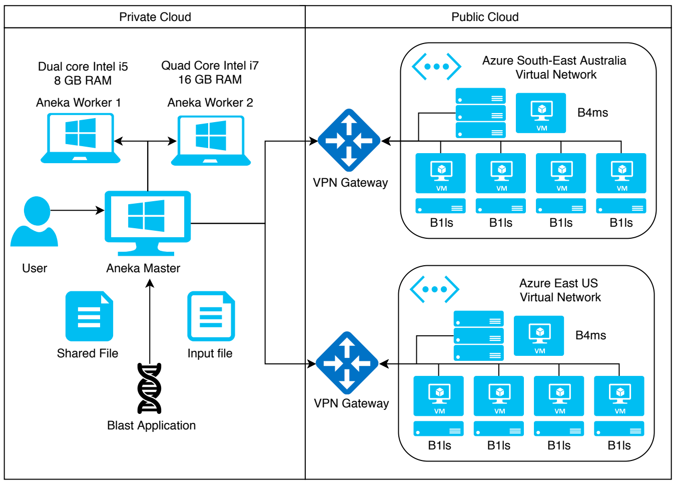


Figure 3: Experimental test-bed for Shared file aware Scheduling

(a) 4 - Single Core Azure B1ls virtual machines with 0.5 GiB RAM, SSD Storage and 64-bit Microsoft Windows Server 2016

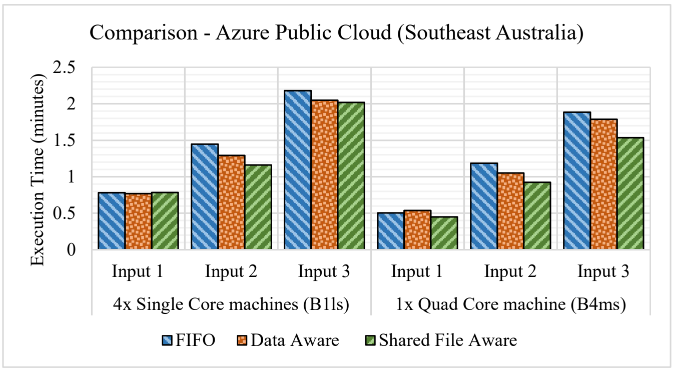
(b) 1 - Quad Core Azure B4ms virtual machine with 16 GiB RAM and SSD Storage and 64-bit Microsoft Windows Server 2016

The Aneka Master container was installed in a Dell XPS 13 Laptop at University of Melbourne, Australia and all Azure VMs are connected to the Master device through Azure Virtual Private Network (VPN) and Azure Gateway. A diagrammatic representation of the test-bed is shown in Figure 3. The master and worker nodes in the private cloud are connected through high-speed 1 GiBps LAN with very low data transfer time among private cloud machines. The data transfer from master device to the public cloud VMs are through Internet. The master device was connected to Internet with uplink bandwidth of 867 Mbps.

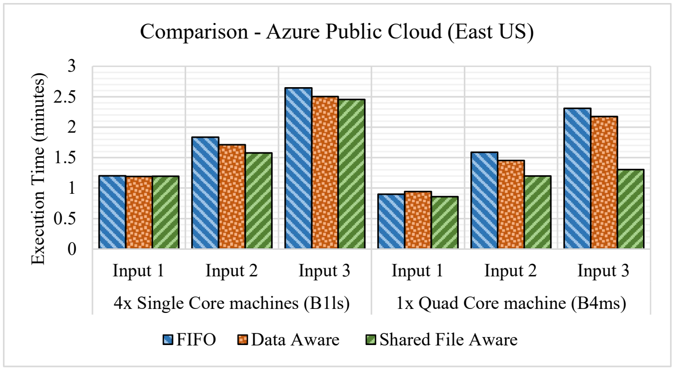
6.2.2. *Experimental Results*

In order to test the performance of the proposed shared-file aware task scheduling and resource provisioning algorithms we used the BLAST application dataset and ran different Bags-of-tasks on the deployment described in Section 6.2.1. We compared the following algorithms: First In First Out (FIFO), Data Aware scheduling, and Shared file aware scheduling. The execution times for the different cloud settings for these algorithms are shown in Figure 4.

