Universiteit Antwerpen

Customer Tool Support for Trading off Cost and QoS in IaaS Service Procurement

Proefschrift voorgelegd op 22th August 2014 tot het behalen van de graad van Master in de Wetenschappen, bij de faculteit Wetenschappen, aan de Universiteit Antwerpen.

Promotoren:
Prof. dr. Jan Broeckhove
dr. Kurt Vanmechelen

Hoang Trung Hieu

CoMP Research Group Computational Modelling and Programming

Contents

List of Figures

List of Tables

Acknowledgement

Foremost, I would like to express my sincere gratitude to my promoters Prof. dr. Jan Broeckhove, dr. Kurt Vanmechelen and my supervisor dr. Ruben Van den Bossche who have supported me throughout my thesis with their patience, motivation and immense knowledge. Without their encouragement and guidance, this thesis would not have been completed or written.

I am also grateful to Belgian Technical Cooperation for having given me the opportunity to pursuit my study here in Belgium.

Last but not least, I would like to thank my wife and my daughters. Their unconditional support and understanding even in my most difficult time are greatly appreciated.

Abstract

IaaS cloud computing infrastructure has become an attractive platform which offers on-demand services to run a wide range of applications, including "batch-type" workloads such as scientific calculations or simulations. Executing a workload tends to have minimum requirements on the resources it consumes (number of cores, speed of CPU or available memory etc). On the other hand, the owner will also require its workload execution to achieve a certain QoS level. A challenging issue here is to select the correct service option among many available options to minimize the execution cost while still meeting provided QoS requirements.

In this thesis, we describe a solution to find out the option which optimizes the execution cost of a workload with a given deadline constraint. In addition, we present an approach to allocate the computing power of existing resources to the workload's tasks in a cost-efficient way.

Finally, we realize the proposed solutions in the design and implementation of a Decision Support System, named Customer Tool. Customer Tool provides cloud users with sufficient information on the (expected) costs that a workloads execution will generate and how the cost level is expected to change with the provided level of QoS. The system is able to gather the necessary information by interrogating the user in a intuitive and attractive way, after which it reports its findings and possibly proposes suggestions to adaptations in the resource or QoS requirements.

CHAPTER 1

Introduction

1.1 Cloud Computing

Cloud computing is emerging as an interesting topic in both research and business fields. Cloud computing refers to applications and services that run on a distributed network using virtualized resources and can be accessed using networking standards. It is realized by the notion that resources are virtual and limitless and that details of the physical systems are abstracted from the user [32].

In fact, cloud computing has been around for a long time which is evolved from grid computing since 1980s. However, until recent years, with the maturity of virtualization technology, cloud computing has been widely accepted and deployed from a small enterprise to multinational corporations. Generally, cloud computing provides computation, software, data access, and storage services that do not require end-user knowledge of the physical location and configuration of the system that delivers the services. The service providers deliver applications via the internet, which are in turn accessed from a Web browser, while the business software and data are stored on servers at a remote location.

With the cloud market expected to grow quickly in the coming years, it is becoming a fierce competitive market with diverse service providers such as Amazon Web Service, Windows Azure, GoGrid, Rackspace etc.

In this chapter, we will study the definition of cloud computing and cloud services, the layers and types of the cloud computing. We also discuss some technologies that enable cloud computing and finally, we consider the benefit and disadvantages when using a cloud environment instead of a private IT infrastructure.

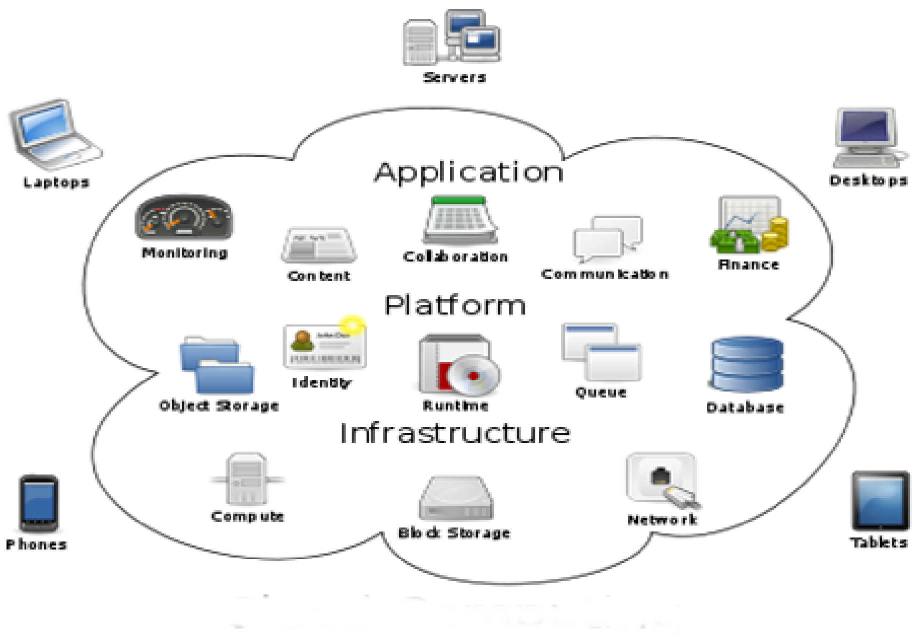


Figure 1.1: Logical Diagram of Cloud Computing [23]

1.1.1 Cloud Computing Definition

Cloud computing represents a significant advance on how services or applications are deployed. The use of *"cloud"* makes reference to the following concepts [2]

- **Abstraction:** Detail of system implementation is hidden from the users. The users are not aware of physical deployment, data location and components communication. They are offered a ubiquitous access to the resources only.
- **Virtualization:** Physical systems are virtualized to provision resources as needed to the users. Costs are charged on actual usage and depending on demand from the users, resources are scalable with agility.

Even though the term *cloud computing* has been used for a while, its definition is still an issue that has not received wide consencus yet. However, among various definitions of cloud computing today, the definition from U.S. National Institute of Standards and Technology (NIST) has been recognized and cited in many studies. According to NIST, we have

Definition:*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.* [28]

1.1.2 Essential Characteristics

The cloud model has described with 5 essential characteristics [28]

On-demand self-service

Cloud service providers can provision resources to its users as needed automatically without requiring human interaction. The users have access to the services and permission to change service configuration following their needs via a web-based service portal (management console)

Broadband network access

With the cloud technologies, users are able to access their resources from anywhere over the network, using a variety of devices such as PCs, laptops, smartphones, and PDAs etc.

Resource polling

Cloud service providers take full control of their physical resources, create a pool of virtual resources and allocate them to serve multiple users. The user generally has no control or little knowledge over the exact location of the provided resources but is able to specify location at a higher level of abstraction.

Elasticity

Resources can be provisioned based on customers' requests. Self-service and resource pooling make rapid elasticity possible. From the user perspective, the capabilities available at cloud providers appear to be unlimited and can be easily requested or released. This will make sure an application will have exactly the capacity it needs at any point of time.

Measured service

Resources usage is metered and reported transparently back to users. The users only pay for what they have used.

1.1.3 Deployment Models

Deployment models identify the purpose of the cloud and the nature of how the cloud is deployed and located. There are four deployment models defined by NIST as follows [28]:

Public cloud

In public cloud, resources are dynamically provisioned to users over Internet using web applications or web services. A public cloud is owned by third-party selling cloud services. Applications and data from different users are mixed and shared the same infrastructure such as servers, storage systems and network from the cloud provider. This is the most popular type of cloud system and is considered as a main stream of cloud systems [23]. Example: Amazon Web Service, Windows Azure, GoGrid.

Private cloud

A private cloud infrastructure is operated for use of an organization only. A private cloud may be managed by the organization itself or a third-party organization. However, the organization has full control over the data, security and quality of service. The private cloud may be either on or off-premise, meaning its deployment is in organization's IT private infrastructure or in a cloud provider infrastructure.

Hybrid cloud

A hybrid cloud is the combination of multiple public and private clouds where those clouds retain their unique identity but bound together as a unit. Hybrid cloud makes determining the distribution and scheduling the execution of applications across public and private cloud challenging issue.

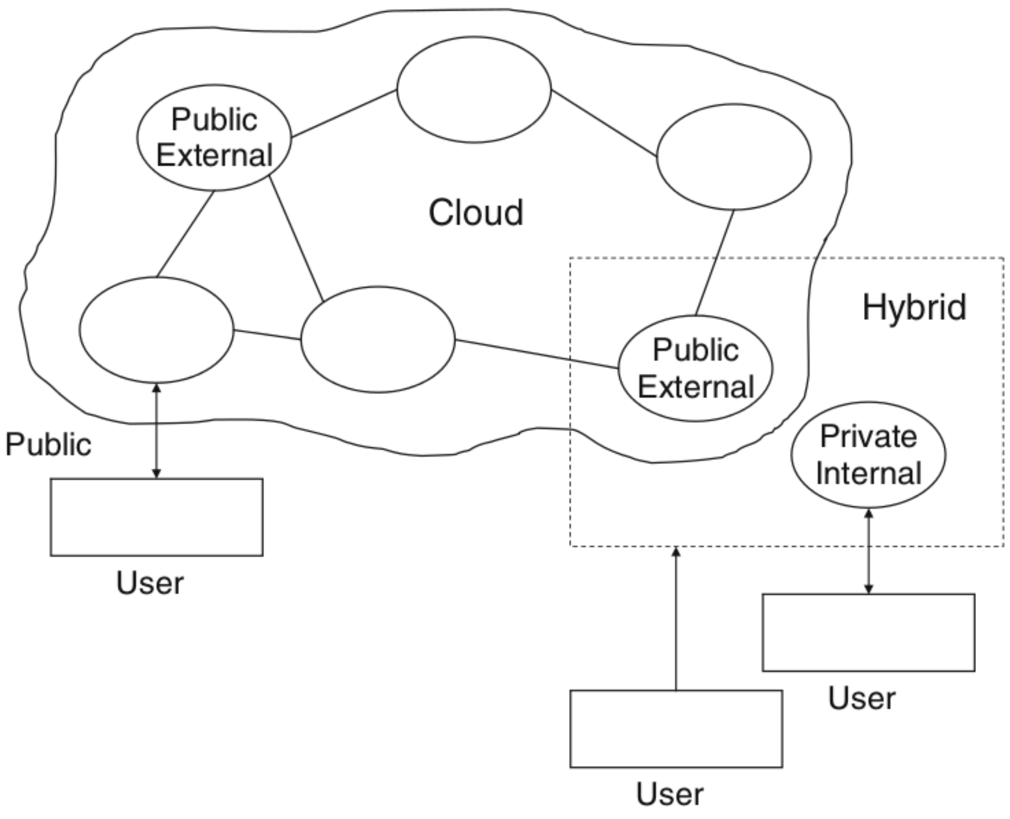


Figure 1.2: Deployment Models [2]

1.1.4 Service Models

As cloud computing has developed, different vendors offer different services associated with their proprietary technologies and business objectives. However, various cloud services can be categorized into three service models which are widely accepted. These 3 models are together referred as SPI (**S**oftware as a Service, **P**latform as a Service, and **I**nfrastructure as a Service) model of cloud computing [32]. Other services such as Storage as a Service (StaaS), Identity as a Service (IdaaS) can be covered by one of these SPI models.

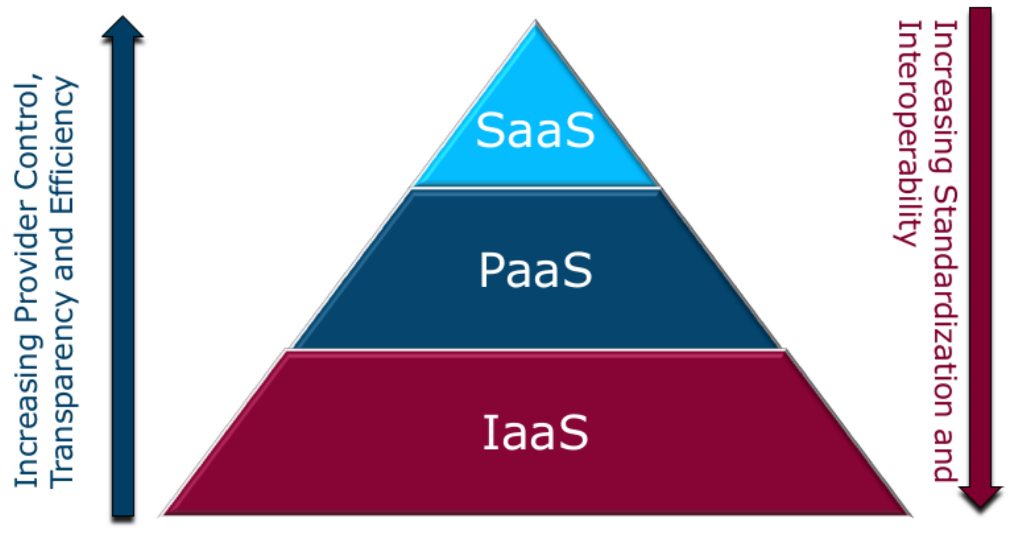

Increasing Provider Control, Transparency and Efficiency
SaaS
PaaS
IaaS
Increasing Standardization and Interoperability

Figure 1.3: Cloud Computing Service Models [21]

Software as a Service (SaaS)

"Software as a service (SaaS) delivers special-purpose software that is remotely accessible by consumer through the Internet with usage-based pricing model" [8]

In SaaS model, the application is accessible to customer via thin-client interface (mostly web-browser). Customers are only responsible for managing their data and user interaction while operation environment with application, management and user interface are the responsibility of service providers. Examples of SaaS are Goolge Docs, Salesforce CRM, Office 365.

Platform as a Service (PaaS)

"Platform as a Service (PaaS) offers a high-level integrated environment to build, test and deploy custom applications. Generally, developers will need to accept some restrictions on the type of software they can write in exchange for built-in application scalability." [8]

PaaS provides virtual machines, operating systems, application services, development frameworks and control structures. Users can deploy any of their applications on the cloud infrastructure but languages and tools used must be supported by PaaS service providers. With PaaS, the users only need to focus on their applications and leave the infrastructure to cloud providers, therefore it helps to speed up the development and deployment process; and automate application scaling. Examples: Force.com, Google App Engine, Windows Azure (Platform)

Infrastructure as a Service (IaaS)

"Infrastructure as a Service (IaaS) provisions hardware, software and equipments (mostly at at the unified resource layer, but can also include part of the fabric layer) to deliver software application environments with a resource usage-based pricing

model. The resulting infrastructure can scale up and down dynamically based on application resource needs." [8]

In this model, an IaaS service provider manages the infrastructure including virtual machines, virtual storages, virtual infrastructure and other hardware assets while users are responsible for the whole deployment process such as operating system, application framework and interaction with the systems. Example: Amazon EC2, GoGrid VM.

1.1.5 Cloud Enablers

The main enablers of cloud computing include virtualization technology, web service & service-oriented architechture, large capital investment by major players & economics of scale [21]

Virtualization technology and Standards

Virtualization allows physical resources to be separated into one or more virtual devices, pooled and shared among different applications in order to gain better server utilization. In cloud computing, servers are virtualized and allocated to users on demand which results in fewer physical servers. Typically, the unit of virtualized devices is a virtual machine with specific hardware configuration, operating systems and application framework.[21]

Web service and Standards

Web service is not a new concept, however it has contributed a lot on the development of cloud computing. Cloud services normally support users interaction via web services, which strictly follow industry standards such as WSDL, SOAP, RESTful etc.

Large capital investment and Economics of scale

Cloud computing development also benefits from the economics of scale enjoyed by providers, who invest a lot of money to build large scale data centers. These data centers typically operate in a more efficient and professional way as compared to small or private ones. Cost savings gained by an increased level of production and investment can be passed proportionally to the users.

1.1.6 Advantages of Cloud Computing

Besides 5 essential characteristics of cloud computing, it is worth considering other additional cloud advantages in this section.

Cost efficiency

Cloud computing is cost effective to be used and implemented. From a service provider's perspective, investment cost benefits from economics of scale and cloud resources operate at higher efficiency with greater resource utilization, thus significant cost reduction can be achieved. From the users' perspective, cost reduction from service providers will be proportionally transfered to the users' cost as well. Further more, there is no need of the users to spend on investing their own private IT infrastructure such as servers, software and license fees from scratch.

Ease of utilization
Depending on the type of services requested, users can concentrate on the implementation of their core business and leave the management such as hardware configuration, software licenses to service providers

Quality of Service
Quality of Service is guaranteed by Service Level Agreement (SLA) signed with service providers.

Reliability
The scale of cloud resources together with geographically distributed data centers allow service providers to provide sufficient redundancy of resources, implement load balancing and plan for failover, disaster recovery scenarios which makes their services reliable and consistently available.

Simplified maintenance and update
IT infrastructure is regularly updated with the latest software and technology by service providers, freeing up their users' time and resources for other tasks. On the other hand, if needed, the users can easily apply patches or upgrades to their applications as the system management is centralized.

Business agility
Cloud computing quickly solves your problems using IT resources; you have to wait no longer to get appropriate solutions [23].

It is worth noting that this is not the exhaustive list of cloud computing advantages. A user with specific type of applications and requirements may find its own advantages as well. It obviously makes the idea of cloud computing an interesting topic both in scientific and business studies.

1.1.7 Disadvantages of Cloud Computing

As stated in previous section, cloud computing offers advantages as compared to previous computing paradigms and it is gradually adopted to many organizations. However, before moving to the cloud environment, a number of concerns should be taken into consideration. Such concerns will be briefly explained in this section

Security and Privacy
Security and Privacy are the biggest concern relating to cloud computing. When using a public cloud service, users need to hand over all their sensitive information to a third-party and many end users might feel uncomfortable to do so. Although users' data is committed to be kept confidential by service providers, all possible alternatives should be explored before making a decision.

Regarding the privacy, it is important that cloud providers must ensure that users' data is not accessed by unauthorized users. It should be kept in mind that even the best service provider can expose some sort of vulnerabilities which potentially put users' data at risk. Companies and users have to trust their cloud service vendors that they will protect their data from unauthorized users.

Provider Lock-In

- Since there is no standard for cloud services, it is not easy to migrate to another cloud provider. Users' applications and data are locked with the current provider because of technical constraints.
- Measurement of resource capacity, usage and user activities is completely under the control of service providers. Therefore, users need to trust providers.
- Users' deployment in cloud may be influenced because of changes in cloud providers business strategy, financial status etc...

Dependent on the network

In order to use and manage cloud services, users will need a broadband Internet connection at all times. Cloud applications could stop working or being out of management if the connection to the cloud services is broken or unreliable.

Control

Service providers take full control of the cloud physical platform. Users have very limited information on the hardware characteristics, service limits and performance.

Support for Legacy System

Cloud providers regularly maintain and upgrade their platform to latest technology, somehow, this practice has an downside which is the lack of support from service providers to upgrade users' legacy IT to cloud environment.

Insufficient SLA

Cloud platforms are universally designed to be able to provide services to users with different requirements. The commitment to QoS such as uptime, Mean Time Between Failure (MTBF), Mean Time To Repair (MTTR) is regulated in Service Level Agreement (SLA). However, provided SLAs are not always sufficient. Some specific users may have requirement that beyond the capacity of cloud providers or the scope of SLA [21].

Regulatory Issues

Since cloud services are delivered in a wide range of geographical locations, there are also concerns related to regulatory, compliance and legal issues when moving to the cloud environment. Different countries or regions may have different policy and regulation on cloud computing services. It is not a rare case that one service used popularly in a country may be considered illegal in another country or region. Illegal or non-compliance services provision can definitely result in substantial civil penalties and harsh criminal punishments that might override the benefits provided by cloud computing.

1.2 Amazon Web Service Overview

In this thesis, as our proposed solutions and the Decision Support Systems are developed based on the services offered by Amazon Web Services (abbreviated AWS).

Additionally, in the next chapters, we use a lot of terms, notions and definitions from Amazon Web Services (AWS). Therefore, it would be useful to introduce AWS before going into the detail solution.

