Cloud Computing for Digital Libraries

Lebeko Bernard Nkoebele Poulo

Supervised by: A/Professor Hussein Suleman

Thesis presented for the degree of Master of Science

Department of Computer Science

University of Cape Town

May 2013

Plagiarism Declaration

I know the meaning of plagiarism and declare that all of the work in the document, save for that which is properly acknowledged, is my own.

Lebeko Bernard Nkoebele Poulo

To 'm'e 'Makutloano Poulo *and* Kali Wilson Poulo Lebeko

...

In memory of ntate Taelo Poulo

Acknowledgements

Special thanks go to my supervisor for the patience he had until I completed this research. You have been amazingly supportive, encouraging and above all, eye-opening. Thanks a million, Hussein. Thanks to the Digital Libraries Research Group for all the interesting paper discussions we had, sharing of ideas and all the company that made my stay at the University of Cape Town pleasant and memorable.

Many thanks to my family; you are too many to enumerate here. To my brother, TEKO, you have truly surprised me, may you be forever blessed with the very best and many, many thanks. Todd Dincher-Poulo, I am grateful for your love, support and encouragement during the toughest moments when I thought all had collapsed in my face. Keneuoe Letšela, you have been a strong and supportive friend and a sister to me, thank you so much, you really have no idea how much your friendship means to me. To my mother, 'm'e 'Makutloano, words can not express how grateful I am to have you. You were with me every step of the way and many, many thanks, *morali'a Baphuthi*. To my nephew, Kadi, you know it's always about you. Thank you for the love and for always believing in me. I love you all.

Gina Paihama, Martha Kamkuemah and Morwan Mohamed Nour, you guys have been an amazingly wonderful family I never had. Thank you for being there when I needed you most. I feel like I have known you all my life. Neann Mathai, thanks for the insight into some of the technical development aspects; so much could have gone wrong, thanks once again. Thanks to all the lab-mates who made my stay in the department worth remembering.

The financial assistance from the Lesotho Government through the NMDS towards partial completion of this research and that of `Amazon.com` through AWS Education towards executing my experiments is hereby acknowledged. However, opinions expressed, and conclusions arrived at, are those of the author and are not to be attributed to NMDS and/or `Amazon.com`.

Abstract

Information management systems (digital libraries/repositories, learning management systems, content management systems) provide key technologies for the storage, preservation and dissemination of knowledge in its various forms, such as research documents, theses and dissertations, cultural heritage documents and audio files. These systems can make use of cloud computing to achieve high levels of scalability, while making services accessible to all at reasonable infrastructure costs and on-demand.

This research aims to develop techniques for building scalable digital information management systems based on efficient and on-demand use of generic grid-based technologies such as cloud computing. In particular, this study explores the use of existing cloud computing resources offered by some popular cloud computing vendors such as Amazon Web Services. This involves making use of Amazon Simple Storage Service (Amazon S3) to store large and increasing volumes of data, Amazon Elastic Compute Cloud (Amazon EC2) to provide the required computational power and Amazon SimpleDB for querying and data indexing on Amazon S3.

A proof-of-concept application comprising typical digital library services was developed and deployed in the cloud environment and evaluated for scalability when the demand for more data and services increases. The results from the evaluation show that it is possible to adopt cloud computing for digital libraries in addressing issues of massive data handling and dealing with large numbers of concurrent requests. Existing digital library systems could be migrated and deployed into the cloud.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The massive amount of data and information produced in recent years constitutes what can be referred to as data or information explosion [66]. Management and storage of this data and information becomes challenging and, over time, data and information systems will have to scale accordingly to cope with this situation. “Big Data" [66] is now part of every sector and function in the global economy. This data needs to be captured, communicated, aggregated, stored and analyzed.

In 2011, the McKinsey Global Institute (MGI) [66] estimated that globally there has been storage of more than 7 exabytes of data by enterprises on disk drives in 2010. Furthermore, MGI stated that consumers alone stored more that 6 exabytes on personal computers and notebooks. Management of this increasingly large amount of data is, therefore, essential. In the current information age, emerging technologies such as cloud computing [50] can arguably be adopted and utilized in order to handle some and/or all of these tasks.

1.1 Cloud Computing

Although there is no consensus on a single definition of **cloud computing** [50], *cloud computing can be defined as a large distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms and services are delivered on-demand to external customers over the Internet* [47]. Cloud computing has been widely

adopted in recent years to solve storage and computational problems in different domains.

Utility computing is a specific subset of cloud computing that provides on-demand infrastructure with the ability to control, scale and configure that infrastructure without the consumer of the resource knowing their physical location [38]. Although sometimes utility computing can be thought of as elastic computing because of its characteristic scalability, elastic computing is a feature of cloud computing that uses computing resources that vary dynamically to meet variable workloads [32]. Utility computing can be argued as a more cost-effective alternative to other forms of high performance computing (HPC) because of the efficiency of shared resources for which the management is normally by a third party [92].

A high level visual model of cloud computing is depicted in Figure 1.1. The figure shows different essential characteristics, service models and deployment models. Cloud services exhibit esential characteristics that demonstrate their relation to, and differences from, traditional computing approaches. A thorough discussion of these concepts is presented in Chapter 2 of this thesis.

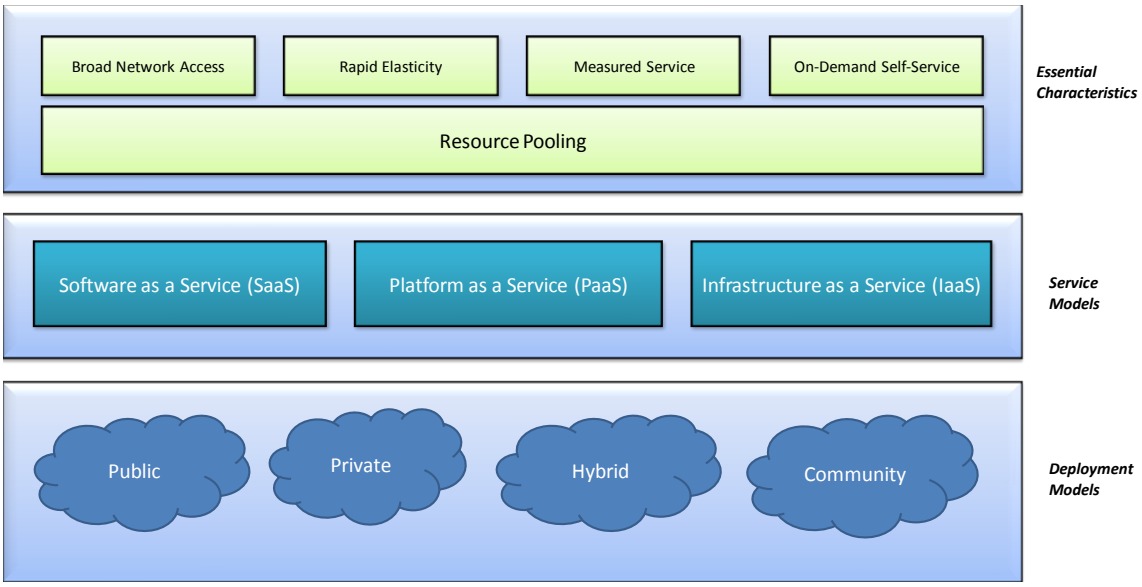


Figure 1.1: An overview of cloud computing showing essential characteristics, service models and deployment models of cloud computing.

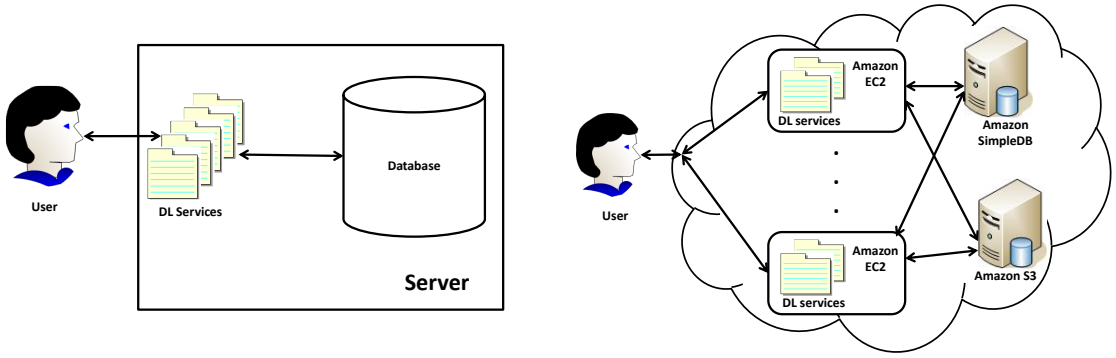
1.2 Motivation

According to a study by the Oracle Corporation [39], data volumes are growing at 40% per year and by 2020 it will have grown to 44 times of its size in 2009. Most systems that are said to scale to cope with large volumes of data store up to 5.1TB of data [31]. Given the amount of research work produced in recent years, in the form of electronic theses and dissertations, research output (Conference papers, Journal articles, workshop articles and books), podcasts, audio files and video clips, systems with limited scalability capabilities will not cope in future. Cloud computing, on the other hand, promises the possibility for unlimited scalability [33] in the management of growing volumes of data.

Digital libraries deal with forever increasing volumes and various forms of data. Over the years, digital library system design and implementation has resulted in production of systems that meet multiple criteria [27]. These include elements of scalability and preservation of data. However, research has to be done to investigate the efficacy of cloud adoption and to verify that cloud computing indeed offers unlimited scalability as the need for more data storage capacity arises. Furthermore, for the majority of applications, databases are the preferred technology for managing and archiving (structured) data sets, but as the size of the data set begins to grow larger than a few hundred terabytes, current databases become less competitive with more specialized solutions, such as the storage services that are part of storage clouds [52]. Utility computing is arguably the better fit here because of its ability to provision services on-demand.

Within utility clouds, digital library services can be designed and implemented to emulate most existing HPC architectures. This is possible because different digital library services are most efficient on different architectures - utility computing offers the flexibility of multiple architectural models to deal with this scenario [92]. Figure 1.2 shows two digital library (DL) architectures; the traditional DL architecture and the cloud computing architecture proposed by this thesis. With the

traditional architecture, DL services are accessible from a manually administered server (therefore, there are administration costs involved) whereas with the cloud DL architecture, services are are accessible from multiple virtual servers.



(a) Traditional Digital Library (DL) architecture (b) Cloud Digital Library (DL) architecture

Figure 1.2: The diagrammatic representation showing the difference between the traditional DL architecture and the proposed cloud DL architecture.

1.3 Problem Statement

This research aims to develop techniques for building scalable digital information management systems based on efficient and on-demand use of generic grid-based technologies. In particular, this study explores the use of existing cloud computing resources offered by the Amazon Web Services. This involves making use of Amazon Simple Storage Service (Amazon S3) to store large and increasing volumes of data, Amazon Elastic Compute Cloud (Amazon EC2) to provide the necessary computational power required and Amazon SimpleDB for data indexing on Amazon S3.

1.4 Research Questions

Specifically, this study addresses the following questions:

1. *Can a typical digital library architecture be layered over an on-demand paradigm such as cloud computing?*

   This research question seeks to investigate if, given a typical digital library system, it is possible to migrate it into the cloud environment. It investigates whether there are any significant modifications necessary to existing digital library systems prior to migration. A proof-of-concept application comprising some typical digital library system services was developed to ascertain whether or not this will be an easy or challenging task to undertake and if, at all, it is possible. The details of the digital library service components design and implementation are discussed in Chapter 4 of this thesis.

2. *Is there linear scalability with increasing data and service capacity needs?*

   This research question can further be divided into two subquestions. The first part will be to investigate whether there will be linear scalability when varying collection sizes are browsed and searched; that is, is there linear scalability when volumes of data increase? The second part of this research question seeks to investigate whether there will be linear scalability when the need for more service capacity arises. In this case, a fixed collection size is browsed and searched while the number of application servers varies. The details of how these two subquestions are answered and the respective findings are presented in Chapter 5.

1.5 Research Contributions

This research explores and gives an insight into the migration of a typical digital library architecture into the cloud environment. It further evaluates if there is

scalability of digital library systems deployed in cloud environments, in terms of the demand for more data and services.

1.6 Scope and Limitations

The focus will be on the core but limited digital library features namely, search and browse. The experiments will also be focused on performance based on the search and browse features. It should be noted that the digital library architectures discussed in this thesis are those relevant to this research.

It will also be infeasible to discuss cloud computing security and the economic value of cloud computing in this thesis. The major focus will be on scalability and service provision in the cloud.

1.7 Research Methodology

The development of a proof-of-concept application comprising typical digital library system services to make use of the Amazon Web Services cloud computing platform was the first step in executing this study. The application comprises a simple user interface for the purpose of querying and results display. Experiments were conducted to validate the study. This evaluation mainly focused on the performance of the search and browse features when subjected to large numbers of requests, large collection sizes and large numbers of concurrent user requests.

1.8 Thesis Organization

The rest of this thesis is organized as follows. **Chapter 2** presents the background to cloud computing, other related high performance computing technologies and an overview of related work. **Chapter 3** gives a technology review of the Amazon

Web Services (AWS) as applicable to this thesis. **Chapter 4** presents the design specification of the application that implements some typical digital library system services as well as its implementation details. **Chapter 5** gives details of the experimental set-up used to test the performance of some design architectures and gives a detailed discussion of findings of this research. Finally, concluding remarks and possible extensions to this research are presented in **Chapter 6**.

Chapter 2

Background and Related Work

The emergence of cloud computing in recent years has attracted researchers, companies and general users across the globe. This has let to the technology largely being used for those applications that are characterized by massive data sets and require significant computing resources [53]. Cloud computing dates back to the early days of Information Technology outsourcing [73]. Different companies or organizations define cloud computing differently in order to describe the kind of services they offer and as a result there is no consensus on a single definition of the term [50]. As stated in Chapter 1, the definition adopted in this thesis is the one by Foster et. al [47].

This chapter presents and discusses different distributed and high performance computing (HPC) technologies with particular reference to how information management systems can continue to make information available and accessible to an ever-larger community of users by consuming cloud/elastic computing resources.

This chapter further presents some trends in distributed and high performance computing by giving an overview of grid computing, cluster computing, cloud computing and related technologies, and how they compare, based on different criteria. The focus is also on computational models and storage models of these technologies (Section 2.1) as used in digital libraries and related systems.

Traditionally referred to as Supercomputing, HPC deals with building hardware and software systems for processing of computationally-intensive jobs [92]. Many forms of HPC exist namely, server farms, grid computing, cluster computing, cloud/utility computing, edge computing, volunteer computing, field-programmable gate arrays

(FPGAs) and general-purpose graphics processing units (GPGPUs). These are in use in recent years to deal with different problems requiring some significant computing power and resources. Each of these technologies is discussed in Section 2.1. Section 2.2 discusses utility and cloud computing. An introduction to Amazon Web Services is given in Section 2.3. Section 2.4 presents cloud computing and information management and related work is presented in Section 2.5.

2.1 Computational and Storage Models

During the early days of studying scalability of online systems, Web server farms and clusters were used in a Web hosting infrastructure as a way to create scalable and highly available solutions [37]. Also referred to as *server cluster, computer farm* or *ranch*, a **server farm**[1] is a group of networked servers, hosted in one location, that streamline internal processes by distributing the workload between individual components of the farm and expediting computing processes by harnessing the power of multiple servers. The following sections discusses different HPC and distributed computing technologies namely, server farms (Section 2.1.1), cluster computing (Section 2.1.2), grid computing (Section 2.1.3), volunteer computing (Section 2.1.4), edge computing (2.1.5), Field-Programmable Gate Arrays (FPGAs) (Section 2.1.6) and general-purpose Graphics Processing Units GPGPUs (Section 2.1.7).

2.1.1 Server Farms

High-scalability server farms [70] have been designed and implemented by Microsoft's development team in order to support over one hundred thousand concurrent users with as few servers as possible. The team achieved a highly scalable

[1] `http://www.webopedia.com/TERM/S/server_farm.html`. [Last accessed on 27 January 2013]

site server by scaling hardware vertically, horizontally and by improving the architecture of the server farm.

Scaling hardware vertically (scale up) means increasing the capacity by upgrading hardware specifications, while maintaining the physical footprint[2] and number of servers in the server farm. This simplifies site management at a higher hardware cost than scaling horizontally [70]. Scaling hardware horizontally (scale out) means increasing capacity by adding servers, that is, adding server machines in parallel to the existing machine(s) [70]. This enables an increase in hardware at a lower cost but requires scaling vertically once site management becomes sufficiently complex. Improving architecture involves improvement of server efficiency by identifying operations with similar workload factors and dedicating servers to each type of operation. The results from these techniques [70], based on an early customer-site performance audit, showed that server farms can be scaled to serve over 100 000 users with as few as 88 front-end servers. The back-end can also be scaled up accordingly by adding servers as needed. The use of server farms has been recommended by the DSpace[3] development team to address some scalability issues because they support the Web server and architecture of DSpace [92].

2.1.2 Cluster Computing

A cluster is a type of parallel or distributed computer system, which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource [19]. Clusters are usually deployed to improve performance and/or availability over that of a single computer while typically, being more cost-effective than single computers of comparable speed and availability [49]. With each machine in a cluster running an independent set of tasks coordinated over a high-speed network, clusters are suited to problems which can be decomposed

[2] `http://www.valid-computing.com/virtual-server-definition.html`

[3] **DSpace** is a software of choice for academic, non-profit and commercial organizations building open digital repositories (`http://www.dspace.org/introducing`).

into separate communicating processes [92]. The computers in a cluster are interconnected among themselves using high-speed networks such as Gigabit Ethernet, SCI[4], Myrinet[5] and InfiniBand[6].

The parallel or distributed computers [86] in a cluster work together to execute compute intensive and data intensive tasks that would otherwise be infeasible to perform on a single computer. The user's requests in a cluster are received and distributed among all the stand-alone computers that form the cluster [86].

Most commercial search engines (e.g. Google) use a cluster architecture for their querying and indexing operations because of the high bandwidth required for efficient indexing for search or browse indices [22, 92].

2.1.3 Grid Computing

With an increasing number of research problems requiring large amounts of computational power, computational grids were developed [46]. The concept of grid computing dates back to the mid-90s and has since then gained much attention in distributed computing research area. Foster and Kesselman [46] define Grid computing as an interconnected system of heterogeneous computational devices under distributed ownership, usually spread over large geographical areas and connected by public or private communication links.

However, Abbas [1] argues that vendors, academics, trade, as well as the popular press have tried to define Grid Computing but could not agree on one definition. Abbas further states that grid computing can enable co-located virtual organizations to share distributed resources to achieve similar functions in a distributed computing paradigm [1].

[4] **SCI** - Scalable Coherent Interface: is a high-speed interconnect standard for shared memory multiprocessing and message passing.

[5] **Myrinet**: is a high-speed local area networking system designed by Myricom to be used as an interconnect between multiple machines to form computer clusters.

[6] **InfiniBand**: is a switched fabric communications link used in HPC and enterprise data centers.

Some types of grids, for instance, computational grids, have been used to address different scientific problems [1]. They represent a new computational framework whose efficient use requires schedulers that allocate user's tasks to the grid resources in a timely manner [42]. Grid computing has a project-oriented business model in which the users or community have a certain number of service units (CPU hours) they can spend [47]. Grids have a five-layer architecture comprising the fabric layer, the connectivity layer, the resource layer, the collective layer and the application layer (see Figure 2.1). The **fabric layer** interfaces to local control, including physical and logical resources such as files. Core communication and authentication protocols supporting Grid-specific network transactions are defined by the **connectivity layer** [51]. The **resource layer** allows the sharing of a single resource and it builds on connectivity layer communication and authentication protocols to define protocols for secure negotiation, initiation, monitoring and control of sharing operations on individual resources [51]. The **collective layer** allows resources to be viewed as collections and sharing of resources and the **application layer** uses appropriate components of each layer to support the application [51]. These layers are shown in Figure 2.1.

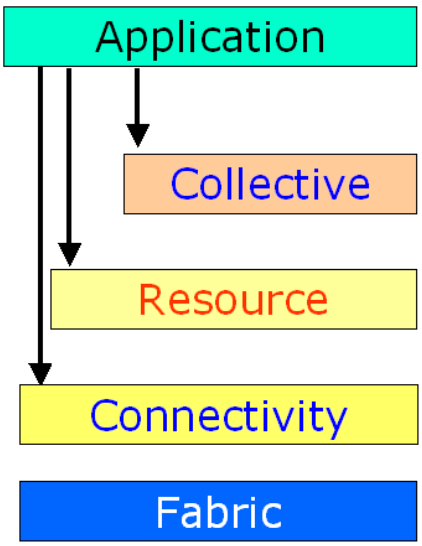


Figure 2.1: A diagrammatic representation of a five-layer Grid architecture [51].

Unlike clouds, grids do not rely much on virtualization [47]. Grids use the MapReduce parallel programming model for large data sets and MPICH-G2, which is a grid-enabled implementation of MPI[7] [63]. The latter gives the familiar interface of MPI while at the same time provides integration with the Globus Toolkit [64]. Applications supported by Grids range from HPC to HTC[8] applications. HPC applications use MPI to achieve the required interprocess communication. Grid security is based on the assumption that resources are heterogeneous and dynamic. Authentication, communication and authorization use public-key based GSI[9] protocols. Storage Resource Broker (SRB) [76], iRODS [20] and DILIGENT [40] are examples of digital library applications that use grids [92]. Grids have been studied and used in digital libraries and a recent study by Parker [83] discusses Grids from a different standpoint, which considers usability as opposed to functionality alone.

2.1.4 Volunteer Computing

Volunteer computing [11] is a form of distributed computing in which the general public volunteers processing and storage to scientific research projects. This is the use of idle desktop computers to perform computations for which dedicated high performance systems can not be secured [92]. BOINC[10] [10] - a platform for public resource distributed computing - is useful for volunteer computing and it facilitates creation of volunteer computing projects [13]. BOINC is designed to support applications that have large computation requirements, storage requirements or both [26]. The first project to use BOINC was the SETI@HOME[11] project [12].

