A Framework for Provisioning Algorithms as a Service

By

Abdullah A. Qasem

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Applied Science

in

Electrical and Computer Engineering

University of Ontario Institute of Technology

Oshawa, Ontario, Canada

ABSTRACT

A FRAMEWORK FOR PROVISIONING ALGORITHMS AS A SERVICE

Abdullah A. Qasem
University of Ontario Institute of Technology, 2016

Advisor:
Professor Qusay H. Mahmoud

Designing, implementing and executing algorithms have become a relevant and important element in various fields. Public users and data researchers are interested in analyzing and interpreting data with shorter execution time and higher performance. Using a scalable and high resource usage environment will help improve algorithm performance. Cloud computing is an environment which provides scalable and high-end virtual resources to achieve high quality services. This thesis presents the design, implementation and evaluation of a framework for provisioning algorithms as a service in the cloud. This framework introduces solutions to help clients overcome different concerns and difficulties, such as looking for an appropriate algorithm, understanding algorithm source code, installing and configuring specific libraries, and achieving high algorithmic performance. The framework provides clients the possibility to discover available algorithms and/or deploy new algorithms over multiple scalable platforms. It also allows clients to analyze data, compare results, and measure algorithm's performance. A prototype implementation of the framework has been developed to demonstrate the feasibility of the solution. The results of evaluating the prototype demonstrate that providing multiple scalability models and high-end web servers will improve algorithm performance and achieve availability and reliability using the framework.

DEDICATION

I would like to dedicate my thesis to my beloved

My Parents.

My Wife and Kids.

Acknowledgments

I would like to express my sincere gratitude to all those who assisted in the completion of my thesis.

Special thanks to my advisor, Professor Qusay H. Mahmoud, to whom I am deeply indebted for providing stimulating suggestions, encouragement and for his insight and patience during my research and in the writing of this thesis. His invaluable advice and support has been greatly appreciated. I also thank him immensely for the funding I received in the form of Graduate Research Assistantships which provides me with the opportunity to continue my Master's studies.

I would also like to thanks my thesis committee: Dr. Ramiro Liscano and Dr. Khalil El-Khatib for their advice and guidance.

I would like to thank all of my family. In particular I owe my deepest gratitude to my great parents, my amazing wife, and my brothers and sisters have been a huge support for me throughout my Master's study and my life in general.

I would like to personally thank my colleague Mohammed Jassas for the encouragement, help and the enjoyable learning environment. I would also like to thank my friends Zubair and Paul, who patiently and expertly read many drafts of this work. I am grateful for your kind words and encouragement.

I certainly cannot forget the entire staff at the Department of Electrical, Computer, and Software Engineering, as well as the Graduate Office, for taking care of all of my administrative and financial requirements. I am particularly thankful for being assigned as a Teaching Assistant, which has greatly enhanced my knowledge and provided me with valuable hands-on experience.

Table of Contents

List of Tables

List of Figures

Acronyms and Abbreviations

AWS	Amazon Web Services
EC2	Elastic Compute Cloud
S3	Simple Storage Service
RDS	Relational Database Service
ELB	Elastic Load Balancing
EMR	Elastic Map-Reduce
SaaS	Software as a Service
REST	REpresentational State Transfer
SOA	Service Oriented Architecture
UDFs	User Defined Functions
SQL	Standardized Query Language
AMI	Amazon Machine Image
SDK	Software Development Kit
SSL	Socket Secure Layer
SSD	Solid State Drive
JHAVE	Java-Hosted Algorithm Visualization Environment
ViSA	Visualization of Sorting Algorithms
DMaaS	Data Mining as a Service
ML	Machine Learning
IPR	Intellectual Property Rights

Chapter 1

1. Introduction

Cloud computing is a common term for every technology service provided over the Internet. It offers appropriate and on-demand hardware and software services for different computing resources, such as networks, servers, and storage areas [1]. The cloud computing environment has many advantages, such as scalability, flexibility, low cost, automated service provision, massive storage capacity, and processing power. Moreover, it gives users the ability to pay for hardware and software services as they consume them [2]. Cloud service providers enable clients to configure, test, and deploy applications on virtual resources, utilizing several infrastructures and various operating systems [3]. To provide more flexibility, cloud providers offer three different types of services, namely: (1) Software as a Service (SaaS), (2) Platform as a Service (PaaS), and (3) Infrastructure as a Service (IaaS).

SaaS is a software distribution model that remotely provides access to a software application and its functions as a web service. PaaS offers many application frameworks and operating systems for users in the cloud, avoiding installing any framework or software on users' local machines. This minimizes development efforts and cost. IaaS offers a group of cloud computing resources, including hardware, network components, servers, and vast storage areas [4].

According to the National Institute of Standards and Technology (NIST) in the United States (US), there are four models of deploying cloud computing. These are private,

public, community, and hybrid cloud [4]. There are many benefits for clients in using cloud computing [5], such as:

1) Clients can avoid spending a large amount of money in data centers and servers before knowing how they are going to use these data centers. In other words, with the cloud concept, clients will pay when using computing resources in accordance to how much they use.
2) Clients can achieve a lower cost than they can obtain on their own. Providers can achieve vast economies of scale because large numbers of customers use cloud services, which can translate into lower cost services.
3) Clients can focus on their solutions and services rather than on the hardware and infrastructure.
4) Clients can easy deploy applications in multiple regions around the world.

Cloud computing allows new business opportunities and options to provide various on-demand services and reduces information technology service costs [6]. Cloud computing service models can be applied dynamically to different domains and software paradigms.

The Amazon Web Services (AWS) is one of the most widely used cloud providers. AWS is a highly scalable, dependable, low-cost infrastructure environment. AWS data centers are located in the U.S., Europe, Brazil, Japan, Singapore, and Australia and serve businesses in nearly 200 countries around the world [7]. Clients can immediately install new applications, scale up as their workload increases, and scale down when the workload decreases. Whether they need one virtual server or hundreds, short term or long term, AWS remains flexible.

AWS offers Software Development Kits (SDK) for multiple languages such as: Java, .NET, Node.js, Paython, and C++ to simplify using AWS services in applications. It also offers several operating system environments to host any application [7]. Clients can select the development platform or programming model for their business. They can also choose which services they employ and how they use them. This flexibility gives clients the ability to focus on solutions, not infrastructure. To ensure the integrity, safety, security and privacy of client data, AWS data centers and services have multiple layers of operational and physical protection [7]. AWS is an infrastructure as a service that offers different kinds of solutions and services, such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (Amazon S3), Amazon Relational Database Service (Amazon RDS), Amazon Elastic Load Balancing (ELB), and Amazon Elastic Map-Reduce (Amazon EMR) [7].

1.1 Motivation

Creating and configuring algorithms to execute specific tasks and examine varied data types has become a challenge for researchers and scientists, as many of them are lacking programming background and/or sufficient time to configure an experiment environment. Furthermore, preparing a formidable environment that provides optimal performance and virtually unlimited resources is a logical solution in helping clients achieving results within an appropriate execution time and acceptable accuracy. In order to understand the challenges and concerns from a user perspective, several points are presented and discussed in the following sections.

- **Discovering suitable algorithms**

Users, especially scientists and researchers, need to analyze collected data to obtain useful and efficient results. Results lead to better decisions to solve problems related to their fields. There are various kinds of algorithms for different problem domains. Moreover, sometimes a large number of algorithms can be used simultaneously during the same analyzing job. Users must make an effort examining these algorithms and choose the one that is the most appropriate.

- **Understanding algorithm source code**

As previously mentioned, there are various algorithms that perform the same job. Therefore, specifically understanding how to implement algorithm source code is imperative in determining the most compatible algorithm. Users need to learn about programming languages as well as their concepts and rules to efficiently merge any chosen algorithm/s within a specific framework.

- **Installing and configuring runtime environments**

Algorithms are written using different programming languages based on programmer backgrounds. Different software is required to run the algorithms, based on which programming language it was written in. The number of software may be increased based on the number of algorithms used. Furthermore, experienced users are generally required to run the software and implement the algorithm.