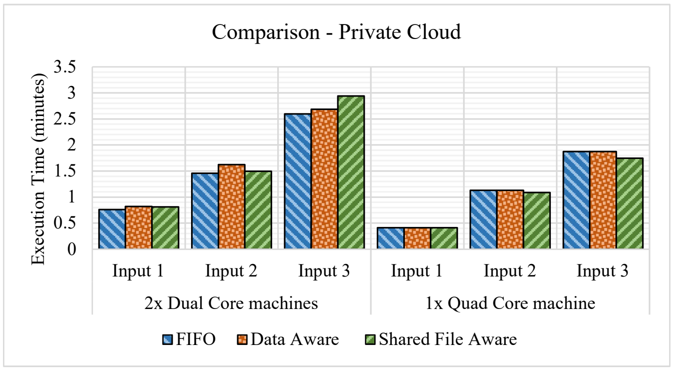
As shown in Figure 4a, time taken by the proposed shared-data aware algorithms is lest compared to the other two: First In First Out (FIFO) and Data-aware algorithm. Here the configuration is where there is only a public cloud used which is close to the Aneka Broker node. As shown by results, the shared-file aware scheduler has much lower execution time compared to the other two policies. The reduction in total execution time is much higher when multi-core systems are deployed. If we use a single quad core machine then file sharing is possible to the maximum extent as at each core we can run separate task with shared data across all cores. This however is limited in the



(a) Execution Times for Public Cloud in South-East Australia



(b) Execution Times for Public Cloud in East US



(c) Execution Times for Private Cloud

Figure 4: Execution Times for different cloud configurations

other case where there are 4 single core machines. Thus, the reduction in the total execution time is higher in the case with a single quad-core machine than in 4 separate single-core machines. This is expected because the shared files are sent only once when a bag of tasks is sent to quad core machine, but is sent multiple times when there are four single-core machines.

The reduction is further increased when the public cloud is far away as shown in Figure 4b. In this case, the public cloud is in East US, away from the Aneka broker which is in Southeast Australia. Here having a shared file aware scheduler allows us to send as many tasks together which have shared data so that this data does not have to be sent multiple times. This saves data transfer time and hence reduce total response time for the system. Here too, the difference is more significant in the multi-core case than in multiple single-core system.

Additionally, the results show that the shared-file aware scheduling provides a higher speedup when the file transfer time to worker devices is higher. The private cloud does not show any improvement compared to Data-aware scheduler as shown in Figure 4c. Instead there is a small amount of overhead for large number of file with higher size when tasks are sent to multiple single core machines and there is no scope of file sharing. The speedup is higher for the public cloud at South-East Australia, compared to private cloud due to higher file transfer time to Azure VMs across the VPN compared to time taken to transfer files on same LAN. The speedup further increases when the file transfer time increases in the case with public cloud at East-US. In this case, the speedup increases to as high as 1.67 (reduction in response time by 40.12%) compared to Data-aware scheduler for the sub-case of Quad-Core B4ms machine at BoT-3. This is due to high transfer time of files from the Aneka master container at CLOUDS lab to Azure VMs at Virginia USA. This shows that the shared-file aware scheduling algorithm provides best results when multi-core machines are deployed and transfer times are very high between Aneka Master and Worker containers.

Finally, as this is a static cloud environment where the number of cloud VMs remain constant at all times, the cost of task execution is same in each case.

6.3. Dynamic Cloud Environment

For the Shared-File Aware provisioning algorithm heterogeneous set of resource pools was used to compare the extent of file sharing and how well the algorithm chooses the type and number of resources to be used from the public cloud. Instead of comparing just execution times as done for shared-file aware scheduler, now tasks have deadlines and deadline-violations are also compared. Hence, a dynamic environment is used here where Aneka can instantiate new resources in the form of Azure VMs for completing tasks in required deadlines.