[7] Message Passing Interface
[8] High Throughput Computing
[9] Grid Security Infrastructure
[10] Berkeley Open Infrastructure for Network Computing
[11] **SETI**: *Search for Extraterrestrial Intelligence*

2.1.5 Edge Computing

Edge computing [82] pushes application logic and underlying data to the edge of the network. The aim is to improve availability and scalability [82]. In edge computing, computers at the edge of the network perform computation, instead of machines at the core [92]. Edge computing has been used in the analysis of caching and replication strategies for Web Applications [88]. Sivasubramanian et. al [88] state that one of the techniques and the simplest way to generate user-specific pages of a website is to replicate the application code at the edge servers while the data is kept centralized. This is also a useful technique used by edge computing products at Akamai[12], for online business solutions.

Akamai edge computing enables companies to deploy and execute J2EE[13] applications or application components onto the Akamai network - one of the largest on-demand distributed computing platforms [2]. There are currently several common Web applications and application components that run on the Akamai edge computing platform [2].

Edge computing lets each edge server generate user-specific pages according to context, session and information stored in the database, thereby spreading the computational load across multiple servers [88]. Some known disadvantages, however, include wide-area network latency incurred during data access and performance bottlenecks as a result of serving the entire system's database needs. A typical application of edge computing in digital libraries is the Bleek and Lloyd [90] project in which a prototype search engine was developed in AJAX to demonstrate how ranked retrieval could be performed on the client using local data, thus the need for network interaction and server resources are eliminated [91].

[12] `http://www.akamai.com`

[13] Java 2 Platform, Enterprise Edition

2.1.6 Field-Programmable Gate Arrays (FPGAs)

FPGAs are microchips that can perform specific tasks in hardware during runtime [92] and they are field programmable devices that feature a general structure that allows very high logic capacity [29].

FPGAs have been evaluated for use in computation-intensive data mining applications [62] in a pilot study that made use of SRAM-based [14] FPGA coprocessors. Further work involving intensive computation using FPGAs was the design of an FPGA-based processor array by Perera and Li [84] for computation of a similarity matrix - a commonly used data structure to represent the similarity among a set of feature vectors, with each matrix element representing the computed similarity measure between two vectors. The experimental results have shown that the processing elements in an FPGA are reconfigurable to perform different functionalities of varying complexities, therefore the processor array on an FPGA can have processing elements with different similarity measures or data mining functions. Data mining applications that require parallelism to achieve better performance make use of processor arrays in FPGAs [84].

Although FPGAs provide inherent parallelism, they are limited by the amount of chip space. However, Mueller et. al [75] demonstrated that their limitations can be managed by an efficient circuit for parallel stream processing. Their assessment of the potential for FPGAs as co-processors for data intensive operations in the context of multi-core systems has shown that FPGA capabilities can be integrated into data processing engines efficiently [75]. FPGAs have no well-known digital library applications that currently make explicit use of them [92].

[14] **SRAM**: Static Random Access Memory

2.1.7 General-purpose Graphics Processing Units(GPGPUs)

GPGPU[15] is the utilization of a graphics processing unit (GPU) to perform computation-intensive tasks that have, in the past, been handled by CPUs[16]. They exploit the increasing computational power of microchips on commodity graphics cards, which are capable of parallel processing, thus they are much faster than traditional CPUs.

There are presently no known digital library systems that explicitly use GPGPUs [92].

2.2 Utility Computing and Cloud Computing

The utility computing model offers a number of benefits to both service providers and users. It is a specific subset of cloud computing that is envisioned as the next generation in HPC [100] where the actual nature of the technology is not obvious to the developer and the end-user [92]. Unlike traditional computing, virtualized resources are created and assigned dynamically to various users when needed.

Utility computing is best suited to those users who have rapidly changing or increasing computing needs. Users can obtain appropriate amounts of computing power from providers dynamically, based on their specific service needs and requirements. However, there are businesses that provide a utility service level agreement (SLA), in which case the customer and the utility computing service provider sign an IT service contract that specifies the minimum expectations and obligations that exist between the two parties [30].

Cloud Computing has recently become a mainstream topic of interest in computing as a utility. Armburst et. al [17] further state that the applications delivered as services over the Internet have long been referred to as *Software as a Service (SaaS)*

[15] `en.wikipedia.org/GPGPU`

[16] Central Processing Units

and the data center hardware and software (operating system, enabling middleware and application software) is what is referred to as a *Cloud*. Cloud computing provides an alternative pay-as-you-go business model, offering users services on-demand.

Cloud computing has a four-layer architecture that comprises the fabric layer, the unified resource layer, the platform layer and the application layer (see Figure 2.2). The fabric layer is represented by the hosting platform in Figure 2.2. The connectivity layer, the resource layer, the collective layer and the application layer are represented by cloud infrastructure services, cloud platform services, and cloud applications, respectively. Cloud services, however, are provided at three different layers - the unified resource layer, the platform layer and the application layer.

In the model shown in Figure 2.2, each layer abstracts the layer below it, exposing interfaces that the layers above build upon. There are no strong dependencies between layers and each layer provides an easy to compose architecture with services from other layers [55]. If needed, horizontal scalability can be provided by individual layers in this case. The components making up the cloud platform in Figure 2.2 are briefly discussed as follows:

- **A hosting platform** provides physical machines, operating systems, network systems, storage systems, power management and virtualization software that are needed by the cloud application [55].

- **Cloud infrastructure services** abstract the hosting platform as a set of virtual resources and manage those resources based on scalability and availability needs. This layer provides compute, storage and network abstract resources which means that underlying physical resources can be accessed without knowing the underlying hardware and software. The services provided by this subsystem are known as *Infrastructure as a Service (Iaas)* [55].

- **Cloud platform services** help with integration of on-premise software with

hosted services because management of software for cloud computing is complex. This platform differentiates one cloud provider from another and the services provided by this layer are referred to as *Platform as a Service (PaaS)*. [94]

- **Cloud Applications** is the layer that connects disparate systems and leverages cloud storage infrastructure to store documents. It mainly house applications that are build for cloud computing which expose Web interfaces and Web services for end users. The services provided on this layer are known as *Software as a Service (SaaS)*. [94]

- **Security services** ensure token provisioning, identity federation and claims transformation, all of which are built on open standards, WS-Security, WS-Trust, WS-Federation, SAML[17] protocols and OpenID for greater interoperability. [55]

- **Management services** cut across all the layers described before. The hosting platform takes advantage of the management interfaces and programs for automated scalability and availability administration. Although cloud hosting and management is the task of the datacenter, customers may need functionality that allows them to easily control their application and post deployment configurations, obtain information about usage statistics and connect their enterprise management systems. [55]

- **Tools (application development SDK)** help customers build, test and deploy applications in the cloud. These can be extensions to existing tools or hosted tools from a specific cloud provider. [55]

In the cloud computing model, resources in the cloud are shared by all users at the same time [47]. The next sections are structured as follows, Section 2.2.1 discusses infrastructure models of cloud computing and Section 2.2.2 presents different service layers of cloud computing namely, Software as a Service (SaaS) (Section 2.2.2.1),

[17] Security Assertion Markup Language

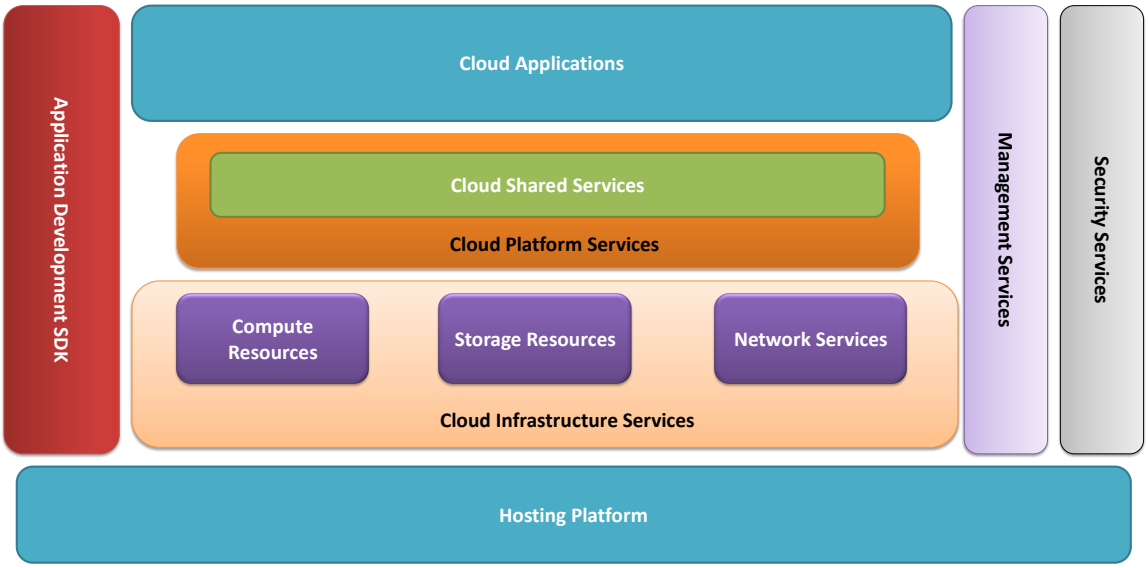


Figure 2.2: A diagrammatic representation of the four-layered cloud architecture [55].

Platform as a Service (PaaS) (Section 2.2.2.2) and Infrastructure as a service (IaaS) (Section 2.2.2.3).

2.2.1 Infrastructure Models of Cloud Computing

There are different architectural considerations for cloud computing particularly, when moving from a standard enterprise application deployment model to one based on cloud computing. Three of these are discussed here. Public and private clouds offer complementary benefits [94], and there is the value of open APIs versus proprietary ones (Figure 2.3).

(a) *Public Clouds:* In this case, the cloud is made available in a pay-as-you-go manner to the general public [17]. Public clouds are run by third parties and applications from different customers are likely to be mixed together on the cloud's servers, storage systems and networks [94]. The general public making use of cloud facilities are not necessarily geographically co-located with the cloud enterprise. In the case of public clouds, the service being sold is *utility*

computing [17]. Examples of public utility computing are *Amazon Web Services* and *Microsoft Azure* [17].

(b) *Private Clouds:* These refer to internal data centres build exclusively for use by one community [17, 94] (see Figure 2.3). These provide clients with control over data, security and quality of service [94] because private clouds can be built and owned by the company's own IT department. In this case, the cloud enterprise and the client are geographically co-located.

(c) *Hybrid Clouds:* These types of clouds combine both public and private clouds (see Figure 2.3) and provide externally provisioned scalability on-demand [94]. Resources of a public cloud can be used to augment a private cloud, which is useful in maintaining service levels in the case of rapid work flows.

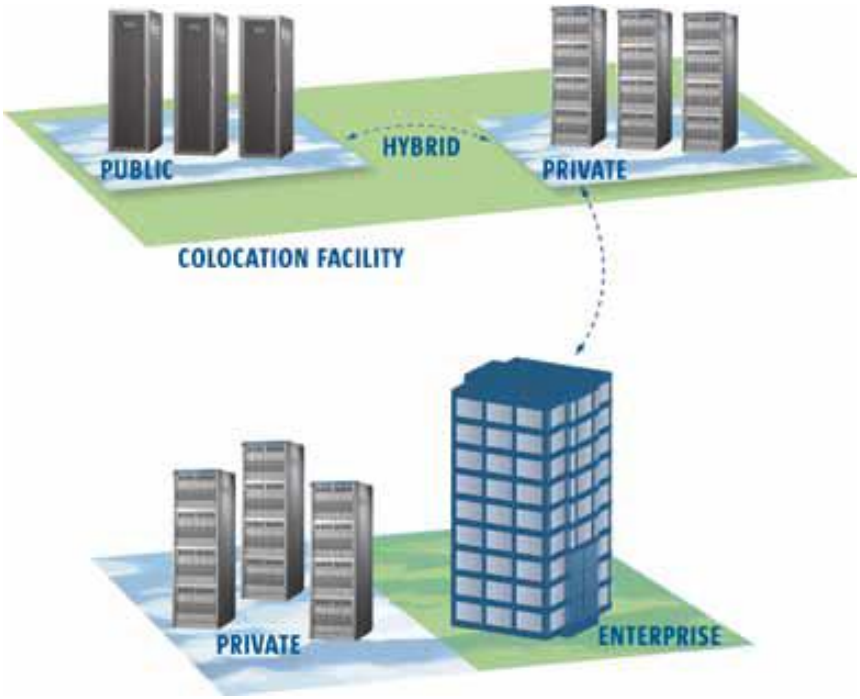


Figure 2.3: Hybrid cloud combining private and public cloud models [94].

2.2.2 Services Layers of Cloud Computing

2.2.2.1 Software as a Service (SaaS)

This is the top layer that features a complete application offered as a service on-demand. A single instance of the software runs on the cloud and serves multiple end users or client organizations [94]. An example of a well-known provider of SaaS is `Salesforce.com`.

2.2.2.2 Platform as a Service (PaaS)

PaaS encapsulates a software layer and provides it as a service that can be used for provision of higher-level services. According to Sun Microsystems [95], PaaS can be provided by integration of an OS, middleware, application software and a development environment, which is then provided to the client as a service. Further, from a client's perspective, PaaS appears as an encapsulated service presented through an API.

2.2.2.3 Infrastructure as a Service (IaaS)

IaaS offers storage [3] and compute [5] capabilities as standardised services over the network. All the hardware, e.g. servers, storage systems, switches, routers and other systems are pooled and made available to handle workloads ranging from application components to HPC applications. The best-known commercial example is Amazon Web Services.

2.3 Amazon Web Services (AWS) and Cloud Computing

Amazon Web Services (AWS) dates back to early 2006 when `Amazon.com` provided companies of all sizes with a Web services platform infrastructure in the cloud. AWS

gives clients the flexibility of using their desired programming model(s) depending on the problems at hand. Clients also have access to computational power, storage, and other Web services on-demand. Clients pay only for what they use and there are no up-front expenses or long-term contracts or commitments, thus making AWS the most cost-effective way to deliver applications to customers and clients.

Different services provided by AWS are summarized in the sections 2.3.1, 2.3.2, 2.3.3, 2.3.4, 2.3.5, and 2.3.6. Details of the services used by the application developed in this thesis are given in Chapter 3.

2.3.1 Amazon EC2

Amazon EC2 [5] is a Web service that enables users to deploy and manage server instances in Amazon's data centres using APIs or available tools and utilities. Users can find Amazon Machine Images (AMIs) and customise them or optionally build AMIs from scratch (this is known as virtualization). These virtual AMIs are based on different operating systems (e.g. Windows and variants of Linux). Then the AMI is bundled so as to obtain an AMI ID to enable deployment of as many instances as desired. Instances can be launched and administered in a similar manner to servers. EC2 is further presented in Chapter 3.

2.3.2 Amazon S3

Amazon S3 [3] is a storage service for the Internet. It provides a simple Web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the Web. Users can create buckets (containers for objects stored in Amazon S3) to store objects[18]. Objects are fundamental entities stored in Amazon S3. Every object has a unique identifier called a key within a bucket. Amazon S3 offers REST and SOAP APIs where users can perform the following operations on buckets: create a bucket, write an object to a bucket, read an object

[18] **An object** is a blob of data, the size of which ranges from 1 byte to 5 TB

from a bucket, delete an object and also list keys contained in one of their buckets. S3 is discussed further in Chapter 3.

2.3.3 Amazon SQS

Amazon Simple Queue Service (Amazon SQS) [7] is a distributed queue system that acts as a buffer between components that produce and save data and offers a reliable, highly scalable, hosted queue for storing messages as they travel between computers in the cloud. Amazon SQS allows decoupling the components of an application so they run independently, with Amazon SQS handling message management among components. It ensures that the message is delivered at least once and supports multiple reading and writing happening on the same queue.

2.3.4 Amazon Elastic MapReduce

Amazon Elastic MapReduce [6] is a Web service that enables businesses, researchers, data analysts and developers to easily and cost-effectively process vast amounts of data using Amazon EC2 and Amazon S3. Amazon Elastic MapReduce focuses mainly on data analysis. Users upload into Amazon S3 the data they need to process along with the mapper and reducer executables that will process the data and then send a request to Elastic MapReduce to start a job flow. Then MapReduce starts an EC2 cluster, which loads and runs Hadoop. Hadoop executes a job flow by downloading data from Amazon S3 onto the cluster of slave instances. Then Hadoop processes the data and uploads the results from the cluster to Amazon S3. Finally, users receive notification that their data analysis is done and they can download the processed data from Amazon S3.

2.3.5 Amazon CloudFront

Amazon CloudFront [4] is a Web service that makes content delivery to users easier and quicker at low latency and high data transfer speeds. It works with Amazon S3 and users are routed to the nearest edge location (geographical location where CloudFront caches copies of users' objects). In the case of Amazon CloudFront, objects refer to files that users need CloudFront to deliver. The origin server stores the original versions of objects.

2.3.6 Amazon SimpleDB

Amazon SimpleDB [8] is a Web service that works in close collaboration with Amazon EC2 and Amazon S3 to provide core database functions of unstructured data and metadata indexing and querying in the cloud.

2.4 Cloud Computing and Information Management

Part of the on-going research in the digital libraries community is focused on the development of techniques for scalable search services [36], addressing issues of scalability for high performance digital libraries on the World Wide Web [14] and expanding digital library services [14]. With increasing volumes of information, some digital library systems fail to scale as expected [85]. Therefore, there is a need to devise generic techniques (not only specific to the WWW) to cope with the ever-increasing demands to archive information using on-demand paradigms such as Amazon's EC2/S3.

The DSpace Foundation [41] and Fedora Commons [60] have developed a new service named DuraSpace to serve academic libraries, universities and other organizations in providing perpetual access to digital content.

2.4.1 DuraCloud

DuraCloud [69] is a Web-based open technology platform aimed at supporting libraries, universities, and other cultural heritage organizations that wish to provide perpetual access to their digital content. The service replicates and distributes content across multiple cloud providers and enables the deployment of services to support access, preservation and re-use. DuraCloud was developed by DuraSpace[19]. Duracloud was not used for experiments in this thesis because it is very specific in its function unlike AWS that provides a suite of services that can be harnessed by different digital library components.

Figure 2.4 shows possible interactions in a particular use case where DuraCloud is used as a backup system for data being stored within an institutional repository. The repository in the diagram is exemplary and it is replaceable with any other software system that is capable of writing to the file system [28].

2.4.2 Fedorazon Cloud Repository

The Fedorazon project [45] dates back to 2004 when cloud computing was adopted as an alternative for development of institutional repositories. The Fedorazon (Fedora Commons + Amazon Web Services) project uses Fedora digital repository software with Amazon's EC2 and S3 and it was tailored to help institutions to launch their own repositories in the cloud with minimal effort and with little or no technical expertise regarding the necessary underlying hardware configurations.

The inception of Fedorazon had close ties with the Grid community and thus the early implementation of Fedorazon was based on the Grid and ASP[20] [45]. The wide open access of the grid allows anyone to deploy any kind of software they want on it

[19]Solutions offered by DuraCloud are described on the project website available at `http://wiki.duraspace.org/display/DURACLOUD/DuraCloud` [Last Accessed on 30 January 2013]

[20]Active Server Pages

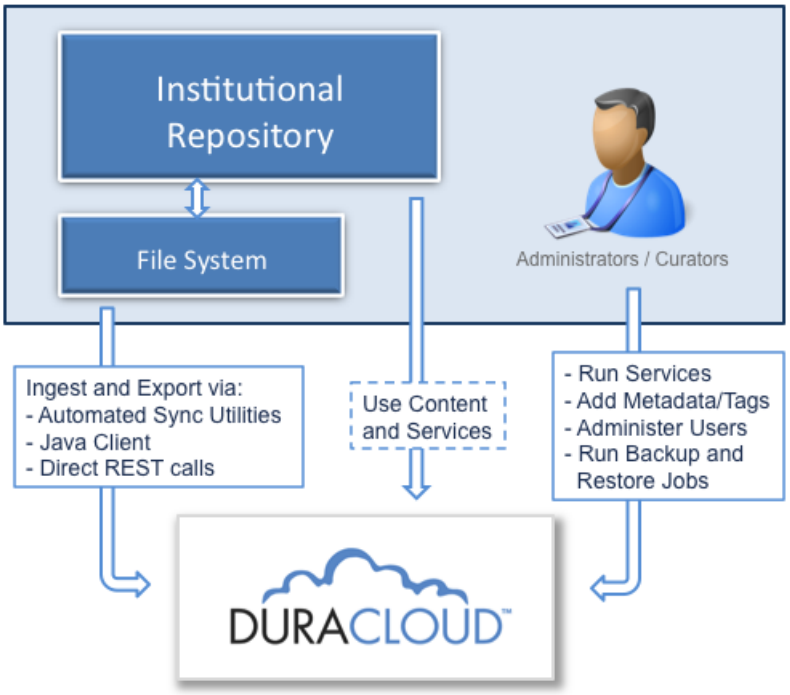


Figure 2.4: DuraCloud in context: A typical DuraCloud utilization scenario [28].

and run it to capacity. However, the emphasis for the Grid is all on computational cycles and very little support for long term storage. As a result, there is a high threshold for understanding and administering the Grid which requires in-house expertise [45].

The Fedorazon project produced of a publicly accessible version of Fedora 3.0 on Amazon Web Services (EC2, S3 and EBS). This means that users can launch an instance of Fedora 3.0 on AWS following a step-by-step procedure outlined in the Fedorazon documentation[21] in order to set up their own institutional repository. The project has also achieved the cost analysis report [45], which attempted to provide pragmatic advice for the actual costs in running any repository within the

[21] http://www.ukoln.ac.uk/repositories/digirep/index/Fedorazon_How_to_Guides

cloud.

At the time of the project conclusion in 2008, there were several academic sector institutions that have utilised Fedorazon for development and testing but there has yet to be a definitive institution to use it as their primary repository instance [45]. In its current state, Fedorazon could be called “Platform as a Service (PaaS)".

2.5 Related Work

The notion of using grid technologies in digital libraries has been addressed before as evidenced by previous works. For example, to address some threats to long term digital preservation, Barateiro et. al [20] present the use of data grids for digital preservation. The authors extend the existing iRODS, an open-source system for data grids based on the distributed client-server architecture. Some extensions made to iRODS were a replication of the iCAT (the database which is a central repository for storage of data in iRODS) to all nodes in the grid and an iCAT recovery mechanism in case of failure/corruption of the central node and an audit service for comparison of iCAT with its replicas [20].