- **Avoiding implementation errors**

Implementing a new algorithm is not an easy task. Many issues could be faced relating to code error, runtime error, or compilation error. Offering various algorithm services as a service will help avoid algorithm implementation errors.

- **Achieving high performance algorithms**

Testing and examining data within a suitable execution time frame is a major concern for users. Aspects that could affect the algorithm's execution time are algorithm code, implementation environment and available resources. Providing high capacity resources in the implementation environment will help users achieve high performance algorithm service/s and decrease the execution time.

Combining multiple data analysis services in one framework allows public users and researchers with limited programming aptitude to conveniently analyze data, present results, and compare them over different algorithm services.

1.2 Thesis Statement and Contributions

The purpose of this research is to design, develop and evaluate a framework for provisioning Algorithms as a Service (AaaS). This framework is offered in the cloud environment to make it easier for public users, data researchers, and algorithm developers to collaborate around algorithms, solve related problems, and compare results and performance with other algorithms. Clients have the ability to develop and deploy new algorithms or search and discover which category of algorithm services they want to utilize from available categories, enter or upload data to cloud web servers, provide the algorithm with the required variables, run the algorithm in the cloud, and then view or download the algorithm results to a local machine.

The contributions of this thesis include:

- A value-added framework for provisioning Algorithms as a Service (AaaS);
- Templates for algorithm developers to contribute, develop, deploy, and examine both sequential and parallel algorithms;
- Scalability models to help public users and data researchers analyze data and compare results using different web server configurations;
- A prototype of the framework using AWS to demonstrate the feasibility of the solution and to be used as a reference for other implementations; and
- A prototype evaluation, with an emphasis on the runtime performance and scalability models performance as well as availability performance.

Algorithms as a Service (AaaS) is a value-added Software as a Service (SaaS) where the framework provides algorithm services as web services using a Web interface. The framework provides clients the ability to select web server configurations, which allow them to host requested services using different web server models and types. Furthermore, the framework allows clients to deploy new sequential or parallel algorithms using scale-up or scale-out services.

1.3 Organization of the Thesis

The chapters of this thesis have been organized as follows:

Chapter 1 explains cloud computing approaches in general, as well as the motivation of the study and the contribution of this work.

Chapter 2 reviews the published related work and highlights the current body of knowledge for the various approaches and process models for all of these approaches.

Chapter 3 contains a preliminary proposal of our solution and describes in detail the framework architecture.

Chapter 4 describes our prototype reference implementation of the proposed framework and provided services.

Chapter 5 presents evaluation results to examine runtime performance based on scalability models and availability of the prototype implementation.

Chapter 6 summarizes our achievements and outlines future extensions.

Chapter 2

2. Background and Related Work

In order to offer a valuable solution, it is important to understand the concept of the Algorithms as a Service and demonstrate the current challenges when implementing algorithm services. This chapter discusses related work that focuses on several methods to provide algorithm services and to avoid implementation issues. This overview is necessary in order to recognize which solution aspects are most important and should be addressed in the proposed solution.

2.1 Software as a Service (SaaS)

SaaS is a software sharing model where an application is hosted in a web server via a service provider and presented to clients over the Internet. It is a more comprehensive distribution model, as basic technologies now support Service-Oriented Architecture (SOA) and web services. There are many advantages to the SaaS, such as compatibility, ease of management, capability to provide regular updates, teamwork collaboration, and global accessibility [8]

The author [9] reviewed already opening market of Machine Learning (ML) algorithms as Software-as-a-Service. He focused on PaaS/SaaS solutions that allow clients to access ML algorithms via Web services. The author presented several software service providers for ML problems. We have selected some of these providers below:

- **BigML** [10] is a SaaS approach to ML algorithms. Under this approach, clients have the ability to setup data sources, initiate, visualize and share decision trees models, from a Web interface or using REST API.
- **BitYota** [11] is a SaaS provider for Big Data warehousing solutions. In addition to providing data integration from different sources (relational, NoSQL, HDFS), it also allows clients to run statistics and summarization queries in SQL92, standard R statistics, and custom functions using JavaScript, Perl, or Python on a parallel analytics engine.
- **Google Prediction API** [12] is Google's cloud-based machine learning tool that can help analyzing data. It is closely connected to Google Cloud Storage, where training data can be stored. Additionally, it offers its services using a RESTful interface, with client libraries allowing programmers to connect from Java, JavaScript, .NET, Ruby, and Python.

After this reviewing, the author concluded that there is still room for a scalable, easy to use, and deployable solution for ML algorithms in the cloud environment to help end-users with less programming or statistical experience.

2.2 Big Data and Parallel Computing

The term “Big Data” is used to describe datasets where the size prevents standard Database software tools to store, manage, and analyze data [13]. Big Data includes commerce transactions, e-mail messages, photos, videos, and unstructured texts on the Web, such as social media and blogs. This type of data requires a different processing paradigm that employs readily-available hardware in parallel [14].

The authors [15] introduced different big data processing techniques, which focussed on the aspects of system and application. Through their study, it was evident that many popular parallel processing models exist, including MPI, General Purpose GPU (GPGPU), MapReduce, and MapReduce-like. MapReduce, a model proposed by Google, is a popular big data processing model that has repeatedly been studied and applied by both industry and academia. The authors divided existing MapReduce applications into three categories: partitioning sub-space, decomposing sub-processes, and approximate overlapping calculations. They also found that some DBMS vendors have recently integrated MapReduce front-ends into their systems, such as HadoopDB. HadoopDB is a hybrid system which efficiently takes the best features from the scalability of MapReduce and the performance of DBMS. The authors concluded that the use of parallelization techniques to implement algorithms is the key to achieving better scalability and performance for processing big data.

Many researchers have aimed to enhance the performance of data mining algorithms by employing distributed data mining tools, such as Hadoop. However, this means that the complexity of employing the tool to distribute the data mining algorithm and follow-up testing is the responsibility of the data mining algorithm developers. The authors [16] proposed a cloud computing framework for distributing and scheduling a cluster-based data mining application and its dataset. The main contribution of the proposed framework is to hide the complexity of the cloud from the end users and reduce the overall execution time of the data mining algorithm, without compromising mining quality.

2.3 Scalability and Performance Improvements

Scalability refers to the capability of an application to address a workload increasing with balanced performance through proportional new resources [17]. The two techniques that can be employed to scale a software system are the scale-up technique and the scale-out technique. The scale-up technique refers to vertical scaling, which involves running the software system on a computer machine that has enhanced features, including greater CPU usage, increased memory capacity, ample disk space, and increased disk bandwidth. The scale-out technique refers to horizontal scaling, which involves distributing the software system over multiple computer machines usually having similar features. The scale-up technique is not practical on the Internet scale since the features of a single computer machine cannot be increased on an unlimited basis [17].

Load balancing is a technique for improving the performance of a software system by equally distributing processing and communications activity across a scale-out network over multiple machines to avoid over loading a single machine. The workload of a machine refers to the total processing time it needs to perform all of the jobs allocated on that machine [18]. Load balancing is particularly important for systems where it is hard to predict the number of requests that will be forwarded to a machine. Applying appropriate load balancing leads to optimal utilization of the available resources, minimizing resources consumption, implementing failover, enabling scalability, avoiding bottlenecks, and reducing response time [19].

The authors [17] have focused on the scalability of SaaS applications. They identified the factors that have considerable impact on scaling a software system, such as software architecture, database access, automated migration, tenant awareness, workload support,

levels of scalability, and recovery and fault-tolerance. For complicated software applications such as SaaS, offering multiple levels of scalability, with different scalability techniques at each level, can improve system scalability. However, these benefits come with higher system complexity. Separating system functions improves system scalability, since each function can be independently developed and scaled. A SaaS system is required to know its expected workload to help the system scale for a particular workload and may allow the scalability techniques to interact with the system fault-tolerance techniques. For instance, offering various copies in several nodes helps the system to easily recover. However, each job to install new copies at different nodes may create an additional workload.