AWS is having a enormous impact in cloud computing services. Indeed, AWS represents the biggest cloud services provider in the marketplace. AWS started providing services since 2006 and today it has serves hundreds of thousands of users worldwide. Inheriting all advantages of cloud computing, AWS allows users to acquire computing power, storage and other services almost instantly without up-front payment or longterm bond. The users are also given the flexibility to choose a development platform and programming model that are best suitable for them.

AWS data centers are deployed in a wide range of locations including US East (Northern Virginia), where the majority of AWS servers are based, US West (northern California), US West (Oregon), Brazil (Sao Paulo), Europe (Ireland), South Asia (Singapore), East Asia (Tokyo), China (Beijing) and Australia (Sydney). Each region, in turn, consists of multiple availability zones, where locate physical data centers providing services to the users. Availability Zones in the same region are engineered to be isolated from failures in other Availability Zones. Such deployment model helps to improve the performance and reliability of the system. The users are also able to benefit from geographically distributed data centers. For example, the users prefer to use services provided by data centers closer to them which thoughtfully can reduce the latency and response time from cloud services.

According to AWS documentation [13] *"AWS is a comprehensive cloud services platform that offers compute power, storage, content delivery, and other functionality that organizations can use to deploy applications and services cost-effectively with flexibility, scalability, and reliability"*.

In this section, a brief introduction to AWS will be given. Although AWS consists of many products, we are going to concentrate on some of its main products and corresponding pricing policy which will be used in the rest of the thesis.

1.2.1 Computing Products

Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provide sizable compute capacity in the cloud and it is the central product in AWS portfolio. Amazon EC2 enables customers to create, use and manage a set of private virtual servers, called instances, running selectable operating systems (i.e different Linux distribution, Windows) over Xen hypervisor. A virtual server in AWS is refered to as an instance.

Amazon EC2 instances are split into several categories to fit different use cases including general purpose, compute optimized, Graphics Processing Unit (GPU), memory optimized and storage optimized instances. In each category, there are a number of instance configurations comprising varying combinations of CPU, memory, storage, and networking capacity and give the flexibility to meet different requirements of the users.

Amazon EC2 is built on commodity hardware, over time there may be several different types of physical hardware underlying EC2 instances. In order to make it easy to compare CPU capacity between different instance types, AWS has defined an Amazon EC2 Compute Unit. The amount of CPU that is allocated to a particular instance is expressed in terms of these EC2 Compute Units. The EC2 Compute Unit (ECU) provides the relative measure of the integer processing power of an Amazon EC2 instance. AWS does not state clearly on what the capacity of EC2 is even though once it was used to mention that one EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

Amazon EC2 is supported by a number of features to build up scalable and enterprise class applications such as Auto Scaling, Elastic IP Address, Elastic Load Balancing, Amazon CloudWatch etc.

In terms of availability, Amazon specifies a Service Level Agreement of 99.95% for their EC2 service.

Since the end of 2013, AWS has launched the next generation of Amazon EC2 instances which offers better performance and pricing policy as compared to the first generation. Table 1.1 shows available instance types and their configuration in AWS (updated in August 2014).

Pricing

Regarding Amazon EC2, pricing is per instance-hour consumed for each instance type. Partial instance-hours consumed are charged as full hours. For example, 5 minute and 60 minute usage are both billed as 1 hour and if an instance is instantiated and terminated twice in an hour, customers will be charged for two instance hours. In all pricing models, users pay for what they have used and there is no minimum fee [1].

On-Demand Instance

With on demand instance, users pay a fixed rate for an hour of usage without upfront payment or long-term commitment. The users can increase or decrease compute capacity depending on the demands of their application and only pay the hourly rate for the instances used

On-Demand Instances are recommended for [13]:

- Users that want the low cost and flexibility of Amazon EC2 without any upfront payment or long-term commitment
- Applications with short term, spiky, or unpredictable workloads that cannot be interrupted
- Applications being developed or tested on Amazon EC2 for the first time

Reserved Instance

Reserved Instance allows users to make a low, one-time payment in order to reserve instances that they plan to use in the future. As a trade-off, the users benefit from significant discount on instance-hour rate for such reserved instances. The users can

[1]Detail on all instance types' rates can be found in [13]

Table 1.1: Instance Type Configuration

	vCPU	**ECU**	**Memory (GiB)**	**Instance Storage (GB)**
General Purpose				
t2.micro	1	Variable	1	EBS Only
t2.small	1	Variable	2	EBS Only
t2.medium	2	Variable	4	EBS Only
m3.medium	1	3	3.75	1 x 4 SSD
m3.large	2	6.5	7.5	1 x 32 SSD
m3.xlarge	4	13	15	2 x 40 SSD
m3.2xlarge	8	26	30	2 x 80 SSD
Compute Optimized				
c3.large	2	7	3.75	2 x 16 SSD
c3.xlarge	4	14	7.5	2 x 40 SSD
c3.2xlarge	8	28	15	2 x 80 SSD
c3.4xlarge	16	55	30	2 x 160 SSD
c3.8xlarge	32	108	60	2 x 320 SSD
GPU Instances				
g2.2xlarge	8	26	15	60 SSD
Memory Optimized				
r3.large	2	6.5	15	1 x 32 SSD
r3.xlarge	4	13	30.5	1 x 80 SSD
r3.2xlarge	8	26	61	1 x 160 SSD
r3.4xlarge	16	52	122	1 x 320 SSD
r3.8xlarge	32	104	244	2 x 320 SSD
Storage Optimized				
i2.xlarge	4	14	30.5	1 x 800 SSD
i2.2xlarge	8	27	61	2 x 800 SSD
i2.4xlarge	16	53	122	4 x 800 SSD
i2.8xlarge	32	104	244	8 x 800 SSD
hs1.8xlarge	16	35	117	24 x 2048

start, monitor and terminate reserved instance in the same way as with on-demand instances. Technically, while running, instances acquired by on-demand or reserved price are exactly the same.

Reserved Instances are recommended for[13]:

- Applications with steady state or predictable usage
- Applications that require reserved capacity, including disaster recovery
- Users able to make upfront payments to reduce their total computing costs even further

There are three type of reserved instance: Light Utilization RI, Medium Utilization RI, and Heavy Utilization RI. The difference between these reservation types is how much you pay upfront for the reservation, and how much you pay by the hour. Generally, Light reservations has lower upfront payments but higher instance-hour price, Heavy reservations has lower instance-hour price but higher upfront payment, and Medium reservations rest somewhere in between.

- Light Utilization RI Light Utilization RIs are ideal for workloads that only run a couple of hours a day or a few days per week.
- Medium Utilization RI Medium Utilization RIs are suitable for workloads that are active most of the time, but have some variability in usage.
- Heavy Utilization RIs Heavy Utilization Ris are most efficiently used when customers commit to always running these instances in exchange for our lower instance-hour fee.

Spot Instance

The Spot Instance rate fluctuates in real time depending on Spot Instance supply and demand. Amazon EC2 sets a Spot Price for each instance type in each Availability Zone, which is the price all users will have to pay to run a Spot Instance for that given period. In order to use Spot Instance, users place a bid price on unused EC2 capacity with a specific instance type. If the users' bid price exceed current Spot Price, requested instances are launched and keep running until Spot Price becomes higher than the bid price or the users choose to terminate those instances. EC2 revokes the resources immediately without any notice when the users bid is less than or equal to the current spot price. The users are charged by the current Spot Price. Apparently, the users will never pay more than how they bid and it introduces possibility to keep the service cost under control. Spot Price offers hourly rates usually lower than the On-Demand rate (recently 86% lower, on average) while giving technically the same performance as On Demand or Reserved Instance.

Spot Instances are recommended for interruption-tolerant tasks and, potentially, accelerate those applications when there are many Spot Instances available [13].

The figure 1.4 gives example of 3 months spot price history in us-east region.

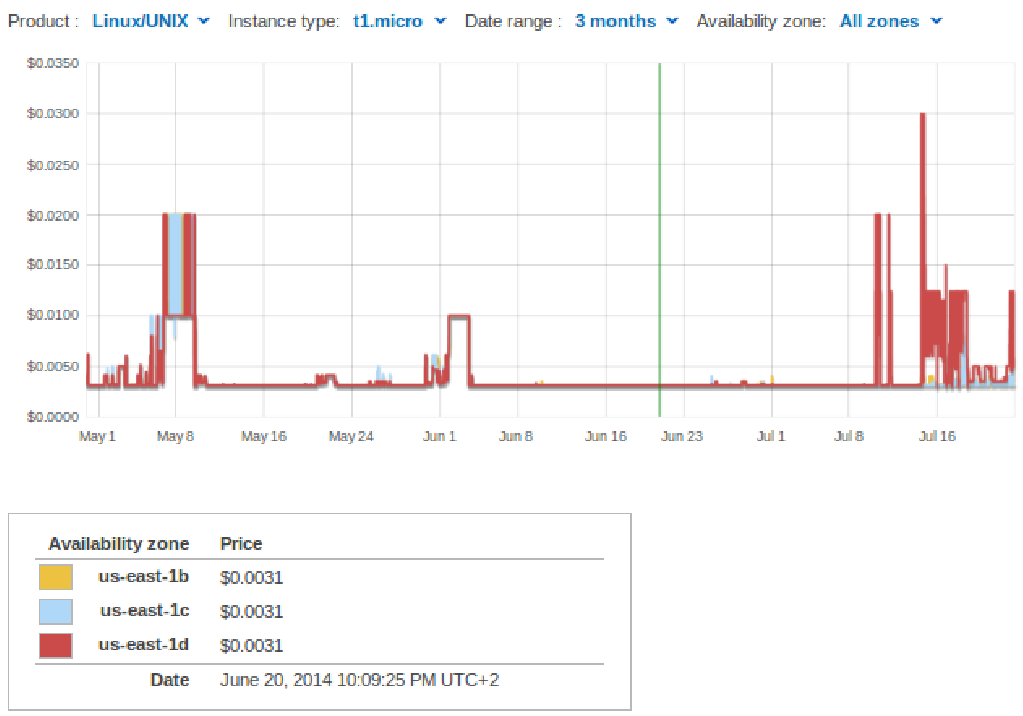
1.2.2 Storage Products

Amazon Simple Storage System (S3)

Amazon Simple Storage System (Amazon S3) is an online storage services. It provides users with a web service that can be used to upload, storage and retrieve any type of data for different objectives such as content storage and distribution; storage for data analysis; backup, archiving, disaster recovery and so forth.

Amazon S3 is designed with a set of following features

- Data uploaded to Amazon S3 are stored in a container called a buckets. Before starting working with Amazon S3 at least one bucket has to be created. The bucket is shared by all users of the system so the name must be globally unique.


Product : Linux/UNIX Instance type: t1.micro Date range : 3 months Availability zone: All zones

	Availability zone	Price
🟨	us-east-1b	$0.0031
🟦	us-east-1c	$0.0031
🟥	us-east-1d	$0.0031
	Date	June 20, 2014 10:09:25 PM UTC+2

Figure 1.4: Spot Instance Pricing History in 3 months

- S3 supports write, read, and delete files with 1 byte to 5 terabytes of data each. Files are saved into and read from buckets with keys . No limits on the number of files a bucket contains.
- The users can choose to store their bucket in a specific location to optimize latency, cost or address regulatory requirement.
- Each object in the bucket has a URL: https://<bucketname>.<S3EndPoint><objectKey>
- Buckets cannot be nested, it is unable to create a bucket within a bucket.

Main advantages of Amazon S3 are the scalability and high availability and theoretically unlimited storage. According to AWS, Amazon S3s standard storage is:

- Designed for 99.999999999% durability and 99.99% availability of objects over a given year.
- Designed to sustain the concurrent loss of data in two facilities.

Reduced Redundancy Storage (RRS)

RRS allows users to reduce their costs by storing non-critical, reproducible data at lower levels of redundancy than Amazon S3s standard storage. Recommended types of data include thumbnails, transcoded media, or other processed data that can be easily reproduced. The RRS does not replicate files as many times as standard Amazon S3 storage but still more cost effective than Amazon S3.

Reduced Redundancy Storage is [13]:

- Backed with the Amazon S3 Service Level Agreement.
- Designed to provide 99.99% durability and 99.99% availability of objects over a given year.
- Designed to sustain the loss of data in a single facility.

Amazon Glacier

As compared to Amazon S3, Amazon Glacier is an extremely low-cost online storage service. This service is optimized for infrequently access of data, or "cold data. Amazon Glacier is suitable for data archiving and backup where customers want to store their data effectively in terms of cost for months or years.

Amazon Glacier is designed to provide average annual durability of 99.999999999% for an archive. The service redundantly stores data in multiple facilities and on multiple devices within each facility [13].

Amazon Elastic Block Store

Amazon EBS provides a persistent storage volumes attached to 1 Amazon EC2 instance at a time. Amazon EBS is used as physical drives to an instance and customers can create a file system on top of that drives. Amazon EBS is automatically replicated in different Availability Zones to give redundancy to customer data. It is noted that Amazon EBS exists independently from life-time of an instance.

There are three EBS volume types:

- General Purpose (SSD): suitable to most of applications such as virtual desktop, development environment...
- Provisioned IOPS (SSD): applications that require high IOPS performance, i.e large database workload
- Magnetic: data that infrequently accessed

Pricing

Costs for Amazon S3, RRS and Glacier are broken down into three components.

- **Storage Pricing:**This is the cost that customers have to pay to store their data in cloud. Storage price is calculated as GB per month and it will decrease when the usage increases.
- **Request Pricing:**Requests to data are charged at units of 1000 or 10000 request. A request is counted when customers data in bucket (files) is accessed via URL.
- **Data Transfer Pricing:**This is the cost to transfer data out of customers bucket. Like Storage Pricing it is calculated as GB per month and goes down if the usage increases. The amount of data transferred to customers bucket is free of charge.

The rates of these price components will be decreased as the users' usage increase. It can be seen from the table 1.3 that the highest rate is S3 storage and the lowest one is Glacier. It reflects the decrease in committed QoS from S3 to RRS and Glacier. It is also noted that Glacier rate is fixed regardless of users' usage because the purpose of Glacier is for "cold data" which is assumed to be in big size.

Table 1.2: Storage Pricing

	S3	**RRS**	**Glacier Storage**
First 1 TB / month	$0.0300 per GB	$0.0240 per GB	$0.0100 per GB
Next 49 TB / month	$0.0295 per GB	$0.0236 per GB	$0.0100 per GB
Next 450 TB / month	$0.0290 per GB	$0.0232 per GB	$0.0100 per GB
Next 500 TB / month	$0.0285 per GB	$0.0228 per GB	$0.0100 per GB
Next 4000 TB / month	$0.0280 per GB	$0.0224 per GB	$0.0100 per GB
Over 5000 TB / month	$0.0275 per GB	$0.0220 per GB	$0.0100 per GB

Regarding Amazon EBS, the rate depends on EBS volume type and the amount of I/O requests

Table 1.3: Storage Pricing

Description	**Pricing**
EBS General Purpose (SSD)	$0.10 per GB-month of provisioned storage
EBS Provisioned IOPS (SSD)	$0.125 per GB-month of provisioned storage $0.065 per provisioned IOPS-month
EBS Magnetic	$0.05 per GB-month of provisioned storage $0.05 per 1 million I/O requests
EBS Snapshots to Amazon S3	$0.095 per GB-month of data stored

CHAPTER 2

Problem Introduction

2.1 Thesis Motivation

Cloud computing introduces an attractive way to alleviate private IT infrastructure construction for companies and organizations whether they are small independent software vendors (ISVs), startups, or large corporations that are looking to cut costs. Nowadays IaaS infrastructure can be used to run a wide range of applications, including both "online" applications such as web-, database- or application servers and "batch-type" workloads such as payroll processing, scientific calculations or simulations.

However, when users want to execute their application workload in the cloud environment, many concerns should be taken into account. The application workloads normally have minimum requirements on the resources they consume (e.g. number or speed of CPU, available memory, disk space, network or disk I/O speed etc.). It is obvious that the more resources can be acquired, the better the performance of workload execution is. In addition, the users always require their applications must achieve a certain level of quality of service depending on users' purposes and business cores. The cloud computing with pay-as-you-go pricing model gives an alternative to users' private IT investment where users only pay for what they use. With such pricing model, another question is how to minimize the cost spending on cloud resources to deploy and complete a workload in a cloud environment with provided quality of service level. This is not a trivial task as there are multiple cloud providers offering services in a various types of configuration, with non-standardized notions and different pricing schemes. Users need assistance in determining and selecting the right option which can meet their requirements from numerous options available.

The goal of this thesis is to build a "Decision Support System" that provides the user with sufficient information on the (expected) costs that a workload's execution will generate and how the cost level is expected to change with the provided level of QoS. The system should thereby gather the necessary information by interrogating the user in a intuitive and attractive way, after which it reports its findings and possibly proposes suggestions to adaptations in the resource or QoS requirements.

The contributions of this thesis are:

- A description of an algorithm to find out the option which optimizes the execution cost of a workload while still satisfying deadline requirement
- An approach that allocates the computing power of existing reserved instances to multiple tasks in a cost-efficient way.
- The design and implementation of a Decision Support System that assists cloud users in selecting the optimal options from the numerous available options. The Decision Support System is to realize aforementioned algorithms in a practical application. The system supports a web-based interface that provide the users an intuitive and attractive way to interact with the Decision Support System.
- An analysis of the workload execution cost with different levels of deadline constraint.

In the next section, before going into the solution and implementation of the DSS, we will give a brief explanation to some terms and assumptions which will be regularly used in the rest of this thesis.

2.2 Quality of Service

Quality of Service (QoS) refers to the level of service which is satisfactory to some users. The term QoS originated in the field of telecommunications and computer networks but it is also expanded to use in other fields as well.

QoS requirements are driven by bussiness demands and normally specified in Service Level Agreements (SLA). A SLA is an important document describing what the definition of 'satisfactory' is. A SLA can either be an informal contract between parties or a legally binding contract.

Regarding software systems, several Quality models exist. Among all, one of the most relevant was established in ISO9126 which classified the software quality in a set of characteristics as flollow: [12]

- **Functionality:** Suitability, Accuracy, Interoperability, Compliance, Security
- **Reliability:** Maturity, Recoverability, Fault Tolerance
- **Usability:** Learnability, Understandability, Operability
- **Efficiency:** Time Behaviour, Resource Behaviour

- **Maintainability:** Stability, Analyzability, Changeability, Testability
- **Portability:** Installability, Replaceability, Adaptability, Conformance

We have to mention here that when considering a workload execution on a software service system, even though all that QoS parameters should be taken into account to some extent, depending on the types of concerned workload, one parameter can have higher priority to the others. For example, with Web or database applications, QoS is often measured by response time. It means the time between a request/call is sent (or received) and when its response is received (or sent) with respect to user load condition. On the other hand, with service that performs heavy computation work and return result to user, completion time is of importance.

In this thesis, due to time and space limitation, the QoS requirements are restricted to deadline and budget constraints only.

2.3 Application Workload

Cloud computing infrastructure can be used to run a wide range of applications, including both "online" applications such as web-, database- or application servers and "batch-type" workloads such as payroll processing, scientific calculations or simulations.

Quality of Service measurement is highly dependant on the characteristics of a workload. In fact, it is impossible to handle all types of workload applications with various requirements on the QoS. Therefore, in order to keep the proposed solution simple and applicable, only batch-type workload is taken into consideration. Many scientific analyses are expressed as batch-type workload and they are commonly used to solve problems in many disciplines such as video coding/encoding, DNA sequence analysis and particle physics simulations... With batch-type workload, the QoS requirements are mainly defined by a specific deadline in which the workload must be completed and/or a maximum budget that a user can spend on the workload execution. The workload execution time is measured as the time duration from the first job in workload submitted until the last job completely executed.

We summarize some main characteristics of a batch-type workload as follows [18, 24]

- Workload consists of a number of independant tasks
- Each task, in turn, consists of multiple jobs. Jobs in a task shares some common features such as resource requirements, class of task (computing-intensive, data-intensive or network-intensive)
- The machine has to meet a certain level of resource capacity in order to be able to host a task in the workload. Resource capacity might be CPU speed, number of cores, size of RAM, storage size etc.