Suleman et. al [93] address two scalability concerns with the aid of migration and replication in component-based digital library systems using grid technology. The results from the experimentation have shown that the component-based digital library system with minimal functionality successfully scales seamlessly with little overhead independent of the number of parameters that can be optimized [93].

The DILIGENT project focuses on integration of grid technology and digital libraries [40] but differs from the cloud computing approach in that it does not address scalable on-demand access to digital libraries. The project aims at supporting virtual research groups by providing the knowledge infrastructure that manages a network of shared resources (archives, databases and software tools) and enables reliable and secure Digital Libraries on-demand [40].

Some repository systems such as DSpace [41] and Fedora [60] may, in future, use storage resource brokers (SRB) to manage access to information and for preservation using some lightweight API [41]. However, these systems have not yet achieved scalability levels required in handling large amounts of data [72, 58]. Thus, it is of paramount importance to move digital library systems to a different ideal such as the use of cloud/utility computing to provide access to large amounts of data and computational resources [79].

The next sections discuss previous work on cloud migration (Section 2.5.1), performance analysis of applications deployed in the cloud (Section 2.5.2) and some related cloud systems (Section 2.5.3). Migrating applications into the cloud environment is discussed here because there have been efforts to migrate existing systems into the cloud [18]. It is also important to highlight performance of applications deployed in the cloud as compared to applications that are hosted on organization's premises.

2.5.1 Cloud Migration

Cloud migration[22] is the process of transitioning all or part of an organization's data, applications and services from on-site premises behind the firewall to the cloud. Cloud migration has gained attention from different communities in recent years. Babar and Chauhan [18] shared their experiences in migrating an open source software system, Hackystat[23], into the cloud. In their study, they presented the existing Hackystat architecture and the enhanced architecture for cloud deployment. They argue that software systems consisting of multiple technologies in their implementation are harder to migrate than systems using stateless components. There

[22] Cloud migration `http://www.webopedia.com/TERM/C/cloud_migration.html`. Last accessed on May 22nd, 2013

[23] **hackystat**: a framework for collection, analysis, visualization, interpretation, annotation and dissemination of software development process and product data. *Source*: `http://code.google.com/p/hackystat/`

was no actual migration performed except for the framework that could be used in the migration of Hackystat into the cloud.

Khajeh-Hosseini et. al [56] discussed the cost-benefit analysis of cloud migration. Their study did not involve actual system migration but concluded that migrating enterprise IT systems to IaaS is more cost-effective than maintaining an on-premises system.

2.5.2 Performance Analysis of Cloud Applications

With an increasing adoption of cloud computing, one of the key issues is performance evaluation of applications deployed in cloud environments. In a study that is very similar to this work in terms of evaluation, Moschakis and Karatza [74] present performance evaluation of integrating mechanisms for job migration and handling of job starvation. Their evaluation was conducted through simulation under varying workloads, job sizes, migration and starvation handling schemes and the results did not show significant improvements in response times in a cloud environment [74].

Khazaei et. al [57] present a novel approximate analytical model for performance evaluation of cloud server farms and solve it to obtain an accurate estimation of the complete probability distribution of the request response time, among other things. The results from the performance of the cloud server farm indicated that their proposed approximation method provided more accurate results for the mean number of tasks in the system, blocking probability, probability of immediate service and response times in the cloud [57]. However, there were longer waiting times for clients in a cloud centre that accommodated heterogeneous services as opposed to its homogeneous equivalent with the same traffic intensity [57].

Performance evaluation of high-speed network interconnects such as InfiniBand on HPC and cloud systems has also be studied [96]. The results showed that the latest version of InfiniBand FDR[24] interconnect gives the best performance in terms of

[24] **FDR**: Fourteen Data Rate

latency and bandwidth on HPC and cloud computing systems [96].

2.5.3 Related Cloud Systems

The are several cloud initiatives and virtualization technologies that exist today. Some of these are briefly discussed in this section. Detailed discussion of all cloud computing vendors and operating systems is out of the scope of this thesis. The cloud computing systems discussed here are based on some of the HPC technologies discussed above.

2.5.3.1 EUCALYPTUS

Eucalyptus (Elastic Utility Computing Architecture Linking Your Programs To Useful Systems) is an open-source software infrastructure for implementing utility computing on clusters. EUCALYPTUS delivers a framework where Amazon cloud facilities can be integrated. Its infrastructure is designed to support multiple client-side interfaces and the current Eucalyptus interface is compatible with the following AWS services used in this thesis; Amazon EC2, Amazon S3 and Amazon EBS[25] [78].

Originally a research project from the Computer Science Department at the University of California, Santa Barbara, Eucalyptus software is now maintained by Eucalyptus Systems (A company founded by the authors of the software). Nurmi et. al [79] present the design of Eucalyptus that emulates Amazon EC2's SOAP and query interfaces and allows users to launch, access and terminate entire virtual machines. It is an open-source framework for cloud computing, implementing Infrastructure as a Service (IaaS).

Its design is simple, flexible and modular and motivated by extensibility and non-intrusiveness [78, 79]. The current design is such that virtual machines running on top of Xen hypervisor [21] are supported. Each high-level system is implemented

[25]Elastic Block Store

as a stand-alone Web service allowing Eucalyptus to expose each Web service as a well-defined, language-agnostic API [43].

The six components of the Eucalyptus cloud architecture are [43]:

(a) *Node Controller (NC)*: the NC executes on any machine hosting VM instances and controls VM activities such as execution, inspection and termination of VM instances [43].

(b) *Cluster Controller (CC)*: The Cluster Controller has the function of gathering information about a set of Node Controllers (NCs) and schedules virtual machine (VM) execution on specific NCs. Executing on a machine that has a network connection to both machines running the NC and the machine running the CLC, the CC also manages the virtual machine networks [43].

(c) *Cloud Controller (CLC)*: The CLC is the entry point into the cloud and it queries other components for information about resources, makes high-level scheduling decisions and makes requests to Cluster Controllers (CC). It is responsible for exposing and managing the underlying virtualized resources e.g. servers, network and storage and it can be accessed through command line tools that are compatible with Amazon's EC2 as well as through a Web-based Eucalyptus Administrator Console [43].

(d) *Walrus*: Users can store persistent data, organized in *buckets* and *objects* using Walrus. Walrus has the following operations for buckets: `create, delete` and `list` buckets; and `put, get` and `delete` on objects. The Walrus interface is compatible with Amazon's S3 [43].

(e) *Storage Controller (SC)*: The storage controller provides similar functionality to Amazon's Elastic Block Store(EBS) [43].

(f) *VMware Broker (Broker or VB)*: this optional Eucalyptus component enables deployment of virtual machines (VMs), VMware infrastructure elements and mediates all interactions between CC and VMware hypervisors either directly

or through VMware vCenter [43]. Figure 2.5 shows the logical relationship

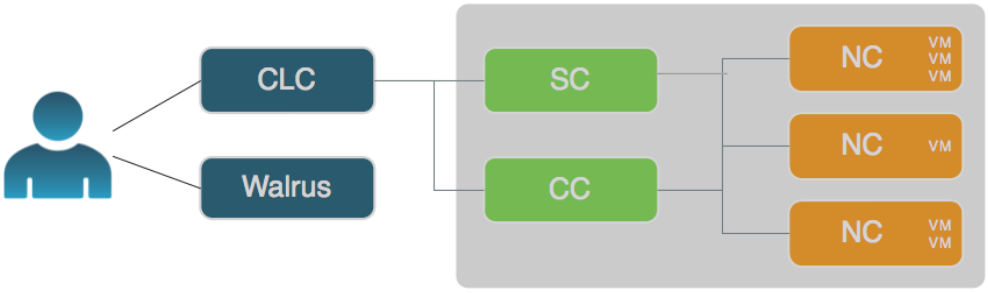


Figure 2.5: The logical relationship between Eucalyptus components [44].

between these components. The cloud components - CLC and Walrus - communicate with cluster components - CCs and SCs. The CCs and SCs, in turn, communicate with the NCs. The network among machines hosting these components must be able to allow TCP[26] connections among them [44].

Euca2ools - the Eucalyptus command line interface for interacting with Web services - has most of its commands similar to EC2, S3 and IAM[27] services, thus generally accepting the same options and environment variables. Figure 2.6 shows the relationship between AWS and Eucalyptus functions. EC2 maps to CLC, EBS maps to SC and S3 maps to Walrus.

2.5.3.2 OpenNebula

OpenNebula [71] is an open-source cloud computing framework that enables creation and management of virtualized infrastructures that provide private, public and hybrid IaaS clouds [89]. OpenNebula architecture allows the use of multiple storage back-ends such as logical volume manager (LVM) and Internet small computer system interface (iSCSI) and different hypervisors such as VMware, Xen and KVM[28] [71].

[26] Transmission Control Protocol
[27] Identity and Access Management
[28] kernel-based virtual machine

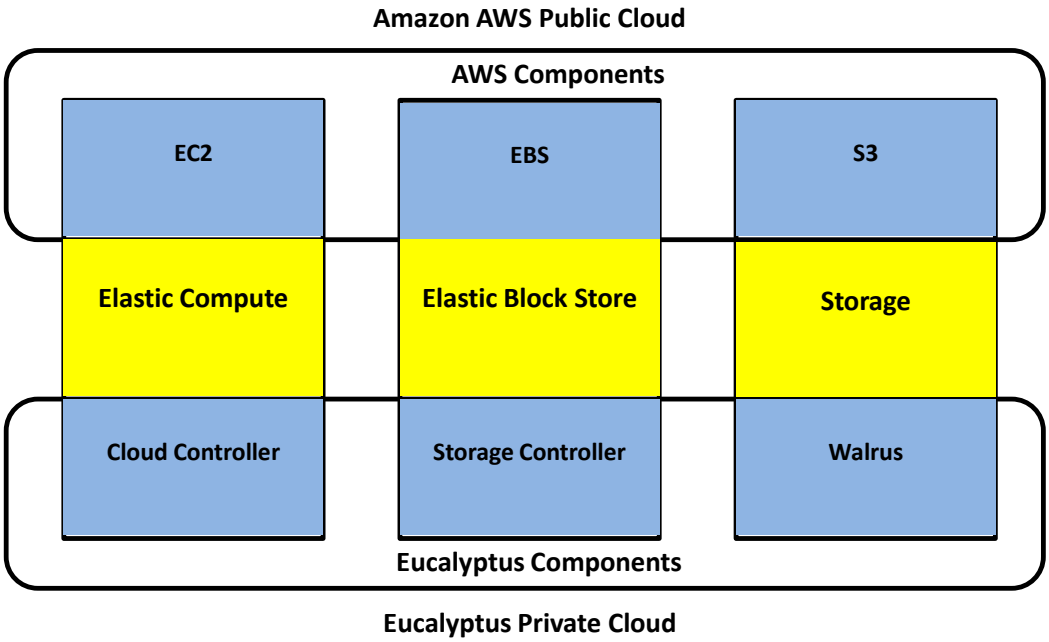


Figure 2.6: The relationship between Amazon AWS and Eucalyptus [44].

Several cloud infrastructures are based on OpenNebula. CERN - the European Organization for Nuclear Research - uses virtualization and OpenNebula for batch processing services [86]. Telecommunications operators such as Telefónica are using OpenNebula to virtualize Web servers, mailing systems and databases [71]. Unlike AWS and Eucalyptus, OpenNebula, by default, uses a shared file system (typically NFS) for all disk image files and files needed to run OpenNebula functions [87]. OpenFlow[29] is one of the applications that use OpenNebula.

OpenFLow provides a way for researchers to run experimental protocols in the networks they use daily [68]. OpenFlow provides network management for CloudNaaS (Cloud Network as a Service) [24] - a networking platform for enterprise applications that extend the self-service provisioning model for cloud services to include networking services [89]. Stabler et. al [89] implemented OpenFLow protocol on

[29] **OpenFlow** is being developed by the Open Networking Foundation (ONF) (`http://www.openflow.org`)

Clemson's OneCloud[30] [89] which was integrated into a virtual machine provisioning engine and provided an API for networking services to end-users. OpenFlow can also serve as a useful campus component in large scale testbeds and Stanford University will adopt it, using commercial Ethernet switches and routers for their networking experiments [68].

OpenNebula supports several standard APIs namely; EC2 Query, vCloud[31] [59] and Open Cloud Computing Interface (OCCI)[32] - an emerging standard defining IaaS APIs that are delivered through the Open Grid Forum (OGF)[33] [23].

2.5.3.3 OpenStack

Openstack[34] is an open-source project initiated by Rackspace and NASA that is built into Ubuntu Sever and Ubuntu is the reference operating system for OpenStack deployments [34]. OpenStack controls and manages compute, storage and network resources aggregated from multiple servers in a data centres [23]. It provides a Web interface (dashboard) and APIs compatible with Amazon EC2 to the administrators and users [23]. The interface allows for flexible provisioning of resources.

2.5.3.4 CloudStack

CloudStack is an open-source software platform that pools computing resources to build public, private and hybrid IaaS clouds [15]. In addition to its own API, CloudStack implements the the Amazon EC2 and S3 APIs, as well as the vCloud API [23].

[30] **One Cloud** is an experimental cloud infrastructure from Clemson University that is based on the OpenNebula cloud framework (`https://sites.google.com/site/cuonecloud`)

[31] VMware's vCloud `http://vcloud.vmware.com/`

[32] Open Cloud Computing Interface. `http://occi-wg.org/`

[33] Open Grid Forum. `http://www.ogf.org/`

[34] OpenStack. `http://www.openstack.org`

There are several cloud offerings in industry today. Some examples of different cloud service provisioning platforms are Red Hat's OpenShift[35], Nimbula[36], Nimbus [77], VMware's Cloud Foundry [97], etc. As mentioned before, discussion of all these is out of the scope of this thesis. Most of these platforms offer similar services, what differs is the vendor of the cloud infrastructure.

2.6 Summary

Different distributed and HPC technologies have been presented in this chapter and, for each of them, there were examples showing if and how they have been used in digital libraries. Some advances in cloud computing were also presented but none of the existing systems uses cloud computing in a manner similar to this work. Most systems are focused on the computational power, data processing and storage without explicit emphasis on evaluation of issues of scalability. A detailed discussion of Amazon Web Services relevant to this thesis is given in Chapter 3 that follows.

[35] OpenShift. https://openshift.redhat.com/app/

[36] Nimbula. https://nimbula.com

Chapter 3

Amazon Web Services

Amazon Web Services (AWS) provides a full suite of services designed to solve application growth needs through on-demand scalability, processing and storage. This chapter gives an overview of Amazon Web Services used in the design and implementation as discussed in Chapter 4.

The AWS examples presented in this chapter are those that were used in the implementation and deployment of the search and browse digital library system services. There are several services offered by AWS that will not be presented in this chapter. Specific services that will be discussed include Amazon Elastic Compute Cloud (Amazon EC2[1]) (Section 3.1), Amazon Simple Storage Service (Amazon S3[2]) (Section 3.3), Amazon SimpleDB[3] (Section 3.4) and Amazon Elastic Block Store (Amazon EBS[4]) (discussed under Section 3.2). Section 3.5 presents some request and response mechanisms on EC2, S3 and SimpleDB.

3.1 Amazon EC2

EC2 is a Web service that provides resizable computing capacity in the cloud and it refers to the servers in Amazon's data centers that are used to build and host software systems [5]. EC2 is physically a large number of computers on which Amazon provides time to paying customers [53]. EC2 is based on Xen virtualization

[1] Amazon EC2 and EC2 will be used interchangeably
[2] Amazon S3 and S3 will be used interchangeably
[3] Amazon SimpleDB and SimpleDB will be used interchangeably
[4] Amazon EBS and EBS will be used interchangeably

technology [21]. This allows one physical computer to be shared by several virtual computers, each of which hosts different operating systems (OSes). Each virtual OS has its own management strategy. In principle, Xen virtualization can allow any sort of operating system to be hosted [53].

EC2 provides users with virtual hosts based on Linux and Windows operating systems. These have a range of 32- and 64-bit kernels that support Windows and common Linux distributions such as Ubuntu and Fedora Core. The components and features that EC2 provides can be accessed using a Web-based GUI[5], command line tools and API[6] calls.

EC2 comprises *Amazon Machine Images (AMIs)*, Regions and Available zones, storage, databases, networking and security, and other components such as monitoring, auto scaling and elastic load balancing, and AWS identity and access management(IAM).

3.1.1 Amazon AMIs

An Amazon AMI is a template that contains a software configuration such as an operating system, a server and applications from which *instances* can be launched [5]. An instance is a copy of an AMI running as a virtual server in the cloud. Multiple and different instance types of the same AMI can be launched. The hardware of the host computer used for an instance is determined by the instance type [101].

Each instance type offers different compute and memory capabilities [5]. It is important for users to select an instance type based on the amount of memory and computing power that they will need for the application or software that they plan to run on the particular instance. Table 3.1 [7] gives a snapshot of instance types available to users to run their software applications. A full list of instance types is available in [5].

[5]Graphical User Interface

[6]Application Programming Interface

[7]*Source*: Amazon Elastic Compute Cloud User Guide, API Version 2012-12-01

Type	Name	EC2 Compute Units (ECU)	Virtual Cores	Memory	Instance Store Volumes	Platform	I/O
Micro	t1.micro	Up to 2 (for short periodic bursts)	1	615 MiB	None (uses EBS for storage)	32-bit and 64-bit	Low
M1 Small	m1.small	1	1	1.7 GiB	150 GiB (1 x 150 GiB)	32-bit and 64-bit	Moderate
High-CPU	c1.medium	5	2 (with 2.5 ECUs each)	1.7 GiB	340 GiB (1 x 340 GiB)	32-bit and 64-bit	Moderate
High-Memory Extra Large	m2.xlarge	6.5	2 (with 3.25 ECUs each)	17.1 GiB	410 GiB (1 x 410 GiB)	64-bit	Moderate
High-Memory Cluster Eight Extra Large	cr1.8xlarge	88	16 (with 2 x Intel Xeon E5-2670, eight-core)	244 GiB	240 (2 x 120 GiB)	64-bit	Very High (10Gbps)
High I/O Quadruple Extra Large	hi1.4xlarge	35	8 (with 4.37 ECUs each)	60.5 GiB	2 TiB (2 x 1 TiB SSD)	64-bit	Very high (10 Gbps Ethernet)
High Storage Eight Extra Large	hs1.8xlarge	35	16 (8 cores + 8 hyperthreads)	117 GiB	48 TiB (24 x 2 TiB hard disk drives)	64-bit	Very high (10 Gbps Ethernet)
Cluster Compute Eight Extra Large	cc2.8xlarge	88	16 (2 x Intel Xeon E5-2670, eight-core “Sandy Bridge" architecture)	60.5 GiB	3360 GiB (4 x 840 GiB)	64-bit	Very High (10 Gbps Ethernet)
Cluster GPU Quadruple Extra Large	cg1.4xlarge	33.5	8 (2 x Intel Xeon X5570, quad-core “Nehalem" architecture), plus 2 NVIDIA Tesla M2050 “Fermi" GPUs)	22.5 GiB[a]	1680 GiB (2 x 840 GiB)	64-bit	Very High (10 Gbps Ethernet)

Table 3.1: Some examples of available Amazon EC2 Instance Types[5]

[a]The cg1.x4large instance type has 1 GiB reserved for GPU operation. The 21.5 GiB doesn’t include the on-board memory of the GPUs, which is 3 GiB per GPU for the NVIDIA Tesla M2050.

Figure 3.1 illustrates how different instance types can be launched from one AMI. That is, three host computers each has at least one instance launched from them and all the instances on the three computers were launched from the same AMI although launched on different hosts. Instances continue running until they are terminated or they fail. There are many available public AMIs but users can also create their own AMIs using command line tools to come up with an AMI that meets their specific needs.

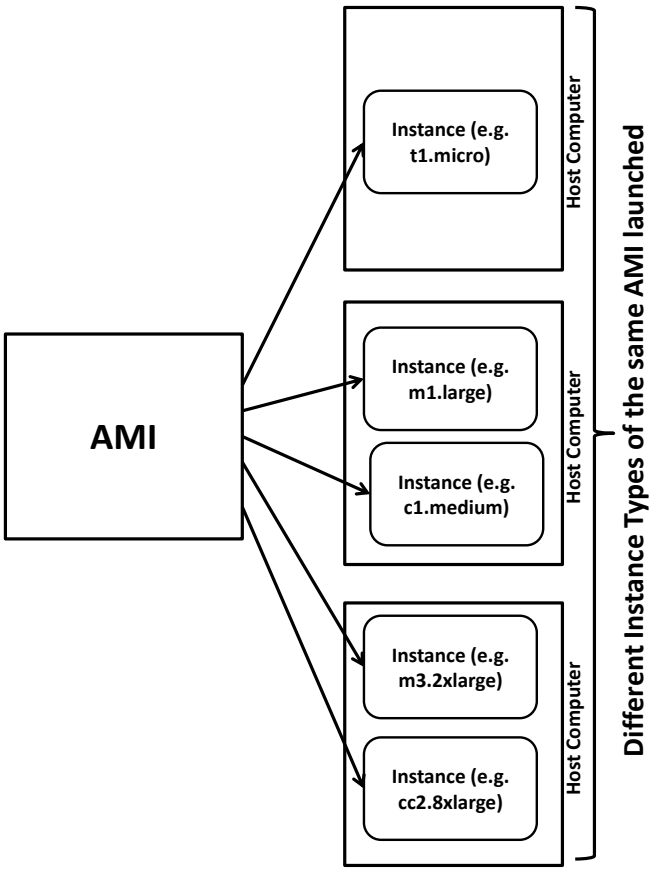


Figure 3.1: Launching different instance types from one AMI.

In order to gain insight into which instance type is best-suited for an application, the following definitions of instance types and families need to be borne in mind [5, 101], examples of which are given in Table 3.1.

- *Micro*: this instance type provides a small amount of consistent CPU resources

and enables users to burst CPU capacity when additional cycles are available. These are well-suited for lower throughput applications and websites that consume significant compute cycles periodically e.g. t1.micro.

- *Standard*: these instance types have memory-to-CPU ratios suitable for most general-purpose applications e.g. m1.small.

- *High-CPU*: these instance types have proportionally more CPU resources than memory (RAM).These are well-suited for computationally-intensive applications e.g. c1.medium.