The authors [20] presented a careful analysis of Auto Scaling techniques currently available in cloud computing. They first provided background information about Auto Scaling, discussed the main benefits, and provided definitions of key concepts. Next, they presented a classification of current Auto Scaling techniques to provide developers and researchers with an understanding of existing Auto Scaling techniques. They then described the commercial and academic platforms, and finally, took into consideration the challenges and future trends of Auto Scaling in cloud computing.

2.4 Security and Privacy

Data security and privacy are major concerns surrounding cloud computing [21]. Many methods have been introduced to ensure data security and privacy in the cloud, which includes constructing secure channels to transfer data to the cloud servers, using a

cryptographic process to encrypt data before storage, and authenticating user identifications before storing or retrieving data.

The authors [21] proposed a cloud computing business solution based on the idea of separating the storage service from the encryption/decryption service. They found that cloud computing providers offer encryption as a service to help their customers to store the data securely. However, if the encryption service and the storage service are provided via the same service provider, the service provider's operators have the ability to use customers' decryption keys and internal authentication privileges to access their data. From the customers' perspective, the stored data can be accessed and affected in the cloud environment. In their proposed solution, an encryption/decryption service, as well as a storage service, is not provided by the same provider. Thus, these services will be segregated between service providers. The authorities of the storage service provider are storing user data which has already been encrypted through the encryption/decryption service without any access to the data decryption key. On the other hand, the encryption/decryption service only has authority to manage the key required to encrypt and decrypt user data. The main concept of their solution is splitting management responsibility, thus minimizing operational risk and avoiding the risk of illegal access to clients' data.

The authors [22] proposed multi-level encryption and verification system which handles the practical issues faced in order to achieve data privacy and security in the cloud. The system provides insights on how and where this multi-level encryption and data integrity solution is applied on what kind of storage device for satisfying the data security and privacy needs for different service models in cloud environment. The authors

highlighted what kind of encryption approach and methods are reasonable for maintaining between the data security and performance trade-off. For both cloud user and provider' perspective, the proposed integrated data encryption architecture is practically helpful in reducing data security and privacy risk in cloud environment.

2.5 Algorithms as a Service

An algorithm is an effective method to solve an issue or provide a solution [23]. It is generally applied for data analysis, interpretation, calculation and other related mathematical and computer processes. The algorithm is also used for processing data in a variety of ways, including data insertion, performing data searches, or sorting data.

The authors [24] described JHAVE, a Java-Hosted Algorithm Visualization Environment. JHAVE is a client-server environment for providing algorithm visualizations over the Web. In JHAVE's client-server model, the client applet delivers user requests to the server. The server will in turn run the program and generate the script file for that algorithm. The algorithm is then sent back to the client. Where the algorithm requires input from the user, an input generator object is delivered by the server to the client. The user's input is returned by the client to the server as a dataset, which in turn will be used when running the algorithm.

The authors [25] described ViSA, a tool for the Visualization of Sorting Algorithms. ViSA is an easy-to-set-up and fully automatic visualization system that provides dataset entry and animation control with step-by-step explanations and sorting algorithm comparisons. The purpose of ViSA was to develop an educational tool that would engage

students in the learning process, helping them to acquire knowledge about well-known sorting algorithms.

The authors [26] have used SQL, extended with UDFs and stored procedures, to demonstrate a cloud computing application that presents data mining algorithms as a service. The solution provides several improvements to reduce input/output, minimize data redundancy, and avoid transferring massive amounts of data, which normally causes bottlenecks in the cloud environment. To estimate the run time, the application uses a simple linear cost model that calculates local input/output speed, cloud input/output speed, transmission speed, dataset sample size, and the physical Database operator.

The authors of [27] developed a cloud server for providing data mining services publically. This solution is proposed to avoid higher costs, decrease execution time, and reduce the wastage of resources that result from the communication between a web server and a client in a general client-server model. In this model, there is a common library which needs to be shared between the client-side and the server-side. This could cause issues because all clients must be aware of this library. Furthermore, the solution aims to prevent sensitive information leaks when clients want to take advantage of the services provided by the cloud. Using this service, client database will be secured since clients only need to transmit the essential parts of the database to the cloud server in an indexed format based on the service requested by clients. Thus, no encryption and decryption operations need to be performed in the cloud and no large storage space is required.

The authors [28] developed a visual interactive and parallel data mining service called DMaaS (Data Mining as a Service) based on Hadoop and Mahout. Mahout is an open source library containing data mining algorithms that can be utilized for multiple

functions such as data clustering, data classification, association analysis, and recommendation [28]. DMaaS aims to simplify the data mining process by offering a cloud data processing platform in conjunction with a visual user-friendly interface, to interact with the data mining analysis and results.

The authors [29] proposed a model that introduces data mining techniques as cloud-based services, which are available to clients on demand. The data mining algorithms, which are widely recognized in the industry, have been implemented as Map-Reduce jobs, and are executed as services in the cloud environment. The client simply selects or uploads the dataset to the cloud, provides the data mining algorithm with the required variables, executes the job request to be processed, and receives the job results. The main contribution of this study is to present the performance analysis of running the data mining algorithm and the simple integration of cloud-based services.

Cloud'N'Sci [30] provides an intriguing business model for both algorithm developers and users. Cloud'N'Sci marketplace allows algorithm developers the opportunity to introduce their algorithms to global markets and makes their algorithms available to a wide scope of multiple business applications. Algorithmic power is provided to users in the form of data refining services, which transform raw data automatically, or simply, data in and data out. This makes application integration easier and protects developer Intellectual Property Rights (IPR) by keeping the exact behaviors of the algorithms unknown. Application developers benefit from the algorithm marketplace by reducing research and development risks and costs, and enhancing data quality.

Algorithma [31] offers a platform for sharing algorithm knowledge in a way that is scalable, compassable, and easy-to-integrate. The service can be used in two basic ways:

either by calling each available algorithm in the system via its REST API, or by writing and submitting the algorithm to be used as a web service. Algorithms can be open source, or closed, where the exact behaviors of the algorithms are not publically available. Each algorithm has its own interactive console page, so developers can test directly on the web without needing to write and implement code. The web console is a specialized console that can be used to test an algorithm by supplying the correct inputs required to execute the algorithm.

2.6 Limitations of Related Work

The previous related work provides us with a good view of the existing solutions about the algorithms as a service and helps us discover some limitations to focus on, in the proposed solution. The limitations are presented below.

- **Providing available and reliable services.** The existing algorithms solutions did not discuss multiple techniques such as, Load Balancing and Auto Scaling, to achieve system availability and reliability, since the services are offered publically for users and many users are expected to request and obtain benefit from the services.
- **Providing multiple templates to develop new algorithms**. Most of the existing solutions do not offer suitable templates for algorithm developers to contribute, develop, and deploy new algorithms easily.
- **Offering multiple services to deploy sequential and/or parallel algorithms.** Most of the existing solutions did not give the developers the ability to add new algorithms and choose an appropriate platform to host such algorithms.

- **Giving clients the ability to select hosting web server features and models.** Although some companies offer sequential and/or parallel algorithm platforms for the clients, they did not give them the ability to host the algorithms on selectable environments. In other words, the clients did not have the ability to run the algorithms on multiple web server models and features.

2.7 Summary

The chapter discussed the challenges posed by configuring the algorithms on a local machine and as a service over the Internet. This helps to determine which aspects are most important and should be addressed in the proposed solution. Finally, we reviewed currently existing algorithms as a service marketplace and their solutions to discover their limitations. The next chapter presents the proposed solution and the framework architecture.

Chapter 3

3. Proposed Framework

This chapter presents the architecture of a framework for provisioning algorithms as a service. This framework is designed to assist public users and data researchers who wish to benefit from available algorithm jobs as well as algorithm developers who are interested in deploying and examining new services. The framework is hosted in the cloud environment in order to provide a scalable, high-end environment. The chapter focuses on multiple aspects within the proposed solution in order to provide distinctive services. Moreover, the chapter focuses on the architectural details of the framework and provides an in-depth look at the component of the framework. Finally, it describes the main services in the framework.

3.1 Solution Requirements

The goal of the proposal is to develop an easy-to-use framework for public users, data researchers, and algorithm developers. The focus is on different requirements of the solution in order to provide distinctive services as discussed below.