6.3.1. Hybrid Cloud Setup

Similar to the setup described earlier, the private Cloud was deployed in CLOUDS Lab, University of Melbourne, Australia. Public cloud resources were provisioned from Microsoft Azure when additional resources are required for meeting the task deadlines. The initial start-up time, used by the proposed algorithm, of the Azure VMs is assumed to be 150 seconds which was average of 10 boot-ups conducted for recording this time. Since the master container has a relatively high internet connection speed, for the experiments NetLimiter Version 4 was used to limit both the upload and download bandwidth of the master node to 50 MB/s. The Aneka Dynamic provisioning service was used to boot up new VMs and extend resources to public cloud. The following resources in private cloud and different types of resource pools were used:

I Private Cloud:

(a) Four Quad Core laptops with Intel i5-7200 CPU @ 2.50GHz, 4.00 GB DDR4 RAM, SSD Storage and 64-bit Windows 10. These were in CLOUDS Lab at University of Melbourne, Victoria, Australia.

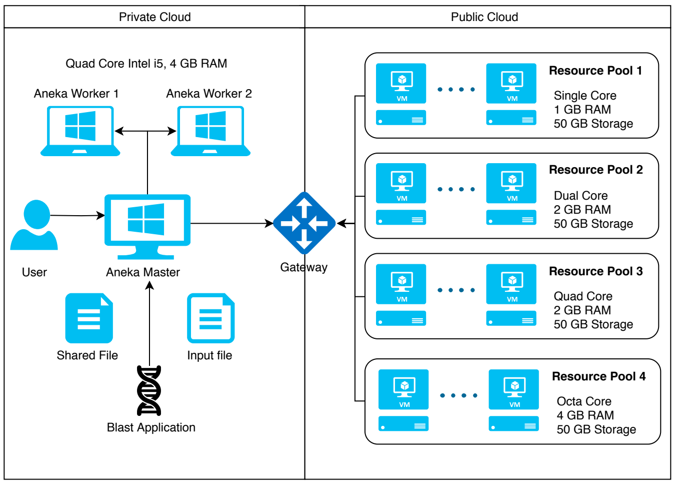


Figure 5: Experimental testbed for Shared file aware Provisioning

Table 3: Experiment parameters

Parameter	Value
Average task completion time	10 minutes
Boot up time	2.5 minutes
Internet speed	50 MB/s
Shared file size	1132 MB
Number of private cores	8
Number of task	50
Input File size	100 MB
transferTimeShared	3 Minutes
transferTimeFile	0.4 Minutes
Azure B1S machine cost[1]	0.02 USD/hour
Azure A1 machine cost[1]	0.08 USD/hour
Azure A2 machine cost[1]	0.15 USD/hour
Azure A3 machine cost[1]	0.33 USD/hour

II Azure Public Cloud: (Cost and details from Azure[1])

(a) Resource Pool 1: Single Core Azure B1s virtual machines with 1a GiB RAM, SSD Storage and 64-bit Microsoft Windows Server 2016

(b) Resource Pool 2: Dual Core Azure A1 virtual machine with 2 GiB RAM and SSD Storage and 64-bit Microsoft Windows Server 2016

(c) Resource Pool 3: Quad Core Azure A3 virtual machine with 2 GiB RAM and SSD Storage and 64-bit Microsoft Windows Server 2016

(d) Resource Pool 4: Octa Core Azure A4 virtual machine with 4 GiB RAM and SSD Storage and 64-bit Microsoft Windows Server 2016

A diagrammatic representation of this test-bed is shown in Figure 5. The Bag-of-Task parameters, file and network parameters are mentioned in Table 3.

Table 4: Number of Cores used for different deadlines

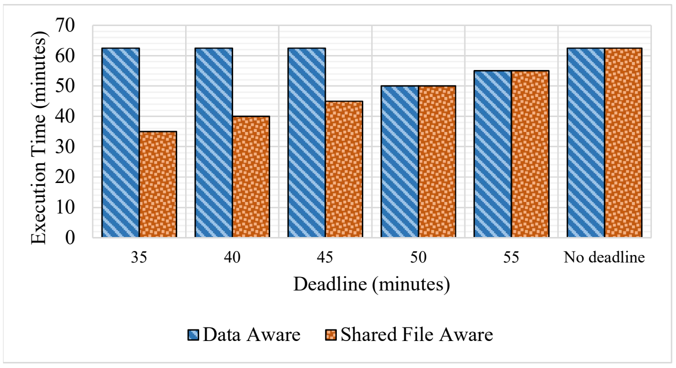
Number of Cores Used					
Deadline	8-core	4-core	2-core	1-core	Total cores
35	2	0	1	1	19
40	1	0	0	0	8
45	0	1	0	1	5
50	0	0	1	1	3
55	0	0	1	0	2