- *High-Memory*: these instance types have proportionally more memory resources. These are well suited for high-throughput applications, for instance, database and memory caching applications e.g. m2.xlarge.

- *High-Memory Cluster*: these instance types have large amounts of memory as well as high CPU and network performance and they are well suited for in-memory analytics, graph analysis and scientific computing applications.

- *High I/O*: these instance types provide tens of thousands of low-latency, random I/O operations per second (IOPS) to an application. These are well-suited for NoSQL databases, clustered databases and OLTP (online transaction processing) systems e.g. hi1.4xlarge.

- *High Storage*: these instance types provide very high storage density and high sequential read and write performance per instance. They are well-suited for data-intensive applications, for instance, data warehousing, Hadoop/MapReduce and parallel file systems e.g. hs1.8xlarge.

- *Cluster Compute*: these instance types have a very large amount of CPU coupled with increased networking performance. These are well-suited for High Performance Computing (HPC) applications and other network-intensive distributed computing applications e.g. cc1.4xlarge.

- *Cluster GPU*: these instance types provide general-purpose graphics processing units (GPUs), with proportionally high CPU and increased network performance for applications that benefit from highly parallelized processing. These are well-suited for HPC applications as well as rendering and media processing applications e.g. cg1.4xlarge.

3.1.2 Other EC2 Components

Other important EC2 components include:

- *Databases*: EC2 instances can be used to run a database and store data within an EBS volume[5]. Advanced database functionality can be provided by services such as SimpleDB[8].

- *Networking and security*: EC2 allows for assignment of instances to user-defined security groups that define firewall rules for instances. When an AMI instance is launched, it can be assigned to as many groups as the user desires [5].

- *Monitoring*: EC2 provides basic monitoring and Amazon CloudWatch provides advanced monitoring services [5].

- *Auto scaling*: this allows for automatic scaling of EC2 capacity up or down depending on whether there is a demand for more instances to maintain performance, or the demand drops, respectively, to minimise costs [5].

- *Elastic load balancing*: this service automatically distributes incoming traffic across multiple EC2 instances [5].

- *Identity and Access Key Management (IAM)*: this service provides users with unique security credentials and access privileges [5]. The pre-requisite is that a user has to sign up for an AWS account so that they can be assigned security

credentials - an access key ID and a secret access key. An access key ID[8] is essentially a **username** and it is an alphanumeric text string that uniquely identifies a user who owns an AWS account. A secret access key plays the role of a **password** hence it is known to the user who owns an AWS account only.

3.1.3 Regions and Availability Zones

Amazon EC2 instances can be placed in multiple locations. Locations on Amazon EC2 include Availability Zones and regions. Regions are dispersed and located in separate geographic areas [5, 101]. Availability Zones are distinct locations within a region that are engineered to be isolated from failures in other Availability Zones and provide inexpensive, low latency network connectivity to other Availability Zones in the same region [5]. Therefore, instances can be launched in separate regions allowing for applications to be designed in such a way that they are closer to specific customers or other requirements.

One of the advantages of instances that are launched in separate Availability zones is that it helps to protect applications in case of failure of a single location. In order for the user to find out the Availability zones and Regions available to them, they issue the following commands, respectively, from the command line:

```
prompt> ec2-describe-availability-zones

AVAILABILITYZONE us-east-1a available us-east-1
AVAILABILITYZONE us-east-1b available us-east-1
```

```
prompt> ec2-describe-regions

REGION ap-northeast-1 ec2.ap-northeast-1.amazonaws.com
REGION ap-southeast-1 ec2.ap-southeast-1.amazonaws.com
```

[8] http://www.bucketexplorer.com/documentation/amazon-s3--what-is-my-aws-access-and-secret-key.html

There are several API command-line tools for Amazon EC2 (detailed documentation of all the command line tools is available in [5]) but most AWS services are easily accessible via a Web interface today.

During data processing on an instance, the storage options available to users on EC2 are EBS, Amazon instance store and S3. Figure 3.2 shows the relationships among these different forms of EC2 storage. Instance store provides temporary block-level storage for use with an EC2 instance during processing and, for persistent block-level storage, EBS is used. EBS volumes are created from a snapshot which is, in turn, stored on S3.

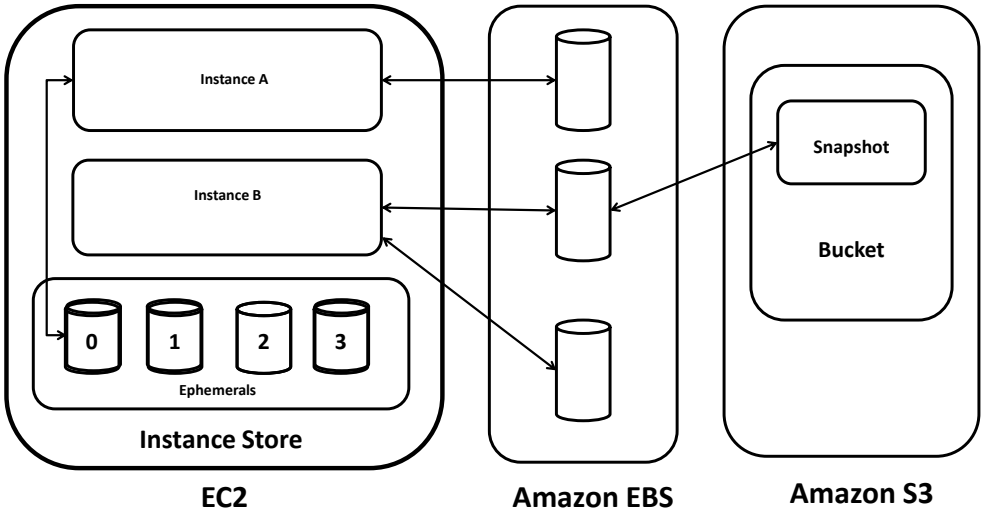


Figure 3.2: The relationship among Amazon EC2 forms of storage.

3.2 Amazon EC2 Storage

Amazon EC2 storage is provided by two components, Amazon EBS (Section 3.2.1) and Amazon instance store (Section 3.2.2).

3.2.1 Amazon EBS

EBS is used where persistent block-level storage is required. Best suited for applications that require a database, a file system, or access to raw block-level storage, EBS volumes are essentially hard disks attached to a running instance[5].

As shown in Figure 3.2, multiple volumes can be attached to an instance. For purposes of data back-up, users can create a snapshot of an EBS volume and store it in Amazon S3. New Amazon EBS volumes can be created from a snapshot, and be attached to another instance. Volumes can be detached from instances and be attached to other instances.

When an EBS volume is created, it goes into a “pending" state. In this state it can either be in “available" state or “deleted" state. If the volume is in “available" state, then it can be attached to an instance. In this case it is in “attached" state. Otherwise the volume can be deleted if not used. In this case it goes into the “deleted" state. In the “attached" state, the EBS volume can be detached and go back into the “available" state, in which case it can he attached again or deleted, and the cycle repeats itself. Figure 3.3 represents this scenario.

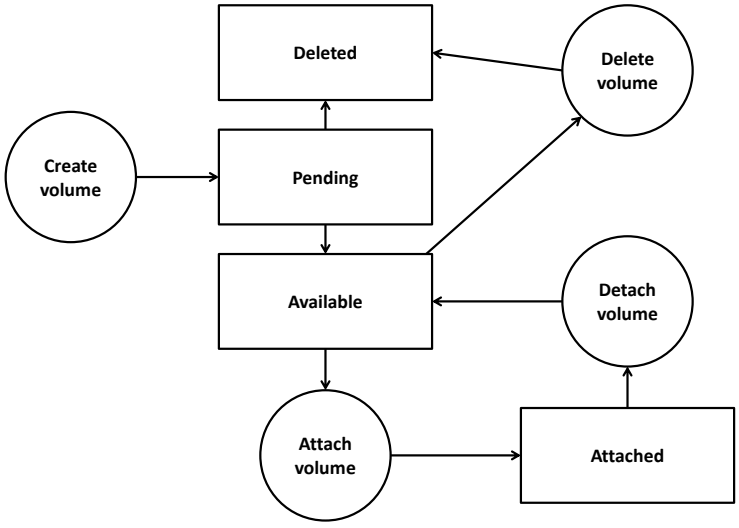


Figure 3.3: The diagram illustrating the life cylce of Amazon EBS[5]

3.2.2 Amazon Instance Store

With the exception of `micro` instances, all instance types offer instance store, which provides instances with temporary, block-level storage that is physically attached to the host computer. The data on an instance store volume does not persist when the associated instance is stopped or terminated. Detailed information on instance ranges is given in the Amazon EC2 User Guide API Version 2012-12-01 in [5].

An Amazon EC2 instance store consists of one or more instance store volumes. These volumes are usable only from a single Amazon EC2 instance during its lifetime; they can not be detached and then attached to another instance. Instance store volumes are mounted before they can be used[5]. The instance type also determines the type of hardware for instance store volumes. A high I/O instance (for example, `hi1.4xlarge`) uses solid state drives (SSD) to deliver very high random I/O performance. This works best for very low latency storage needs. Instance stores can be used with EBS and in turn with Amazon S3 (see section 3.3 that follows).

3.3 Amazon S3

S3 allows storage and retrieval of any amount of data from anywhere on the Web. In Amazon S3, data is stored in *buckets*. A bucket[3] is a container for *objects* stored in S3 and it follows a specific naming convention. Objects[3] are fundamental entities stored in S3. They consist of data and metadata. The metadata is a set of name-value pairs that describe the object. Although developers can specify other metadata fields when storing the object, S3 default metadata fields include the date the object was last modified, and standard HTTP metadata such as Content-Type.

A unique identifier of an object within a bucket is called a *key*. Also referred to as the filename, exactly one key belongs to one S3 object. Thus, to uniquely identify each S3 object, the combination of a bucket, key, and version ID is used. It is

also important to choose a Region where created buckets will be stored. The S3 user guide API Version 2006-03-01 has an easy-to-use Web interface from which S3 services can be accessed [3].

Amazon S3 buckets can be created from the Web user interface provided by AWS. Any data stored on S3 can also be browsed through the same interface. A bucket can be deleted if it is empty. Buckets containing objects can only be deleted if the objects are deleted first.

Objects can be uploaded into an S3 bucket. Once uploaded, users can perform various operations on them. Users can download the object, edit the object properties, copy an object to a different location and even delete an object, if not needed. It is also possible to create folders in S3, to group objects just like creating folders on a computer system. In order to index and query S3 data and metadata, SimpleDB is used. Section 3.4 below gives an overview of SimpleDB.

3.4 Amazon SimpleDB

AWS offers a Web interface to create and store multiple data sets[8]. It simplifies querying of data stored in it. Amongst other operations on the data stored in it, SimpleDB supports a write operation (for example, `PutAttributes`) and two types of read operations[98], differentiated by calls to GetAttributes: *consistent read* - which ensures that the value returned always comes from the most recently completed write operation; and *eventually consistent read* - which does not guarantee this. The Amazon SimpleDB data model, which is different from the relational database data model, is discussed in Section 3.4.1 and further details on limits and queries on SimpleDB are discussed in Section 3.4.2.

3.4.1 Amazon SimpleDB Data Model

Amazon SimpleDB allows developers to organize their structured data in domains within which they can perform the following operations; put data , get data or run queries. Each domain has items in it that are described by attribute name-value pairs[35]. For instance, consider a spreadsheet analogy. The Amazon SimpleDB components are analogous with those of the spreadsheet[8] as shown in Figure 3.4.

- *Customer Account* is represented by the entire spreadsheet. It refers to the Amazon Web Services account to which all domains are assigned.
- *Domains* are represented by the domain worksheet tabs at the bottom of the spreadsheet. Domains are similar to tables that contain similar data. Queries are executed against a domain, but cannot be executed across different domains.
- *Items* are represented by rows. Items represent individual objects that contain one or more attribute name-value pairs.
- *Attributes* are represented by columns. Attributes represent categories of data that can be assigned to items.
- *Values* are represented by cells. Values represent instances of attributes for items. An attribute can have multiple values.

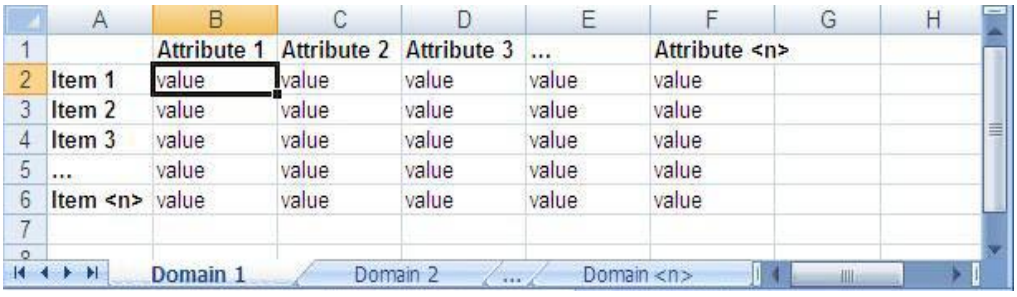


Figure 3.4: SimpleDB data model represented by a spreadsheet[8].

Table 3.2 shows sample data stored in Amazon SimpleDB.

ID	Car Make	Model	Year	Colour	Price	Registration Number
CAR_001	Toyota	Fortuner	2005	Red	R325K	CA149-851
			2009	Plum	R200K	CY149-851
			2012	Black	R400K	CA200-400
CAR_002	Volkswagen	Toureg	2010	Grey	R350K	CA201-401
			2006	Blue	R200K	CZ149-851

Table 3.2: Illustration of the Amazon SimpleDB data model.

3.4.2 Amazon SimpleDB limits and queries

The number of domains that can be created per account on Amazon SimpleDB is limited to 250, each domain size is up to 10GB and can store up to 1 billion attributes. A detailed list of all limits to be considered when working with Amazon SimpleDB can be obtained in [8].

Amazon SimpleDB supports `select`, which takes similar query expressions to the standard `SQL select` statement. The following is an example of a query expression in Amazon SimpleDB:

```
select <output_list>
from <domain_name>
[where <expression>]
[<sort_order>]
[limit <limit>]
```

In the `<output_list>`, the user can specify whether they want to retrieve all attributes, item names only, an explicit list of attributes or the total number of items available in a particular SimpleDB domain (represented by `count(*)` function). SimpleDB also supports sorting on a single attribute, in ascending or descending order.

3.5 Request and Response Handling on EC2, S3 and SimpleDB

Many services offered by AWS use both SOAP[9] and REST[10] API calls. However, Amazon SimpleDB does not support SOAP API calls as of June 1, 2012 [11]. Unless otherwise stated, examples of API calls that follow for EC2, S3 and SimpleDB use the REST API calls. It should be noted that EC2 and S3 still support SOAP API calls. The requests are sent in a URL encoded form as specified in RFC3986[25].

Section 3.5.1 that follows presents Amazon EC2 API requests and responses and Section 3.5.2 presents Amazon S3 API requests and responses.

3.5.1 Amazon EC2 API Requests and Responses

Amazon EC2 supports Query requests and SOAP requests. Query requests on EC2 are HTTP or HTTPS requests that use the HTTP verb GET or POST and a Query parameter named *Action*. For example, the EC2 command

```
ec2-run-instances
```

issued from the command line is encoded in a URL in a Web browser, using a GET request as follows:

```
https://ec2.amazonaws.com/?Action=RunInstances&ImageId=<image_id>&MaxCount=
<max_count>&MinCount=<min_count>&Placement.AvailabilityZone=<availability_
zone>&Monitoring.Enabled=<boolean_value>&AWSAccessKeyId=<ACCESS_KEY_ID>&Version
=<API_Version>&Expires=<timestamp>&Signature=<signature>&SignatureVersion=
<version_number>&SignatureMethod=<signature_method>
```

The first part of the URL (`https://ec2.amazonaws.com/`) represents the **endpoint**, which is the Web service entry point to act on. The **Action** (`?Action=RunInstances`) is the action performed on the endpoint.The remainder of the

[9] Simple Object Access Protocol

[10] REpresentational State Transfer

[11] `https://forums.amazon.com/ann.jspa?annID=1488`

URL is **parameters**, which are request parameters issued in the query. The <image_id> is the identity of the Amazon AMI from which the instances(s) are launched. The <boolean_value> can either be true or false depending on whether the instance is monitored or not. Instance monitoring is performed by viewing the status checks and scheduled events for an instance. A status check gives the information that results from automated checks performed by EC2. These automated checks detect whether specific issues are affecting a running instance. Detailed operational visibility of each instance is given by the status checks[5] as well as the data provided by Amazon CloudWatch[9]. The signature sent in the request is URL encoded as mentioned above. Detailed list of all EC2 actions is given in [5].

Amazon EC2 also uses the SOAP API calls to make requests. SOAP request messages use HTTPS and have to be hashed and signed for integrity and non-repudiation. In response to either a Query or a SOAP request, an XML data structure is returned. The EC2 response includes the request ID in the `requestId` element, which is a unique string assigned by AWS. The response structure is shown below:

```
<DescribeKeyPairsResponse xmlns="http://ec2.amazonaws.com/doc/2012-12-01/">
  <requestId>7a62c49f-347e-4fc4-9331-6e8eEXAMPLE</requestId>
  <keySet>
        <item>
              <keyName>gsg-keypair</keyName>
              <keyFingerprint>
                    00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
              </keyFingerprint>
        </item>
  </keySet>
</DescribeKeyPairsResponse>
```

The structure of an XML response is specific to the associated request. In general, the response data types are named according to the operation performed and whether the data type is a container (can have children)[5]. Examples of containers include `groupSet` for security groups and `keySet` for key pairs (see XML response above)[5]. Item elements are children of containers, and their contents vary

according to the container's role.

3.5.2 Amazon S3 API Requests and responses

Amazon S3 supports two types of requests: requests made using a user's AWS account (or IAM[12] credentials) and the REST API calls. The Amazon S3 REST API uses a custom HTTP scheme based on a keyed-HMAC[13] for authentication[61]. To authenticate a request, selected elements of the request are concatenated first to form a string. Then the AWS Secret Access Key is used to calculate the HMAC of that string [61]. and we call the output of the HMAC algorithm the "signature" because it simulates the security properties of a real signature. Finally, the signature is added as a parameter of the request.

An example of a request making use of an AWS account or AMI credentials given below uses the AWS S3 code and libraries provided on the AWS website (`http://aws.amazon.com/code/Amazon-S3`). The example uses Java code, which was also used to develop typical digital library services presented in this thesis as discussed in Chapter 4. Firstly, an instance of `AmazonS3Client` is created and then a **request** is sent to Amazon S3.

```
//user authentication
AWSCredentials myCredentials = new BasicAWSCredentials(myAccessKeyID,
mySecretKey);

//create an instance of AmazonS3Client
AmazonS3 s3client = new AmazonS3Client(myCredentials);

//send a sample request (list objects in a given bucket).
ObjectListing objectListing = s3client.listObjects(new
ListObjectsRequest().withBucketName(bucketName));
```

The Amazon S3 REST API calls use the following `HTTP` methods; `PUT`, `GET` and `DELETE`[54]. A `GET` request is used to get objects from an S3 bucket. A `PUT` request is used to upload objects into S3 buckets. A `DELETE` request is used to

[12]Identity and Access Management

[13]Hash Message Authentication Code

delete objects from S3. An example of a successful S3 REST request to create a bucket is depicted below:

```
PUT /onjava HTTP/1.1
Content-Length: 0
User-Agent: jClientUpload
Host: s3.amazon.com
Date: Wed, 02 Jan 2013 14:34:45 GMT+02:00
Authorization: AWS <ACCESS_KEY_ID:signature>
```

The response of the above successful request is as follows:

```
HTTP/1.1 200 OK
x-amz-id-2: <metadata_header>
x-amz-request-id: <request_id>
Date: Wed, 02 Jan 2013 14:34:01 GMT+02:00
Location: /onjava
Content-Length: 0
Server: AmazonS3
```

3.5.3 Amazon SimpleDB API Requests and Responses

Firstly, a SimpleDB request has the following set of **mandatory parameters**; *Action, AWSAccessKeyID,DomainName, Signature, SignatureVersion, Timestamp and Version.* SimpleDB has a set of **conditional parameters**, namely, *Attribute.X.Name, Attribute.X.Value, AttributeName, ItemName and SignatureMethod.* These are conditional because they are not applicable to some operations. For instance, the `Attribute.X.Value` parameter is required by `PutAttributes` and `BatchPutAttributes` and it is optional for `DeleteAttributes` and `BatchDeleteAttributes`. **Optional parameters** in SimpleDB are *Attribute.X.Replace, MaxNumberOfDomains, MaxNumberOfItems, NextToken and SelectExpression.*

Some important terminology related to Amazon S3 requests and responses handling is explained below.

- *Action*: this is the name of the action, for instance, `BatchPutAttributes`.
- *AWSAccessKeyID*: the user's AWS Access Key ID.

- *DomainName*: the name of the domain used in the operation.
- *Signature*: this is an HMAC-SHA1 or HMAC-SHA256 signature calculated using the user's Secret Access Key.
- *SignatureVersion*: the AWS signature version, which is currently the value 2.
- *Timestamp*: this is the day, date and time of the request.
- *Version*: this is the version of the API used. In this thesis, API Version 2009-04-15 has beed used.
- *Attribute.X.Name*: this is the name of the attribute associated with an item.
- *Attribute.X.Value*: this is the value of an attribute associated with an item.
- *AttributeName*: the name of the attribute to return. It is optional for the `GetAttributes` request.
- *ItemName*: this is a unique identifier of an item. It is required by the following operations: `PutAttributes`, `BatchPutAttributes`, `GetAttributes`, `DeleteAttributes` and `BatchDeleteAttributes`.
- *SignatureMethod*: this is required when using signature version 2 with REST requests. It explicitly provides the signature method HmacSHA1 or HmacSHA256.
- *Attribute.X.Replace*: this is a flag used to specify whether to replace an attribute/value or to add an attribute/value. It defaults to `false`.
- *MaxNumberOfDomains*: this is the maximum number of domain names that can be returned. For example, an operation like `ListDomains` uses `MaxNumberOfDomains`.
- *MaxNumberOfItems*: this is the maximum number of items to return in a response.

- *NextToken*: it is a string that tells Amazon SimpleDB where to start the next list of domain or item names.
- *SelectExpression*: it is a string that specifies the query that is executed against the domain.