3.1.1 Different Algorithms Different Clients

Offering a framework that enables the collection of the various categories of algorithms in one environment will be a motivational factor for clients with specific requirements to utilize these algorithm services. In other words, researchers who are working in different

fields of science can find some interesting algorithm services to invoke and examine. Furthermore, the proposed solution grants developers the possibility to host new algorithms privately under their account, or deploy the new algorithms as a service for the public. This will provide an attractive environment to implement and run the algorithms. Offering a private and high-end hardware environment to develop and examine algorithms will encourage developers to test new algorithms and compare results and performance using different web server configurations.

3.1.2 User-Friendly Interface

Providing simple requirements when developing a new service is an important aspect, as this will alleviate difficulties when utilizing services. Moreover, offering simple and clear steps to run any system will encourage clients with any level of experience to work with it. The proposed framework provides a user-friendly interface that does not require any knowledge of algorithm configuration and/or programming language for clients to run available algorithm jobs. Furthermore, users do not need to work with a database or specific data format to upload or download data. They only need to enter or upload input datasets and view or download results. All algorithm jobs follow the same workflow to process data, as shown in Figure 3.1.

Figure 3.1: Framework workflow for processing data

In Figure 3.1:

- Data Input: a dataset that will be processed by the algorithm. For simplicity, the framework supports various data formats to be uploaded to cloud web servers. Every algorithm has an instructions link to describe data preparation steps that must be completed in order for the algorithm to read the data properly.
- Execution: steps involved in running a desired algorithm job. In some algorithms, multiple helping functions are required to execute the desired algorithm.
- Data Output: a file which contains the desired algorithm results.

On the other hand, the framework offers multiple templates and classes to help developers develop and deploy new algorithms. The templates were built based on the same workflow which we provide in the available algorithm jobs. As a result, algorithm developers are required to add the algorithm source code in the execution section of the template and add any helping function which is required to execute that algorithm in another specific section.

3.1.3 Data Security and Privacy

Data security and privacy are significant factors regarding the cloud environment. Offering a secure area to store user data is an important requirement. Users, and particularly researchers, have concerns about data privacy when they want to analyze data in public services. In order to alleviate this concern and achieve data security and privacy in the proposed solution, we applied different solutions which are described in the following points:

- **Socket secure layer**

Socket Secure Layer (SSL) is a popular technique to establish an encrypted channel between a client local machine and web servers [32]. Typically, because data is sent between browsers and web servers in normal text, the data becomes vulnerable to eavesdropping. If attackers are able to interrupt data being transferred between a browser and a web server, they can manipulate or copy that information. SSL is an effective method that helps transmit data to cloud servers in a secure channel.

- **Cryptography and authentication**

Since SSL is not responsible for encrypting data in the cloud, we use a cryptography algorithm to encrypt algorithm results before storage. An authentication method is applied to identify a user before retrieving the result. In order to provide data privacy we apply the idea of separating a storage service from an encryption and decryption service [21]. The framework creates an encryption and decryption key for a user within the registration process and separately saves all user identity information in a separate database server rather than the data storage area and application web servers. The storage area is storing algorithm job results that have already been encrypted via the encryption and decryption service in application servers, with a user decryption key already stored in the Database server.

- **Deletion of client input data**

The last solution applied to achieve added data privacy involves deleting all client input files from the web servers at the end an algorithm job. The input data, which is uploaded

to be processed by the algorithm, is temporarily stored in the web server storage during the execution process and removed when the execution process is done.

3.1.4 Scalability and Availability

Since the framework will serve multiple clients, it is important to apply some techniques to achieve a scalable and available framework. We applied Load Balancing and Auto Scaling technologies as a solution to improve the scalability and availability of the proposed solution. The Load Balancing is responsible for distributing expected incoming application workloads across multiple web servers [33]. The Load Balancing server is also responsible for ensuring that only healthy web servers receive requests by detecting unhealthy web servers and changing new request paths toward the remaining healthy web servers [33]. After a failed web server is repaired, the Load Balancer will resume sending new requests to that web server. The Auto Scaling is responsible for scaling the application horizontally based on the web server CPU utilization and the Load Balancing workload [34]. The Auto Scaling will launch new web servers if the CPU utilization of the current web servers is increased, and delete any web server if its CPU utilization is decreased. This will help in fulfilling application availability and resource management.

3.1.5 Scalability and High Performance

One of the main contributions in this solution is improving algorithm performance using different scalability models and techniques. Offering multiple on-demand web servers that have high-end resources, and providing a simple solution to launch and terminate those servers, will allow users and developers to achieve high performing algorithm services. Since the framework will be offered in the cloud environment, more flexibility is granted

when provisioning vertical scaling using a single high performance web server and horizontal scaling using distributed web servers.

3.2 Framework Architecture

An overview of the framework architecture is shown in Figure 3.2, where the core layers of the framework and their components are presented. As illustrated in this diagram, **the first layer is an application layer** which works as an entrance to the services are offered by the framework. This layer contains available algorithm services to be run over the default platform. It also allows clients to re-run the available algorithm services over a scale-up service using the default solution image as a service. The default solution image is a full backup of our solution, including operating system and service implementation, which can automatically be re-installed in a new web server to run the framework services. In order to utilize algorithm services, clients only need to enter or upload the specific part of their data which they want to analyze and examine. This input data is stored in the requested web server storage and deleted after the execution process ends. On the top of the application layer, we apply the first security layer which is responsible for the verification of client authentications by using their usernames and passwords and constructing a channel to transfer data from the client machine to the web servers in the cloud using the SSL.

The second layer is a platform layer which offers two different services for algorithm developers to deploy and host new algorithms. These services are the scale-up service and the scale-out service.

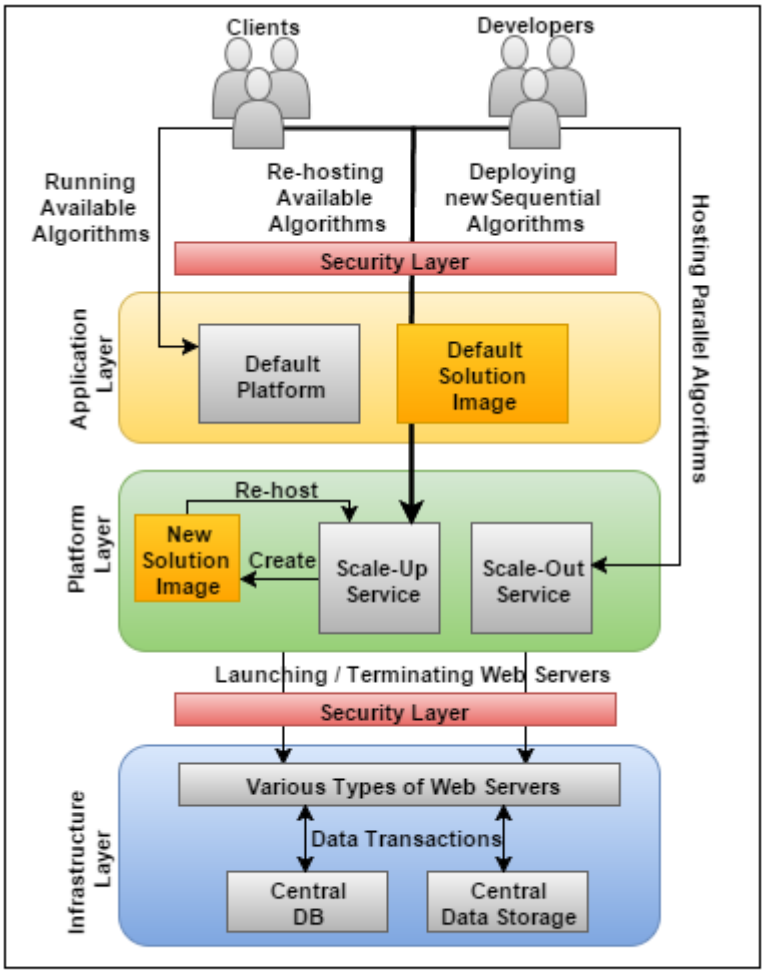


Figure 3.2: Framework architecture

The scale-up service allows developers to develop and deploy new sequential algorithm services using a high-end web server configuration, analyze data, and compare results and performance with the available algorithm services. Furthermore, this service will provide developers with the ability to create a new solution image, which in turn will incorporate their new algorithm services. The new solution image may then be re-hosted over different high-end web server configurations. The scale-out platform allows algorithm developers to host new parallel and distributed algorithm services over distributed web servers. The scale-out platform authorizes developers to select the number of web servers and their models and features to execute the algorithm service.