6.3.2. Experimental Results

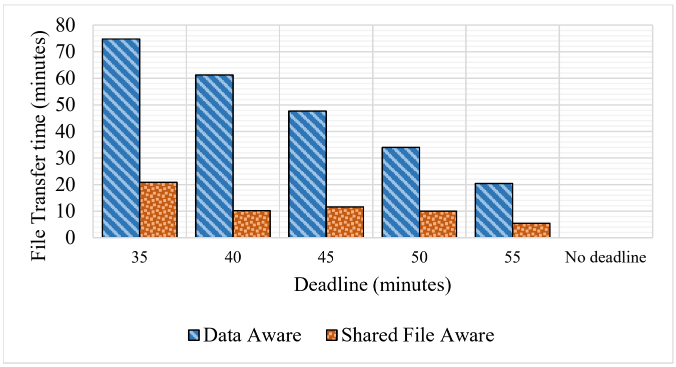
To measure and compare the performance of the shared-file aware dynamic resource provisioning algorithm, BLAST BoTs were submitted to the Aneka Master and used the Aneka Dynamic provisioning to boot up public cloud VMs as and when required. To demonstrate the robustness of the proposed provisioner, tasks with different deadlines was submitted. For different deadlines, the total number of cloud resources provisioned in terms of number of cores and their distribution is shown in Table 4. The Data-aware approach does not consider the heterogeneous number of cores in different machines and hence boots up 10 machines for 10 tasks needs to send shared file 10 times. As these machines are boot up in parallel, the boot up time is 2.5 minutes but the file transfer time is very high for these tasks. Hence, as shown in Figure 6, for tight-deadline cases the file transfer time and hence execution time is much higher of Data-aware algorithm as compared with proposed algorithms.

As data aware approach sends a separate instance of the shared file for each task, the file transfer time is so high that for small deadline cases ≤ 45 minutes (which is the transfer time), the Data-aware provisioner does not send any task to the cloud and executes them in private cloud itself. Due to this, the data-aware provisioner is unable to finish tasks within deadline when it is ≤ 45 minutes. However, the shared-file aware provisioner is able to finish tasks within deadlines for all cases. Elaborating further, Figure 6a shows that for deadline ≤ 45 minutes the total execution time is constant and equal to 60 minutes as it does not send tasks to public cloud. However, shared data aware algorithm is able to identify tasks where file sharing is possible to reduce file transfer times (as shown in Figure 6b) and hence complete tasks in the required deadlines. This shows that for deadline-critical applications data-aware provisioner is unable to utilize cloud resources due to its ignorance to the number of cores of cloud resources. Hence for deadline critical applications, shared-file aware provisioner is much better in terms of utilization of resources (compute and network bandwidth) and providing results in required deadlines.

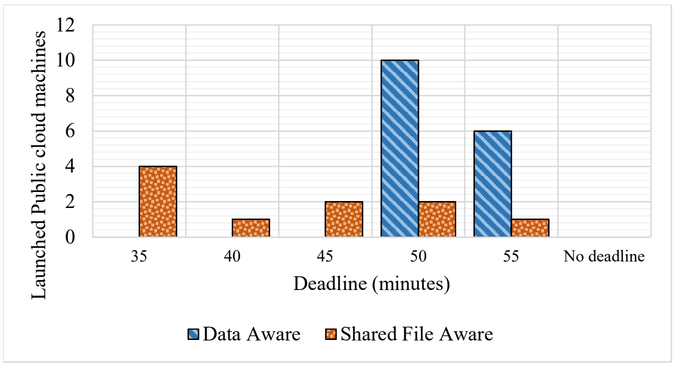
Experiments comparing cost of execution of the BLAST job using the two scheduling algorithms is shown in Figure 7. For the cases where deadline is less than 45 minutes, the Data-aware algorithms is not able to complete within the deadline