With these definitions in mind, the next example uses `PutAttributes` on `CAR_001` from Table 3.2 using SimpleDB REST API to demonstrate a request and response. The item name is **CAR_001**, with attributes *car make, model, year, colour, price* and *registration number*. The first two attributes have a single value and the rest of the attributes are multivalued. The attribute **Year** has values `2005, 2009` and `2012`. **Colour** attribute has the values `red, plum` and `black`. Lastly, the attribute values for **registration number** are `CA149-851, CY149-851` and `CA200-400`.

```
https://sdb.amazonaws.com/?Action=PutAttributes&Attribute.1.Name=Make&Attribute.1.Value
=Toyota&Attribute.2.Name=Model&Attribute.2.Value=Fortuner&Attribute.3.Name=Year
&Attribute.3.Value=2005&Attribute.3.Name=Year&Attribute.3.Value=2009&Attribute.3.Name
=Year&Attribute.3.Value=2012&Attribute.4.Name=Colour&Attribute.4.Value=Red&Attribute.
4.Name=Colour&Attribute.4.Value=Plum&Attribute.4.Name=Colour&Attribute.4.Value=Black&
Attribute.5.Name=Price&Attribute.5.Value=R325K&Attribute.5.Name=Price&Attribute.5.
Value=R200K&Attribute.5.Name=Price&Attribute.5.Value=R400K&Attribute.6.Name=
RegistrationNumber&Attribute.6.Value=CA149-851&Attribute.6.Name=RegistrationNumber&
Attribute.6.Value=CY149-851&Attribute.6.Name=RegistrationNumber&Attribute.6.Value=
CA200-400&AWSAccessKeyId=[AWS_ACCESS_KEY_ID]&DomainName=CarModels&ItemName=CAR_001
&SignatureVersion=2&SignatureMethod=HmacSHA256&Timestamp=2013-01-11T15%3A03%3A05-07
%3A00&Version=2009-04-15&Signature=[valid_signature]
```

When the `PutAttributes` request has been successful, the following response is returned in an XML format:

```
<PutAttributesResponse>
       <ResponseMetadata>
          <RequestId>490206ce-8292-456c-a00f-61b335eb202b</RequestId>
          <BoxUsage>0.0000219907</BoxUsage>
       </ResponseMetadata>
</PutAttributesResponse>
```

`BoxUsage` gives an indication of the system resources that were utilized to complete a request.

3.6 Summary

In this chapter, an overview of the Amazon Web Services used in this thesis has been presented. It can be argued that Amazon Web Services offers a well-established, well-documented and stable platform to develop applications. A proof-of-concept application comprising typical digital library services was designed and developed to make use of the services presented here. The design and implementation details are given in Chapter 4 that follows.

Chapter 4

Digital Library Services

The basic research question this thesis attempts to answer is whether a typical digital library architecture can be migrated into a cloud environment and also to investigate the scalability issues involved when the demand for more data and services increases. With these research questions in mind, an experimental set of limited typical digital library services was designed and implemented on AWS. This chapter outlines the design and implementation of two typical digital library services: search and browse. The chapter also outlines how AWS services were integrated in the development of these services. Section 4.1 presents cloud computing architectural design considerations that are of importance to this thesis. The overall architecture of the systems is discussed in Section 4.2.

4.1 Architectural Design Considerations

This section discusses different architectures that can be used to implement scalable on-demand digital library systems on top of a cloud environment. Some of the architectures considered here emulate some of the parallel programming architectures such as shared-memory machines and distributed-memory machines. All such architectures are characterized by a master/manager paradigm. Typically, the master node (or server) (as shown in figures 4.1, 4.2, 4.3 and 4.4) is used to steer/proxy connections and manage application servers. The master achieves this via a Web user interface. It also handles requests/responses and monitors machines (represented by nodes numbered 1 to 4 in the diagrams that follow).

Four possible architectures that take advantage of scalability within utility clouds are illustrated and discussed in sections 4.1.1 (the proxy architecture), 4.1.2 (the redirector architecture), 4.1.3 (the round-robin architecture) and 4.1.4 (the client-side architecture) that follow. A further presentation and discussion of these design architectures can be found in [92].

4.1.1 The Proxy Architecture

This type of architecture, illustrated in Figure 4.1, works like the shared-memory machines. The “Master" or “Manager" node acts as a proxy for all external connections to and from the service nodes. It keeps track of which nodes are available at any given point in time. Should any node fail, the “master" or “manager" redirects connections/services to the available nodes. This architecture is characteristic of cluster computing.

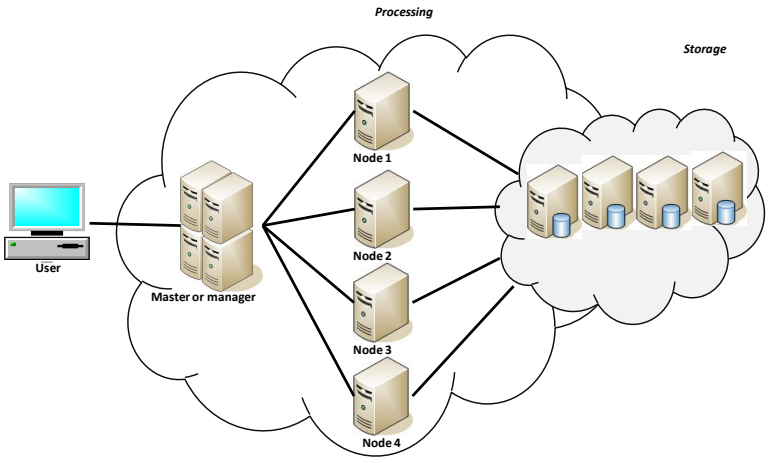


Figure 4.1: The Proxy Architecture: The “master" or “manager" acts as a proxy between users and the nodes in the cloud.

(a) **Advantage**: should any failures occur, there will always be continuity of service as requests will be redirected to the working and/or available nodes.

(b) **Disadvantage**: there is a possibility of a bottleneck on the “master" node during simultaneous service requests.

4.1.2 The Redirector Architecture

Figure 4.2 shows an architecture that emulates the grid. In this case the “master" or “manager" node serves as a lookup table for service nodes. External clients in this case make direct connections to the nodes in the cloud.

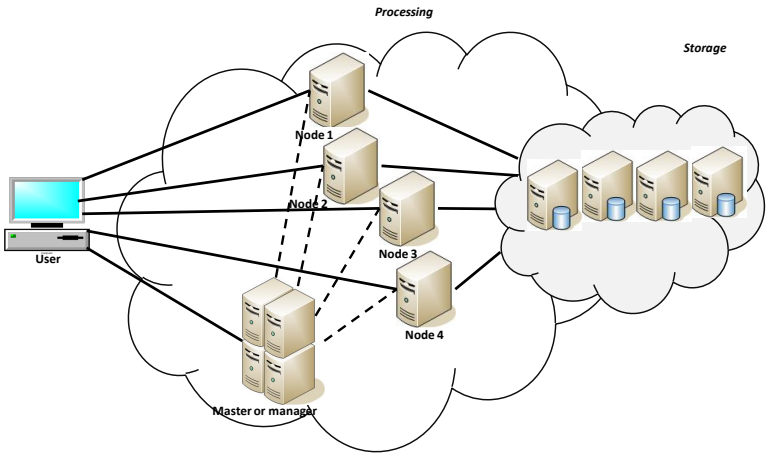


Figure 4.2: The Redirector Architecture: The “master" or “manager" acts as a look up table for service nodes and steers connections between users and the nodes in cloud

(a) **Advantage**: An obvious advantage is that there are fewer possible bottlenecks.

(b) **Disadvantage**: The possible disadvantage is that in the event of one machine failing, requests can still be sent to that machine if the requests are sent directly from the user instead of the “master" node.

4.1.3 The Round-Robin Architecture

In this scenario (Figure 4.3), clients use the DNS system or any similar address resolution system to obtain addresses of the next node to use in the cloud using a round-robin approach. The “master" keeps IP addresses of all machines and the information about which machine(s) has/have failed. Although easy to implement, round-robin DNS has problematic drawbacks such as those arising from

record caching in the DNS hierarchy itself, as well as client-side address caching and reuse, the combination of which can be difficult to manage. Thus, round-robin cannot be solely relied upon for service availability. An architecture of this nature is characteristic of Web server farms.

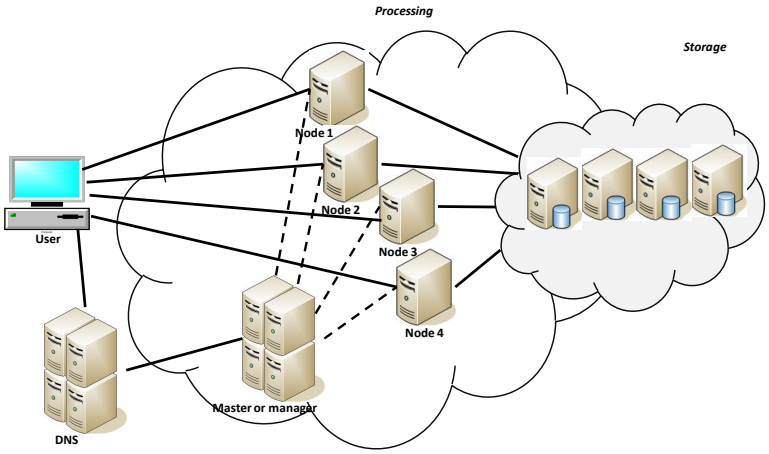


Figure 4.3: The Round-Robin Architecture: Clients use DNS system to obtain addresses of the next machine to use.

4.1.4 The Client-Side Architecture

In the architecture depicted in Figure 4.4, the “manager" or “master" keeps information about the state of all service nodes and can, therefore, send this list of nodes to the client when requested. This architecture is rather complex although it arguably provides high levels of scalability and reliability.

4.2 System Architecture

Digital libraries can be designed to take advantage of services provided by these architectures in order to achieve flexibility and high levels of scalability. An application was designed on top of a cloud environment using one of the architectures, particularly the proxy architecture with some elements of the client-side architecture. Figure 4.5 shows the overall architecture of the typical digital library services

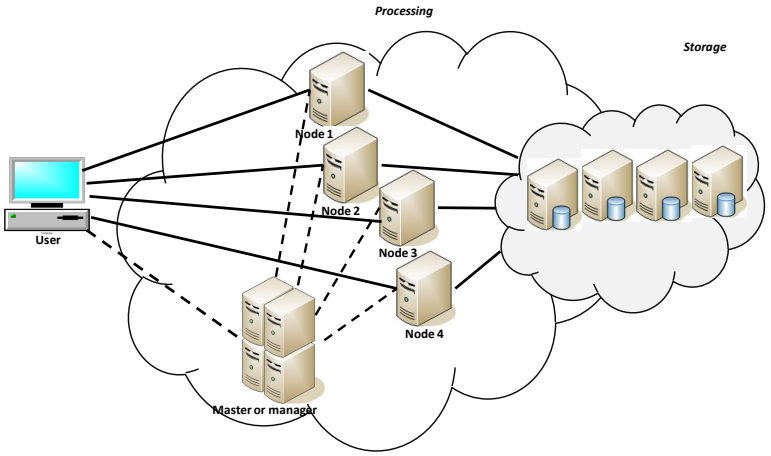


Figure 4.4: The Client-Side Architecture: The “master" or “manager" sends a list of nodes to the client upon request. This list is used to rotate requests to service nodes.

implemented. The Web user interface in Figure 4.5 is discussed further in Section 4.3. Typical digital library services (search and browse) are presented in Section 4.4.

4.3 The Web User Interface

The Web user interface is a light-weight process that provides an entry point for queries to the browse and search features [76]. The user interface is text-based and it issues queries and waits for responses. A query response consists of a list of document identifiers ranked by values which indicate the probability that the document satisfies the information need represented by the query. The query results are presented such that the highest ranked documents are at the top [76].

4.4 Typical Digital Library Services

This thesis looks into the implementation of two typical digital library services: browse and search. These services used the metadata harvested from two collections

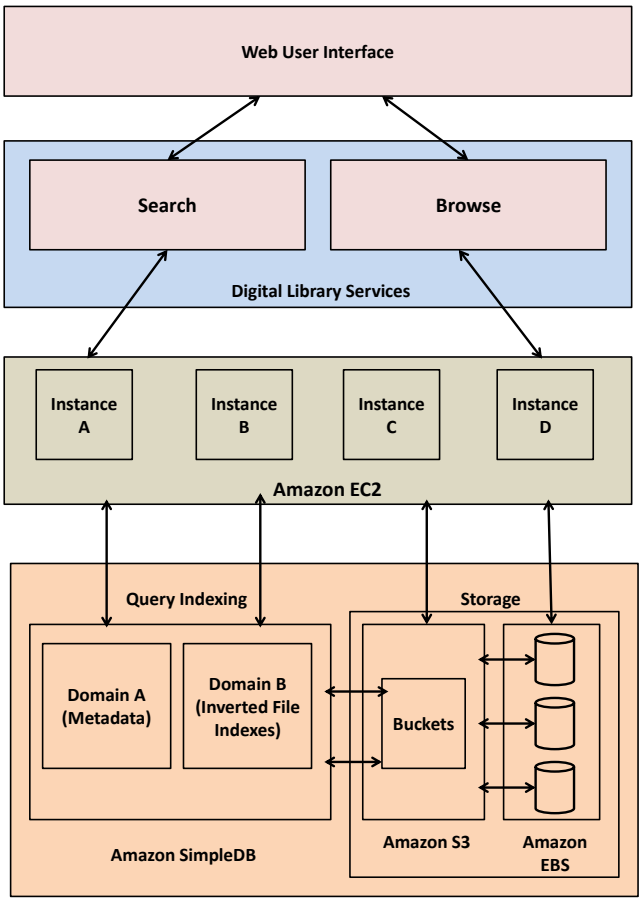


Figure 4.5: The high-level architecture of the system showing different components.

- The Networked Digital Library of Theses and Dissertations (NDLTD) and The South African National Electronic Theses and Dissertations (SA NETD) portal using the metadata harvester discussed in Section 4.4.3.

4.4.1 Browse

Paihama [81] defines browsing as the process of going through a collection of items using specific criteria to find the items of interest. In our experimentation, the criteria specified was the title of the document, the author and the date that the document was published. In all the three criteria, the collection can be browsed in ascending/descending order.

The browse feature is accessible through the light-weight Web user interface described in section 4.3 and accepts users' queries to browse the collection by author, title and date. When the collection is browsed, users select the criteria they wish to browse by. The results of the browsing request are displayed back on the Web user interface. The application uses SimpleDB for indexing and querying operations. Communication between the application and SimpleDB is over the REST[1] API.

On the front end, a user's browse request is submitted in a URL of the form:

```
ec2-50-17-126-167.compute-1.amazonaws.com:8080/SimpledbBrowsinServlet/Index?act
ion=browse&category=<browse_category>&order=<browsing_order>
```

The first part of the URL, `ec2-50-17-126-167.compute-1.amazonaws.../Index`, is the endpoint, followed by the *Action* (`&action=browse`) and the parameters (*category (title, author or date)*) and *order (ascending or descending)* in the browse request.

The back-end processing is more elaborate. When a request is send to SimpleDB, the following action is executed:

```
select * from <domain_name>
where itemName()=<item_name>
order by <sort_order>
limit <limit>
```

The `select` operation returns a set of `Attributes` for `ItemNames` that match the select expression. In addition to this, the response metadata is also returned in the response. This includes the *RequestId* and *BoxUsage*. `BoxUsage` refers to the system resources utilized during query execution. Queries with a higher `BoxUsage` value are the more expensive queries. The general response XML from a SimpleDB `select` request is as follows:

```
<SelectResponse>
       <SelectResult>
         <Item>
              <Name>item_name</Name>
              <Attribute><Name>attribute_name</Name><Value>attribute_value</Value>
```

[1] REpresentational State Transfer

```
        </Item>
      </SelectResult>
      <ResponseMetadata>
            <RequestId>request_ID</RequestId>
            <BoxUsage>amount_of_system_resources_used</BoxUsage>
      </ResponseMetadata>
</SelectResponse>
```

A `select` operation uses `NextToken` to tell SimpleDB where to start the next list of domain or item names. If `ListDomains` and/or `GetAttributes` are called successive times with the `NextToken`, they will return up to `MaxNumberOfDomains` more domain names and `MaxNumberofItems` more item names respectively, each time. A SimpleDB `select` operation is limited to 2500 items per request. If the limit is specified in the `select` expression, results will be returned with the `NextToken`. Due to the nature of the SimpleDB data model with the possibility of some attributes having multiple values, each time a request is issued, the collection is browsed from the beginning but the results are displayed starting from the last `NextToken` that was returned.

For example, if the collection is to be browsed in descending order of *titles* with 50 results to be displayed per page, then the following `select` expression is issued:

```
      select * from DLCloud
      where itemName() is not null
      order by Title desc
      limit 50
```

The `select` response of the results returned is in an XML format. The snapshot of the results, shown below, represents a sample of one of the records retrieved in an XML format.

```
<SelectResponse>
  <SelectResult>
      <Item>
        <Name>oai_techreports.cs.uct.ac.za_102</Name>
        <Attribute><Name>Title</Name><Value>A Lightweight Interface to Local Grid
            Scheduling Systems</Value>
        <Attribute><Name>Author</Name><Value>Christopher Parker</Value>
```

```
        <Attribute><Name>Description</Name><Value>Many complex research problems
            require an immense amount of computational power to solve. In order to
            solve such problems, the concept of the computational Grid was conceived.
            Although Grid technology is hailed as the next great enabling technology in
             Computer Science, the last being the inception of the World Wide Web, some
             concerns have to be addressed if this technology is going to be successful
            .</Value>
        <Attribute><Name>Date</Name><Value>May 2009</Value>
        <Attribute><Name>Identifier</Name><Value>http://pubs.cs.uct.ac.za/archive
            /00000521/01/thesis.pdf</Value>
      </Item>
  </SelectResult>
  <ResponseMetadata>
       <RequestId>b1e8f1f7-42e9-494c-ad09-2674e557526d</RequestId>
       <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</SelectResponse>
```

The XML file returned is parsed and displayed to the user in a readable manner. The application uses Java servlets to achieve this. The application uses the IAM credentials for communication with AWS in order to parse the returned XML for display on the Web user interface.

4.4.2 Search

Search is the process by which a client can retrieve an item with specified properties among a collection of items stored on a server [81]. A data collection (or collection of documents) stored on S3 is indexed by SimpleDB. SimpleDB indexes metadata on full-text of the relative documents. Figure 4.6 shows how files and their metadata are stored on S3 and SimpleDB respectively. When the collection is searched, the indexing is performed by SimpleDB, as presented in Section 4.4.2.1. Inverted index on SimpleDB is discussed in Section 4.4.2.2. Index maintenance during document additions, modifications and deletions is discussed in Section 4.4.2.3. An algorithm for querying SimpleDB is presented in Section 4.4.2.4.

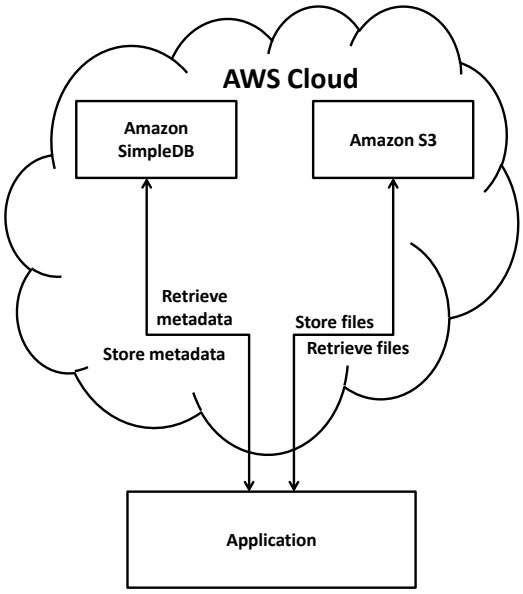


Figure 4.6: Illustration of data and metadata storage in the AWS cloud using S3 and SimpleDB [35].

4.4.2.1 Metadata Indexing

The main focus of this thesis is not development of new information retrieval algorithms. Despite this, there are some considerations in developing a search feature on top of SimpleDB. Existing information retrieval engines such as Lucene [65] can not be used to develop the searching and indexing components on SimpleDB because of SimpleDB's data model, as discussed in Chapter 3.

In order to make querying faster, an *index* is used [48]. An index is a data structure that maps *terms* to the *documents* that contain them. With an index, query processing can be restricted to documents that contain at least one of the query terms. Many different types of indices exist. The most efficient index structure for text querying is *an inverted index* [48]. An inverted index is a collection of lists, one per term, recording the identifiers of the documents containing that term. Both metadata and full text of documents are indexed by SimpleDB. A detailed discussion of indexing and inverted files can be found in Frakes et. al [48].

SimpleDB is arguably a persistent hashtable of hashtables. Each row, also called an

item in the outer hashtable, has another hashtable with up to 256 key-value pairs, called *attributes*.

4.4.2.2 Inverted Index on SimpleDB

An *inverted index* (or *inverted file*) representation on SimpleDB can be performed by mapping the inverted file on top of the attributes. That is, have one SimpleDB domain with one word (term). Then the attributes store the list of URLs (or documents) containing that word. A single URL (or document) contains many words in it, therefore, it is important to have a separate SimpleDB domain containing a mapping from a hash of URL to URL and use the hash URL in the inverted file. This helps keep the inverted file relatively smaller.

To achieve this, SimpleDB uses `CreateDomains` to create the domains (if this is the first time indexing occurs), a domain of terms and a domain of URLs and `PutAttributes` to add terms (attributes) to the domains. To avoid re-inventing the wheel, Ferret's default Analyzer [67] was used to tokenize the input and filter the terms. Stemming of terms uses the Porter Stemming algorithm [99]. Algorithm 4.1 depicts creation of an inverted file entry on SimpleDB.