The third layer is an infrastructure layer which has multiple web servers to host the framework platform, contains a storage area to store the algorithm results, and also has a database server to store client information. The database information is managing data transactions and authentications between the web servers and the storage area. Moreover, the framework grants clients the ability to store the results inside the storage area in an encrypted format. The clients also have the ability to delete the algorithm results from the storage area after downloading them. The second security layer, which is the upper infrastructure layer, functions as a firewall to monitor the traffic that will be allowed to access the infrastructure components.

Integration of these layers helps providing highly dynamic services and achieving a high performance framework. Allowing clients to launch a suitable and on-demand web servers based on their requirements is another big advantage to the proposed framework.

3.3 Framework Services

The following sections discuss the main services which are offered by the framework.

3.3.1 Searching and Running Algorithm Services

The framework offers multiple categories of algorithms as a service to give clients the opportunity to easily select an appropriate algorithm service and analyze data. Furthermore, any requirements for programming algorithms or implementing hardware components are avoided. Clients simply register with the framework, and then begin invoking these services using the default platform or run the available algorithm services over a high performance web server using the scale-up service.

3.3.2 Developing and Deploying New Algorithm Services

This service allows algorithm developers to develop and deploy new algorithms and run them using the framework services. With the proposed solution, developers can examine the accuracy and performance of their algorithms utilizing high-end resources in the cloud and achieving a pay-per-use concept. Developers can host their own algorithms privately under their own account or publicly to be run by the clients. The framework provides templates and libraries to help algorithm developers develop new algorithms using the default workflow for processing data. When the participating developers decide to offer their algorithms as an open source, clients will be able to test and run these algorithm services using the default interface. All available algorithms will be offered for developers as templates to build on and deploy as a new algorithm.

3.3.3 Platforms as a Service

Providing multiple platform services to host and run algorithms will help users to analyze data and help developers to develop and examine algorithms in a more flexible and robust environment. Clients have the option of running and/or hosting algorithms using either a high vertically scaled service, or a horizontally distributed scaled service. These services allow clients to run sequential and parallel algorithms. The framework also allows clients options to select web server models and configuration features to host the requested service.

3.3.3.1 Vertical Scaling Platform

Vertical scaling (scale-up) service provides clients the ability to activate the algorithm services inside different web server features, which in turn will help clients analyze data

effectively and improve algorithm performance. The proposed solution provides a scale-up service allowing clients to run available algorithm services over a high web server resource usage using a default solution image. An example would be web servers with multiple CPU usage and massive memory capacity. A further advantage is that the framework allows developers to create a new solution image to involve new algorithm services which are registered under their account, and may host this new image in another vertical scaled web server to examine the algorithm's accuracy and performance.

3.3.3.2 Horizontal Scaling Platform

Horizontal scaling (scale-out) service offers a distributed environment that gives users and developers an opportunity to run Hadoop Map-Reduce algorithms to analyze vast amounts of data [35]. This environment makes it easier and faster to process and distribute large amounts of data over multiple web servers. It also achieves cost-effectiveness since the algorithm services are offered based on a pay-per-use concept. Therefore, less execution time results in cost reduction. The service also grants users and developers an opportunity to run Hadoop Map-Reduce algorithms on distributed and high performance web servers.

3.4 Summary

Offering a framework for analyzing data and examining an algorithm's accuracy and performance will allow public users, researchers, and developers to gain benefit from this framework. The chapter described the solution requirements for offering attractive services. The proposed solution provides simple requirements, as well as security and privacy methods, to avoid difficulties and data falsification when utilizing the algorithm services. The proposed solution also applies the scalability techniques to achieve system

availability and improve performance. The chapter then introduced concepts and detailed information regarding the framework architecture and its layers. Finally, it discussed the main services that are provided in the framework. The next chapter will discuss the implementation of a reference prototype based on the proposed framework.

Chapter 4

4. Prototype Implementation

This chapter provides, in general, an overview of the implementation phases. The chapter discusses the prototype configuration, which may be used as a reference for any implementation. We used AWS to host the infrastructure layer, as well as the ASP.NET framework and C# programming language to build and implement the platform and application layers. The chapter is organized as follows: Section 4.1 will discuss the implementation phases, as well as the prototype configuration and its components. Section 4.2 provides a discussion of the rationale and the implementation language. Section 4.3 details the user-interface. Finally, Section 4.4 describes the implementation of services in the prototype.

4.1 Implementation Phases

In order to build our prototype implementation, we divided our work into three phases, as seen in Figure 4.1.

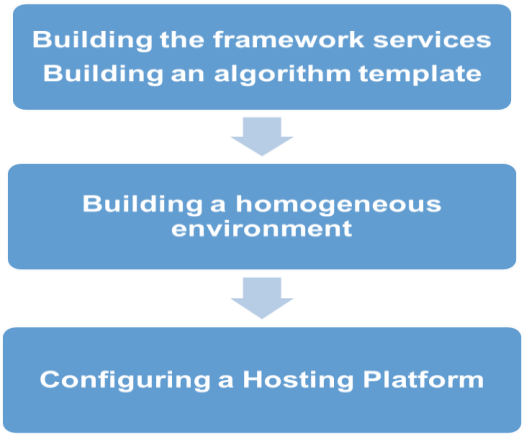


Figure 4.1: Implementation phases

The first phase involves developing the framework services and building an algorithm template to deploy the current algorithms. The template would also be available to developers to develop and deploy new algorithms. The second phase entails selecting and configuring a hosting platform to host and run the framework services. Finally, the third phase requires building many classes and installing required libraries to provide a homogeneous environment between the framework services and the hosting platform resources.

4.2 AWS

Hosting the framework on a highly scalable and multi-service infrastructure is critical in achieving a more realistic implementation. The features of the infrastructure, such as the number of CPU cores and the memory capacity, have greatly impacted the experiment in terms of achieving algorithm results and minimizing the execution time. We decided to take the virtualization route using the Infrastructure-as-a-Service (IaaS) solution package offered by Amazon via their Amazon Web Services (AWS). AWS offers freedom in configuring a virtualized distributed architecture using its large array of IaaS services [7]. In addition, the prototype instantly benefits from the security and reliability of the AWS infrastructure. AWS also provides a web solution to host a .NET web application on its web services and applies the pay-per-use concept, which has opened up many more options for experimentation.