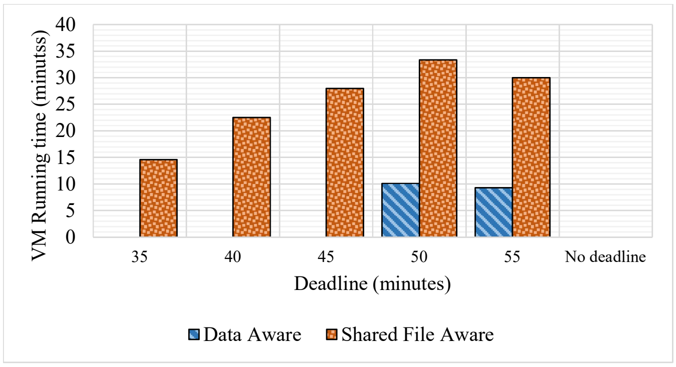
(a) Execution Time comparison



(b) File transfer time comparison



(c) Comparison of number of launched VMs in public cloud



(d) Comparison of VM Run time in public cloud

Figure 6: Comparison of Data Aware and Shared File aware provisioning algorithms

[1]Azure Pricing calculator at `https://azure.microsoft.com/en-au/pricing/calculator/`

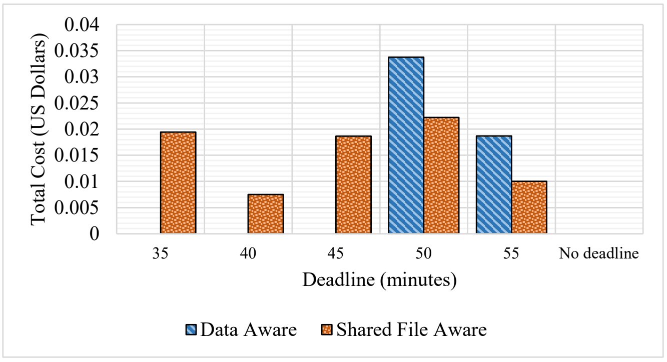


Figure 7: Comparison of total cost for job completion in public cloud

and does not use any resources from the public cloud. Even though it has zero cost of public cloud usage, it is of little use as deadlines were not met. In the case of shared-data aware algorithm, the system is not only able to complete the tasks in the required deadlines but also has lower monetary expenditure in cases where deadline is more than 45 minutes. This is because number of VMs instantiated is much lower (as data aware approach starts new VM for every task) even though VM run-time is higher. Hence, the shared-data aware algorithm further provides better service affordability and is much more promising in providing results withing deadlines, important for critical cases like healthcare applications [4].

7. Conclusions and Future Work

Efficient utilization of the public cloud resources as an extension to the private resources is a complex problem for data-intensive applications. This problem is complicated further due to the heterogeneity of cloud environments. One approach to efficiently utilize multi-core resources to prevent transfer of shared-files multiple times across the network is provided in this work. Prior work on Data-Aware algorithms exploit this file -sharing capability with limited perspective. This work provide novel Shared-File aware Task Scheduling and Dynamic resource provisioning to provide better quality of service in terms of response time, lower number of deadline violations and lower resource consumption both in terms of compute and network bandwidth. The shared-file aware scheduling algorithm tries to efficiently reduce the number of dependent task groups and have maximum extent of file sharing which reduces the number of transfers of shared files. File sharing can also be used in dynamic provisioning. The shared-file aware dynamic resource provisioning algorithm utilizes the fact that file sharing can be achieved to maximum extent by employing mutlicore systems. The larger tasks are sent to machines with highest number of cores to reduce the number of transfers of shared files and hence be able to complete tasks within deadlines as well as reduce resource utilization of both public and private clouds.