- Jobs in a task are fed to virtual machines by using a large queue which technically is able to hold all unprocessed jobs in that task. In addition, jobs are fairly distributed to running machines and a machine can only process at most one job at a time.

For the rest of the thesis, the term batch-type workload and scientifc workload are used interchangeably and refered to the same type of workload described in this section.

2.4 Assumptions

2.4.1 Assumptions

Before going into the solution, we need to make the following assumptions:

- As jobs in a task share common features, in order to decrease the complexity of the solution, it is assumed that each task can be executed only by 1 type of virtual machine.
- Each job may contain part of the work that must be done serially by a single processor. This part of the job does not run any faster on a parallel collection of processors. Only the part that can be parallelized runs faster. The proportion of the latter part to the total work of the job is called parallelization degree.
- A job is indivisible process and the execution of a job is uninterruptable. It means a job can only start being executed and finish on a single machine.
- The monetary cost of workload execution is minimum if the cost of all tasks execution in the workload are minimum either.
- Making accurate prediction on jobs'runtimes is beyond scope of this thesis. However, it is believed that during daily business operation in which the workloads tend to be executed repeatedly [6]. In the worst case, by experiments, the runtimes of jobs on a specific virtual machine can be estimated. It is noted that the jobs' runtimes vary depending on the type of a virtual machine it is running on. For example, a job may take 120 hours to complete on 1Ghz CPU/1 core/512MB RAM but the same job only needs 80 hours if running on 1Ghz/2 core/1GB RAM. The same remark is applicable to jobs' parallelization degrees and data transfered in/out cloud environment.
- Even though, it could take some time since an instance acquisition request until it is ready to use, we can assume that such startup time can be well observed and the instances from cloud providers can be acquired as long as they are needed with little or no delay.

2.4.2 Amdahl's Law

In order to ensure whether the execution of a workload satisfies deadline constraint, we should have the ability to estimate the runtime of each job in workload on any instance type. As mentioned in previous section, the job runtimes could be predicted during the operation of the organizations. However, in fact it is not feasible to compute job runtimes in all available virtual machine configuration offered by cloud providers but only one or few configurations that are commonly used to host the workload. In AWS, a virtual machine configuration is refered as an instance type. These two terms will be used interchangeably in the rest of this thesis.

Since the job runtimes are only known on a few instance types, it is required that there must be an algorithm to convert the job runtimes from a instance type to another. The conversion algorithm should take into account all the properties of the involved instance types to achieve accurate conversion. Therefore, we believe that the Amdahl's law is a good solution to this problem.

Amdahl's law is a model for the relationship between the expected speedup of parallelized implementations of an algorithm relative to the serial algorithm, under the assumption that the problem size remains the same when parallelized. According to Amdahl, *"As we gain more parallel power, we solve equally sized problems faster"* [20].

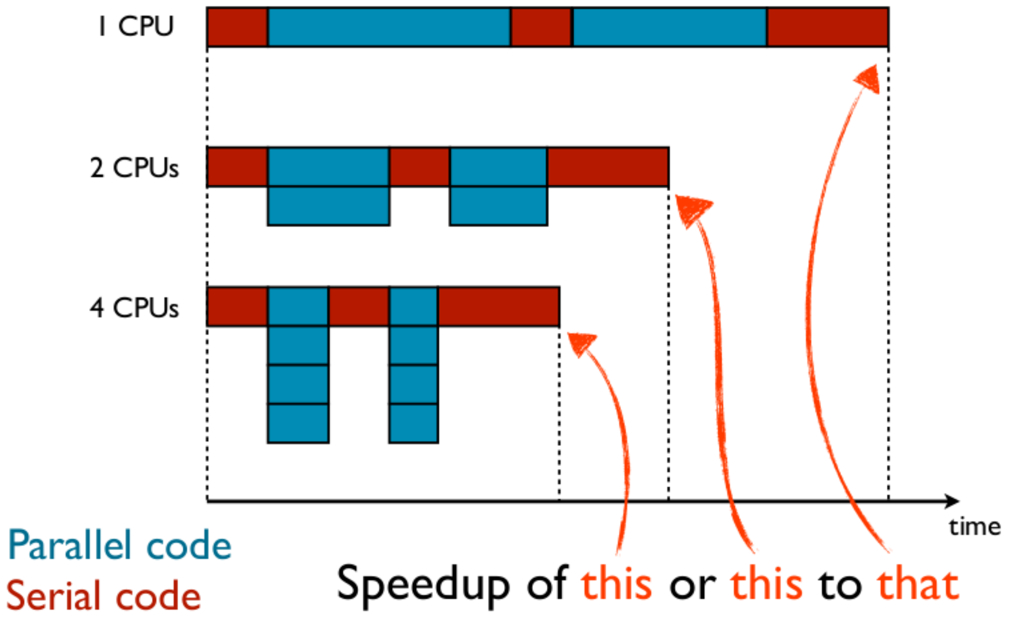


Figure 2.1: Amdahl's law [20]

If we have
t_1: time to run a job in instance type with 1 CPU
t_n: time to run a job in instance type with n CPUs
p: fraction of a job that can be ideally parallelized

$(1-p)$: fraction of a job that cannot be parallelized

Then the speedup S_n will be[20]

$$S_n = \frac{t_1}{t_n}$$

$$S_n = \frac{pt_1 + (1-p)t_1}{pt_n + (1-p)t_n}$$

$$S_n = \frac{pt_1 + (1-p)t_1}{\frac{pt_1}{n} + (1-p)t_1}$$

$$S_n = \frac{1}{\frac{p}{n} + (1-p)}$$

This expression is known as Amdahl's law. This is in almost all cases the best speedup one can achieve by doing work in parallel, so the real speed up S_n is less than or equal to this quantity.

Figure 2.2 shows how the speedup changes when the number of processors increases, with a couple of job's parallelization degrees.

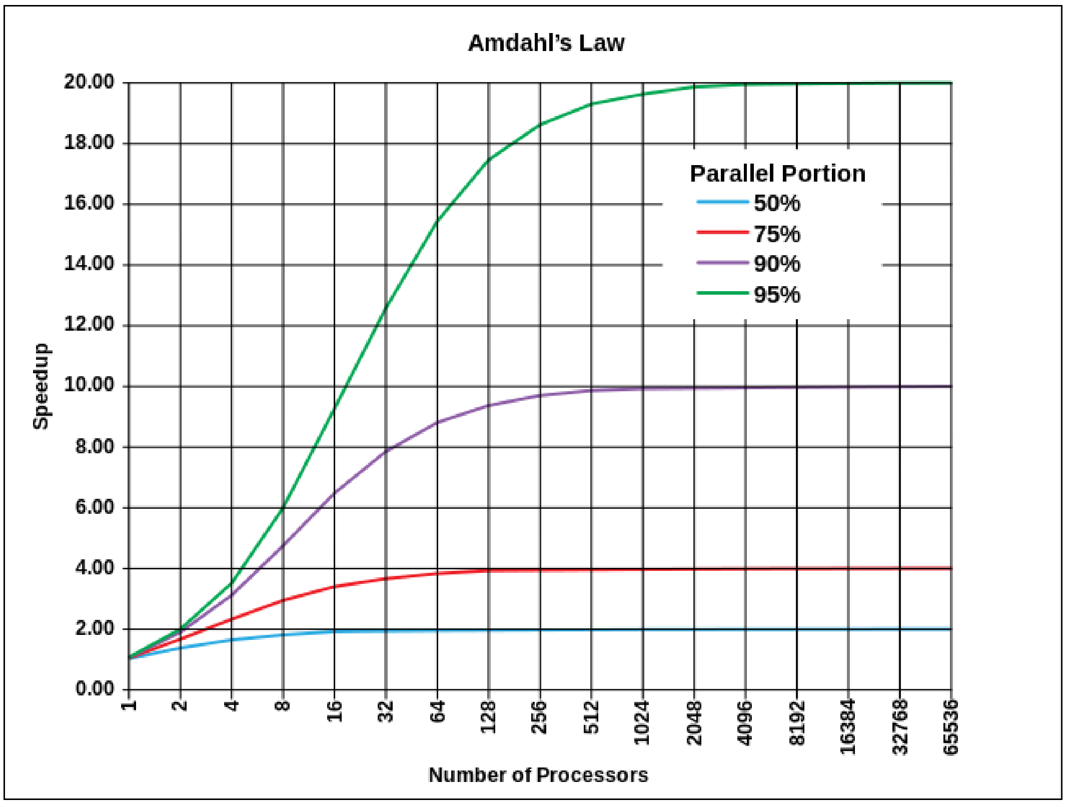


Figure 2.2: Amdahl's law [20]

It can be deducted from the Amdahl's law expression that as the number of core goes to infinite, the maximum speedup of a job will be

$$S_{\propto} = \frac{1}{(1-p)}$$

One limitation when using Amdahl's law to convert the runtime between the instance types is that the internal memory is not taken into consideration.

We will extend Amdahl's law to cover also the case when jobs are running in instance types with different CPU speeds and different number of CPUs. Apparently, if a proportion of job cannot be parallelized, in some cases, it is still possible to gain speedup by using faster CPU. We assume that speedup is linearly proportional to speed of CPU on which that job is running. We will go into detail on the algorithm to convert job runtime between instance types in following section.

Assume that we know the runtime of a job on an instance type u with configuration

CPU_u: CPU speed

RAM_u: Internal memory

$CORE_u$: Number of cores

t_u: Job runtime

We want to estimate the runtime of that job on an instance type v with configuration

CPU_v: CPU speed

RAM_v: Internal memory

$CORE_v$: Number of cores

t_v: Job runtime

p with $0 \leq p \leq 1$: The parallelization degree of the job, which is known upfront

The formula of the conversion can be written as

$$t_v = \frac{CPU_u}{CPU_v}.(1 - p + p\frac{CORE_u}{CORE_v})t_u = convert_{u->v}(t_u) \tag{2.1}$$

To better understand the law, we can go through an example. If for a given job in which 80% of the operations are parallelizable and its runtime on an instance type with 1.00GHz CPU, single core is 100 hours, then by applying (2.1), we can estimate the runtime of that job on 2.00Ghz, quadcore instance type is

$$\frac{1.00}{2.00}.(1 - 0.8 + \frac{0.8}{4}).100 = 20 \text{ (hours)}$$

CHAPTER 3

Proposed Solution

3.1 Problem Statement

A user has a scientific workload which is executed in the cloud environment. The workload consists of multiple tasks and each task has a number of independant jobs. The runtime of an individual job on an specific instance type is known upfront and the user requires the workload execution must be completed before a given deadline.

In this chapter, we will propose the algorithms to find out the option which optimizes the monetary cost of the workload execution while still being able to meet a given deadline constraint.

An option is defined as a set of specific instance types and the corresponding number of instances needed to complete a user workload.

We will develop the algorithms from simple case when there is only on-demand instance types available to more complex case in which user has purchased a number of reserved instances.

3.2 On-demand Instances

This section will handle the simple case in which only on-demand instances are taken into consideration. The variables and their meanings are summarized in Table 5.1

The monetary cost to complete the entire workload is also the cost to finish all tasks in it.

$$\text{Workload Cost} = \sum (T_i \text{ Cost}) \text{ with } 1 \leq i \leq N$$

Since a task in the workload is independent and can be executed without any interference from other tasks, the problem of finding optimal option to complete the workload within a given deadline is divided to finding optimal option for every individual task in workload.

$$Min(\text{Workload Cost}) = \sum Min(T_i \text{ Cost}) \text{ with } 1 \leq i \leq N$$

Table 3.1: System model description

Notions	Meaning
W	Workload
N	Number of tasks in W
T_i	The i^{th} task in W
n_i	Number of jobs submitted in $\mathrm{T_i}$
$J_{i,j}$	The j^{th} job in $\mathrm{T_i}$
v	Instance type v
$r_{i,j,v}$	The estimated runtime of $\mathrm{J_{i,j}}$ on instance type v
$p_{i,j}$	The estimated parallelization degree of $\mathrm{J_{i,j}}$
c_v	The cost per hour of instance type v
D	Time to deadline
B	Budget

3.2.1 Job runtimes and parallelization degrees are fixed

In this case, the runtimes and parallelization degrees of all jobs in a task are supposed to have the same values, meaning

$$r_{i,j,v} = r_{i,k,v} \text{ with } j \neq k \text{ and } j,k \in [1..n_i]$$

$$p_{i,j} = p_{i,k} \text{ with } j \neq k \text{ and } j,k \in [1..n_i]$$

Since the number of instance types offered by cloud providers is limited (i.e at present, AWS provides 46 instance types) and with the fact that jobs in a task are carried out by only 1 instance type, it is feasible to use brute-force algorithm in order to find out the optimal solution.

Therefore, we will compute the monetary costs to complete a task before deadline with all available instance types then examine the results to pick up the instance type with the best performance in term of monetary cost.

Assume that by experiments, the job runtimes on a specific instance type u are known upfront. By using Amdalh's law, it is possible to estimate the job runtimes on another instance type v.

$$r_{i,j,v} = convert_{u->v}(r_{i,j,u})$$

Regarding instance type v, a number of jobs that can be completed by a single instance during deadline time D is

$$\lfloor \frac{D}{r_{i,j,v}} \rfloor = \lfloor \frac{D}{convert_{u->v}(r_{i,j,u})} \rfloor$$

The number of job should be rounded down because a job is indivisible so even the deadline does not reach yet, if the time left is less than job runtime then that instance cannot be used to run another new job completely. In order to complete n_i jobs in T_i, the number of instance type v needed is

$$\lceil \frac{n_i}{\lfloor \frac{D}{r_{i,j,v}} \rfloor} \rceil$$

Instances are always charged hourly. Even a fraction usage of an instance-hour is counted as a full hour. It means regardless of 1 minute or 59 minutes, you still need to pay the same amount for 1 full hour of usage. So, the computed number of instance type v must be rounded up to adapt to that billing model. The corresponding cost is

$$\lceil \frac{n_i}{\lfloor \frac{D}{r_{i,j,v}} \rfloor} \rceil .c_v$$

By examine the cost on all available instance type, we can retrieve the optimal one with the lowest cost

$$Min(\lceil \frac{n_i}{\lfloor \frac{D}{r_{i,j,v}} \rfloor} \rceil .c_v) \text{ with } v \in \{\text{all available instance types}\}$$

```
function getEstimatedCostWithSpecificInstance(Task T,
    Instance type U) {
     int n = number of jobs in T;
     int rv = estimated runtime of jobs on instance type V
    which is known;
     int ru = runtime of jobs on U;
     int numberOfInstance = Number of U needed to complete T
    within deadline;
     int estimatedCost;
     if U does not meet T requirement {
         return "Instance configuration does not meet task
    requirement"
     }
     ru = Convert from rv using Amdalh law;
     numberOfInstance = roundup(n * (deadline/ru));
     estimatedCost = Number of U needed * Cost of U per hour;
     return numberOfInstance and estimatedCost;
}
```

Listing 3.1: Get estimated cost using a specific instance type

```
function getOptimalOptionForATask(Task T) {
    int optimalCostForTask;
    int optimalOptionForTask;
    for each instance type U in available instance types{
        instanceUsage and estimateestimatedCost =
    getEstimatedCostWithSpecificInstance(T, U);
        if (estimatedCost < optimalCost) {
            optimalCostForTask = estimatedCost;
            optimalOptionForTask = instanceUsage;
        }
    }
    return optimalOptionForTask and optimalCostForTask;
}
```

Listing 3.2: Get optimal option for a task in workload

```
function getOptimalOptionForWorkload(Workload W) {
    int optimalCost;
    Object optimalOption;
    for each Task T in W {
        optimalOptionForTask and optimalCostForTask =
    getOptimalOptionForATask(T);
        optimalCost + = optimalCostForTask;
        Put optimalOptionForTask into optimalOption
    }
    return optimalCost and optimalOption;
}
```

Listing 3.3: Get optimal option for workload

3.2.2 Job runtimes and parallelization degrees are different

In fact, it is not practical to assume that the job runtimes in a task are always the same. Depending on the applications, estimation can be difficult and sometimes impossible. Even one job may have different runtimes when being executed on a machine with the same configuration. Therefore, instead of finding a fixed runtime for all jobs in a task, it will be more feasible to build up a statistic model for job runtimes. Normally, it will not cost much because application inherent to business core of user, so it is supposed to be repeatedly executed in a regular basis. In the worst case, by experiments we are also able to have job runtimes statistic model relatively accurate. It is noted that this remark is also applicable to other parameters relating to all jobs in a task including parallelization degrees and data transfer requirements (in/out cloud environment). At this point of time, the tool supports 2 random models, normal and uniform distribution respectively.

The algorithm to find optimal option for each task execution is slightly different from the case with fixed parameters. It is worth noting that even only one instance type used but the usage time of each instance could be different reflecting the fluctuated job runtimes and parallelization degrees. For example, in order to finish a task T within 100 hours using solely m3.medium type, the option might be 10 instances running for 100 hours, 5 instance running for 90 hours and 5 running for 80 hours.

Regarding the task T_i, we start with the first instance v. Jobs in T_i can be completed within time period of D by the 1^{st} instance v is $J_{i,j}$ with $j = [1..k_1]$ with k_1 satisfying

$$\sum_{j=1}^{k_1} r_{i,j,v} \leq D \text{ and } \sum_{j=1}^{k_1+1} r_{i,j,v} \geq D$$

The running time of the 1^{st} instance v is

$$\sum_{j=1}^{k_1} r_{i,j,v}$$

Similarly, the 2^{nd} instance v will execute in time period D jobs $J_{i,j}$ with $j = [(k_1 + 1)..k_2]$, in which

$$\sum_{j=k_1}^{k_2} r_{i,j,v} \leq D \text{ and } \sum_{j=k_1}^{k_2+1} r_{i,j,v} \geq D$$

The running time of the 2^{nd} instance v is

$$\sum_{j=k_1}^{k_2} r_{i,j,v}$$

If we keep increasing the number of instances v one by one until all the jobs in T_i complete then the number of instance v and their running time needed to satisfy deadline D can be retrieved. The number of instance is I_i with

$$\sum_{j=k_{I_i}}^{n_i} r_{i,j,v} \leq D$$

The execution cost of T_i is

$$\sum_{j=1}^{n_i} r_{i,j,v}.c_v$$

By examining the cost on all available instance types, we can retrieve the optimal option with the lowest cost. Even the execution cost can be easily computed in this case but we still need to go through the steps above to get the running time of each instance in the optimal option. Indeed, this solution covers the case in Section 3.2.1 and gives more accurate results.

```
function getEstimatedCostWithSpecificInstance(Task T,
    Instance type U) {
     int executionTime = 0;
     int numberOfInstanceUsed = 0;
     float estimatedCost = 0;

     /*Store different usage time of instances*/
     int[] instanceUsage;
     if U does not meet T requirement {
     return "Instance configuration does not meet task
    requirement"
     }
     for (i = 1; i <= number of jobs in T, i++) {
     jobRuntime = Compute runtime of ith job on U using Amdalh
     law;
     if (jobRuntime > deadline) {
         return "Unable to complete this task on instance u
    before deadline"
     }
     if ((executionTime + jobRuntime) > $deadline) {
         estimatedCost += executionTime * (Cost of u per hour)
    ;
         Push executionTime to instanceUsage array;
         executionTime = jobRuntime;
     } else {
         executionTime += jobRuntime;
     }
     }
     return instanceUsage and estimatedCost;
}
```

Listing 3.4: Get estimated cost using a specific instance type

3.3 Reserved Instances

In previous section, only on-demand instances are taken into consideration. However, many cloud providers also deliver prepaid instances (or reserved instances as the term used by AWS). This type of offer gives users the ability to maximize their cost savings by purchasing reserved instances that meets their businesss needs upfront. The reserved instances allow users to make low, one-time payment to reserve resources and then gain a big discount on the hourly rate when those resources are in use.