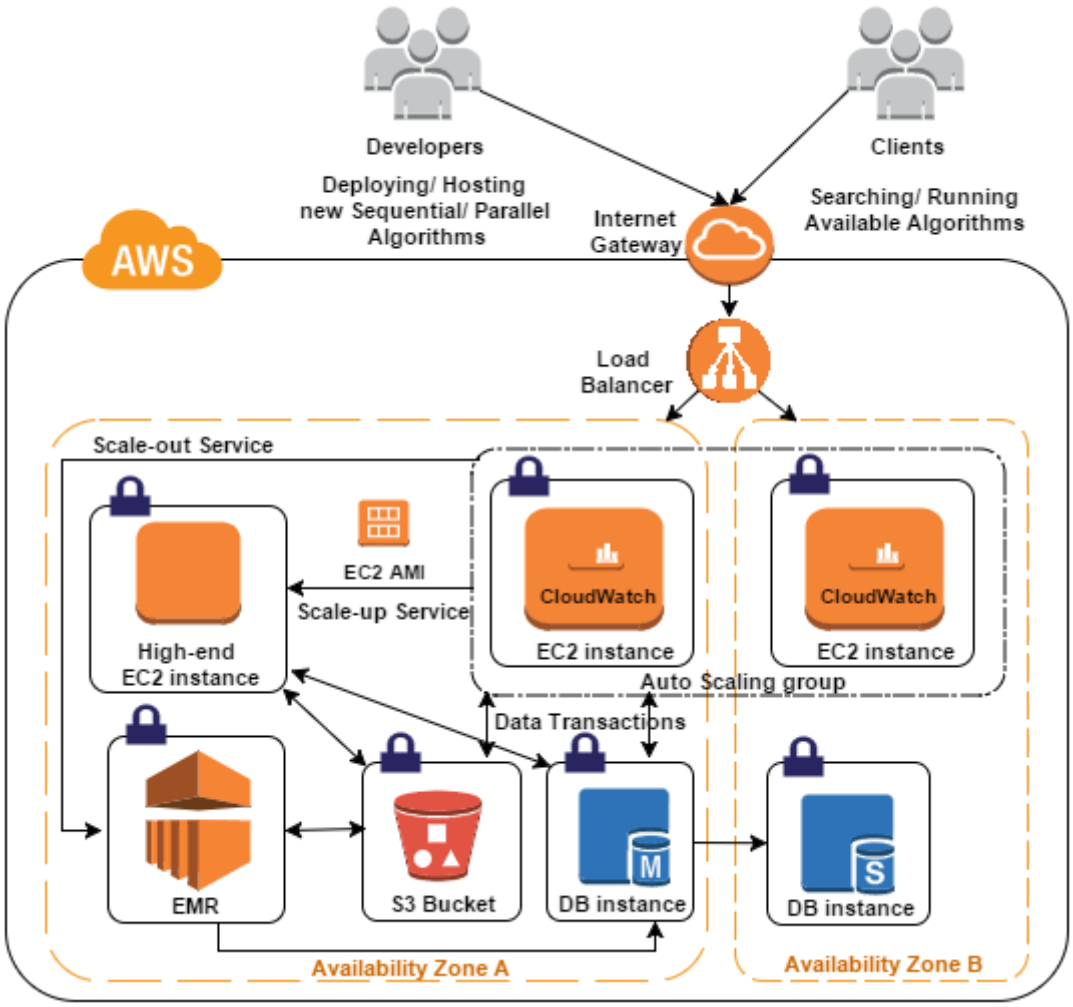


Figure 4.2: AWS configuration for the prototype implementation

Figure 4.2 illustrates the design of the prototype implementation on AWS. Public users, data researchers, and algorithm developers have the ability to request services from the application by utilizing available algorithms or hosting new algorithms. All requests are first processed by a single load balancer which distributes traffic across default web servers. Clients can select the most appropriate scale services to process their requests. The services can be hosted over multiple models of web servers to run the available algorithm services, or deploy new sequential or parallel algorithms. These web servers are connected with central data storage to store algorithm output results and a central database to store

application and client information. Moreover, the application provides a cryptography algorithm as an optional service that is executed in the cloud to ensure that algorithm results are stored in an encrypted format. Finally, once the algorithm process ends, clients are provided the opportunity to download the result. Completion of the execution process will delete all client input files from the web server storage. This will increase data privacy and effectively minimize the storage area in the cloud.

The following sections briefly explain the components that we used to host a scalable and high performance web application on the AWS infrastructure.

4.2.1 EC2

We used Amazon Elastic Compute Cloud (EC2) to create a virtual server that hosts the prototype implementation. Amazon EC2 is a web service developed to provide turnkey web scale computing solutions for clients and flexible compute resources in the cloud [36]. EC2 offers numerous choices of operating systems pre-packaged as an Amazon Machine Image (AMI) [36].

The AMI provides the information required to start an EC2 instance in the cloud [36]. It initiates permissions that monitor AWS accounts which can use the AMI to start EC2 instances. It also contains a template for the instance root volume, such as an operating system and an application server, and a block device mapping that describes the volumes that connect to the instance when it starts [36].

We use Amazon Cloud Watch to monitor Amazon EC2 instances in near real-time. It automatically provides metrics for CPU utilization, latency, and request counts for EC2 instances. Amazon Cloud Watch also provides Amazon EC2 instance metrics aggregated

by Auto Scaling group and by Elastic Load Balancer [36]. This is discussed in detail in Sections 4.1.6 and 4.1.7.

For the prototype, we launched two EC2 instances running Microsoft Windows 2012 R2 Standard edition with 64-bit architecture. All default EC2 instances were powered by hardware model t2.micro. This model offers a single core with a high frequency Intel Xeon processor with turbo up to 3.3GHz and approximately 1GB RAM. It also provides a balance of compute, memory, and network resources [36], as shown in Table 4.1 [37].

Table 4.1: Cloud EC2 Instance Features

Model	vCPU	Mem (GB)	SSD Storage (GB)	Networking Performance	Physical Processor	Clock Speed (GHz)
t2.micro	1	1	EBS Only	Low to Moderate	Intel Xeon family	Up to 3.3

4.2.1.1 EC2 Security Group

A security group serves as a firewall that monitors the traffic allowed to access one or more EC2 instances [36]. One or more security group/s can be allocated for an EC2 instance to control the traffic of that EC2 instance. For the implementation, we created one security group as a reference for all EC2 instances that were launched, and also for any new EC2 instances. We then selected inbound HTTP rules for that security group. Furthermore, we configured the security group to accept any IP address. This allows users to reach and access the prototype from any physical location.

4.2.2 S3

Amazon Simple Storage Service (S3) provides developers and IT groups with a scalable, durable, and secure data storage service [38]. Amazon S3 gives users the ability to store and/or retrieve any amount of data, at any time, from anywhere on the web. Moreover, Amazon S3 offers a pay-per-use storage model for multiple uses, including cloud applications, data distribution, data backup and archiving, and Big Data analysis. Amazon S3 stores files as an object in a location called a "bucket" [38]. A single Amazon S3 storage bucket was created to store all output data created from the results of the algorithm services. This S3 bucket is connected with all EC2 instances as a centralized storage area. We provided an encryption option service to grant clients the ability to store the result in encrypted format in the storage area. In this implementation, we used an Advanced Encryption Standard (AES) symmetric algorithm because it uses stronger protocols than asymmetric encryption [21]. In doing so, the resulting data is more securely stored in the cloud.

4.2.3 RDS

In order to run the Database server, Amazon Relational Database Service (RDS) was used. Amazon RDS is a web service that enables users to easily establish, operate, and scale a relational Database in the cloud [39]. The Database instance was created running Microsoft SQL Server Express Edition that is powered by hardware class db.t2.micro with allocated storage of 20GB. Furthermore, a multiple availability zones Database instance was applied. This means that Amazon RDS automatically provides a synchronous standby replica in several availability zones [39]. In order to protect the latest Database updates against DB

server failure, updates to the main DB server are asynchronously sent to the standby replica across availability zones [39]. Finally, we created a security group that grants the application servers the permission to access the Database server.

4.2.4 ELB

Amazon Elastic Load Balancing (ELB) is a technique designed to help users promote the availability and scalability of their software systems [33]. The main objective of ELB is to automatically distribute incoming application workloads amongst multiple EC2 instances in the cloud [33]. This assists in providing the desired amount of Load Balancing capability required to distribute application workloads and fulfils maximum levels of fault tolerance in the application.

In order to configure the Amazon ELB, we selected HTTP as the protocol between the EC2 instances and the ELB. We also selected 80 as the open port number for both the ELB and the EC2 instances, which allows the ELB to receive new requests from users, and then forward these requests to the EC2 instances. A response timeout of 10 seconds was set, which means that the ping request must receive a response within 10 seconds to determine whether the node is healthy and able to process incoming traffic. For a health checks interval, which is the time between every health check operation, the time of 30 seconds was set. The number of sequential health check success occurrences before confirming an EC2 instance as being healthy, was set at 2. Furthermore, the number of sequential health check failure occurrences before confirming an EC2 instance as being unhealthy, was also set at 2. In order to promote the availability of the ELB, subnets were selected from different availability zones. It should be noted that EC2 instances were not

registered to the ELB since the Auto Scaling group is responsible for launching the instances and attaching them to the ELB.