As part of future work, we propose to include in the dynamic provisioning algorithm support for integration of multiple cloud providers with different cost models. Among the different cloud providers or the geographically distributed cloud resources from the same providers, the algorithm can utilize the different data transfer times among different VMs and resource pools. Apart from the current Bag-of-Tasks model, other models like Workflow and Map-Reduce can also be explored to provide shared-file aware algorithms optimized for such models. Another direction is to integrate edge resources with cloud resources and modify the proposed algorithms so that they are applicable to Edge and Fog computing paradigm. Other QoS parameters and budget constraints can also be imposed in the scheduling and provisioning policies to further enhance the user experience and hence the improve the policies. Moreover, we also propose to work on managing the shared files in more secure manner especially for mobile cloud or edge services and also consider privacy and security aspects of scheduling and resource selection. We propose to add filtering approach based on privacy vulnerability and sensitivity of data on the Aneka platform [23, 24]. Mobile edge computing system has more demands and thus it has the need for enhancing Aneka to adopt such systems. Similarly, performance and reliability to get on and off-loading for data can be made available [2] for large scale applications.

Acknowledgements

This research work is supported by the Melbourne-Chindia Cloud Computing (MC3) Research Network and Australian Research Council. We thank Nipam Basumatary (Indian Institute of Technology, Madras, India) and Shashikant Ilager (CLOUDS Lab, University of Melbourne, Australia) for their valuable comments on improving the quality of presentation.

References

[1] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, S. Ullah Khan, The rise of "big data" on cloud computing: Review and open research issues (2015). doi:10.1016/j.is.2014.07.006.

[2] F. Zhang, J. Ge, C. Wong, C. Li, X. Chen, S. Zhang, B. Luo, H. Zhang, V. Chang, Online learning offloading framework for heterogeneous mobile edge computing system, Journal of Parallel and Distributed Computing 128 (2019) 167–183.

[3] H. Gelernter, J. Birnbaum, M. Mikelsons, J. Russell, F. Cochrane, D. Groff, J. Schofield, D. Bromley, An advanced computer-based nuclear physics data acquisition system, Nuclear Instruments and Methods 54 (1) (1967) 77–90.

[4] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems 29 (7) (2013) 1645–1660. doi:10.1016/j.future.2013.01.010.

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

[6] A. Nadjaran Toosi, R. O. Sinnott, R. Buyya, Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using Aneka, Future Generation Computer Systems 79 (2018) 765–775. doi:10.1016/j.future.2017.05.042.

[7] S. Abdi, L. PourKarimi, M. Ahmadi, F. Zargari, Cost minimization for deadline-constrained bag-of-tasks applications in federated hybrid clouds, Future Generation Computer Systems 71 (2017) 113–128. doi:10.1016/j.future.2017.01.036.

[8] 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.

[9] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds, Future Generation Computer Systems 48 (2015) 1–18. doi:10.1016/j.future.2015.01.004.

[10] R. Van 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.

[11] L. Zuo, L. Shu, S. Dong, Y. Chen, L. Yan, A multi-objective hybrid cloud resource scheduling method based on deadline and cost constraints, IEEE Access 5 (2017) 22067–22080. doi:10.1109/ACCESS.2016.2633288.

[12] C. Vecchiola, X. Chu, R. Buyya, Aneka: A software platform for.NET based cloud computing, in: Advances in Parallel Computing, Vol. 18, 2009, pp. 267–295. doi:10.3233/978-1-60750-073-5-267.

[13] S. C. Nayak, C. Tripathy, Deadline sensitive lease scheduling in cloud computing environment using AHP, Journal of King Saud University - Computer and Information Sciences 30 (2) (2018) 152–163. doi:10.1016/j.jksuci.2016.05.003.

[14] S. C. Nayak, S. Parida, C. Tripathy, P. K. Pattnaik, An enhanced deadline constraint based task scheduling mechanism for cloud environment, Journal of King Saud University - Computer and Information Sciencesdoi:10.1016/j.jksuci.2018.10.009.

[15] S. C. Nayak, C. Tripathy, Deadline based task scheduling using multi-criteria decision-making in cloud environment, Ain Shams Engineering Journal 9 (4) (2018) 3315–3324. doi:10.1016/j.asej.2017.10.007.

[16] C. T. Fan, Y. S. Chang, S. M. Yuan, VM instance selection for deadline constraint job on agent-based interconnected cloud, Future Generation Computer Systems 87 (2018) 470–487. doi:10.1016/j.future.2018.04.017.