Algorithm 4.1 *Inverted index on SimpleDB*

```
1: procedure ADDINVERTEDFILEENTRY(term, vector)
                /* read the index object from SimpleDB */
                /* extract the entire text from a given document */
                /* break the text into tokens/terms (SimpleDB attributes) */
                /* filter stop words from the terms */
                /* apply stemming to the terms */
2:      current_term ← null
3:      current_term_line ← ∅
4:      last_saved_term ← null
                /*iterate on character level */
```

```
5:    for each stemmed version of the term, enumerate(vector) do
6:        index_term, term_line ← getTermLine(term, position)
7:        if current_term ≠ index_term and current_term ≠ null then
8:            make a new entry of the term in the domain of terms
9:            last_saved_term ← current_term
10:       end if
11:       current_term_line ← term_line
12:       current_term ← index_term
13:       current_term_line[bucket] ← current_term_line.get(bucket, "  ")
14:       save the entry in inverted file
15:       if current_term ≠ last_saved_term then
16:           add it to the inverted file.
              /* add current term, update current term line */
17:           last_saved_term ← current_term
18:       end if
19:   end for
20: end procedure
              /* save index object to SimpleDB */

end
```

4.4.2.3 Index Maintenance on SimpleDB

When the data collection changes, the index needs to be updated to reflect the changes in the collection [76]. Index update on SimpleDB involves **addition** of documents, **modification** of documents and **deletion** of documents.

SimpleDB uses `PutAttributes` and `BatchPutAttributes` calls for document additions and modifications. The two API calls differ in that `BatchPutAttributes` is used to generate multiple put operations in a single call. With document additions, new documents are added to the index object on SimpleDB. For document modifications, the original entry of the item to be modified is first deleted

from the index using the `DeleteAttributes` call, then the new version of the object is added to the index object, and stored on SimpleDB. Modifying a SimpleDB item that does not exist causes an error. Furthermore, because SimpleDB makes multiple copies of the data and uses the *eventual consistency* update model (see Chapter 3), an immediate `GetAttributes` or `Select` operation (both read operations) after a `PutAttributes` or `DeleteAttributes` operation (both write operations) might not return updated data.

The deletion of documents (using `DeleteAttributes` or `BatchDeleteAttributes` calls) is whereby entries of documents deleted from SimpleDB are deleted from the index object. `DeleteAttributes` will delete an item if all its attributes have been deleted. If `DeleteAttributes` is called without being passed attributes or values specified, all the attributes associated with that specific item are deleted. However, `DeleteAttributes` is *idempotent* because running it multiple times on the same item does not cause an error.

4.4.2.4 Querying

SimpleDB allows queries to be performed through the Query API operation on one domain at a time. This service understands queries expressed in a simple language specific to SimpleDB. SimpleDB responses do not include attribute values, meaning that when a SimpleDB query operation is issued, the service returns a set of item names that match the query. These item names do not include the items' attribute names or values, which means that a follow up `GetAttributes` operation is required. The `GetAttributes` operation retrieves the attributes stored in a named item. For instance, if a query returns 50 items, then the application has to perform 50 subsequent `GetAttributes` operations to retrieve all the data.

SimpleDB does not support sorting of query results, which means that the results are returned in their lexicographical order. The result sorting has to be explicitly specified by the application. Therefore, the results sorting is done at inverted file

level.

There is a time limit imposed by SimpleDB on the queries issued and if a query exceeds this limit, the entire query operation fails and the service returns a `RequestTimeout` error message. Any partial results that may have been generated before the timeout occurred are not returned.

Queries are issued from the Web user interface (section 4.3). The user's request is sent in a URL of the form:

```
ec2-50-17-126-167.compute-1.amazonaws.com:8080/SimpledbBrowsinServlet/Index?act
ion=search&query=computer+science&searchType=title
```

The query text (or sentence) is tokenized to obtain all terms (attributes) in the order in which they appear in the sentence. The stop words are removed and the stemming process is applied to the terms. For each stemmed term, if the index of the term already exists in the domain for terms, then its URL hash is retrieved, and a node is added to its posting list. Since each SimpleDB item is limited to 256 attribute-value pairs, new items are created each time the current item reaches the upper limit (256). A SimpleDB response is 1MB in size and if the response is larger than 1MB, a `nextToken` is returned. This `nextToken` will be used to get remaining results until all results are displayed (paging). Algorithm 4.2 shows pseudocode for querying SimpleDB.

Algorithm 4.2 *Querying SimpleDB*

```
1: procedure QuerySimpleDB(query)
2:     URL_hash_of_terms ← ∅
3:     terms ← query.tokenize()
4:     for term in terms do
5:         inverted_file_entry ← this.sdb_index.getInvertedFileEntry(term)
               /* print inverted file entry */
6:         URL_hashes ← extractHashListFromInvertedFile(
               inverted_file_entry)
```

```
            /* print URL hashes
 7:         for URL_hash in URL_hashes do
 8:             URL_hash_of_terms[URL_hash] ← URL_hash_of_terms.get(
            URL_hash, [ ]) + [term]
 9:         end for
10:     end for
11:     for URL_hash in URL_hash_of_terms do
12:         results ← (length(URL_hash_of_terms[URL_hash], URL_hash))
13:     end for
            /* printing inverted file results and URLs with matches */
14:     results ← [ ]
15:     for matches, URL_hash in results do
16:         look up URL with URL_hash
17:         print URL_hash
18:     end for
19:     return results
20: end procedure
```

end

The algorithm for extracting the URL hash list of pages from the inverted file entry is given by the pseudocode of Algorithm 4.3.

Algorithm 4.3 *Extracting URL hash list from an inverted file entry*

```
1: procedure EXTRACTHASHLISTFROMINVERTEDFILE(inverted_file_entry)
2:     pages ← inverted_file_entry.split(pageSeparator)
           /*extract positions */
3:     URL_hash_list ← [ ]
4:     for page in pages do
5:         if page ← empty then
6:             continue
```

```
7:          end if
                /* append page URL hashes to the the URL hash list */
8:          URL_hash_list ← URL_hash_list + [page.split(
                positionSeparator)[0]]
9:      end for
                /* sort the URL hash list */
10:     URL_hash_list.sort()
11:     return URL_hash_list
12: end procedure
```

end

Depending on the query complexity, there is a single SimpleDB access or multiple accesses. Index merging involves concatenating posting lists that belong to the same term into a single domain, resulting in one single index instead of multiple indices.

4.4.3 The Metadata Harvester

The Metadata harvester is a simple Perl script that is used to obtain the metadata records from the two collections namely, the Networked Digital Library of Thesis and Dissertations (NDLTD) and the South African National Electronic Thesis and Dissertations (SA NETD). The harvested metadata files were then ingested into S3. At the time of metadata harvesting, the best way available to ingest the files into S3 was through JetS3t[2]. JetS3t is a free, open-source Java toolkit and application suite for S3, CloudFront content delivery network and Google Storage for Developers. The JetS3t toolkit provides Java programmers with a powerful yet simple API[3] for interacting with storage services and managing data stored there. However, due to recent developments and API version upgrades, S3 has a simple easy-to-use Web

[2] `http://www.jets3t.org`

[3] Application Programming Interface

user interface provided by the AWS management console from which data can be ingested. Figure 4.7 shows a simple metadata harvester that stored the harvested metadata files on Amazon S3.

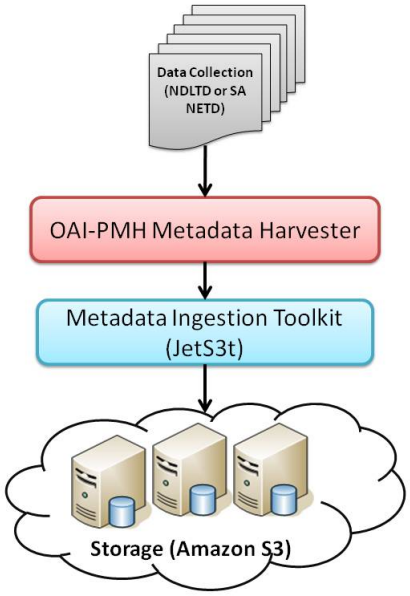


Figure 4.7: Metadata harvesting from NDLTD and SA NETD.

4.5 Summary

Cloud architectures provide an opportunity and platform for digital library design and development over existing cloud platform APIs. The digital library services presented in this thesis used some AWS services such as EC2, S3 and SimpleDB. Despite its ability to simplify querying and indexing of structured data, SimpleDB's data model remains complex in terms of the ease with which querying can be done. SimpleDB allows querying on attribute name level and, therefore, results sorting remains an open question in SimpleDB. In order to sort by value, the developer has to explicitly develop a sorting method using any of the existing efficient sorting algorithms. Also eventual consistency may entail some issues. "Real-time" updates may be a requirement for DLs. For example deletion of a document, update of a

licence or addition of a document. The next chapter presents the evaluation of the services described here and discusses the findings of the evaluation experiments.

Chapter 5

Evaluation

This chapter presents and discusses how the application comprising typical digital library service components was tested and evaluated. One of the important aspects that the evaluation addresses is the scalability issues involving deployment of digital library services in the cloud. Section 5.1 gives and overview of the resources used in the setup of the experiments. A brief overview of experiments is given in Section 5.2. The three experiments performed in this thesis are presented and discussed in sections 5.3, 5.4 and 5.5

5.1 Experimental Setup

For performance evaluation, the application was deployed on an Amazon EC2 instance. Apache JMeter was used for simulation of users and user requests. For these experiments, an Amazon EC2 instance of type t1.micro was launched for server-side processing. For simplicity purposes, a 32-bit Ubuntu AMI was used because it had the same architecture as the desktop computer on which the application was developed prior to deployment on EC2. The firewall was configured to open ports 22 for SSH, 80 for HTTP, 8080 for Glassfish and 4848 for Glassfish Admin, then the instance was launched. A key-pair was created and downloaded to the local machine.

Glasshfish is an open source, production-ready, Java Enterprise Edition-compatible application server [1]. The application developed used Glassfish as the Web server.

[1] `http://glassfish.java.net/public/getstarted.html`

Therefore, as a requirement and for compatibility purposes, Glassfish was installed on all EC2 instances used in the experiments that follow.

Apache JMeter (hereinafter referred to as JMeter) [16], a 100 percent pure Java application designed to test and measure performance, was used for load testing of the application by recording the time it takes to perform a search and/or browse query. It may be used as a highly portable server benchmark as well as multi-client load generator [16].

It should be noted that “thread group" in the following refers to the simulated number of users accessing the Web application. “Loop Count" represents the number of times the Web application was accessed by the simulated number of users. In JMeter terminology, “ramp" up defines the amount of time between start up of threads [80]. JMeter also has logic controllers that determine the order in which the **samplers** are processed. Samplers tell JMeter to sent requests and wait for a response and they are processed in the order in which they appear in the tree. The Logic controller used in the experiments that follow was the “Random controller", which alternates among each of the other controllers for each loop iteration, picking one at random at each pass. *Samplers* perform the actual work on JMeter by generating one or more sample results.

For the purpose of presentation of experimental results that are discussed in the sections that follow, the following listeners were added to the thread group on JMeter: view results in table, aggregate report and view results tree. *View Results in Table* is a listener that creates a row for every sample result, making it easy to generate Excel spreadsheets for analysis of results. *Aggregate Report* creates a table row for each differently named request in a test. It gives the totals of the response information and provides request count, minimum, maximum, average, error rate, approximate throughput (request/second) and Kilobytes per second throughput. *View Results Tree* displays a tree of all sample responses, allowing for easy viewing of responses of any sample.

These listeners were able to give a better indication of performance of the Web

application because they produce a tabular representation of results. The actual data from the experiments was analyzed using an Excel spreadsheet. The graphs of time taken to process each request against the requests' timestamps were plotted using the actual data from these experiments.

- **Ubuntu Instance - Server Configuration**: An Amazon EC2 instance was launched using the AWS management console. Once the instance was running, an SSH connection to the server was opened[2]. For example,

  ```
  ssh ubuntu@ec2-23-20-202-114.compute-1.amazonaws.com
  ```

  An SSH client was used with the information provided, that is, the EC2 keypair that was downloaded to the local machine and was associated with the instance launched. The first time the connection was made, Ubuntu needed to be updated with the latest changes using the commands:

  ```
  sudo add-apt-repository deb http://archive.canonical.com/ oneric partner
  sudo apt-get update
  sudo apt-get upgrade
  ```

  Since Glassfish server 3.1 was the server used to run the application on the local machine, it was then downloaded and installed on the instance at this stage:

  ```
  wget http://dlc.sun.com.edgesuite.net/glassfish/3.1/release/glassfish-3.1.zip
  unzip glassfish-3.1.zip
  ```

- **Ubuntu Instance - Install Oracle Java JDK7 via PPA**: The Oracle Java JDK 7 was required because it includes JRE and the Java browser plugin. It was installed via the PPA[3] because this provides the full Oracle JDK7 package. The following commands were used to add the PPA and install the latest Oracle Java (JDK) 7 in Ubuntu:

[2] `http://www.cecs.csulb.edu/~monge/classes/423/2011Spring/project/aws_ec2_glassfish_instance.html` (Last accessed on January 2, 2013)

[3] `http://www.webupd8.org/2012/01/install-oracle-java-jdk-7-in-ubuntu-via.html?m=1` (Last accessed on January 2, 2013)

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-jdk7-installer
```

- **Ubuntu Instance - Glassfish Configuration and Launch**: The Glassfish tool pkg was run as follows:

  ```
  ~/glassfish3/bin/pkg
  ```

  Finally Glassfish was started with the following command:

  ```
  ~/glassfish3/glassfish/bin/asadmin start-domain --verbose
  ```

  Now, using a Web browser, the Glassfish administration console was opened and configured with the security realm and JDBC resources. The Web application to be run on the EC2 instance was deployed from the Glassfish admin console.

5.2 Overview of Experiments

To determine service and data scalability needs of the application, a series of experiments were carried out:

(a) **Service scalability** - performance testing of the application for its ability to handle numerous search and browse requests (response time) using multiple varying number servers. In the light of this, experiments were carried out to determine the response time for searching different categories of words and browsing the collection using different criteria. In particular, timed-experiments were carried out to search for popular words, unpopular words, multiple popular words, multiple unpopular words and a hybrid of popular and unpopular words to determine if the response times are stable and if there are variations that occur because of multiple server front-ends.

(b) **Data Scalability** - performance testing of the application for its ability to handle searching and browsing functions over varying collection sizes for a fixed number

of servers. The response time was recorded for both searching and browsing through data collections of different sizes. The results and interpretation are elaborated under section 5.4

(c) **Processor load testing**: A set of experiments in this category were aimed at determining the behaviour of the application when subjected to a large number of concurrent requests.

5.3 Experiment 1: Service Scalability

5.3.1 Search

The aim of this experiment is to investigate the time taken for a DL based on AWS to respond to **search requests** in a server farm configuration. This experiment determines if the response times are stable and if there is variation that occurs because of the multiple server front-ends.

5.3.1.1 Methodology

JMeter was set up to simulate fifty (50) users accessing one Web service ten (10) times for the search function. The Web service was hosted on four (4) identical Amazon EC2 instances (servers). In order to determine the order in which the Samplers are processed a logic controller was added. In this particular case, a Random Controller. The Random Controller picks a random sampler or sub-controller at each pass, so all the servers had an equal chance of being selected for processing. The 500 (50 users x 10 requests per user) requests were distributed amongst the four servers.

In the case of a search, the experiments were carried out at least five times for the five different types of queries: popular words, unpopular words, multiple popular words, multiple unpopular words and a hybrid of popular and unpopular words.

Query complexity was determined based on the size of the inverted file size when results were retrieved from S3.

(a) *Popular words*: This part of the experiment was divided into two categories. The first category was the case where one hundred (100) results were displayed per page. The second category was whereby all results were displayed on one page. When a query of a popular word (for example, “computer") was submitted, the results were returned in an inverted file from S3. In general, larger inverted files contain the most popular words in documents. Therefore, the largest inverted file was the one with more occurrences of the popular words in a query string.

(b) *Unpopular words*: Contrary to popular words, unpopular words (for example, “immunochemical") were contained in the smallest inverted files when results were retrieved from S3.

(c) *Multiple popular words*: An example of multiple popular words can be “cloud computing". Retrieval of the words in this type of a query requires multiple SimpleDB operations. SimpleDB uses the following operators to combine expressions: INTERSECTION, OR, AND, NOT and UNION in the case of multiple-word search. Multiple words were obtained from the largest inverted files containing individual words in a query string and the results were merged using one of the afore-mentioned SimpleDB operators, depending on the nature and type of query.

(d) *Multiple unpopular words and/or a hybrid of popular and unpopular words*: Examples of multiple unpopular words and a hybrid of popular and unpopular words are “carbohydrate conformations of pneumococcal antigens" and “computational chemistry and immuno-chemical dynamics", respectively. These category of words were obtained from combinations of small and large inverted files containing individual words in the query string and the results were merged

using the operators mentioned before. These queries also involved multiple SimpleDB operations.

Since the experiments described above were executed at least 5 times, the averages of the times were computed and used in analysis of the results.

5.3.1.2 Results and Discussion

The results are shown in Figure 5.1. The graph in Figure 5.1(a) was obtained from the average response times of the 500 requests. Figure 5.1(b) shows the case in which the data was partitioned into an average of blocks of 50 requests.

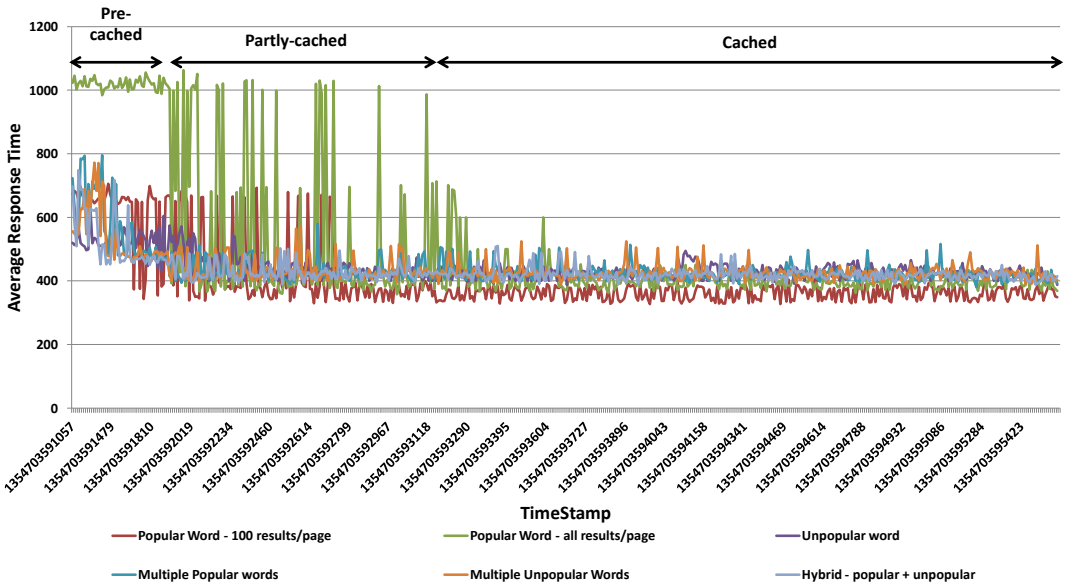
The results show that there was a noticeable time to connect to AWS services (see “pre-cached" part of the graph in 5.1(a)) at the start of all experiments. The oscillations in response times for queries of different complexities suggest that there were multiple AWS back-end servers with caches of data that were not completely shared. The results further show that there was stable response times and that there was some variability because of multiple server front-ends but nothing significant after the initial cache priming stage (see “cached" section of the graph in 5.1(a)).

5.3.2 Browse

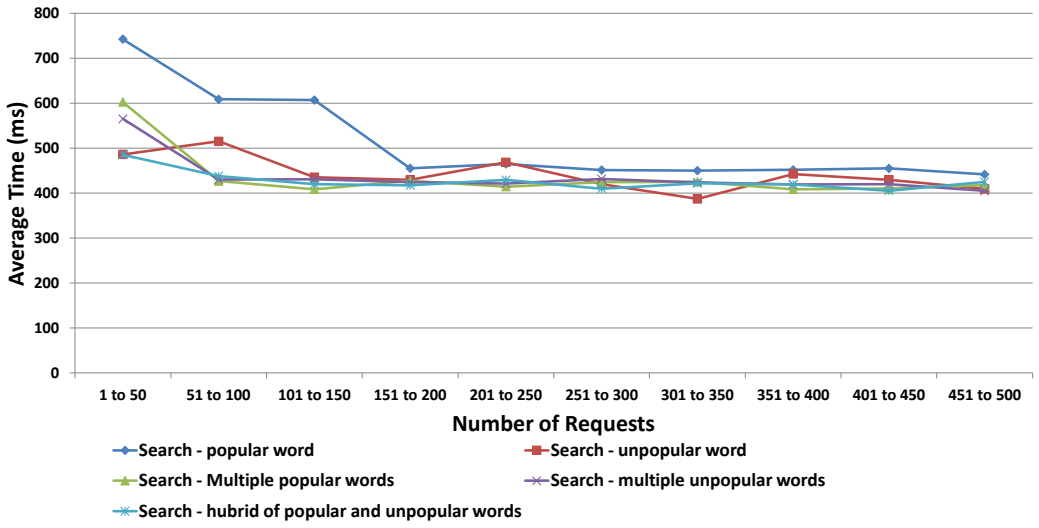
The aim of this experiment is to investigate the time taken for a DL based on AWS to respond to **browse requests** in a server farm configuration. This experiment was also performed to determine if the response times are stable and if there is variation that occurs because of the multiple server front-ends when browsing the collection.

5.3.2.1 Methodology

JMeter was set up to simulate fifty (50) users accessing one Web service ten (10) times for the browse function. The Web service was hosted on four (4) identical



(a) Combined graph of the average response time in milliseconds (ms) vs. request timestamp when processing queries of different complexities.



(b) Graph of average response time in milliseconds (ms) vs. the number of requests when processing queries of different complexities.

Figure 5.1: Results obtained from processing queries of different complexities.

Amazon EC2 instances (servers). In order to determine the order in which the Samplers are processed a logic controller was added. In this particular case, a Random

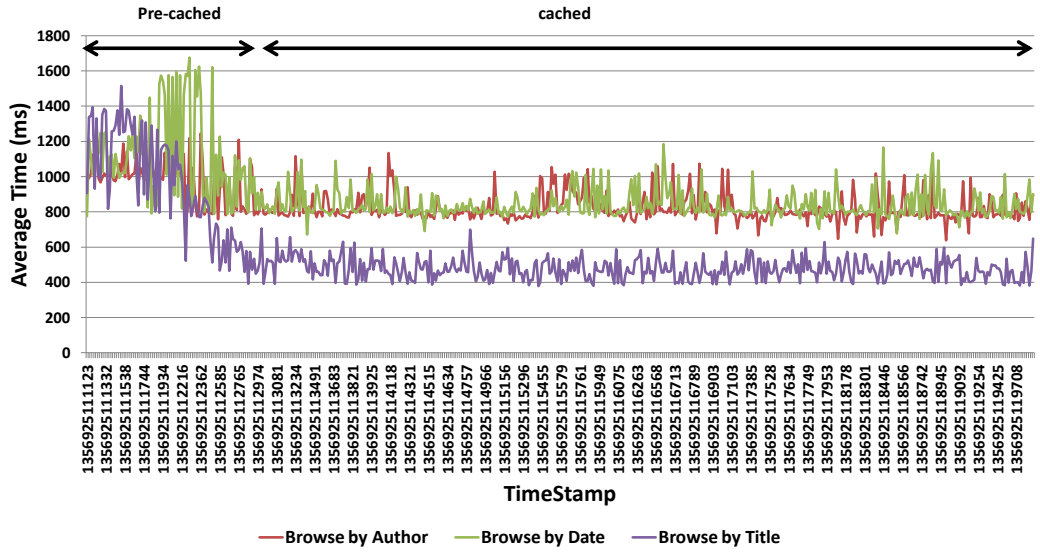
Controller. The Random Controller picks a random sampler or sub-controller at each pass, so all the servers had an equal chance of being selected for processing. The 500 (50 users x 10 requests per user) requests were distributed amongst the four servers.

The experiments were repeated at least 5 times for each browsing criteria. The averages were computed from the experiment runs and for each browsing category a graph of time (in milliseconds) against request timestamp was plotted. A further comparative analysis of the three browsing categories was carried out by partitioning the results into blocks of 50 and taking the average of each block. This gave a better comparison of the average response time against the number of requests for all the three browsing categories (author, title and date).

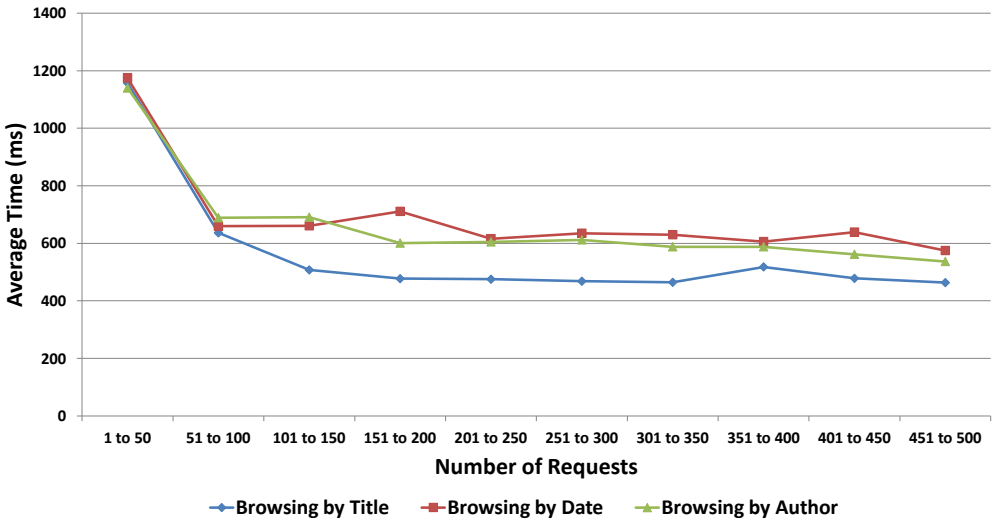
5.3.2.2 Results and Discussion

The results of this experiment are as shown by Figure 5.2 that follows.Figure 5.2(a) shows a case whereby the average response times of the 500 requests were used to generate the graph of average response time against request timestamp. The graph of Figure 5.2(b) shows the case whereby the average of blocks of 50 requests was used to generate the graph.

The results show that the average response time is slightly higher when browsing the collection. This could be due to the size of the response returned by a browse request. However, as in the case of a search operation, the results show that there was a noticeable time to connect to AWS services (see “pre-cached" part of the graph in 5.2(a)) at the start of all experiments. The oscillations in response times for different browsing criteria suggest that there were multiple AWS back-end servers with caches that were not completely shared. The results further show that there were stable response times and that there was some variability because of multiple server front-ends but nothing significant after the initial cache priming stage (see “cached" section of the graph in 5.2(a)).



(a) Graph of average response time in milliseconds (ms) vs. request timestamp obtained from browsing the collection by different criteria.



(b) Combined graph of time in milliseconds (ms) vs. the number of requests processed when browsing by different criteria.

Figure 5.2: Results obtained from browsing the collection by different criteria.

5.3.3 Varying the number of EC2 instances

The aim of this experiment is to determine if the number of instances on which the application was run does affect the response time when browsing or searching SimpleDB data.

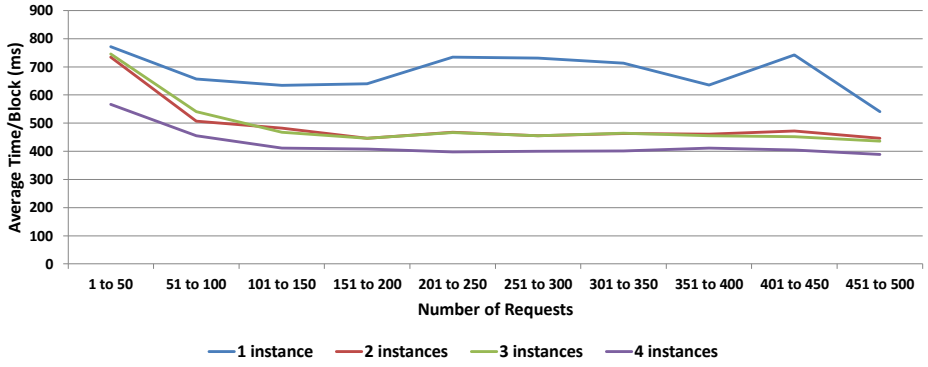
5.3.3.1 Methodology

The experiment was setup such that JMeter simulated 50 users accessing one Web service 10 times as in the previous experiments. The Web service was hosted on one, two, three and four EC2 instances respectively. In all the four cases, the experiment was repeated at least 5 times and the average response time was computed. For the case of one EC2 instance, there was no need for load distribution so the Random Controller was not used. However, for the remaining cases (2, 3, and 4 instances), the Random Controller was used in order to distribute the load evenly to all the EC2 instances. The results of the experiment are represented graphically in the section that follows.

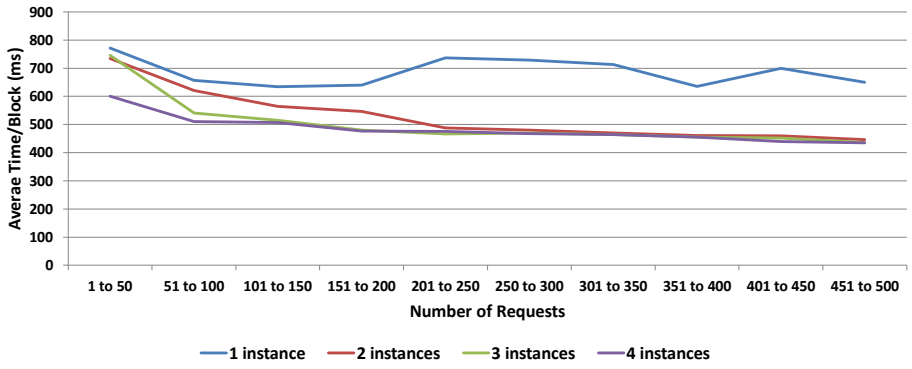
5.3.3.2 Results and Discussion

The results of this experiment are represented by the graphs of Figure 5.3. For further analysis and interpretation purposes, the results were divided into blocks of 50 requests per block. The average of each block was computed and the graphs of average response time (in milliseconds) vs. number of requests was generated. The results of all the different numbers of instances were plotted on the same graph in order to have a clear picture of the comparison of performance when using different numbers of instances.

Figure 5.3(a) depicts the results obtained from varying the number of instances when searching the collection. In this case, the average response time is significantly higher when using one instance and it increases slightly as the number of



(a) Graph of time in milliseconds (ms) vs. number of requests when searching the collection over a varying number of EC2 instances.



(b) Graph of time in milliseconds (ms) vs. number of requests when browsing the collection over a varying number of EC2 instances.

Figure 5.3: The results obtained from varying the number of instances when browsing and searching the collection.

requests increases. This is caused by a possible bottleneck as the instance (server) gets over-whelmed with numerous requests, thus slowing it down. There is no distinct difference in response time between 2 and 3 instances. The performance of the application is similar in both cases and an additional fourth server shows a significant performance improvement. The average response time is lowest when using 4 instances, showing that the number of instances does impact on the application performance when searching a digital collection.

Figure 5.3(b) on the other hand, shows a scenario in which the number of instances was varied when browsing the collection. The graph shows that using one instance

suffers possible performance bottlenecks and therefore the average response time is higher than all the other 3 cases (2, 3 and 4 instances). With browsing, the difference is significant at the start up to 200 requests, after which there is no major difference when using 2, 3 and 4 instances. However, a closer look at the results indicated that the case of four instances show better performance in comparison to all the other cases with an average of 483 milliseconds.

The speedup graph obtained when the number of EC2 instances was varied is shown in Figure 5.4.

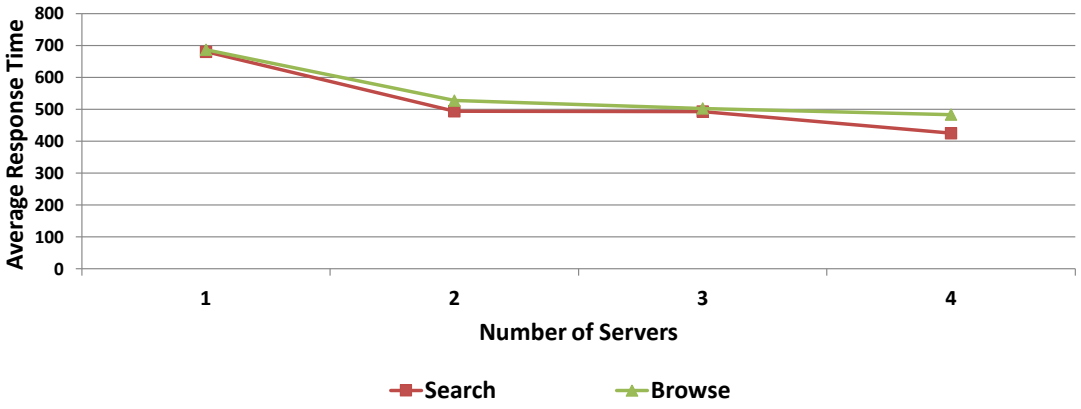


Figure 5.4: Speedup graph showing average response time in milliseconds (ms) vs. the number of servers.

5.4 Experiment 2: Data Scalability

Experiments were carried out in order to determine the performance of the browse and search features of the application for different collection sizes. These experiments were intended to reveal whether or not the application could cope with increasing volumes of data in digital collections.

5.4.1 Search

The aim of this experiment is to investigate the time taken to process typical requests when the collection size differs.

5.4.1.1 Methodology

To determine the effect of time on differing collection sizes, Apache JMeter was used as in the case of service scalability discussed before. JMeter was set up to simulate 50 users accessing one Web service 10 times.

One of the important considerations in this case are the number of servers. For instance, this first experiment involved the use of four (4) identical servers and was run with collection sizes of 4000, 8000, 16000 and 32000 records. The experiment was run at least five times for each collection size. The average of the results was computed and for each collection size, the average response time obtained was further partitioned into blocks of 50 requests to give a better analysis of the results. The graph of average response time (in milliseconds) against number of requests for each collection size was plotted.

5.4.1.2 Results and Discussion

The results were represented as average response time against collection size as shown in Figure 5.5 and Table 5.1. The data from this experiment was partitioned into blocks of 50 requests and the values in Table 5.1 were obtained by taking an average of each block. In this case there is still a noticeable time to connect to AWS at the start of the experiment. The results suggest that the average response time does not differ much. However, more experimentation is needed with larger datasets. This is dependent on the nature of the requests and the sizes of responses, but there is no unexpected overhead from the cloud service layer.

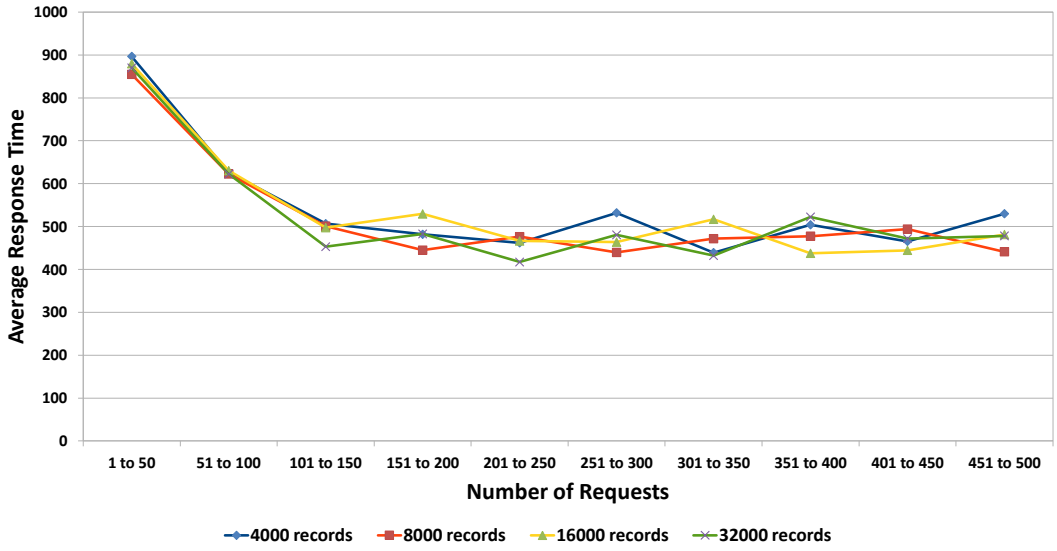


Figure 5.5: Average response time in milliseconds (ms) vs. the number of requests processed when searching collections of different sizes.

	Average Response Time (ms)			
Requests	**4000 records**	**8000 records**	**16000 records**	**32000 records**
1 to 50	897	855	880	870
51 to 100	626	623	630	622
101 to 150	507	501	497	453
151 to 200	482	445	529	483
201 to 250	462	476	466	417
251 to 300	532	440	464	481
301 to 350	439	472	517	432
351 to 400	504	477	438	522
401 to 450	465	494	445	472
451 to 500	530	441	481	478

Table 5.1: Average response times in milliseconds (ms) against the number of requests processed when searching collections of different sizes.

5.4.2 Browse

The aim of this experiment was to determine the performance of the browse feature for a fixed number of servers (four servers in this case) when browsing varying collection sizes.

5.4.2.1 Methodology

To determine the time taken to browse collection sizes of 4000, 8000, 16000 and 32000 records, JMeter was setup and used as described in Section 5.4.1.

5.4.2.2 Results and Discussion

The graph of average response time taken to browse the collection was plotted against the number of requests. The results are as shown by Figure 5.6 and Table 5.2. The data from this experiment was partitioned into blocks of 50 requests and the values in Table 5.2 were obtained by taking an average of each block.

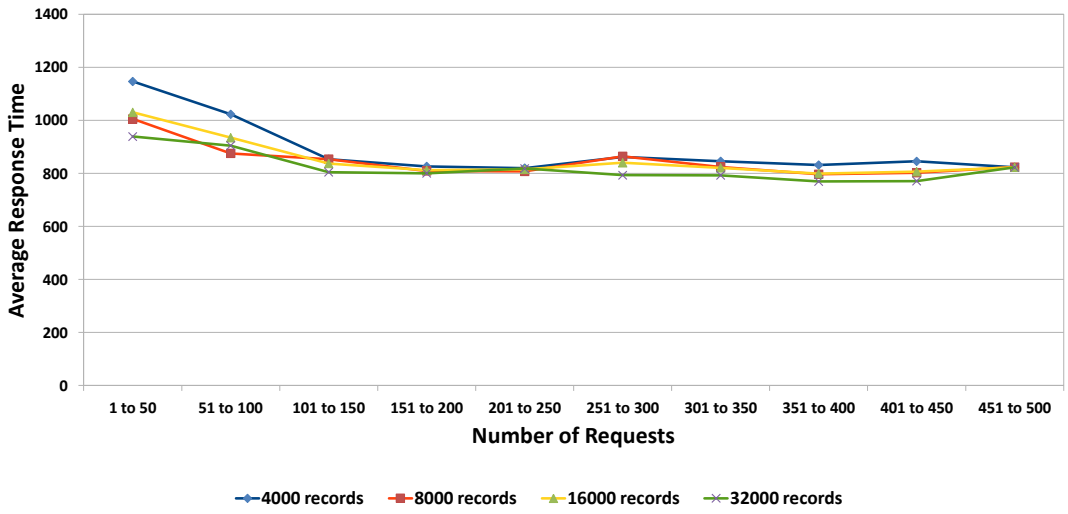


Figure 5.6: Average response time in milliseconds (ms) vs. the number of requests processed when browsing collections of different sizes.

	Average Response Time (ms)			
Requests	**4000 records**	**8000 records**	**16000 records**	**32000 records**
1 to 50	1147	1005	1030	939
51 to 100	1023	876	935	905
101 to 150	854	854	837	805
151 to 200	827	809	812	801
201 to 250	820	808	815	818
251 to 300	863	865	840	793
301 to 350	846	825	821	793
351 to 400	832	797	799	770
401 to 450	846	803	807	771
451 to 500	823	824	823	823

Table 5.2: Average response times in milliseconds (ms) against the number of requests processed when browsing collections of different sizes.

The response time for browsing is higher than that of searching because there was a large number of results retrieved when the collection was browsed. As with the search feature, there is a noticeable time taken to connect to AWS. The average response time does not greatly impact on browsing collections of different sizes but more experimentation is needed with larger datasets.

5.5 Experiment 3: Processor Load Testing

The idea behind load testing is to determine the volume of requests that the application could process for an increasing number of concurrent users.

The aim of this experiment was to investigate the time taken to process typical requests when the number of sequential requests varies, with different number of concurrent requests.

5.5.1 Methodology

JMeter was set up to simulate different numbers of users accessing one Web service. The first scenario was a simulation of 5 users, each accessing the Web service 10 times. The second and subsequent scenarios were a simulation of 20, 50, 100, 250 and 500 users, each accessing the Web service 10 times for a search query.

The Web service was again hosted on 4 identical Amazon EC2 instances (servers) in this case. In order to determine the order in which the Samplers are processed, a logic controller was added and in this particular case, it was a Random Controller. The Random Controller picks a random sampler or sub-controller at each pass, so all the servers have an equal chance of being selected for processing. This experiment was repeated at least 5 times for search and browse for each of the 5, 10, 20, 50, 100, 250 and 500 users simulated. The overall average response time for each case was computed and used to generate a graph of average response time against the number users.

5.5.2 Results and Discussion

The results of this experiment are as shown by Figure 5.7 that follows.

The results show that the average response times are relatively low when there are fewer users for both search and browse. The average response time shows a slight increase when the number of concurrent users is increased. As expected, the quantity and age of requests does not impact on the response times. However, more experimentation is needed with a simulation of a large number of concurrent users.

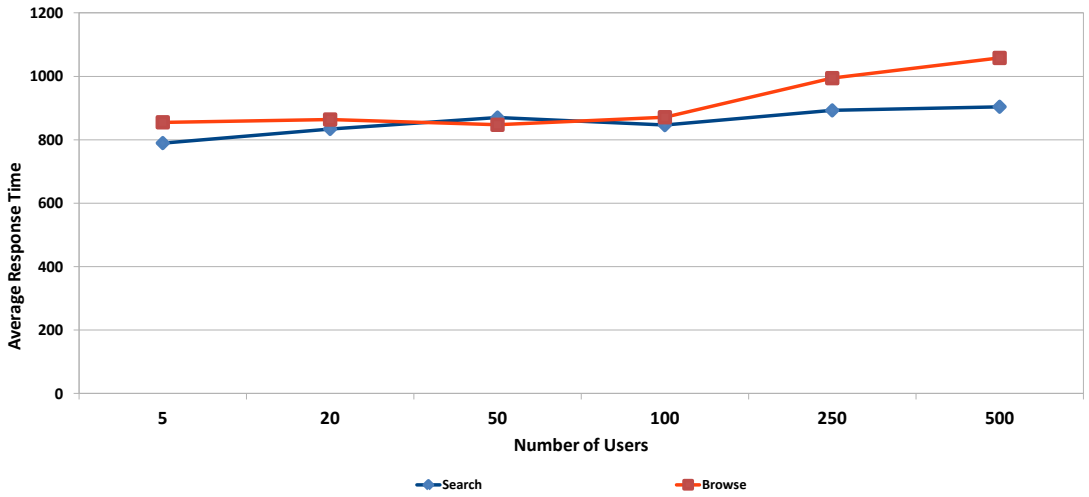


Figure 5.7: Graph of average response time in milliseconds (ms) vs. number of users serviced for a fixed number of concurrent requests.

5.6 Summary

The results of the experiments have shown that there are stable response times and some degree of variability because of multiple front-end servers when browsing and/or searching a collection of a fixed size. There are no significant response time differences after the results caching phase. When processing typical requests on varying collection sizes in the cloud, response times do not differ much although more experimentation with larger datasets is needed. Lastly, request sequencing has shown that the quantity and age of requests does not have an impact on response times.

Chapter 6

Conclusion

6.1 Concluding Remarks

Dealing with large volumes of data can be a daunting task, especially when there are no tools and appropriate technologies at one's disposal. Cloud computing has gained popularity in recent years and it has been adopted by several institutions, industries and/or individuals for use in different domains. It therefore provides a means for developing systems that can scale as needed. The cloud application proposed and developed on top of the Amazon Web Services cloud computing stack for this thesis gave a better understanding of the pros and cons of developing Digital Library applications in AWS. The complexity of applications differs but development of digital library service components in the cloud has proven to be feasible.

Performance evaluation of the application deployed in the cloud has shown that response times are not greatly affected by differences in request complexity, collection sizes or request sequencing. There is a noticeable time taken to connect to AWS services. In production systems this should really be persistent, like ODBC/JDBC connections in persistent database-driven Web applications.