Suppose that a user has purchased a number of reserved instances of different types, how to allocate the computing power of these instances to tasks and jobs in a workload is not a trivial problem. In fact, the user has many options to assign

Table 3.2: Extended system model description

Notions	Meaning
W	Workload
N	Number of tasks in W
$\mathrm{T_i}$	The $\mathrm{i^{th}}$ task in W
$\mathrm{n_i}$	Number of jobs submitted in $\mathrm{T_i}$
$\mathrm{J_{i,j}}$	The $\mathrm{j^{th}}$ job in $\mathrm{T_i}$
v	Instance type v
$n_{ri}(v)$	Number of reserved instance type v available
$\mathrm{r_{i,j,v}}$	The estimated runtime of $\mathrm{J_{i,j}}$ on instance type v
$\mathrm{p_{i,j}}$	The estimated parallelization degree of $\mathrm{J_{i,j}}$
$\mathrm{c_v}$	The cost per hour of instance type v
D	Time to deadline
B	Budget
$AT_k(v)$	Available hours of k^{th} reserved instance v
RI	all types of reserved instance available
$n_{ri}(v)$	Number of reserved instance v available
$t_{ri(v)}$	Time left until reserved instance v expires
$c_{ri(v)}$	The cost per hour of reserved instance type v
$C(i,v)$	The cost generated when running task T_i by an instance type v
$ET(i,v)$	The execution time when running task T_i by an instance type v
$min(C_{OD}(T_i))$	The optimal cost when running T_i by on-demand instances only

reserved instances to tasks which will generate different monetary cost. For example, if the user needs to allocate 3 reserved instances of different types to 3 tasks, he already has 6 options. The number of options will explode if the user has purchased many reserved instances or the number of tasks increases. It is required to find the option which can give good performance in terms of cost-effectiveness. Brute-force strategy is not an efficient solution to this problem.

There have been a number of studies concerning this problem, and most of them go to solving an linear programming problem which gives more accurate results but with the compromise of algorithm complexity.

Another solution, which is simpler, is to distribute computing power of the reserved instances to jobs in the workload proportionally to the jobs' runtimes. However, this solution has some limitations. It ignores the fact that cloud providers charge their virtual machines on hourly basis, meaning that even if an instance is activated for only 1 minute, the user still has to pay for the whole hour of usage. In addition, the jobs in the workload are indivisible and executed uninterruptedly on a single instance only. Therefore, by distributing the computing power of the reserved instances proportionally to runtimes, there is high possibility that a job can be partially executed by the reserved instances and the rest of that job is completed by another on-demand instance which violates our assumption.

We will propose a different approach to allocate reserved instances to tasks in the workload in an cost-efficient way.

Technically, a reserved instance has the same operational characteristics and performance as an on-demand instance of the same type while running. However, we will consider that they are of different types because the hourly rate of a reserved instance is always lower than an on-demand instance.

Step 1

In the first step, we apply the algorithm in section 3.2 but this time, all types of existing reserved instances need to be included. The fixed payment for reserved instances is skipped in the cost computation. The output of the algorithm are options (e.g instance type and number of instances) that gives the lowest execution cost for each task in the workload, provided that a task is solely completed by one instance type.

- With tasks which the optimal option uses an on-demand instance type, it means the execution of such tasks does not benefit from available reserved instances and they can be excluded from the next steps in our solution. The workload consisting of such tasks is called W_{OD}.
- With tasks which the optimal uses a reserved instance type, we need to have further investigation because it is likely that we can make use of reserved instances to lower the execution cost. The workload consisting of such tasks is called W_{RI}.

Step 2

In the second step, we build a list of possible combinations of a task in the workload and an instance type in the existing reserved instances. Each item in the list is presented as (i, v) for the task T_i and instance type v. The items in this list are sorted descendingly following their cost-effectiveness. The definition of cost-effectiveness will be explained later in this section

Based on this list, reserved instances will be assigned to tasks in order to gain cost-efficient performance. It is worth noting that the users also preferably use reserved instances with shorter expiration time. For example, if there are 2 reserved instances with expiration time 10 and 20 hours respectively, the user would prefer to use the one with 10 hours expiration time first.

Step 3

Suppose that (i, v) is the first item in the list. It means that executing task T_i on an instance type v gains the most efficient performance in terms of cost. We will allocate $n_{ri}(v)$ instances v to run task T_i. At this point, we have 2 possibilities:

- **Case 1:** Within D, $n_{ri}(v)$ reserved instances v are not able to complete task T_i. The remaining jobs in T_i will be executed by on-demand instances of the same type v since following assumption in Section 2.4, a task can only be executed by a single instance type. The execution cost of T_i will be the sum of reserved instances' cost and on-demand instances' cost. However, in case a user has purchased expensive reserved instances, by using such expensive ones to run

T_i, the user risks that the cost may go up significantly. Instead, it is possible to use cheaper on-demand instances and still satisfying the deadline constraint. In order to avoid such unexpected situation, it is required to compare the computed execution cost with $min(C_{OD}(T_i))$, the minimum cost when using on-demand instances only.

- If the computed cost is higher than $min(C_{OD}(T_i))$, it means that the execution of T_i does not benefit from reserved instances v. We will pick up the next item in the ordered list and go back to **Step 3**
- If computed cost is lower than $min(C_{OD}(T_i))$, it means that cost savings can be achieved by using reserved instances v. The option using both reserved and on-demand instances v for T_i is selected. In addition, we need to compute the left of available hours of each reserved instance v after executing T_i. Even T_i cannot be comleted by reserved instances v, it is possible that there are still some unused hours of reserved instances v. We then take the next item (j, u) in the list with $i \neq j$ as the task T_i has been completed already and go back to **Step 3**. If $(u = v)$, we can use the left of available hours of reserved instances v to execute T_j

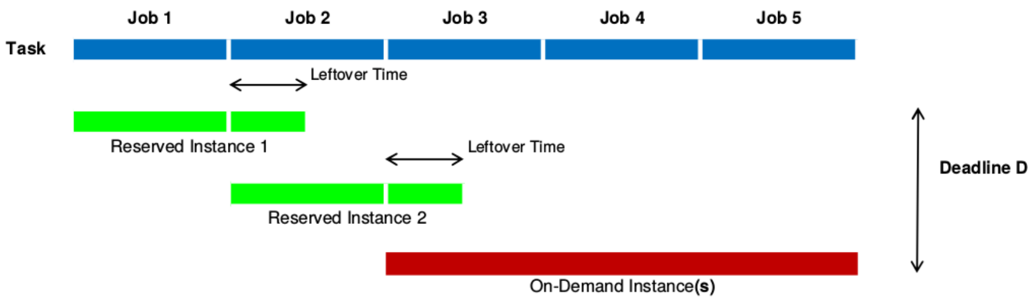


Figure 3.1: Example for Case 1

- **Case 2:** Within D, the task T_i can be completed by $n_{ri}(v)$ reserved instances v. The T_i execution cost will be

$$(\sum_{j=1}^{n_i} r_{i,j,v}).c_{ri(v)} \tag{3.1}$$

Similar to **Case 1**, the computed execution cost needs to be compared with $min(C_{OD}(T_i))$.

  - If the computed cost is higher than $min(C_{OD}(T_i))$, no cost savings can be gained from reserved instances v. We continue with the next item in the list and return to **Step 3**.

– If the computed cost is lower than $min(C_{OD}(T_i))$, the option using reserved instances v for T_i is selected. We need to compute the left of available hours of each reserved instance v after completing T_i. Then we pick up the next item (j, u) in the list which has $i \neq j$ and go back to **Step 3**. If $(u = v)$, we can use the left of available hours of reserved instances v to execute jobs in T_j.

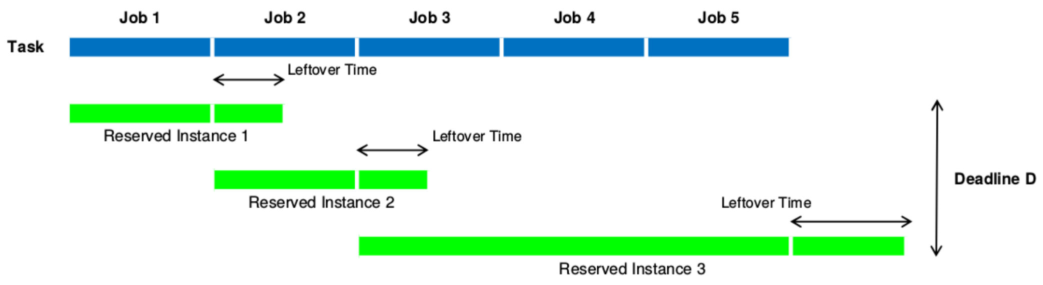


Figure 3.2: Example for Case 2

These steps are applied until one of the following conditions is met

- The cost-efficient option for all tasks in W_{RI} are selected.
- We reach the end of the list and there are remaining tasks under consideration. In this case, with the tasks that execution options have not been selected yet, they do not gain cost savings by using reserved instances. Therefore, the optimal options finding in **Step 1** will be selected for these tasks.

In the next section, we will exlain in more detail about the computation in each step of the algorithm. It is assumed that the jobs'runtimes on instance type u, ($r_{i,j,u}$ with $i = 1..N$, $j = 1..n_i$), are aware upfront.

We define the *load unit computed on instance type* v is the number of jobs that can be finished within 1 hour (billing unit) by a single instance v. Given such definition, the load of a job $J_{i,j}$ computed on instance type v is indeed the runtime $r_{i,j,v}$. Since jobs' runtimes on the instance type u are known upfront, it is possible to compute the load of $J_{i,j}$ on any other instance type v using Amdalh's law. Thus, we have the load of task T_i computed on v will be:

$$\sum_{j=1}^{n_i}(r_{i,j,v}) = \sum_{j=1}^{n_i} convert_{u->v}(r_{i,j,u})$$

The number of available hours of k^{th} reserved instance v is represented by $AT_k(v)$ with $1 \leq k \leq n_{ri}(v)$. Reserved instances v are sorted ascendingly following their available hours, meaning $AT_m(v) \leq AT_n(v)$ if $m < n$ and $1 \leq m, n \leq n_{ri}(v)$.

The maximum value of $AT_k(v)$ is the deadline D since the execution of the workload is required to complete within D. We can use reserved instances during that time only.

Compute the execution cost and the left of available hours of reserved instances in Case 1

Assuming that $n_{ri}(v)$ reserved instances v are allocated to run T_i. The number of jobs in T_i that are completed by the first reserved instance v is k_1 with k_1 satisfying

$$\sum_{j=1}^{k_1} r_{i,j,v} \leq AT_1(v) \leq \sum_{j=1}^{k_1+1} r_{i,j,v}$$

After running k_1 jobs, the leftover of usable hours of 1^{st} reserved instance v is

$$AT_1(v) - \sum_{j=1}^{k_1} r_{i,j,v}$$

Next, the first and the second reserved instances together can complete k_2 jobs in T_i with

$$\sum_{j=k_1}^{k_2} r_{i,j,v} \leq AT_2(v) \leq \sum_{j=k_1}^{k_2+1} r_{i,j,v}$$

and the leftover of usable hours of 2^{nd} reserved instance v is

$$AT_2(v) - \sum_{j=k_1}^{k_2} r_{i,j,v}$$

Apply this rule repeatedly, we have the number of jobs in T_i that can be executed by $n_{ri}(v)$ reserved instances v is $k_{n_{ri}(v)}$ with

$$\sum_{j=k_{(n_{ri}(v)-1)}}^{k_{n_{ri}(v)}} r_{i,j,v} \leq AT_{n_{ri}(v)}(v) \leq \sum_{j=k_{(n_{ri}(v)-1)}}^{k_{n_{ri}(v)}+1} r_{i,j,v}$$

and the leftover of available hours of $(n_{ri}(v))^{th}$ reserved instance v is

$$AT_{n_{ri}(v)}(v) - \sum_{j=k_{(n_{ri}(v)-1)}}^{k_{n_{ri}(v)}} r_{i,j,v}$$

At this point, we are no longer able to use reserved instance v and the remaining of the task T_i will be executed by on-demand instances v. The load of the remaining jobs would be

$$\sum_{j=(k_{n_{ri}(v)}+1)}^{n_i} r_{i,j,v}$$

Total cost will be the sum of reserved instance cost and on-demand instance cost, which is

$$(\sum_{j=1}^{k_{n_{ri}(v)}} r_{i,j,v}).c_{ri(v)} + (\sum_{j=k_{n_{ri}(v)}+1}^{n_i} r_{i,j,v}).c_v$$

Figure 3.1 introduces a simple example describing the computation.

Compute the left of available hours of reserved instances in Case 2

The number of jobs in T_i that can be completed by the first reserved instance v is k_1 with k_1 satisfying

$$\sum_{j=1}^{k_1} r_{i,j,v} \leq AT_1(v) \leq \sum_{j=1}^{k_1+1} r_{i,j,v}$$

and the leftover of usable hours of that reserved instance is

$$AT_1(v) = AT_1(v) - \sum_{j=1}^{k_1} r_{i,j,v}$$

Next, the first and the second reserved instances together can complete k_2 jobs in T_i with

$$\sum_{j=k_1}^{k_2} r_{i,j,v} \leq AT_2(v) \leq \sum_{j=k_1}^{k_2+1} r_{i,j,v} \tag{3.2}$$

and the leftover of usable hours of 2nd reserved instance v is

$$AT_2(v) - \sum_{j=k_1}^{k_2} r_{i,j,v} \tag{3.3}$$

Apply this computation repeatedly until s^{th} reserved instance that we have

$$\sum_{j=k_(s-1)}^{n_i} r_{i,j,v} \leq AT_s(v)$$

and the leftover of usable hours of s^{th} reserved instance v is

$$AT_s(v) - \sum_{j=k_(s-1)}^{n_i} r_{i,j,v}$$

In this case, T_i can be completed by using reserved instances v. The leftover of usable hours of 1^{st} to s^{th} reserved instances can be computed as in Formular (3.2) and we have not yet used any computing power of $(s+1)^{th}$ to $(n_{ri}(v))^{th}$ reserved instance v.

Figure 3.2 gives better explanation of the computation in this section.

Cost Effectiveness

Cost effectiveness value is to measure how efficient it is when executing a task on a specific instance type. Suppose we have a task T_i and an instance type v. The execution cost of T_i on a single instance v is

$$C(i,v) = \sum_{j=1}^{n_i} (r_{i,j,v}).c_v$$

with the execution time

$$ET(i,v) = \sum_{j=1}^{n_i} r_{i,j,v}$$

If task T_i is executed by instance type s, the cost is

$$C(i,s) = \sum_{j=1}^{n_i} (r_{i,j,s}).c_s$$

If we have

$$C(i,v) \leq C(i,s)$$

Apparently, executing T_i on v is more efficient than on s because the user pays less to complete T_i on v.

However, the issue is how to compare the cost-effectiveness when executing T_i on v and T_k on s with $i \neq k$ and $v \neq s$. As the characteristics of T_i and T_k are different, we cannot use purely the execution cost to address the cost effectiveness.

Suppose the cost to execute T_k by s is

$$C(k,s) = \sum_{j=1}^{n_k} (r_{i,j,s}).c_s$$

with the execution time

$$ET(k,s) = \sum_{j=1}^{n_k} r_{i,j,s}$$

Recalling that job runtimes on instance type u known upfront, we will use task execution on u as a reference in order to compare the cost effectiveness of 2 combinations (i,v) and (k,s). The following rules will be applied

- If $C(i,v) > C(i,u)$, $C(k,s) > C(k,u)$ and $ET(i,v) < ET(i,u)$, $ET(k,s) < ET(k,u)$, we consider it is more efficient when executing T_i on v if

$$\frac{C(i,v) - C(i,u)}{ET(i,u) - ET(i,v)} \leq \frac{C(k,s) - C(k,u)}{ET(k,u) - ET(k,s)}$$

  It means we need to pay less money on (i,v) to gain the same reduction in execution time as compared to (k,s).

- If $C(i, v) < C(i, u)$, $C(k, s) < C(k, u)$ and $ET(i, v) > ET(i, u)$, $ET(k, s) > ET(k, u)$ we consider it is more efficient when executing T_i on v if

  $$\frac{C(i, u) - C(i, v)}{ET(i, v) - ET(i, u)} \geq \frac{C(k, u) - C(k, s)}{ET(k, s) - ET(k, u)}$$

  It means in order to save the same amount of money, we need to compensate less on the extension of execution time with (i, v) than (k, s).

CHAPTER 4

Customer Tool Trading off Cost and QoS

In order to realize the approaches in chapter 3 in a practical application, we build a software system which provides users with sufficient information to evaluate their workload execution in the cloud environment, called Customer Tool Trading off Cost and QoS (abbr. Customer Tool). Customer Tool also gives information on how the cost level expected to change with the provided level of QoS.

4.1 System Architecture

Customer Tool is a web-based application which allows users to interact via a web-based interface. The system consists of four components: database, configuration management, logic and user interface components. We will explain detail of each component in the next sections.

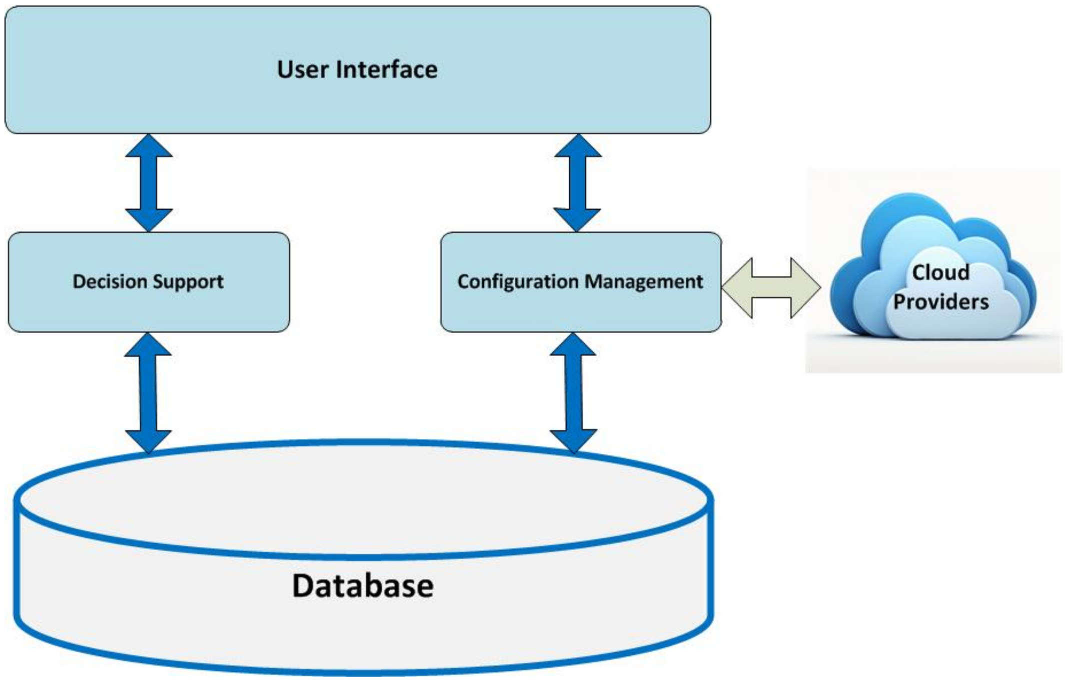


Figure 4.1: The Architecture of Customer Tool

4.1.1 Configuration Management

Service Configuration Model

Configuration Management component is to maintain a basic model related to services delivered by cloud providers. Since cloud services are offered in different configurations and there is no standardized naming and terminology for them, it is necessary to have a model which is able to capture and describe the cloud services in common terms. The model should be abstract and flexible enough to accommodate new services with little or no changes to the existing implementation.

By studying from a number of cloud providers and refer to other studies [34], we have the general service model in Table 4.1.

Pricing Model

In theory, as the main characteristic of a cloud computing is Pay as You Go, the cloud computing should offer a straightforward model for consuming resources. Users should be able to spin up and down resources as they need and only pay for what they use. Unfortunately, cloud pricing models are complicated which makes purchasing decisions for the users and the comparison across multiple providers a challenge. There is a large range of pricing models expressed in non-standardized terms that can make cloud pricing ambiguous for consumers. Each provider has its own pricing policy and the pricing policy can vary significantly. For example, it can be as simple as in on-demand plan or as complicated and unpredictable as spot instance price in AWS.