[17] I. A. Moschakis, H. D. Karatza, Multi-criteria scheduling of Bag-of-Tasks applications on heterogeneous interlinked clouds with simulated annealing, Journal of Systems and Software 101 (2015) 1–14. doi:10.1016/j.jss.2014.11.014.

[18] F. Xiong, C. Yeliang, Z. Lipeng, H. Bin, D. Song, W. Dong, Deadline based scheduling for data-intensive applications in clouds, Journal of China Universities of Posts and Telecommunications 23 (6) (2016) 8–15. doi:10.1016/S1005-8885(16)60064-X.

[19] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, R. Buyya, The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds, in: Future Generation Computer Systems, Vol. 28, 2012, pp. 861–870. doi:10.1016/j.future.2011.07.005.

[20] M. Kalra, S. Singh, A review of metaheuristic scheduling techniques in cloud computing, Egyptian Informatics Journal 16 (3) (2015) 275–295. doi:10.1016/j.eij.2015.07.001.

[21] R. Sandhu, A. N. Toosi, R. Buyya, An API for Development of User-Defined Scheduling Algorithms in Aneka PaaS Cloud Software, in: B. B. Gupta, P. D. Agarwal (Eds.), Handbook of Research on Cloud Computing and Big Data Applications in IoT, IGI Global, 2019, pp. 170–187. doi:10.4018/978-1-5225-8407-0.ch009.

[22] G. P. Rédei, Blast (basic local alignment search tool), in: Encyclopedia of Genetics, Genomics, Proteomics and Informatics, 2008, pp. 221–221. doi:10.1007/978-1-4020-6754-9_1879.

[23] X. Xu, X. Zhao, A framework for privacy-aware computing on hybrid clouds with mixed-sensitivity data, in: 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, IEEE, 2015, pp. 1344–1349.

[24] S. Christey, J. Kenderdine, J. Mazella, B. Miles, Common weakness enumeration, Mitre Corporation.

Shreshth Tuli is an undergraduate student at the Department of Computer Science and Engineering at Indian Institute of Technology - Delhi, India. He is a national level Kishore Vaigyanic Protsahan Yojana (KVPY) scholarship holder for excellence in science and innovation. He is working as a visiting research fellow at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, the University of Melbourne, Australia. Most of his projects are focused on developing technologies for future requiring sophisticated hardware-software integration. His research interests include Internet of Things (IoT), Fog Computing, Network Design, Blockchain and Deep Learning.

Rajinder Sandhu obtained his M.E. with honours from Thapar University, Patiala in 2013 and B.Tech with Distinction from MMEC, Mullana in 2011. He completed his Ph. D. in cloud computing and Big Data to Guru Nanak Dev University, Amritsar under the guidance of Dr. Sandeep Sood. He has published his research work in esteemed Scientific Citation Index journals of Elsevier, John Wiley and Springer. He also filed two patents in Indian Patent Office. He served as Honorary Academic Visitor by Department of Computing and Information Systems, University of Melbourne, Australia. He is reviewer of many reputed SCI journals of Elsevier, Wiley and Springer. He has delivered multiple expert talks on cloud computing for various workshops and FDPs of reputed universities like JNU-Delhi, PEC-Chandigarh and IIT-Kharagpur. His current research areas are cloud computing, Big Data and Internet of Things (IoT).

Rajkumar Buyya is a Redmond Barry Distinguished Professor 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 served as a Future Fellow of the Australian Research Council during 2012-2016. He has authored over 700 publications and seven text books including Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=132, g-index=280, 92500+ citations). "A Scientometric Analysis of Cloud Computing Literature" by German scientists ranked Dr. Buyya as the World's Top-Cited (#1) Author and the World's Most-Productive (#1) Author in Cloud Computing. Dr. Buyya is recognized as a "Web of Science Highly Cited Researcher" for four consecutive years since 2016, a Fellow of

IEEE, and Scopus Researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier for his outstanding contributions to Cloud computing. He served as the founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Editor-in-Chief of Journal of Software: Practice and Experience, which was established over 45 years ago. For further information on Dr. Buyya, please visit his cyberhome: www.buyya.com