There is a ramp-up time where internal caching has a small impact on the results. This occurs consistently for all requests and request types. This will not affect busy services but may have a small impact on services that are rarely used. Oscillation in response times suggests that there are multiple AWS back-end servers with caches

that are not completely shared. This results in a degree of unpredictability in results but this averages out over a period of time.

6.2 Limitations

In developing the services, many features had to be redesigned from typical database-driven versions. This may constrain what is possible in, for example, query syntax, and may also affect the viability of other services because of the limitations of S3/SimpleDB.

Experiments carried out in this thesis were using one type of Amazon EC2 instance, which could have also had a performance bearing. A more elaborate approach would be to deploy an application of digital library service components on different Amazon EC2 instance types. Another aspect of this research was the use of Amazon SimpleDB for indexing and querying the digital collection. Amazon SimpleDB has a high degree of query results caching, which could possibly affect the experimental results. The use of different indexing services can therefore give a better indication of performance of digital library service components deployed in clouds.

While this work was underway, Amazon Web Services released the beta version of another service - Amazon CloudSearch[1] - which supposedly provides a fully-managed search service that automatically scales with increasing amounts of searchable data. This service was not used in this thesis because the development of digital library service components was a way of experimenting with application deployment in the cloud.

There are open questions that this thesis did not answer, in particular those related to building a full set of digital library services in the cloud and optimizing the inverted file indexing on SimpleDB.

[1] `http://aws.amazon.com/cloudsearch` [Last accessed on 6 February 2013]

6.3 Future Work

Some possible extensions to this work are highlighted in sections 6.3.1, 6.3.2 and 6.3.3 that follow.

6.3.1 Implementation of a Full Prototype Digital Library System

Development of a full prototype digital library system can give a better indication of the ease of development and deployment and/or migration of existing digital library architectures into the cloud environment.

6.3.2 Experimentation

The set of experiments conducted in this thesis were a subset of a series of experiments that could be carried out to determine digital library scalability in the cloud. It would also be useful to use a larger collection size of up to millions of records when testing for data scalability. Another consideration could be simulation of an even larger population of users, perhaps millions of concurrent user accesses.

The experiments presented in this thesis used a synthetic load for performance testing. It is however necessary to test the application using a realistic load. The realistic load follows actual load patterns which can be obtained from existing log files of real-world websites.

6.3.3 Cloud Security

Data security in the cloud is another possible extension to this work. It is imperative for users to know that all their data stored in public clouds is secure and safe from third party intrusions. An evaluation of cloud security for digital library services in public clouds can be incorporated into this study.

Bibliography

[1] Abbas, A. *Grid Computing: A Practical Guide to Technology and Applications*, first ed. Networking Series. Charles River Media, INC, 2004.

[2] Akamai Technologies, Inc. Akamai Solutions for Cloud Computing: Accelerate, Scale and Fortify Applications and Platforms Running in the CLoud. [Online: Last Accessed 27 January 2013], 2009.

[3] Amazon Web Services. *Amazon Simple Storage Service, Developer Guide*, March 2006. Website: `http://aws.amazon.com/s3`.

[4] Amazon Web Services. *Amazon CloudFront, Developer Guide*, June 2008. `http://aws.amazon.com/cloudfront`.

[5] Amazon Web Services. *Amazon Elastic Compute Cloud, Developer Guide*, March 2009. Website: `http://aws.amazon.com/EC2`.

[6] Amazon Web Services. *Amazon Elastic MapReduce, Developer Guide*, March 2009. `http://aws.amazon.com/mapreduce`.

[7] Amazon Web Services. *Amazon Simple Queue Service, Developer Guide*, February 2009. Website: `http://aws.amazon.com/sqs`.

[8] Amazon Web Services. *Amazon SimpleDB, Developer Guide*, April 2009. `http://aws.amazon.com/simpledb/`.

[9] Amazon Web Services. *Amazon Simple Queue Service - Developer Guide, API Version 2010-08-01*, August 2010. Website: `http://aws.amazon.com/cloudwatch`.

[10] Anderson, D. P. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing* (Pittsburgh, PA, USA, November 2004).

[11] Anderson, D. P., Christenson, C., and Allen, B. Designing a Runtime System for Volunteer Computing. In *SC '06, Proceedings of the ACM/IEEE Conference on Super Computing Applications and Computer Architecture* (Tampa, FL, USA, November 2006).

[12] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. SETI@HOME: An Experiment in Public-Resource Computing. *Communications of the ACM 45*, 11 (November 2002), 56–61.

[13] Anderson, D. P., and Fedak, G. The Computational and Storage Potential of Volunteer Computing. In *CCGRID '06, Proceedings of the 6th International Symposium on Cluster Computing and the Grid* (May 2006), pp. 73–80.

[14] Andresen, D., Yang, T., Egecioglu, O., Ibarra, O. H., and Smith, T. R. Scalability Issues for High Performance Digital Libraries on the World Wide Web. In *Proceedings of IEEE ADL '96, Forum on Research and Technology Advances in Digital Libraries* (Washington, DC, 1996), pp. 139 – 148.

[15] Apache CloudStack. *Apache CloudStack 4.0.0-incubating: CloudStack API Developer's Guide*, May 2012.

[16] The Apache Software Foundation. *Apache JMeter User Manual.* `http://jmeter.apache.org/usermanual/intro.html`.

[17] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009. `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html`.

[18] Babar, M. A., and Chauhan, M. A. A Tale of Migration to Cloud Computing for Sharing Experiences and Observations. In *SECLOUD '11:*

Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing (Waikiki, Honolulu, HI, USA, May 2011).

[19] Baker, M., Apon, A., Buyya, R., and Jin, H. Cluster Computing and Applications. *Encyclopedia of Computer Science and Technology 45* (September 2000).

[20] Barateiro, J., Antunes, G., Cabral, M., Borbinha, J., and Rodrigues, R. Using a Grid for Digital Preservation. In *Proceedings of the 11th International Conference on Asian Digital Libraries, ICADL* (Bali, Indonesia, December 2008), Digital Libraries: Universal and Ubiquitous Access to Information, pp. 225–235.

[21] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the Art of Virtualization. In *SOSP '03: Proceedings of the Nineteenth ACM symposium on Operating Systems Principles* (New York, NY, USA, 2003), ACM, pp. 164 – 177.

[22] Barroso, L. A., Dean, J., and Hölzle, U. Web Search for a Planet: The Google Cluster Architecture. *IEEE Computer Society* (March-April 2003).

[23] Beloglazov, A., Fotuhi Piraghaj, S., Alrokayan, M., and Buyya, R. Deploying OpenStack on CentOS Using the KVM Hypervisor and GlusterFS Distributed File System. Tech. Rep. CLOUDS-TR-2013-3, Cloud Computing and Distributed Systems Laboratoy, The University of Melbourne, Melbourne, Victoria, Australia, August 2012.

[24] Benson, T., Akella, A., Shaikh, A., and Sahu, S. CloudNaaS: A Cloud COmputing Networking Platform for Enterprise Applications. In *SOCC '11: Proceedings of the 2nd ACM Symposium on Cloud Computing* (Cascais, Portugal, October 2011).

[25] Berners-Lee, T., Fielding, R., and Masinnter, L. RFC3986 Uniform Resource Identifier (URI): Generic Syntax, January 2005.

[26] BOINC. *Computing with BOINC*, December 2011. `http://boinc.berkeley.edu/track/wiki/ProjectMain` [Last accessed on 27 January 2013.

[27] Borgman, C. L. Challenges in Building Digital Libraries for the 21st Century. In *In Proceedings of the 5th International Conference on Asian Digital Libraries: Digital Libraries: People, Knowledge and Technology, ICADL 2002* (Singapore, December 2002), Springer.

[28] Branan, B. *DuraCloud Architecture*, November 2012. [Online] Last Accessed on 30 January 2013.

[29] Brown, S., and Rose, J. Architecture of FPGAs and CPLDs: A Tutorial. *IEEE Design and Test of Computers 13*, 2 (1996), 42–57.

[30] Buco, M. J., Chang, R. N., Luan, L. Z., Ward, C., Wolf, J. L., and Yu, P. S. Utility Computing SLA Management based upon Business Objectives. *IBM Systems Journal 43*, 1 (2004), 159 – 178.

[31] Bui, H., Bui, P., Flynn, P., and Thain, D. ROARS: A Scalable Repository for Data Intensive Scientific Computing. In *Proceedings of the First Conference on Data-Intensive Distributed Computing* (Chicago, IL, USA, June 2010).

[32] Buyya, R., Broberg, J., and Goscisnki, A. *Cloud Computing: Principles and Paradigms*. John Wiley & Sons, Inc., 2011.

[33] Camposano, R. Electronic Design in the Cloud. Tech. rep., Nimbic Inc., 2011.

[34] Canonical Limited. Creating open cloud. Tech. rep., Canonical Group Limited, May 2012. White paper.

[35] Chaganti, P., and Helms, R. *Amazon SimpleDB Developer Guide: Scale your application's database on the cloud using Amazon SimpleDB.* Packt Publishing, June 2010.

[36] Cheng, X., Dolin, R., Neary, M., Prabhakar, S., Kanth, K. V. R., Wu, D., Agrawal, D., Abbadi, A. E., Freeston, M., Singh, A., Smith, T., and J, S. Scalable Access within the Context of Digital Libraries. In *IEEE Proceedings of the International Conference on Advances in Digital Libraries, ADL* (1997), pp. 70 – 81.

[37] Cherkasova, L. Scalable Web Hosting Service. Tech. Rep. HPL-1999-52(R.1), HP Laboratories Palo Alto, October 1999.

[38] Danielson, K. Distinguising Cloud Computing from Utility Comuting, March 2008.

[39] Dijcks, J.-P. Oracle: Big Data for the Enterprise. Oracle White Paper, 2012.

[40] The DILIGENT Project. *DILIGENT: A Digital Library Infrastructure on Grid Enabled Technology*, 2007. Website: `http://www.diligentproject.org`.

[41] The DSpace Foundation. *DSpace 1.5.2 Manual*, 2002 – 2009. `http://www.dspace.org/`.

[42] Duran, B., and Xhafa, F. The effects of two replacement strategies on a Genetic Algorithm for Scheduling Jobs on Computational Grids. In *SAC '06, Proceedings of the 21st Annual ACM Symposium on Applied COmputing* (Dijon, France, April 2006).

[43] EUCALYPTUS Systems, Inc. *Eucalyptus 3.2.0 Administration Guide: 2012-12-16 API Version*, December 2012.

[44] Eucalyptus Systems, Inc. *Eucalyptus 3.2.0 Installation Guide: API Version 2012-12-16*, December 2012.

[45] Flanders, D. F. Fedorazon - Cloud Repository. Tech. rep., Birkbeck College, 2008.

[46] Foster, I., and Kesselman, C. *The Grid 2: Blueprint for a New Computing Infrastructure*, second ed. Morgan Kaufmann, 2004.

[47] Foster, I., Zhao, Y., Raicu, I., and Lu, S. Cloud Computing and Grid Computing 360-Degree Compared. [Online], 2008. `http://www.citebase.org/abstract?id=oai:arXiv.org:0901.0131`.

[48] Frakes, B., and Baeza-Yates, R. *Information Retrieval: Data Structures and Algorithms*, 1st ed. Prentice Hall PTR (ECS Professional), June 1992.

[49] Gandotra, I., Abrol, P., Gupta, P., Uppal, R., and Singh, S. Cloud Computing Over Cluster, Grid Computing: a Comparative Analysis. *Journal of Grid and Distributed Computing 1*, 1 (2011).

[50] Geelan, J. Twenty-one experts define cloud computing. [Online], January 2009. `http://cloudcomputing.sys-con.com/node/612375/print`.

[51] GRIPSI-2007. An Overview of Grid Computing and Globus Toolkit, November 2007.

[52] Grossman, R. L., Gu, Y., Sabala, M., and Zhang, W. Compute and storage clouds using wide area high performance networks. *Future Generation Computer Systems*, 29 (July 2008), 179–183.

[53] Hazelhurst, S. Scienctific Computing using high-performance computing: a case study of using Amazon Elastic Computing Cloud. In *Proceedings of SAICSIT 2008: Riding the Wave of Technology* (Wilderness Beach Hotel, Wilderness, South Africa, October 2008), C. Cilliers, L. Barnard, and R. Botha, Eds., pp. 94 – 103.

[54] Heuveneers, E. Introduction to Amazon S3 with Java and REST. Online, August 2007. [Last Accessed on 20 January 2013].

[55] Joseph, J. Patterns for High Availability, Scalability and Computing Power with Windows Azure. *MSDN Magazine* (May 2009).

[56] Khajeh-Hosseini, A., Greenwood, D., and Sommerville, I. Cloud Migration: A Case of Migrating and Enterprise IT System to IaaS. In *2010 3rd IEEE International Conference on Cloud Computing* (Miami, FL, USA, July 2010).

[57] Khazaei, H., Mišić, J., and Mišić, V. B. Performance Analysis of Cloud Computing Centers Using $M/G/m/m + r$ Queuing Systems . *IEEE Transactions on Parallel and Distributed Systems Volume 23*, Number 5 (May 2012), 936–943.

[58] Krafft, D. B., Birkland, A., and Cramer, E. J. NCore: Architecture and Implementation of a Flexible, Collaborative Digital Library. In *Proceedings of the Joint Conference in Digital Libraries* (Pittsburg, PA, USA, June 2008).

[59] Krieger, O., McGachey, P., Kanevsky, A., and The VCD Team. Enabling a Marketplace of Clouds: VMware's vCloud Director. *SIGPOS Operating Systems Review Volume 44*, Number 4 (2010), 103 – 114.

[60] Lagoze, C., Payette, S., Shin, E., and Wilper, C. Fedora: An Architecture for Complex Objects and their Relationships. *International Journal on Digital Libraries 6*, 2 (April 2006), 124 – 138.

[61] Lascelles, F. Standardized HMAC, OAuth RESTful Authentication Schemes. [Last Accessed on 20 January 2013], 2013.

[62] Leung, K. T., Ercegovac, M., and Muntz, R. R. Exploiting Reconfigurable FPGA for Parallel Query Processing in Computation Intensive Data mining Applications. In *UC MICRO Technical Report* (Los Angeles, CA, USA, 1999).

[63] Lin, J., and Dyer, C. *Data-Intensive Text Processing with MapReduce.* Morgan and Claypool Publishers, 2010.

[64] Lobodzinski, B., Wilk, K., and Plaszczak, P. Globus Toolkit 4.0 - Grid Engine 6 Interface: description. Tech. rep., Gridwise Tech, 2006.

[65] Lucene. *Lucene Search Engine*, October 2012. [Last Accessed on 22 January 2013].

[66] Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., and Hung Byers, A. Big Data: The next frontier for Innovation, Competition and Productivity. Tech. rep., McKinsey Global Institute, May 2011.

[67] Mauldin, M. L. *Information Retrieval by Text Skimming.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, August 1989.

[68] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review Volume 38*, Number 2 (April 2008).

[69] McLean, B. DuraCloud: Federated Repositories and Cyberinfrastructure Open technologies and services for managing durable data in the cloud. Tech. rep., DuraSpace, October 2009. [Report Presentation by the CTO - Duraspace].

[70] Microsoft TechNet. Site Server - Building High-Scalability Server Farms. Tech. rep., Microsoft Corporation, San Diego, CA, USA, September 1999. [Last accessed on 27 January 2013].

[71] Milojičić, D., Llorente, I. M., and Montero, R. S. OpenNebula: A Cloud Management Tool. *IEEE Internet Computing* (March/April 2011). Trend Wars Interview.

[72] Misra, D., Seamans, J., and Thoma, G. R. Testing the Scalability of a DSpace-based Archive. In *Proceedings of the IS&T Archiving* (Bern, Switzerland, June 2008), pp. 36 – 40.

[73] Mohamed, A. A history of Cloud Computing. [Online], March 2009. `http://www.utilitycomputing.com/AHistoryOfCloudComputing20090327.asp`.

[74] Moschkis, I. A., and Karatza, H. D. Performance and Cost Evaluation of Gang Scheduling in a Cloud Computing System with Job Migrations and Starvation Handling. In *ISCC '11: Proceedings of the 16th IEEE International Symposium on Computers and Communications (ISCC)* (Kerkyra(Corfu), Greece, June/July 2011), pp. 418–423.

[75] Mueller, R., Teubner, J., and Alonso, G. Data Processing on FPGAs. In *Proceedings of VLDB Endowment* (August 2009), pp. 910–921.

[76] Nakashole, N. A Hybrid Scavenger Grid Approach to Intranet Search. Master's thesis, University of Cape Town, Rondebosch, Cape Town, South Africa, February 2009.

[77] Nimbus. Nimbus Platform and Infrastructure. `www.nimbusproject.org` [Online: Last accessed on 10 February 2013].

[78] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems. Tech. Rep. 2008-10, University of California Santa Barbara, Computer Science, Santa Barbara, California 93106, August 2008.

[79] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. The Eucalyptus Open-source Cloud-computing System. In *Proceedings of Cloud Computing and its Applications* (October 2008).

[80] Nyirenda, M. Universal Web Application Server. Master's thesis, University of Cape Town, Rondebosch, Cape Town, South Africa, December 2007.

[81] Paihama, J. K. d. R. Meta-standardisation of Interoperability Protocols. Master's thesis, University of Cape Town, Rondebosch, Cape Town, South Africa, June 2012.

[82] Pang, H., and Tan, K.-L. Authenticating Query Results in Edge Computing. In *ICDE '04, Proceedings of the 20th International Conference on Data Engineering* (Washington DC, USA, 2004).

[83] Parker, C. A Light Weight Interface to Local Grid Scheduling Systems. Master's thesis, University of Cape Town, Rondebosch, Cape Town, South Africa, May 2009.

[84] Perera, D. G., and Li, K. F. Parallel Computation of Similarity Measures Using an FPGA-Based Processor Array. In *AINA '08, Proceedings of the 22nd International Conference on Advanced Information Networking and Applications* (Okinawa, Japan, March 2008), pp. 955–962.

[85] Potgieter, J. OLAP Data Scalability. White Paper, October 2003.

[86] Sadashiv, N., and Dilip Kumar, S. M. Cluster, Grid and Cloud Computing: A Detailed Comparison. In *ICCSE '11, Proceedings of the 6th International Conference on Computer Science and Education* (SuperStar Virgo, Singapore, August 2011).

[87] Sempolinski, P., and Thain, D. A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science* (Indianapolis, IN, USA, Novemeber/December 2010), pp. 417–426.

[88] Sivasubramanian, S., Pierre, G., and van Steen, M. Analysis of Caching and Replication Strategies for Web Applications. *IEEE Internet Computing* (2007).

[89] Stabler, G., Rosen, A., and Goasguen, S. Elastic IP and Security Groups Implementation Using OpenFlow. In *VTDC '12: Proceedings of the 6th International Workshop on Virtualization Technologies in Distributed Computing* (Delft, The Netherlands, June 2012).

[90] Suleman, H. Digital Libraries Without Databases: The Bleek and Lloyd Collection. In *Proceedings of Research and Advanced Technology for Digital Libraries, 11th European Conference (ECDL 2007)* (Budapest, Hungary, September 2007), N. F. Kovacs, Laszlo and C. Meghini, Eds., pp. 462–465. [Available: `http://pubs.cs.uct.ac.za/archive/00000433/01/ecdl_2007_dlwd.pdf`].

[91] Suleman, H. in-Browser Digital Library Services. In *Proceedings of Research and Advanced Technology for Digital Libraries, 11th European Conference (ECDL 2007)* (Budapest, Hungary, September 2007), N. F. Kovacs, Laszlo and C. Meghini, Eds., pp. 462–465. [Available: `http://pubs.cs.uct.ac.za/archive/00000434/01/ecdl_2007_ajax.pdf`].

[92] Suleman, H. Utility-based High Performance Digital Library Systems. In *Proceedings of Second Workshop on Very Large Digital Libraries* (Corfu, Greece, October 2009).

[93] Suleman, H., Parker, C., and Omar, M. Lightweight component-based scalability. *International Journal on Digital Libraries 9* (2008), 115–124.

[94] Sun Microsystems, Inc. *IntroductIon to Cloud Computing ArChiteCture.* 4150 Network Circle, Santa Clara, CA 95054 USA, June 2009. White Paper.

[95] Sun Microsystems, Inc. *Take Your Business to a Higher Level: Sun Cloud Computing.* 4150 Network Circle, Santa Clara, CA 95054 USA, March 2009.

[96] Vienne, J., Chen, J., Was-ur-Rahman, M., Islam, N. S., Subramoni, H., and Panda, D. K. D. Performance Analysis and Evaluation of InfiniBand FDR and 40GigE RoCE on HPC and Cloud Computing Systems.

In *HOTI '12: Proceedings of the 20th IEEE Annual Symposium on High-Performance Interconnects* (Santa Clara, CA, USA, August 2012).

[97] VMware. Cloud Foundry - The Industry's Open Platform as a Service. `http://www.cloudfoundry.com` [Online: Last accessed on 10 February 2013].

[98] Wada, H., Fekete, A., Zhao, L., Lee, K., and Liu, A. Data Consistency Properties and the Trade-Offs in Commercial Cloud Storages: the Consumers' Perspective. In *Proceedings 5th Biennial Conference on Innovative Data Systems Research, CIDR'11* (Asilomar, CA, USA, January 2011), pp. 134 – 143.

[99] Willet, P. Porter Stemming Algorithm: then and now. *Program: Elecronic Library and nformation Systems 40*, 3 (2006).

[100] Yeo, C. S., de Assuncao, M. D., Yu, J., Sulistio, A., Venugopal, S., Placek, M., and Buyya, R. Utility Computing and Global Grids. [Online], 2006. `http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0605056`.

[101] Yigitbasi, N., Iosup, A., Epema, D., and Ostermann, S. C-Meter: A Framework for Performance Analysis. In *Proceedings of 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09* (Shanghai, China, May 2009), pp. 472 – 477.