To address such problem, we propose a generic pricing model for cloud services

Table 4.1: Services Model

PARAMETERS	**POSSIBLE VALUES**
Instance Type	
CPU speed (Ghz)	> 0
Internal Memory (MB)	> 0
Number of cores	$1 \geq cores \geq 32$
Architecture	32-bit or 64-bit
OS Platform	Linux or Windows
Storage	
Storage size (GB)	> 0
Data Center	up to cloud providers
IO Request Requirement	High or Low
Network	
Data Transfer In (GB)	> 0
Data Transfer Out (GB)	> 0

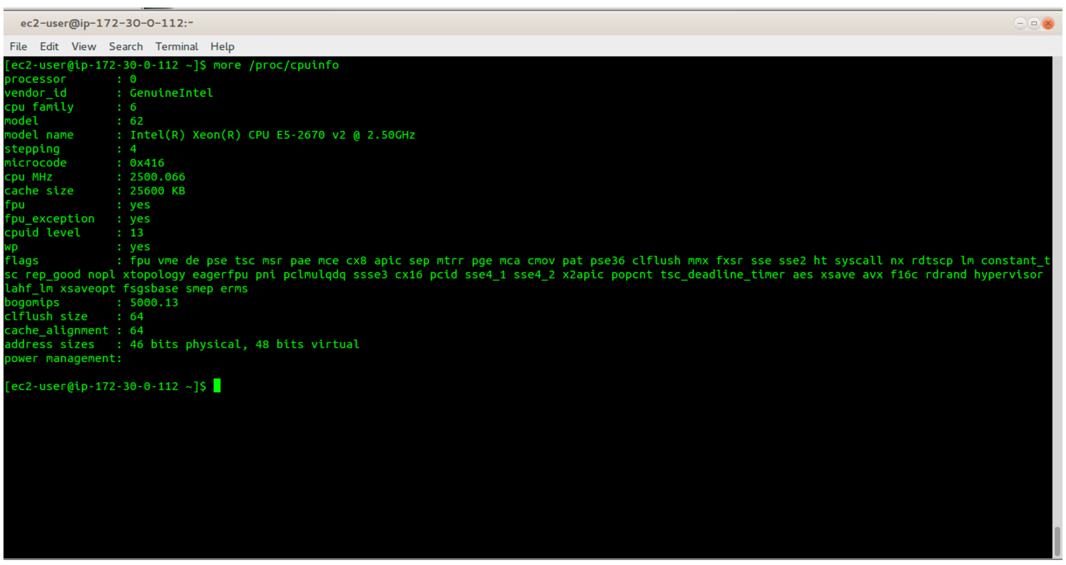
in Table 4.2.

Table 4.2: Pricing Model

PARAMETERS	**POSSIBLE VALUES**
Service Configuration	
On Demand	Price per hour
Prepaid	Prepaid term, fixed price and price per hour
Spot Instance	Spot Price
Storage	
Storage Price	Rate tiers based on storage size
Network	
Data Transfer Price (GB)	Rate tiers based on transfer size

Adapter module in Configuration Management component provides a common interface to convert service configurations and service prices from different cloud providers to the generic model mentioned earlier. For example, regarding CPU speed on a virtual machine, a basic product that offered by all providers, Amazon Web Service refers to Amazon EC2 instance type with the CPU speed computed on ECU unit. On the other hand, other providers such as a Cloud Server of GoGrid or Virtual Machine of Windows Azure do not declare or make commitment on the CPU speed of their virtual machines. According to GoGrid documentation, "CPU allocation will vary depending on the profile of physical node on which Cloud Server resides". In order to eliminate this discrepancy, we can acquire a virtual machine from each cloud provider and measure the CPU speed on that virtual machine by a specific software tool (i.e "more /proc/cpuinfo" on Linux OS).

Another functionality of Adapter module is to facilitate the interaction between Customer Tool and the cloud providers. The Adapter module allows users to retrieve


ec2-user@ip-172-30-0-112:~
File Edit View Search Terminal Help

```
[ec2-user@ip-172-30-0-112 ~]$ more /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 62
model name      : Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz
stepping        : 4
microcode       : 0x416
cpu MHz         : 2500.066
cache size      : 25600 KB
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_t
sc rep_good nopl xtopology eagerfpu pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor
lahf_lm xsaveopt fsgsbase smep erms
bogomips        : 5000.13
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:

[ec2-user@ip-172-30-0-112 ~]$ █
```

Figure 4.2: Running "more /proc/cpuinfo" on t2.micro instance

information and update changes from the cloud providers such as pricing policy, available instance types, available data center, price history and so forth. At this point of time, Customer Tool is only working with Amazon Web Service but it can be easily expanded to support other cloud providers. To do so, we need to write a new class implementing adapter interface using suitable set of APIs supplied by cloud providers.

4.1.2 Database

Service configuration and pricing policy after being converted to generic parameters by the Adapter module will be stored in database and can be fetched later using standard SQL queries.

Notions, terminologies and information specific to a cloud provider are kept in database although such information has no impact on the algorithm. However, with the assumption that users have little or no knowledge on the detail of the providers' products and they are dependant completely on Customer Tool in selecting right options to execute their workload, it is expected that sufficiently detailed information should be provided in the outcome of the tool.

Figure 4.3 shows the entities and their relationship in database design. The instance_type and price_model entities are to store services from different provides in the general service model and pricing model. instance_type entity can be updated with latest changes from cloud providers by using the functionality in the Adapter module.

Beside cloud providers' data, workload/task templates, users settings and preferences are also stored in database for management tasks. As can be seen from figure 4.4, the characteristics of a task in a workload is covered by entity u_user_task_template.

```
interface ICloudAdapter {
    /**
     * Retrieve instance type configuration, price and
    convert to generic model
     */
    public function updateOnDemandPrices();

    /**
     * Retrieve price for reserved instances and convert to
    generic model
     */
    public function updateReservedInstancesPrices();

    /**
     * Retrieve storage offers, price and convert to generic
    model
     */
    public function updateStoragePrices();

    /**
     * Retrieve purchased reserved instances
     */
    public function retrieveExistingReservedInstances(
    $accessKeyId, $secretAccessKey);
}
```

Listing 4.1: Common Interface for Adapter

The workload/task templates save users' time in providing information for frequently-used workload patterns. User management functionality will be discussed in more detail in section 4.1.4

4.1.3 Decision Support Component (DSC)

DSC is the core component of the system. Generally speaking, input of DSC is user's workload properties and requirements on level of QoS. QoS in Customer Tool is expressed as constraints on execution deadline and budget. The outcome of DSC is information of the options which assists user in deciding and picking up a suitable one for his workload.

Figure 4.5 depicts the structure of DSC. DSC consists of a couple of modules, each module carries out one functionality of the system or provide services to other modules in DSC.

Cost Computation Module (CCM)

As mentioned in section 2.3, a workload has multiple tasks in it. CCM functionality is to calculate generated cost when execute a separate task on a specific instance type provided that the task must be finished before a given deadline. CCM is in fact the implementation of algorithm in chapter 3. One noted issue is that before

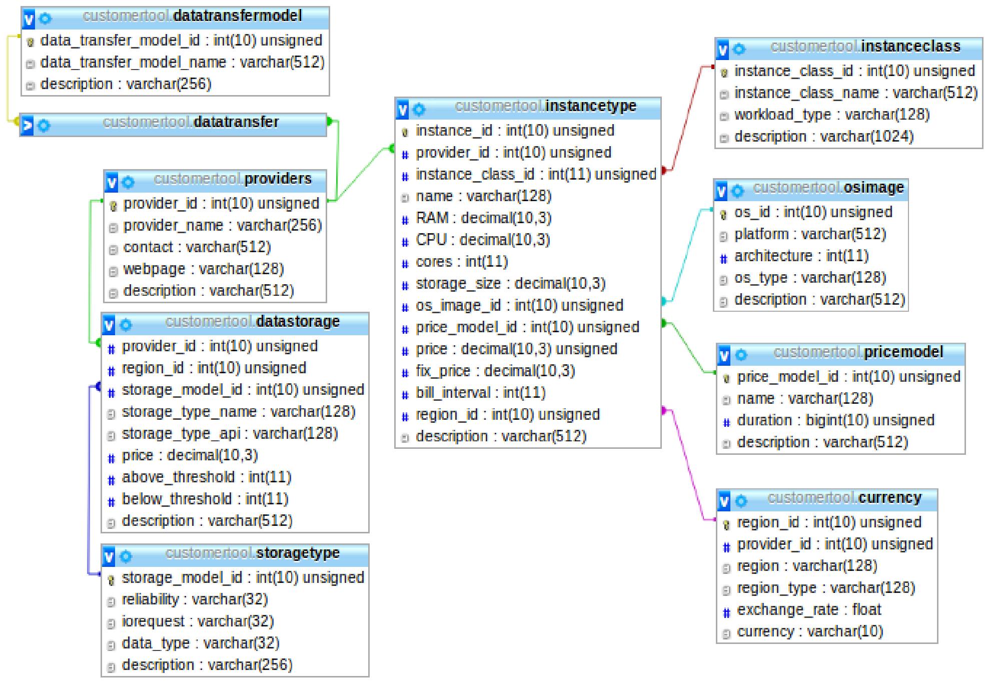

customertool.datatransfermodel
data_transfer_model_id : int(10) unsigned
data_transfer_model_name : varchar(512)
description : varchar(256)
customertool.datatransfer
customertool.providers
provider_id : int(10) unsigned
provider_name : varchar(256)
contact : varchar(512)
webpage : varchar(128)
description : varchar(512)
customertool.datastorage
provider_id : int(10) unsigned
region_id : int(10) unsigned
storage_model_id : int(10) unsigned
storage_type_name : varchar(128)
storage_type_api : varchar(128)
price : decimal(10,3)
above_threshold : int(11)
below_threshold : int(11)
description : varchar(512)
customertool.storagetype
storage_model_id : int(10) unsigned
reliability : varchar(32)
iorequest : varchar(32)
data_type : varchar(32)
description : varchar(256)
customertool.instancetype
instance_id : int(10) unsigned
provider_id : int(10) unsigned
instance_class_id : int(11) unsigned
name : varchar(128)
RAM : decimal(10,3)
CPU : decimal(10,3)
cores : int(11)
storage_size : decimal(10,3)
os_image_id : int(10) unsigned
price_model_id : int(10) unsigned
price : decimal(10,3) unsigned
fix_price : decimal(10,3)
bill_interval : int(11)
region_id : int(10) unsigned
description : varchar(512)
customertool.instanceclass
instance_class_id : int(10) unsigned
instance_class_name : varchar(512)
workload_type : varchar(128)
description : varchar(1024)
customertool.osimage
os_id : int(10) unsigned
platform : varchar(512)
architecture : int(11)
os_type : varchar(128)
description : varchar(512)
customertool.pricemodel
price_model_id : int(10) unsigned
name : varchar(128)
duration : bigint(10) unsigned
description : varchar(512)
customertool.currency
region_id : int(10) unsigned
provider_id : int(10) unsigned
region : varchar(128)
region_type : varchar(128)
exchange_rate : float
currency : varchar(10)

Figure 4.3: Model Entities and Relationship

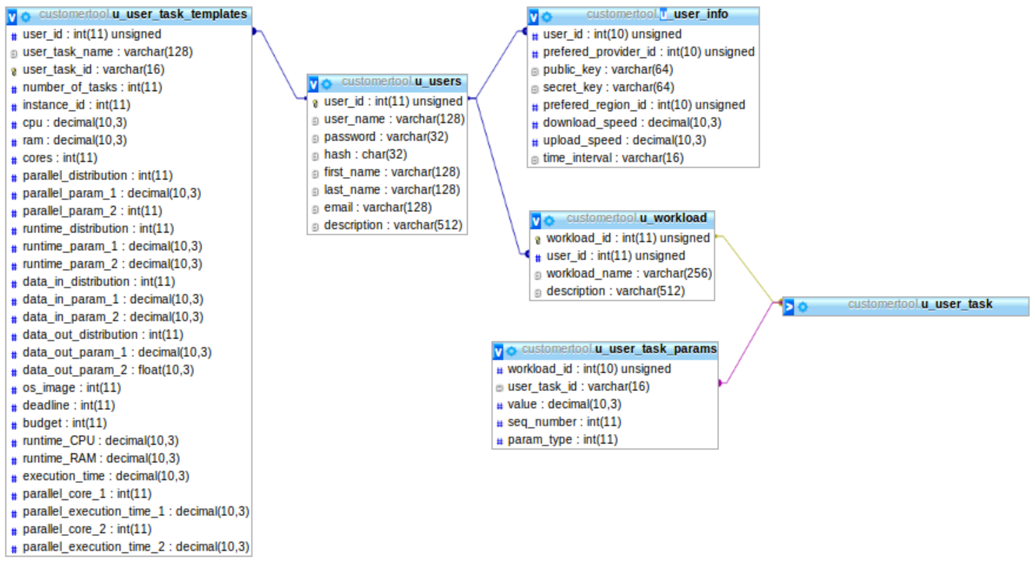

customertool.u_user_task_templates
user_id : int(11) unsigned
user_task_name : varchar(128)
user_task_id : varchar(16)
number_of_tasks : int(11)
instance_id : int(11)
cpu : decimal(10,3)
ram : decimal(10,3)
cores : int(11)
parallel_distribution : int(11)
parallel_param_1 : decimal(10,3)
parallel_param_2 : int(11)
runtime_distribution : int(11)
runtime_param_1 : decimal(10,3)
runtime_param_2 : decimal(10,3)
data_in_distribution : int(11)
data_in_param_1 : decimal(10,3)
data_in_param_2 : decimal(10,3)
data_out_distribution : int(11)
data_out_param_1 : decimal(10,3)
data_out_param_2 : float(10,3)
os_image : int(11)
deadline : int(11)
budget : int(11)
runtime_CPU : decimal(10,3)
runtime_RAM : decimal(10,3)
execution_time : decimal(10,3)
parallel_core_1 : int(11)
parallel_execution_time_1 : decimal(10,3)
parallel_core_2 : int(11)
parallel_execution_time_2 : decimal(10,3)
customertool.u_users
user_id : int(11) unsigned
user_name : varchar(128)
password : varchar(32)
hash : char(32)
first_name : varchar(128)
last_name : varchar(128)
email : varchar(128)
description : varchar(512)
customertool.u_user_info
user_id : int(10) unsigned
prefered_provider_id : int(10) unsigned
public_key : varchar(64)
secret_key : varchar(64)
prefered_region_id : int(10) unsigned
download_speed : decimal(10,3)
upload_speed : decimal(10,3)
time_interval : varchar(16)
customertool.u_workload
workload_id : int(11) unsigned
user_id : int(11) unsigned
workload_name : varchar(256)
description : varchar(512)
customertool.u_user_task
customertool.u_user_task_params
workload_id : int(10) unsigned
user_task_id : varchar(16)
value : decimal(10,3)
seq_number : int(11)
param_type : int(11)

Figure 4.4: Database model representing system administration

doing the actual cost calculation, CCM also check whether a instance type matches the minimum configuration (i.e CPU speed, number of cores, RAM) set by the user

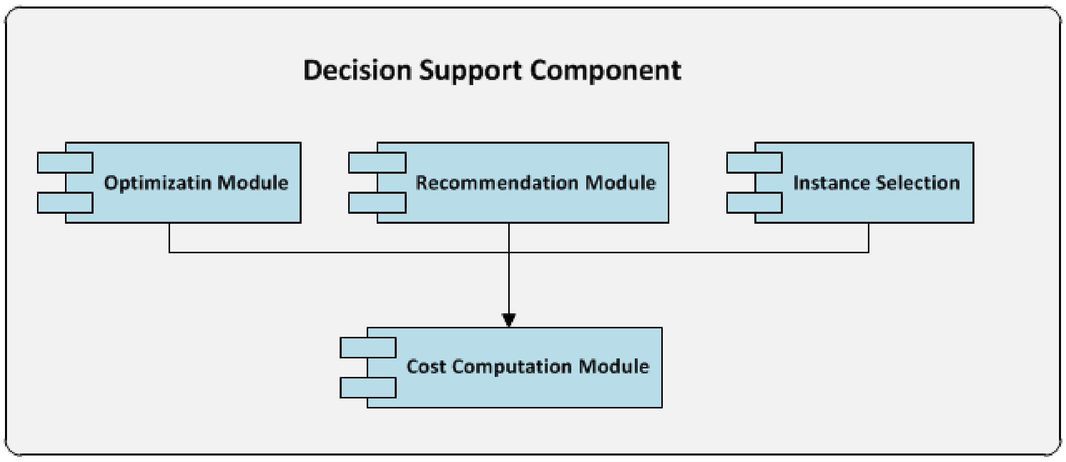


Figure 4.5: Decision Support Component Architecture

for that task. If not then the option with such instance type becomes invalid. Most of other modules in DSC are built on top of CCM.

Storage Cost

Beside computing, an application workload typically requires some sort of data processing. In addition, tasks in the workload can also share a set of data in the cloud. It is a common case because tasks are assumed to be loosely coupled or independant. Therefore it is likely that the only way to exchange information among them is using common data storage.

Cloud providers offer many storage systems that can be deployed for the workload application. The storage systems are differentiated by several properties such as data reliability, IO requests, storage capacity, durability, price policy and so forth.

Instead of having a user getting lost in many different terms and definitions from multiple cloud providers, the user should need to provide his requirements in high level of abstraction and the system will be smart enough to recommend him qualified storage systems with the lowest cost.

Network Cost

Cloud providers not only charge data stored in but also the data transferred in/out the cloud environment. Each task in a workload normally has its own data that is required to upload or download to/from cloud. Therefore, to have more accurate estimation of monetary cost when executing workload, the cost for data transfer should be taken into account.

Optimization Module (OM)

Regarding each task in a workload, by using CCM, the estimated cost to finish that task before the deadline by using any specific instance type can be retrieved. OM iterate through the results of all available instance types and select the option with the lowest monetary cost. An option for a task includes an instance type, the number of instances and real execution time (in some cases, if an instance type with high capacity is used, the workload execution may finish before the deadline)

Apply the same algorithm for all tasks in the workload, OM proposes the optimal options to complete tasks within the deadline, which also the optimal option for the workload.

Recommendation Module (RM)

The option with lowest execution cost is not always the users' desire. A user, in some cases, would be willing to pay higher cost but gain other benefits such as shorter execution time, less number of instances used and so forth.

To assist the user in such cases, in addition to the result from Optimization Module, RM gives the user a list of recommendations on the other options which satisfy the deadline contraint. These options use different instance types to the optimal option. Obviously, they are not the best options in term of cost, but by going through them, the user has a general idea on how the cost might change depending on the instance types with the same level of QoS.

On the other hand, RM proposes a number of the optimal options for the workload execution with different level of QoS in the region of the QoS required by the user. For example, if the given deadline is 100 hours, RM will compute the optimal options for the deadlines of 50 to 90 hours and 110 to 150 hours.

The objective of this functionality is to show the correlation between level of QoS and the estimated execution cost. Again, based on the output, the user has more information which may help him in making a suitable decision. Depending on his preferences, the user can choose to relax the constraint to save execution cost or to pay more and gain a shorter execution time.

Instance Selection Module (ISM)

Given a task in the workload, CCM enables the user to find out an instance type which can complete task before the deadline with minimum cost. However, as mentioned earlier, the cost savings are not always the highest priority of the user, he may be interested in other criteria. With ISM, the user has the ability to choose their desired criteria and ISM will propose them with the most suitable instance type that matches his criteria.

At this time, ISM supports 4 criteria as following:

- **Cost Effectiveness:** select the instance type which gives minimum execution cost
- **Time Effectiveness:** select the instance type which gives the shortest execution time. In this case, the user priority is to finish the workload as soon as possible.
- **Ratio Cost to Time Effectiveness**: select the instance which gives minimum ration of execution cost and time. The ratio means how much a user needs to pay for an hour of workload execution.