4.2.5 Auto Scaling

Amazon Auto Scaling helps to ensure that there are a sufficient number of EC2 instances available to process the load on an application [34]. Collections of EC2 instances contained within Auto Scaling is called an Auto Scaling group [34]. The specification of the minimum number, the maximum number, and the desired number of instances in each Auto Scaling group must be defined. In defining the desired number, Auto Scaling guarantees that the group has this number of instances available. Furthermore, by defining scaling policies, Auto Scaling can launch or terminate instances as the load on the application increases or decreases [34].

In order to apply the Auto Scaling group, we first selected an AMI that contains the whole solution implementation, including the preferred type of operating system and web-server models. This AMI serves as a template that the Auto Scaling group will use to launch the EC2 instances. The Auto Scaling group was defined to have a minimum EC2 instance size of one, the required EC2 instance size of two, and the maximum EC2 instance size of three. In order to build a fault-tolerance application, we selected two different availability zones. Thus, if one availability zone has a negative issue, traffic will be routed to the healthy zone [34]. The Auto Scaling group was configured to scale out by one instance, any time a change in running EC2 instance capacity was measured by the Amazon Cloud Watch. The Auto Scaling group automatically adds an EC2 instance when CPU utilization is high (breaches the alarm threshold: CPU utilization > 60% for 2 consecutive periods of 300 seconds), and removes the EC2 instance when CPU utilization is low (breaches the

alarm threshold: CPU utilization < 10% for 6 consecutive periods of 300 seconds). In other words, if the average CPU utilization of the EC2 instances increases more than 60% for consecutive periods of 10 minutes, the Auto Scale group is responsible to launch one new EC2 instance to the running web servers using the template defined earlier. However, if the average CPU utilization of EC2 instances decreases less than 10% for consecutive periods of 30 minutes, the Auto Scale group is responsible for removing one instance from the web server group.

We enabled Amazon ELB and Amazon Auto Scaling services to ensure that EC2 instances, which are launched via the Auto Scaling service, are automatically registered with the ELB, and that the EC2 instances that are terminated by the Auto Scaling service, are automatically deregistered from the ELB.

4.2.6 EMR

Amazon Elastic Map-Reduce (EMR) is a web service that can quickly and cost-effectively process large amounts of data [40]. It simplifies Big Data processing, and provides a managed Hadoop framework that allows developers distributing and processing massive amount of data across dynamically scalable Amazon EC2 instances [40]. For the prototype, a Hadoop streaming service was implemented. This is a utility that comes with Hadoop, enabling developers to develop Map-Reduce algorithms in any of the following supported languages: Ruby, Perl, Python, PHP, R, Bash, and C++ [40]. Developers must follow the directions in Hadoop's documentation to write their streaming program. The programs read input data from standard input and output data through standard output in Amazon S3.

4.3 Implementation Language

Since the AWS is selected as a cloud provider for the implementation of the prototype, C# programming language and ASP.NET were used to build the prototype inside these instances. These were selected since AWS provides a software development kit (SDK) for the .NET framework to make it simpler for Windows developers to build .NET applications that lead to pay-per-use, reliable, and scalable AWS services [7]. In order to offer more helpful resources, the AWS SDK for .NET contains the AWS .NET library, Visual Studio project templates, C# code samples, and support documents [7].

Overall, the choice of implementation language is not a crucial factor for the development of an implementation of the proposed framework. Implementing the prototype in other languages that have AWS SDK, such as Python and Java, would offer valuable insights. However, this falls outside of the scope of this thesis.

4.4 User Interface

The implemented prototype aims to offer simple and clear steps to utilize the services. This helps public users and researchers, who have limited experience with algorithms and data storages, to benefit from the services. Moreover, detailed documentation for how to run each service is available in the service helping links. We describe the general workflow for using the framework below:

1. Registration: This is the first step to use the services. There are two levels under the registration process as shown in Figure 4.3, which are basic registration and advanced registration.

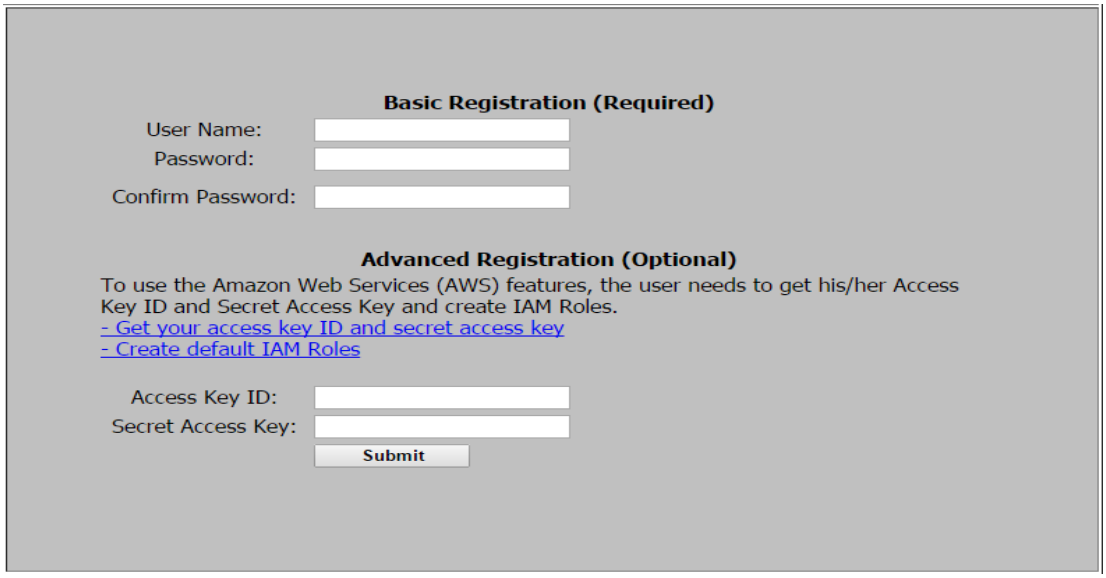


Figure 4.3: Framework registration

a. Basic Registration: Under this registration, framework clients have to input a unique username and password. This level of registration allows clients to analyze limited amounts of data via textbox input, using the available algorithms over the default web servers.

b. Advanced Registration: Under this registration, clients have full access to the framework services and can analyze much larger amounts of data via file uploads. Clients must provide their AWS account information within the registration process as shown in Figure 4.3.

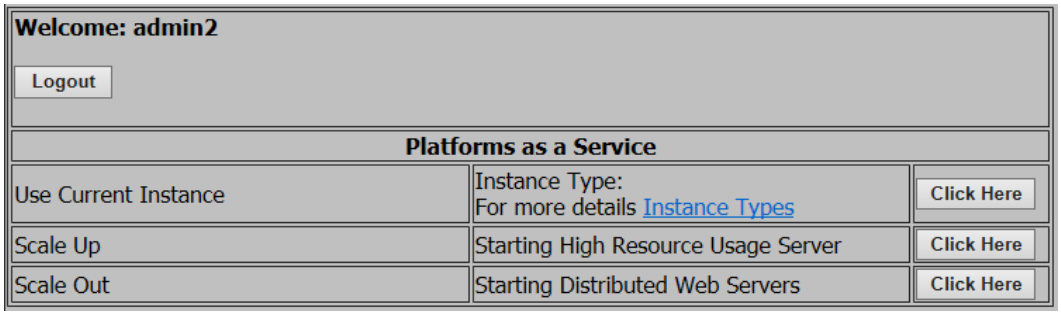


Figure 4.4: Main menu for platforms as a service

2. Choose a platform: Figure 4.4 shows the platforms available as a service. Clients can choose running or hosting algorithms within the framework default web servers, a high performance web server, or distributed web servers.
3. Choose an algorithm: Clients can view lists of the currently available algorithms by selecting the algorithm category under general jobs in the left sidebar, as shown in Figure 4.5. Each algorithm includes a description link. Moreover, in some algorithms, clients will be prompted to enter specific parameters before executing that specific algorithm job. For example, in the K-Means algorithm, a client must provide the algorithm with additional input parameters, which are the number of clusters and iterations.