- **Smallest Number Of Instances**: select the instance type which gives the smallest number of used instances to complete the task within a deadline. This criterion is suitable for a user who wants to reduce the instance management overhead.

The instance types sharing common usage patterns are normally grouped into several families ((i.e general purpose, computing-intensive, memory-intensive or storage-intensive)). ISM offers the ability to evaluate on all available instance types or restrict to only one specific instance family. This functionality is useful in a case the user already has general knowledge on the appropriate requirement of resources for his workload execution.

For example, in a big bio-informatics data processing application, the user is aware that his application is memory intensive by monitoring the RAM consumption during the execution. For that he preferably considers instance types in the memory-intensive family only. Theoretically, the instance types from other group, i.e computing intensive, could be working either but with limited internal memory, the execution can be not optimized[13].

4.1.4 User Interface Component (UI)

UI component provides users an intuitive and attractive way to interact with the Customer Tool. In order to achieve practical usage, UI component has following features:

- Simplify the selection of workload characteristics and cloud service configuration parameters. The UI is designed with the idea that users have little knowledge on the services offered by many cloud providers. To provide the workload characteristics, the UI allows the users to go through a set of question-based widgets and they will need to answer simple questions which require very basic knowledge on IT. More specialized parameters and terms, such as parallelization degree, random distribution etc. will be automatically calculated or explained clearly in the screen itself.
- The findings and suggestions from Customer Tool are presented to the users in an intuitive and consistent ways.
- The user is able to work with Customer Tool interactively. It means that the users can change any parameter at any time and the results will be reflected such change and presented to user almost instantly.
- Support user management functionality.
- Perform as much computation work on the client side as possible to reduce the load on server

UI is divided into four modules

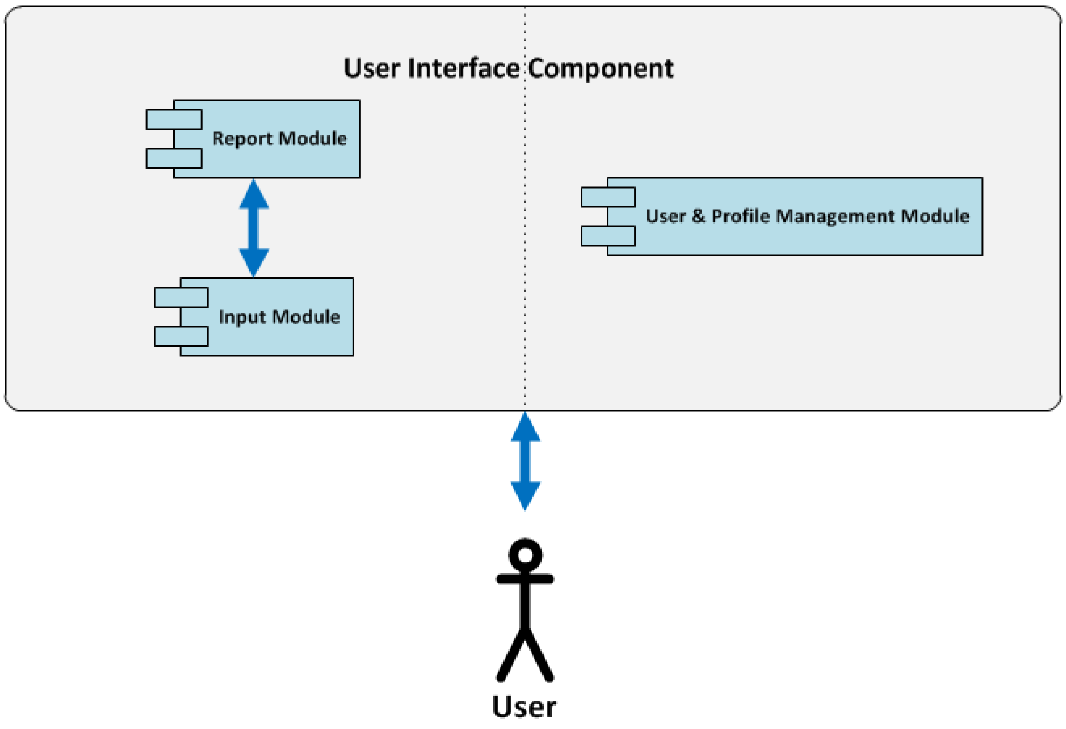


Figure 4.6: Modules in User Interface Component

Input Module

This module allows users to intuitively provide parameters relating to the cloud service configuration, workload characteristics and the QoS requirements.

Regarding the workload, the main functionality of this module is implemented in Task Detail dialog. By using Task Detail dialog, the users can provide information for a separate task in the workload. Task Detail consists of 5 tabs, each goes with its own information category [1].

- **General Information:** minimum requirements of instances that are capable of hosting that task. The requirements include CPU speed, number of cores, RAM and OS platform.
- **Job runtime:** estimated the runtimes of jobs in a task
- **Parallelization Degree:** proportion of each job that can be speedup when being run in multiple-core instances.
- **Data Transfer:** size of data needed to transfer out or in cloud environment for each job in task
- **Storage:** The storage is data that is used and shared by all jobs in workload. Using this widget, user is able select storage size, time usage, reliability and

[1]The screenshots will be presented in Chapter 5

durability requirements and based on that, widget will recommend suitable storage type. If user is aware of which storage type should meet his expectation, he also has the option to pick it up directly

- Constraints: Requirement on level of QoS from user including budget, deadline or reliability

At present, the jobs' runtimes, parallelization degrees and data transfer can have fixed values or fluctuate following uniform/normal distribution. If it is uniformly/normally distribution, Customer Tool has the functionality to support generating automatically and visualizing those values via a line chart in corresponding tab.

Report Module

With a set of the user's parameters, Report Module will be able to provide the user with sufficient information to make suitable decision. The outcomes from this part include

- Optimal option which minimizes the execution cost of the workload and satisfies the deadline constraint
- A report of options which can satisfy the deadline constraint
- A report of options which satisfy different deadlines in the region of the given deadline. For example, if the user expects his workload completing in 100 hours, this report will give the optimal execution costs for that workload with the deadlines, says from 50 to 150 hours. The user can change the deadline range for the report. The purpose of this report is to show how the cost level is changed following different level of QoS

The reports are displayed to the user both in grid-based or chart-based views. By the chart-based views, the trend of changes can be easily identified.

To achieve the desgin goal that the user can see any update in the outcomes as soon as there is a change in the input parameters without having to navigate back and forth different views, the reports are also presented in a floating, zoomable and draggable dialog.

The user is able to choose whether the storage and data transfer cost are taken into account to compute the execution cost and produce reports.

User Management Module

This module implements a couple of functionality including user authentication, user and profile management. The Customer Tool is accessible to a user if his account exists in database with valid credentials. The user is able to create, edit his account and reset password in case it is lost by using User Management module.

A user profile contains the information such as user's preferences of data center, cloud provider, login to public cloud providers and so forth.

In order to avoid tedious work of repeatedly typing in similar information to Customer Tool, this module enable the user to save frequently-used task or workload templates to the database.

4.2 Technical Stack

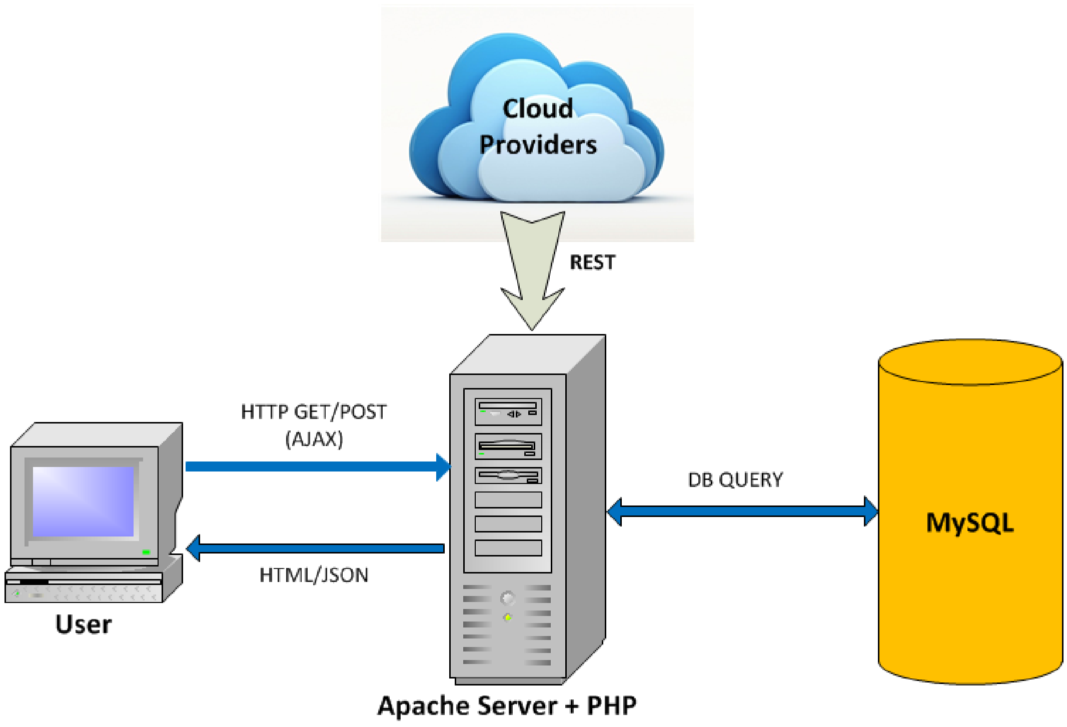


Figure 4.7: Customer Tool Deployment

In the database layer, we choose MySQL for a couple of reasons. It is a light-weight, open-source database which is a popular choice to use in many web applications. Furthermore, MySQL is also supported and easily deployed in multiple platform, from Windows to Linux.

The Apache HTTP Server has been the dominant web server on the public Internet. Apache is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation. A wide variety of features is supported, and many of them are implemented as compiled modules which extend the core functionality of Apache. These can range from server-side programming language support to authentication schemes [9].

Regarding application layer, PHP has been selected. PHP is a popular language for web development with various number of tools and libraries. During its evolution, PHP has gradually set a standard of global acceptance and web-applications written in PHP can be deployed on most of servers. Another important reason is that popular cloud providers always offer a set of APIs to interact with their services on PHP.

The User Interface layer is implemented in JavaScript using some popular frameworks including jQuery for client logic & interaction, jQuery UI and Twitter Bootstrap for predefined look&feel, jqGrid for rich-features data grid widgets and Google Chart for drawing chart.

- PHP serves the HTML pages
- PHP retrieves service configuration from MySQL database, performs computation and returns results to the user
- The Client Web Browser renders the HTML pages and performs client actions through Java Script technology.
- AJAX (Asynchronous Javascript XML) is used to avoid full refresh of the page. Its also used for pure data exchanges, based on JSON (Javascript Object Notation).
- REST APIs are used to communicate with cloud providers

The technical stack consisting of Linux, the Apache HTTP Server, the MySQL relational database management system, and the PHP programming language is known as LAMP model. LAMP model has been proved to be suitable for building dynamic web applications for a long time.

CHAPTER 5

Experiments and Evaluation

5.1 Experimental Environment

Customer Tool is deployed in Amazon Web Services environment. AWS has been choosen because as compared to the other service providers, it is currently the most popular, feature-rich and multi-platform supported commercial cloud. Two t2.micro instances are used to set up Customer Tool. One instance hosts MySQL database while Apache web server and web application are running on the other. Both the instances' images are based on the standard 64-bit Amazon Linux AMI images. The Amazon Linux AMI is a supported and maintained Linux image provided by Amazon Web Services for use on Amazon Elastic Compute Cloud (Amazon EC2). It provides a stable, secure, and high performance execution environment for applications running on Amazon EC2. Additionally, the Amazon Linux AMI also includes packages that enable easy integration with AWS, including launch configuration tools and many popular AWS libraries and tools.However we still need to install some miscellaneous packages to run and manage the Customer Tool.

The t2.micro instance has following hardware configuration: variable ECU, 1 GiB memory, EBS Storage only.

The database is initially populated with sufficient information from AWS such as instance types, storage types, pricing model etc. In future, it is able to extend Customer Tool to support the other service providers by implementing an new adapter module in PHP and populating information of wanted service providers into the database.

5.2 Use Cases

In order to demonstrate the usage of Customer Tool, we introduce an example of an user who needs to analyse a big bioinformatics data file for his study. According to user's analysis strategy, the data file is split into a number of small data chunks and each data chunk will be assigned to run in one machine. The machine has to meet a minimum configuration such as CPU speed, RAM and number of cores. The data chunks are analysed using the same algorithm and when the results from all data chunks are ready they are merged into a consolidated report. There is no precedence relationship among those data chunks and the processing of a each one is independant to the others.

We also assume that by repeatedly performing the analysis on the same type of bioinformatics data, the user has good estimation on the execution time for a data chunk, the propotion of job that can be speeded up using multi-core machines, the data transfer and data storage needed.

To align the application with our model, the bioinformatics data file can be considered as a task in the workload and the small chunks of that data file are jobs in the task.

5.2.1 Use Case 1

In the first scenario, the user only uses on-demand instances and does not purchase any reserved instance. The estimated runtimes, parallelization degrees are considered to be fixed.

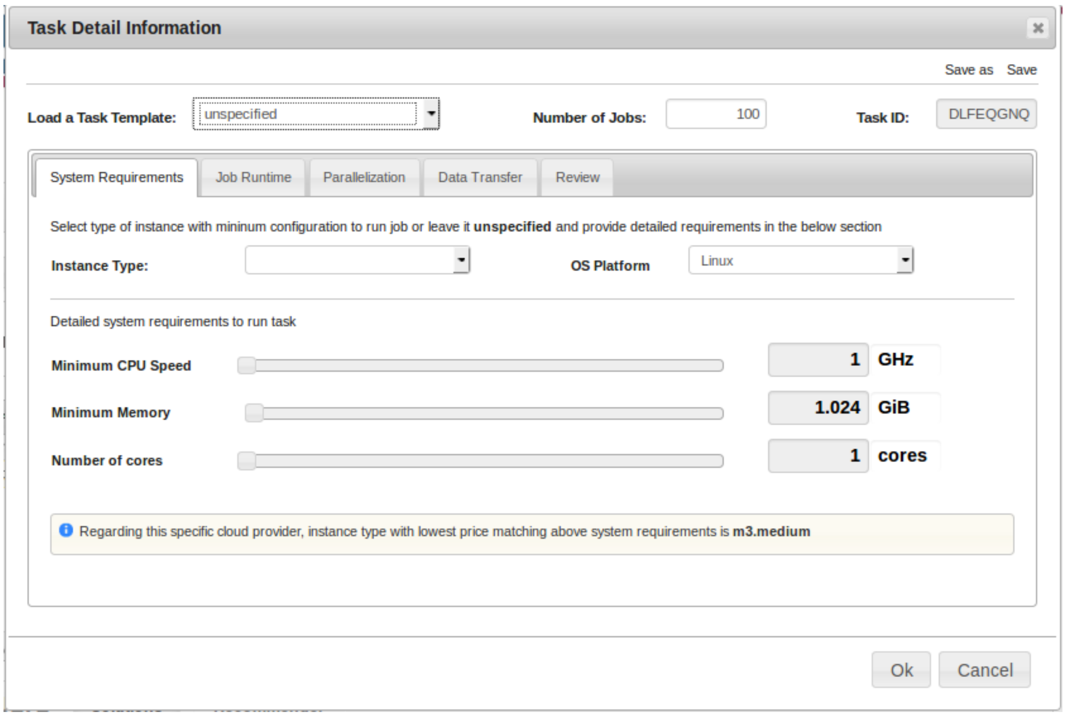
Before executing the workload, the user needs to ensure there is no change in AWS products. Customer Tool provides a funcationality which is able to communicate directly with AWS via REST to retrieve and update the AWS's latest information into the database. The information includes instance types' characteristics, instance prices, storage prices. However, in order to make API call to the AWS, the user should provide his valid AWS credentials in the Customer Tool settings.

Table 5.1 summarizes information passed to Cloud Tool

Table 5.1: User Workload Characteristics

Description	**Value**	**Note**
Number of Tasks	1	
Number of Jobs in Task	100	
Jobs' Runtime	100	
Jobs' parallelization degrees	50%	
Data Transfer In	0GB	
Data Transfer Out	0GB	
Data Storage	50GB	
Required Deadline	80 hours	

For the first step, the user should choose the prefered cloud provider and region (if available) in the homepage. Based on that, the corresponding currency will be


Task Detail Information

Save as Save

Load a Task Template: unspecified

Number of Jobs: 100

Task ID: DLFEQGNQ

System Requirements | Job Runtime | Parallelization | Data Transfer | Review

Select type of instance with mininum configuration to run job or leave it **unspecified** and provide detailed requirements in the below section

Instance Type:

OS Platform: Linux

Detailed system requirements to run task

Minimum CPU Speed: 1 GHz

Minimum Memory: 1.024 GiB

Number of cores: 1 cores

Regarding this specific cloud provider, instance type with lowest price matching above system requirements is **m3.medium**

Ok Cancel

Figure 5.1: Task Detail Information: System Requirement Tab

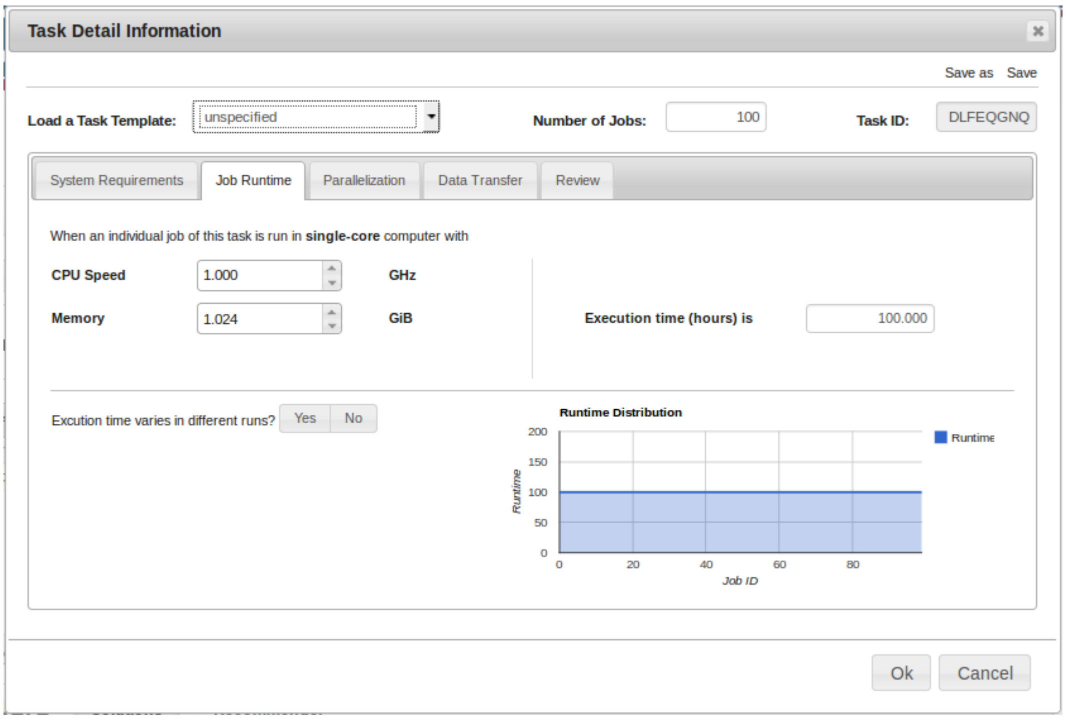
retrieved from database and apply for the computation afterward. Customer Tool does not support the combination of instance types from different regions or from different cloud providers. We currently choose to use all services in the same region.

The user easily specifies necessary information about the workload and the requirement on level of QoS by using UIs widgets. The Customer Tool is designed with the assumption that its users have little or no knowledge on any specific cloud product. Therefore, the UI widgets will guide the user through a number of simple steps to ease the input process. The user do not need to do any extra computation here. For example, if the user does know how the parallelization degree is defined, in the user interfaces, by providing the job runtimes on 2 different instance types with different number of cores, the system will automatically compute the parallelization degree of that job for the user as in 5.3.

Figure 5.1 shows the UI widget in which the user defines the minimum requirements of an instance type capable of executing a job in the workload. It is worth mentioning that whenever the user changes any parameter in this screen, the Customer Tool will propose eligible instance type with the lowest on-demand price in the **Instance Type** dropdown.

As soon as information is completely filled in it will be sent to sever to process and give feedback to users.

The results are showed in Figure 5.4. It can be seen that the minimum cost to complete provided workload before the deadline is 234.50 USD. This cost can be



Task Detail Information

Save as Save

Load a Task Template: unspecified

Number of Jobs: 100

Task ID: DLFEQGNQ

System Requirements | Job Runtime | Parallelization | Data Transfer | Review

When an individual job of this task is run in **single-core** computer with

CPU Speed 1.000 GHz

Memory 1.024 GiB

Execution time (hours) is 100.000

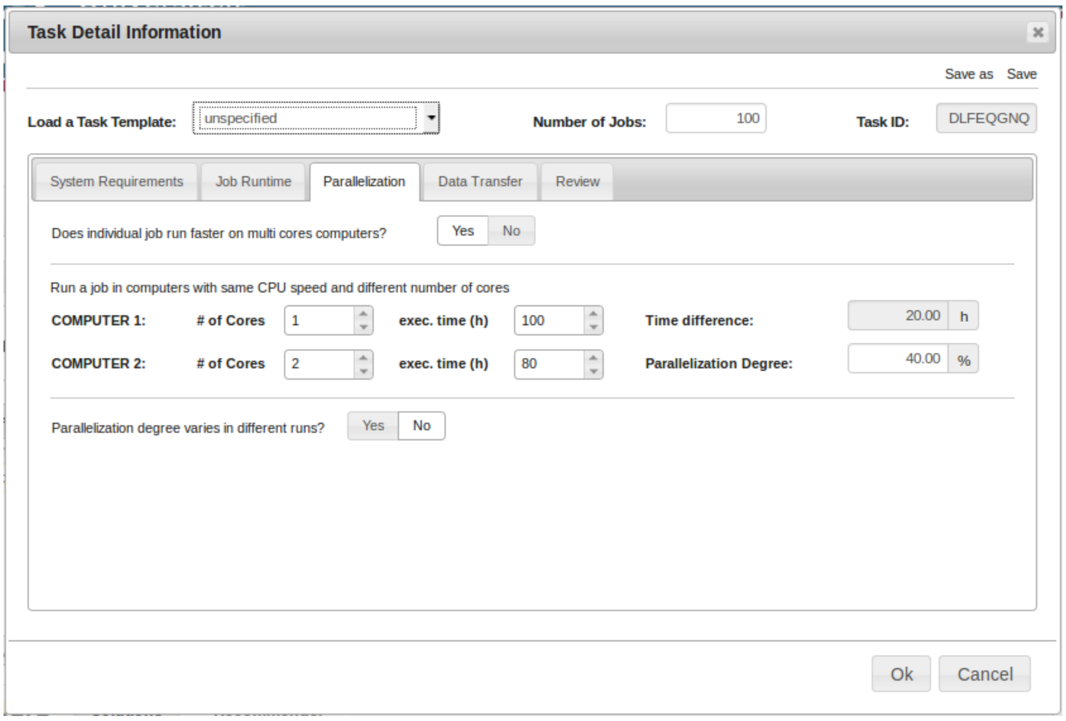
Excution time varies in different runs? Yes No

Figure 5.2: Task Detail Information: Runtime Tab

achieved if we use 50 m3.medium instances in 67 hours. Obviously, the real execution time of the workload is 67 hours, which is 13 hours shorter than the required deadline of 80 hours.

The chart view in Figure 5.6 shows how the execution cost changes when different instance type is used. The m3.medium, which gains the optimal execution cost, is the cheapest instance type in AWS that satisfies the mininum requirement of the workload. The common trend is that the execution cost will increase if the more expensive and higher capacity instance types are used. If the workload is completed by hs1.8xlarge, the cost will go up significantly to 13.340 USD but as a trade-off, the real execution time is only 58 hours. The hourly price of hs1.8xlarge is 4.6 USD with Linux platform, which is the highest price among all available instance types.

As mentioned earlier, in some cases, monetary cost is not the most important factor in the user's decision. The recommendation tab shows how the minimum cost and execution time change with different level of QoS, or the deadline constraint in this case, both in a grid view and a chart view. Based on this tab, the user has sufficient information to make decision of either saving cost but completing his workload in a longer time or decreasing the execution time but spending more. From the Figure 5.7, it can be noticed that the execution cost does not always go up remarkly when the deadline is shortened. The execution cost with the deadline from 40 to 180 hours stays almost the same, around 235 USD and m3.medium is used in all such options. e In order to have the execution of many jobs in a task,


Task Detail Information

Save as Save

Load a Task Template: unspecified

Number of Jobs: 100

Task ID: DLFEQGNQ

System Requirements | Job Runtime | Parallelization | Data Transfer | Review

Does individual job run faster on multi cores computers? [✓ Yes] No

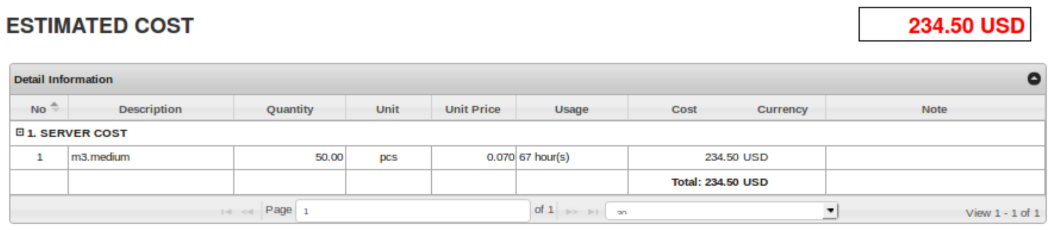
Run a job in computers with same CPU speed and different number of cores

COMPUTER 1:	# of Cores	1	exec. time (h)	100	Time difference:	20.00	h
COMPUTER 2:	# of Cores	2	exec. time (h)	80	Parallelization Degree:	40.00	%

Parallelization degree varies in different runs? Yes [✓ No]

Ok Cancel

Figure 5.3: Task Detail Information: Parallelization Tab


ESTIMATED COST 234.50 USD

Detail Information

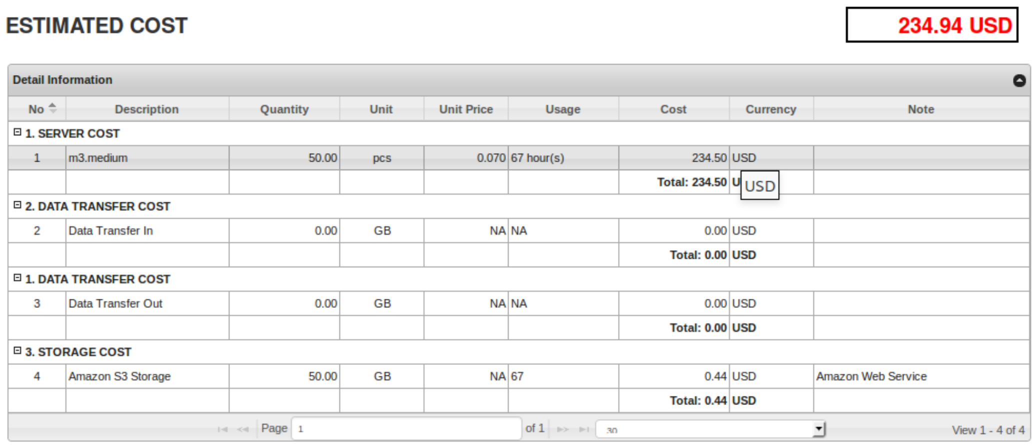
No	Description	Quantity	Unit	Unit Price	Usage	Cost	Currency	Note
1. SERVER COST								
1	m3.medium	50.00	pcs	0.070	67 hour(s)	234.50 USD		
						Total: 234.50 USD		

Page 1 of 1 View 1 - 1 of 1

Figure 5.4: Detail on Optimal Option and Cost in Grid View

we need to increase the number of instances used. With a job runtime of 100 hours on 1GHz/1 core/1.024GB RAM machine, by using Amdalh's law, that job can be speeded and completed only in approximately 34 hours if being run in m3.medium (m3.medium configuration is 3ECU CPU/1 core/3.75GiB RAM). Therefore we can always use m3.medium to complete a job in the task with the deadline greater than 40 hours. In addition, the cost to use one instance in n hours is equal to the cost to use n instances of the same type in 1 hour. It explains why the execution cost does not change much in the range of the deadline from 40 hours onward.

When the deadline goes down to 20 hours, the execution cost increases to 420 USD because with this deadline, m3.medium is no longer capable anymore and c3.xlarge (with configuration 3.5ECU CPU/4 cores/7.5GiB RAM) should be used



ESTIMATED COST

234.94 USD

Detail Information

No	Description	Quantity	Unit	Unit Price	Usage	Cost	Currency	Note
⊟ 1. SERVER COST								
1	m3.medium	50.00	pcs	0.070	67 hour(s)	234.50	USD	
						Total: 234.50	**U** USD	
⊟ 2. DATA TRANSFER COST								
2	Data Transfer In	0.00	GB	NA	NA	0.00	USD	
						Total: 0.00	**USD**	
⊟ 1. DATA TRANSFER COST								
3	Data Transfer Out	0.00	GB	NA	NA	0.00	USD	
						Total: 0.00	**USD**	
⊟ 3. STORAGE COST								
4	Amazon S3 Storage	50.00	GB	NA	67	0.44	USD	Amazon Web Service
						Total: 0.44	**USD**	

Page 1 of 1 30 View 1 - 4 of 4

Figure 5.5: Detail on Optimal Option in Grid View Including Data and Network Cost

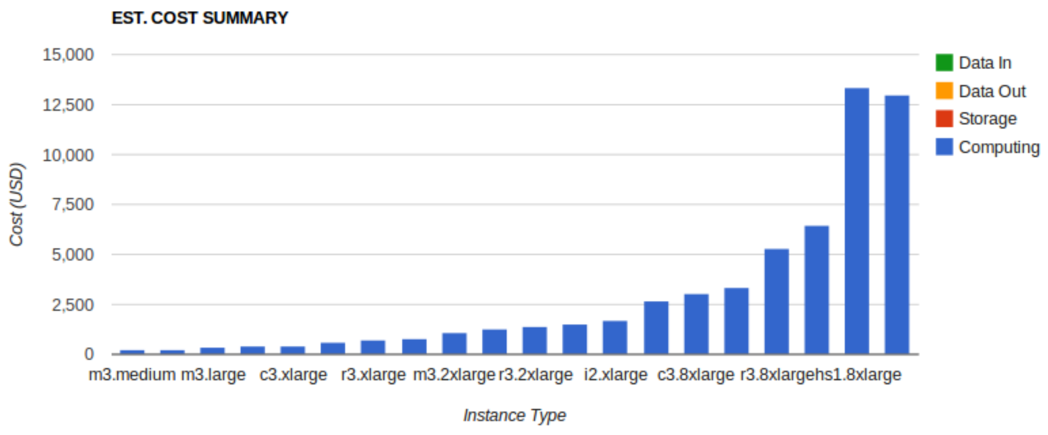


Figure 5.6: Detail on Options and Cost in Chart View

instead. The graph in Figure 5.7 also shows that it is impossible to complete the workload with the deadline lower than 20 hours.

5.2.2 Use Case 2

In this scenario, it is assumed that during the execution each job will need to transfer an amount of data in/out the cloud environment. The user's application also require to store the informations data file in the cloud storage which will be shared among all the jobs.

Data transfer in/out for each job can be provided through Task Detail dialog like job runtimes or paralellization degrees. Regarding the shared data file, the user has 2 options. The user can select a specific data storage type from the dropdown box

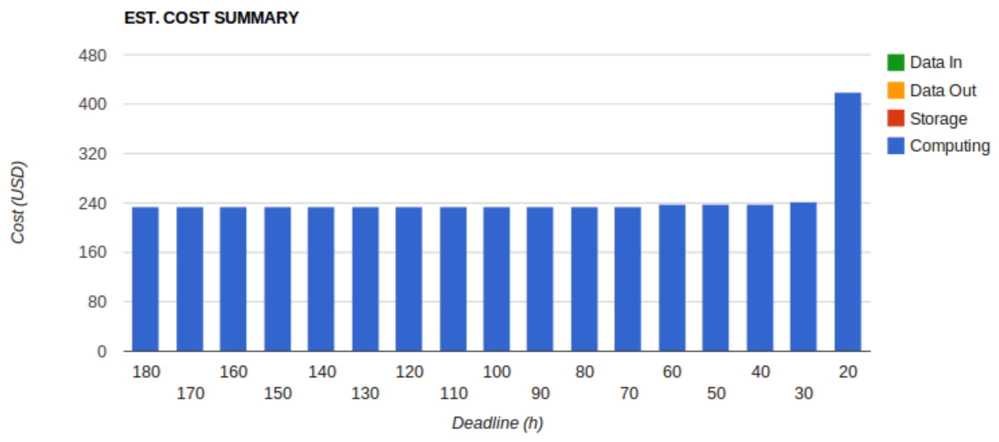


Figure 5.7: Recommendation Chart

or he can specify requirements on Data Reliability and I/O request. In the latter case, Customer Tool will automatically pick up the most suitable storage type.

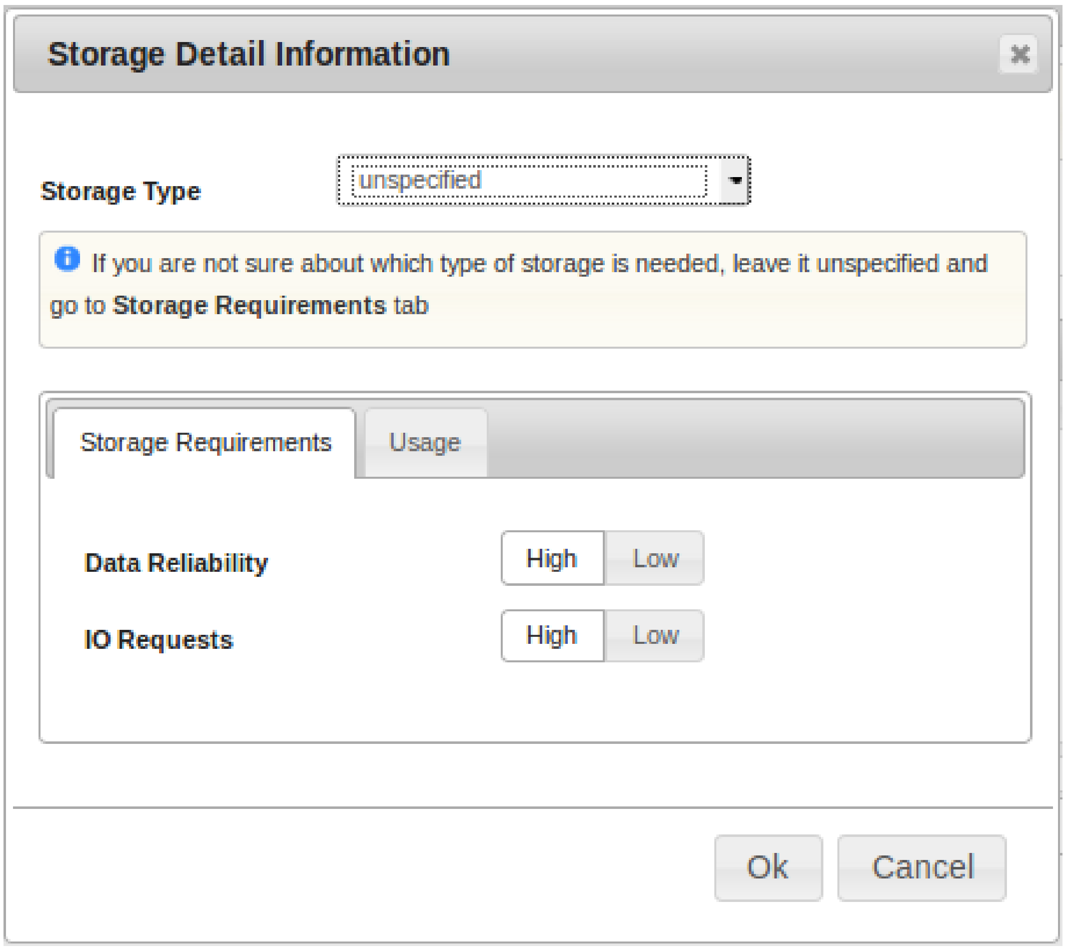
For example, if the user choose both Data Reliability and IO Request requirements to High as in 5.8, Standard Storage will be picked up (Figure 5.10)

In this case, beside the information as in Use Case 1, the results will add both the storage cost of shared data and the transfer cost (or network cost). However, the users have option to include/exclude the storage and network cost into the execution cost.

In the Figure 5.5, the storage cost and network cost are shown to user in separate sections. Regarding the storage cost, AWS uses fixed monthly charges and the price varies depending on storage volume. In the common cases, the bigger the storage volume is used, the cheaper the rate is. In order to be aligned with computing cost, the Customer Tool calculates the real cost users have to pay for storage during the time of workload execution with the assumption that a month has 30 days.

5.2.3 Use Case 3

In this case, it is assumed that the user already purchased a number of reserved instances and prefer to use those instances for the application. With Customer Tool, the user does not need to memorize the reserved instances they bought. Customer Tool provides a functionality to retrieve reserved instances information programmatically via REST calls to AWS. As shown in Figure 5.11 The retrieved information includes instance types, number of instances, fixed price, unit price, leftover before expiration and so forth. The user is able to specify how many reserved instances from the available ones that he wants to execute the workload.



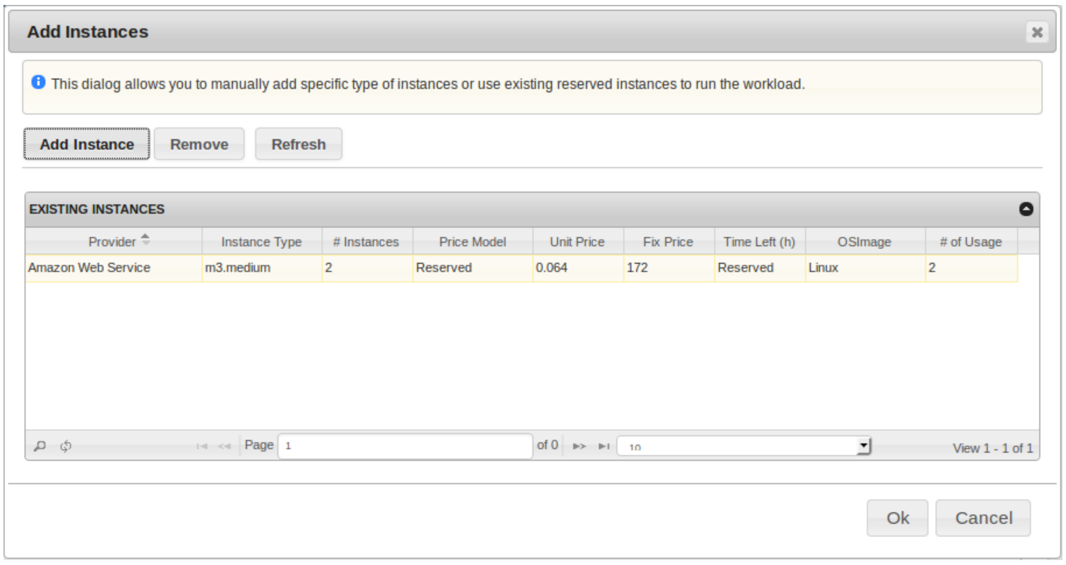
Storage Detail Information

Storage Type: unspecified

If you are not sure about which type of storage is needed, leave it unspecified and go to **Storage Requirements** tab

Data Reliability	High	Low
IO Requests	High	Low

Figure 5.8: Storage Requirement Tab



Add Instances

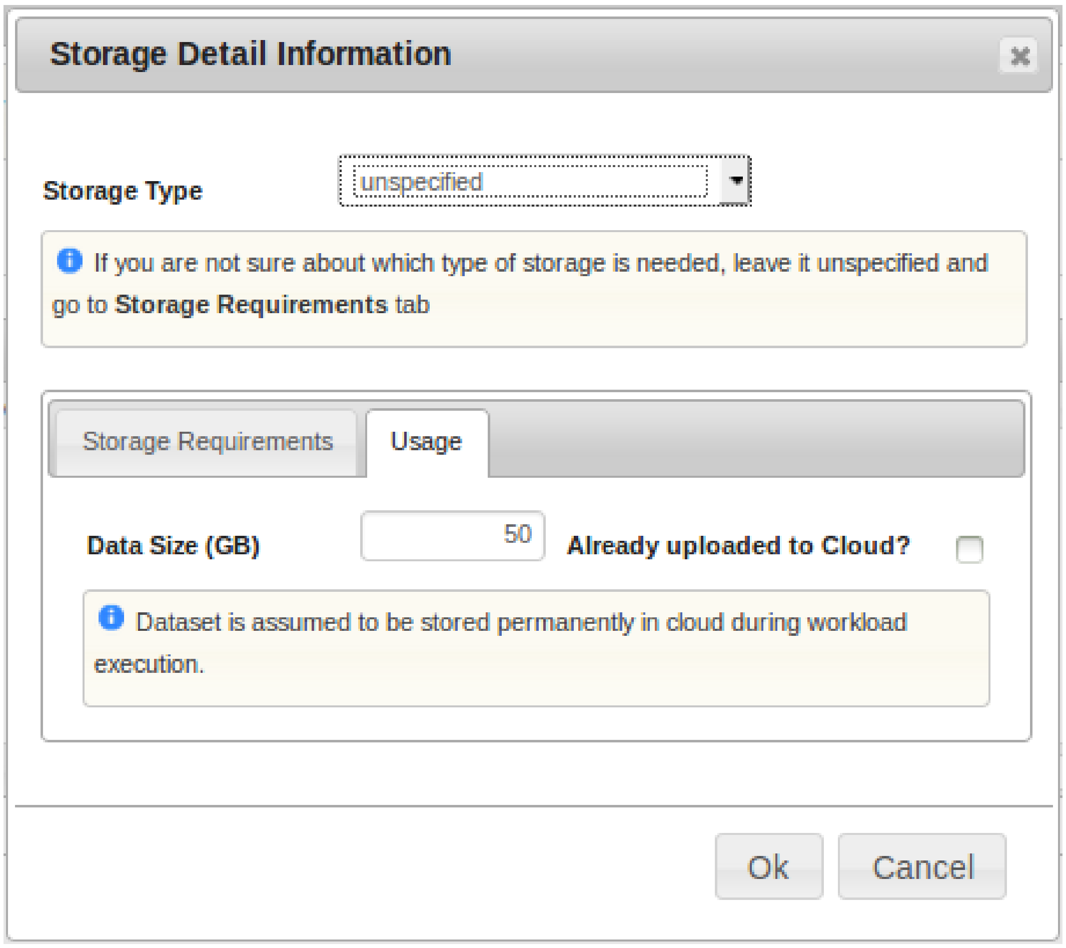
This dialog allows you to manually add specific type of instances or use existing reserved instances to run the workload.

EXISTING INSTANCES

Provider	Instance Type	# Instances	Price Model	Unit Price	Fix Price	Time Left (h)	OSImage	# of Usage
Amazon Web Service	m3.medium	2	Reserved	0.064	172	Reserved	Linux	2

Page 1 of 0 10 View 1 - 1 of 1

Figure 5.11: Add Instances Dialog


Storage Detail Information

Storage Type: unspecified

If you are not sure about which type of storage is needed, leave it unspecified and go to **Storage Requirements** tab

Data Size (GB): 50 Already uploaded to Cloud? ☐

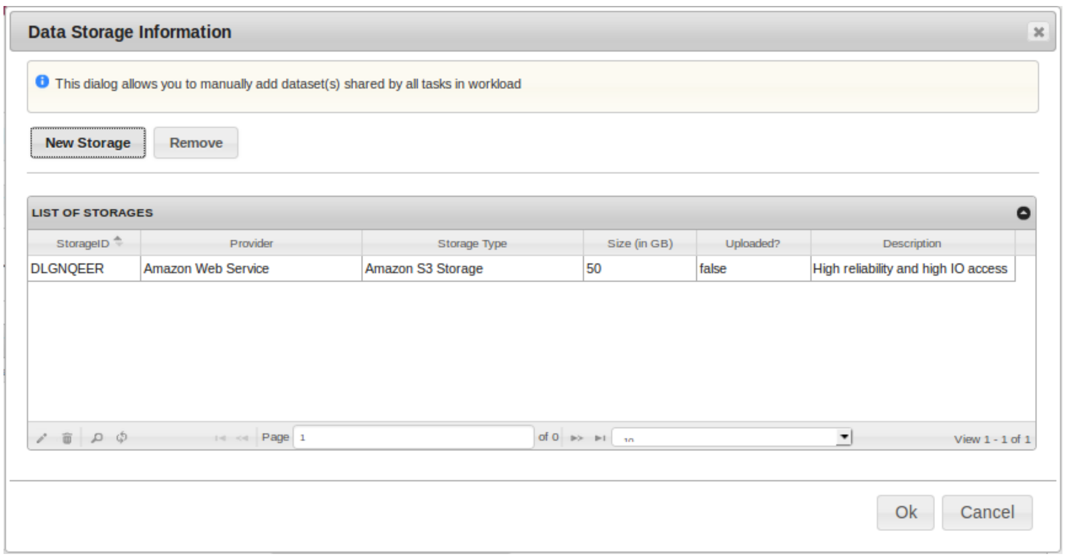
Dataset is assumed to be stored permanently in cloud during workload execution.

Figure 5.9: Storage Usage Tab

5.2.4 Use Case 4

With a given task, the user may want to find out which instance type is the most suitable to his criteria such as cost effectiveness, time effectiveness etc. The instance selection widget will helps the user with such work. In Figure 5.12, the user can choose his desired criteria and the suitable instance type will be proposed together with the corresponding values, i.e execution cost, execution time or number of instances.

Additionally, Instance Selection gives the user 4 bar-charts showing how the corresponding values change when using different instance types. The user is able to choose the instance type family (i.e general purpose, computing-intensive, storage-intensive...) in order to restrict the number of instance types taken into consideration.



Data Storage Information

This dialog allows you to manually add dataset(s) shared by all tasks in workload

New Storage | Remove

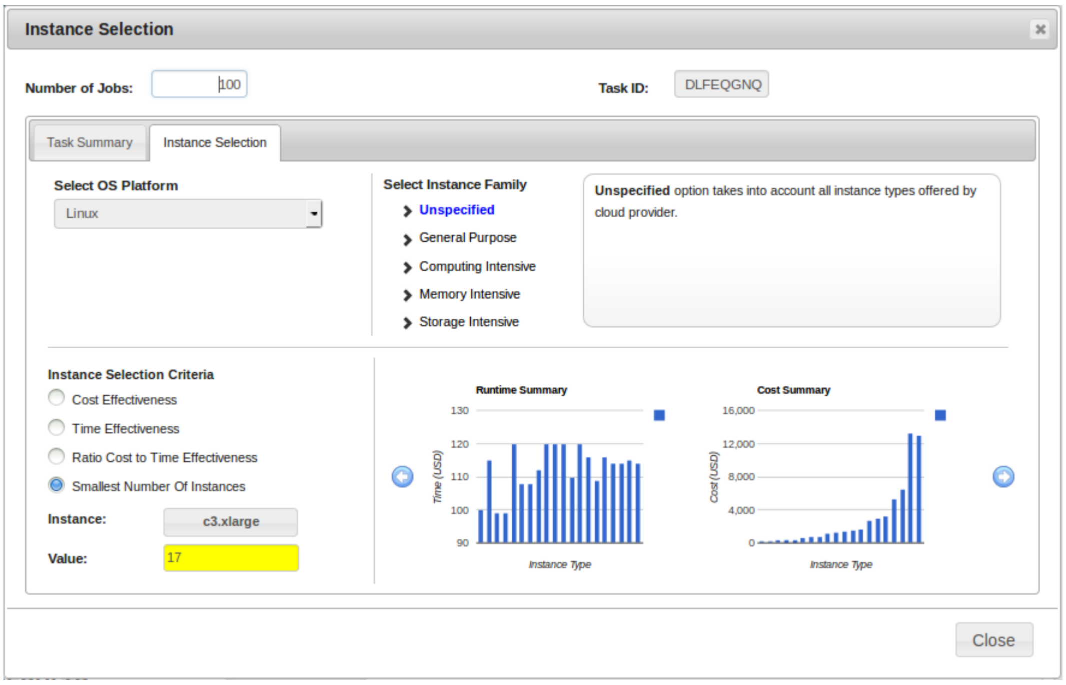
LIST OF STORAGES

StorageID	Provider	Storage Type	Size (in GB)	Uploaded?	Description
DLGNQEER	Amazon Web Service	Amazon S3 Storage	50	false	High reliability and high IO access

Page 1 of 0 10 View 1 - 1 of 1

Ok Cancel

Figure 5.10: Data Storage Information Dialog



Instance Selection

Number of Jobs: 100 Task ID: DLFEQGNQ

Task Summary | Instance Selection

Select OS Platform: Linux

Select Instance Family
- Unspecified
- General Purpose
- Computing Intensive
- Memory Intensive
- Storage Intensive

Unspecified option takes into account all instance types offered by cloud provider.

Instance Selection Criteria
- Cost Effectiveness
- Time Effectiveness
- Ratio Cost to Time Effectiveness
- [✓ Smallest Number Of Instances]

Instance: c3.xlarge

Value: 17

Close

Figure 5.12: Recommendation Chart

CHAPTER 6

Related Work

Many studies has discussed the scientific workloads on cloud environment and how to optimize the workload execution cost while still meeting QoS requirements.

Gideon Juve et al. [18] examine the performance and cost of clouds from the perspective of scientific workflow applications. They use three characteristic workflows to compare the performance of a commercial cloud with other systems, and they analyze the various costs associated with running those workflows in the cloud. In another paper [19], they gives an explanation of an approach that sets up a computational environment in the cloud to support the execution of scientific workflow applications and describe the issues and problems they encountered deploying them on EC2.

In [25], Ming Mao et al. proposed a cloud auto-scaling mechanism to automatically scale computing instances based on workload information and performance desire. The mechanism takes into consideration the delay on instance startup and shut-down activities. [25] also discussed the problem of distributing existing resources to tasks and jobs in the workload. In this paper, the auto-scaling mechanism is reduced to solving several integer programming problems. [24] extends the mechanism to cloud workflows.

There are several papers [5][1][6][30][33] discussing resource allocation and instance consolidation strategies for grid/cloud data centers. In general, the goal is to minimize the workload execution cost while still maintaining QoS requirements.

There have been a variety of systems that assists users in selecting resource configuration in cloud computing.

In [29], Arkaitz and Marty propose an automated approach to selecting the cloud storage service that best matches each dataset of a given application. Their approach relies on a machine read-able description of the capabilities (features, performance, cost, etc.) of each storage system, which is processed together with the users specified requirements.

Paper [3] presents the design and prototype implementation of the Data-centric

Management Framework (DMF) with data models, query languages and sematics that are specially designed to orchestrate infrastructure services. The goal of this work is to improve SLA fulfilment ability of cloud service providers.

In [34], the authors develop a declarative approach for selecting cloud-based infrastructure services. They implemented a recommender system which automates the mapping of users specified requirements to cloud service configurations using a structured data model which can be manipulated through both regular expressions and SQL.

In our Customer Tool, we attempt to take into account not only the resource requirements (i.e CPU speed, RAM, data storage) but also the characteristics of the workloads (i.e job runtimes, paralellization degree etc.). Additionally, the information on how the cost level is expected to change with different level of QoS is provided in an intuitive way to assist users in making decision.

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

Cloud computing is a rapidly emerging paradigm with significant promise to scientific workload execution. However, with pay-per-use pricing from cloud providers, one of the biggest concern when moving to cloud environment is to minimize workload execution cost while still achieving a certain level of QoS. QoS in scientific workload usually relates to deadline and budget constraints. In this thesis we built up a Decision Support System which may help users in answering that concern and assists users in making decision.

We firstly give general introduction to cloud computing and then go deeper into product portfolio and pricing model of Amazon Web Services [13], which is considered to be the key player in the cloud market at present.

We develop an algorithm to estimate minimum monetary cost that a workload execution will generate with constraints on deadline or budget. The algorithm is adjusted in order to adapt to different scenarios such as including reserved instances, different assumptions in jobs' characteristics etc.

In order to reduce the complexity, we do not use an integer programming solver (which is commonly used in other similar studies) to find out the cost-effective option but iterate through all available instance types in a provider to get the most suitable one. This approach is feasible as the number of instance types offered by a cloud provider are limited.

To realize aforementioned algorithms into practice, we have designed and implemented a web-based decision support system, called Customer Tool. Given the characteristics of a workload and QoS requirement, Customer Tool is able to provide users with extensive information on the cost that workload execution will generate and how the cost level may change with different level of QoS requirements. Such information assists users in making decision on service configurations that are aligned with their desire and objectives.

During implementation, we formalize heterogeneous and non-standardized services from different cloud providers into a general model. This model, together with open design, makes it easy to extend Customer Tool support to other service providers, not only AWS.

We demonstrate the realistic usage of Customer Tool through a couple of scenarios. The results and reports are able to provide users with useful information and the accuracy of the estimated cost is acceptable.

7.2 Future Work

For future work, it would be interesting to use a linear programming solver to compute optimal option and by that allow jobs in a task can be executed by different instance types. We also wants to develop new modules and update the database to support other cloud providers such as Windows Azure, Rackspace or GoGrid.

As clouds have become attractive computing platform for a wide range of applications, the Customer Tool can be extended so that it not only supports scientific workload applications but also other types of applications including web applications, database etc. In order to do that, we need to study characteristics of each application type, corresponding QoS requirements and the algorithm to find out optimal option and integrate those into Customer Tool.

In addition, we plan to optimize the PHP implementation on servers. Computation work should be migrated to client side to reduce server load. A PHP framework such as Zend Framework or Symfony can be used to make the system extension and maintenance easier.

Bibliography

[1] Andrzejak, Artur and Kondo, Derrick and Yi, Sangho. Decision Model for Cloud Computing Under SLA Constraints. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '10, pages 257–266, Washington, DC, USA, 2010. IEEE Computer Society.

[2] Armando Escalante Borko Furht. *Handbook of Cloud Computing*. Springer.

[3] Changbin Liu and Jacobus E. Van Der Merwe and et al. Cloud Resource Orchestration: A Data-Centric Approach , 2011.

[4] Darryl Chantry. Mapping applications to the cloud, 2009.

[5] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-Efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference*, pages 228–235. IEEE, 2010.

[6] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads . In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference*, pages 320–327. IEEE, 2011.

[7] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Online Cost-efficient Scheduling of Deadline-constrained Workloads on Hybrid Clouds. *Future Generation Computer Systems*, 29:973–985, 2013.

[8] Raicu I. Foster I., Zhao Y. and Lu S. Cloud computing and grid computing 360-degree compared. In *Proceeding of the Grid Computing Environment Workshop, 2008. GCE*, pages 1–10. IEEE, Nov. 2008.

[9] The Apache Software Foundation. Apache HTTP Server Project.

[10] Archana Ganapathi, Yanpei Chen, Armando Fox, Randy Katz, and David Patterson. Statistics-driven Workload Modeling for the Cloud . In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference*, pages 87–92. IEEE, 2010.

[11] GoGrid. GoGrid Website - http://gogrid.com.

[12] Marc Oriol Hilari. Quality of service (qos) in soa systems. a systematic review. Master thesis, Universitat Politecnica de Catalunya.

[13] Amazon Web Services Inc. Amazon Web Services - http://aws.amazon.com.

[14] Microsoft Inc. Microsoft Azure - http://azure.microsoft.com.

[15] Rackspace US Inc. Rackspace Website - http://www.rackspace.com.

[16] A. Iosup, N. Yigitbasi, and D. Epema. On the Performance Variability of Production Cloud Services. In *Proceedings of the 11th IEEE/ACM Int. Symposium on Cluster, Cloud and Grid Computing*, pages 104–113. IEEE Computer Society, 2011.

[17] B. Javadi, R.K. Thulasiram, and R. Buyya. Statistical Modeling of Spot Instance Prices in Public Cloud Environments. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference*, pages 219–228. Springer-Verlag Berlin, Heidelberg, 2012.

[18] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Benjamin P. Berman, Bruce Berriman, and Phil Maechling. Scientific Workflow Applications on Amazon EC2. In *E-Science Workshops, 2009 5th IEEE International Conference*, pages 59–66. IEEE, 2009.

[19] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, and Bruce Berriman. Data sharing options for scientific workflows on amazon ec2. In *SC10 Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis.*

[20] Przemyslaw Klosiewicz. Shared Memory Parallel Computing. Cluster Computing Lecture, University of Antwerp, 2012.

[21] Kurt Vanmechelen. The Cloud Computing Paradigm. Topic in Distributed Computing Lecture, University of Antwerp, 2013.

[22] C. B. Lee and A. Snavely. On the user-scheduler dialogue: Studies of user-provided runtime estimates and utility functions. *International Journal of High Performance Computing Applications*, 4:495–506, 2006.

[23] Sparx IT Solutions Private Limited. A Complete Reference to Cloud Computing.

[24] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 49:1–49:12, New York, NY, USA, 2011. ACM.

[25] Ming Mao, Jie Li, and Marty Humphrey. Cloud Auto-scaling with Deadline and Budget Constraints. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference*, pages 41–48. IEEE, 2010.

[26] G. Mc Evoy and B. Schulze. Understanding scheduling implications for scientific applications in clouds. In *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '11, pages 3:1–3:6, New York, NY, USA, 2011. ACM.

[27] Rizwan Mian, Patrick Martin, Farhana Zulkernine, and Jose Luis Vazquez-Poletti. Estimating resource costs of data-intensive workloads in public clouds. In *Proceedings of the 10th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '12, pages 3:1–3:6, New York, NY, USA, 2012. ACM.

[28] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing, 2011.

[29] Arkaitz Ruiz-Alvarez and Marty Humphrey. An automated approach to cloud storage service selection. In *Proceedings of the 2Nd International Workshop on Scientific Cloud Computing*, ScienceCloud '11, pages 39–48, New York, NY, USA, 2011. ACM.

[30] Arkaitz Ruiz-Alvarez and Marty Humphrey. A Model and Decision Procedure for Data Storage in Cloud Computing. In *Proceeding CCGRID 12 Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 572–579. IEEE Computer Society, 2012.

[31] Bhanu Sharma, Ruppa K. Thulasiram, Rajkumar Buyya, Parimala Thulasiraman, and Saurabh K. Garg. Pricing Cloud Compute Commodities: A Novel Financial Economic Model. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium*, pages 451–457. IEEE, 2012.

[32] Barrie Sosinsky. *Cloud Computing Bible.* Wiley Publishing, Inc.

[33] Jia Yu, Rajkumar Buyya, and Chen Khong Tham. Cost-based scheduling of scientific workflow application on utility grids. In *Proceedings of the First International Conference on e-Science and Grid Computing*, E-SCIENCE '05, pages 140–147, Washington, DC, USA, 2005. IEEE Computer Society.

[34] Miranda Zhang, Rajiv Ranjan, Surya Nepal, Micheal Menzel, and Armin Haller. A Declarative Recommender System for Cloud Infrastructure Services Selection. In *Proceedings of the 9th international conference on Economics of Grids, Clouds, Systems, and Services*, pages 102–113. IEEE, 2011.