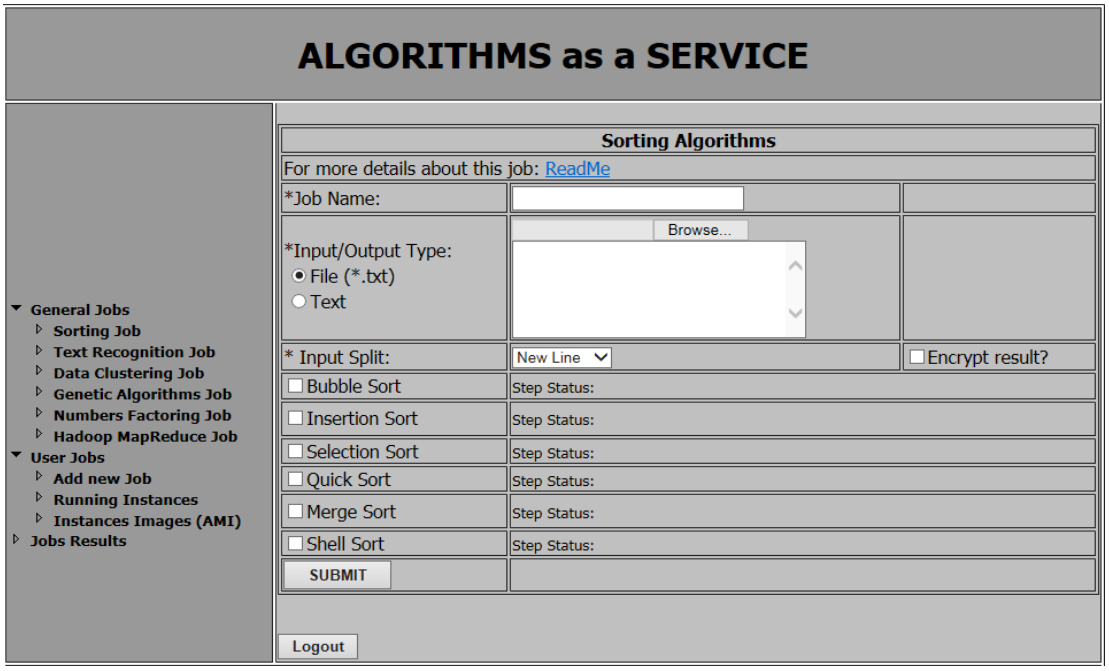


Figure 4.5: Sorting algorithm interface

4. Upload a data set: Clients can upload a data set file or enter the data manually in the input text area. It is important to follow the specific data structure required for the

algorithm to run. The structure must be followed for the job to read and process the data properly.

a. File: For this implementation, the uploaded data set must be in (.txt) format. This will help users easily upload their data file since they do not need to transfer the data file to a specific advanced format.
b. Text: Is applicable for small size data.
c. Input split: Describes the data split in the same line using a given separator (new line, semicolon, comma, or slash).

5. Execute a job: Once clients select a desired algorithm and have uploaded a properly structured data set, the system is ready to execute the job.
6. Download a result: The last step in the workflow is obtaining an algorithm result. After the execution process is done, clients can download the result from the cloud storage to a local machine, as shown in Figure 4.6.

Job Results

		JobName	CreationDate
Download	Delete	SortingTest12015-09-07-222626	9/7/2015 10:26:37 PM
Download	Delete	GA2015-Sep-07-223028	9/7/2015 10:30:34 PM
Download	Delete	GA2015-Sep-07-223028	9/7/2015 10:30:34 PM
Download	Delete	pf_2015-09-07-223205	9/7/2015 10:33:18 PM
Download	Delete	WCHadoop2015-09-07-224840	9/7/2015 10:53:17 PM
Download	Delete	22015-09-20-190802	9/20/2015 7:14:24 PM
Download	Delete	book4-22015-09-20-203802	9/20/2015 8:57:42 PM
Download	Delete	12015-09-25-111549	9/25/2015 11:15:51 AM
Download	Delete	402015-09-25-111609	9/25/2015 11:16:09 AM

Figure 4.6: Results of running algorithms

4.5 Service Implementation

This section describes services implemented in the prototype, which provides an overall picture of the framework architecture and its services. We focus on the main services that were offered as a contribution in the proposed solution.

4.5.1 Utility Services

Before describing the provided services, we need to give consideration to the utility services. These services are not provided to clients as a service to invoke, but rather, offered as a library and classes to implement existing services and to help developers to develop and deploy new algorithm services. Since we applied our solution on AWS, we also built several classes to use as a basis for the framework services, in order to provide a homogeneous environment between the provided services and Amazon services. These helping services make it easier for clients to upload data to the cloud, run or host algorithms over multiple web servers, and download results to their local machines. We built our helping classes based on AWS developer guides to gain whole benefit from Amazon services. Some of these services are described with their source code in Appendix B.2.

4.5.2 Running Algorithms as a Service

The prototype solution provides different open source algorithm categories as a service, which allows users to analyze data in a high resource usage environment.

4.5.2.1 Sorting Algorithms

A sorting algorithm is an algorithm that arranges a list of elements in a specific order [41]. This type of algorithm is divided into two main categories, namely, comparison based and non-comparison based sorting algorithms. The comparison based sorting algorithms include Bubble sort, Quick sort, Insertion sort, Selection sort, Shell sort, and Merge sort. The non-comparison based sorting algorithms include Radix Sort and Bucket Sort. [41]. In the prototype, our focus is on comparison based sorting algorithms and the implementation of different algorithms, which allow users to compare their data with more than one algorithm.

4.5.2.2 Word frequency Algorithm

The word frequency algorithm scans text files and computes how many times a word appears in the text [42].

4.5.2.3 Clustering Algorithm

Data clustering is the process of dividing data items into groups based on their similarity [43]. Thus, items within the same group are similar and items in different groups are dissimilar. K-Means algorithm is the most popular technique for clustering numerical data [43]. In the prototype, we implemented the K-Means algorithm. The K-Means algorithm is useful for unmixed numeric data. Data clustering is classified as one of many machine-learning algorithms, and data clustering can also be applied to execute a particular data analysis task [43].

4.5.2.4 Genetic Algorithm

The genetic algorithm is an evolutionary approach to computing, inspired by Darwin's theory of evolution and biological reproduction, which has the power to determine approximate solutions to optimization problems [44]. For a prototype approach, we applied the traveling salesman problem. The traveling salesman problem is well-used in AI circles. Given a list of cities, a salesman has to identify the most efficient route that enables the salesman to visit every city once, and only once. Any city can be selected as a starting point [44].

4.5.2.5 Number Factoring Algorithm

In numbers theory, the prime factors of a positive integer are the prime numbers that divide that integer exactly [45]. The prime factorization of a positive integer is a list of the integer's prime factors, together with their multiplication [45].

4.5.3 Deploying New Algorithms as a Service

Allowing developers the opportunity to develop and deploy preferred algorithms is one of the contributions in the provided solution. Developers will deploy algorithms and run them without configuring a specific hardware or software environment. Developers will test all source codes privately under their accounts. This allows more privacy while testing algorithms as they will not be shared publicly.

In order to host an algorithm, developers need to choose one of the provided services based on the algorithm type, or more specifically, whether it is a parallel or sequential algorithm. This algorithm will be hosted privately under their account within separate web servers, using the default solution image. This will help achieve system security as the new

algorithm source code will be hosted in separate EC2 instances. Furthermore, developers' privacy will be achieved as the EC2 instances will launch under their own AWS account to host the algorithm source code. The developers have the ability to create a new solution image to involve their new algorithms to be available under their account, and re-run this image using new EC2 instances. The developers also have the ability to delete their work when they desire by terminating the EC2 instances and deleting their own solution image.

We offer a default template and some available algorithms as templates for the developers, which will allow them to develop and host these templates as new algorithms, as shown in Figure 4.7. The template has two files, which are code file and design file. Developers can update the design file based on the new algorithm requirements and add the main algorithm source code and algorithm helping functions to the file code in their sections. An example of the default template code file and design file are presented in Appendix B.1.

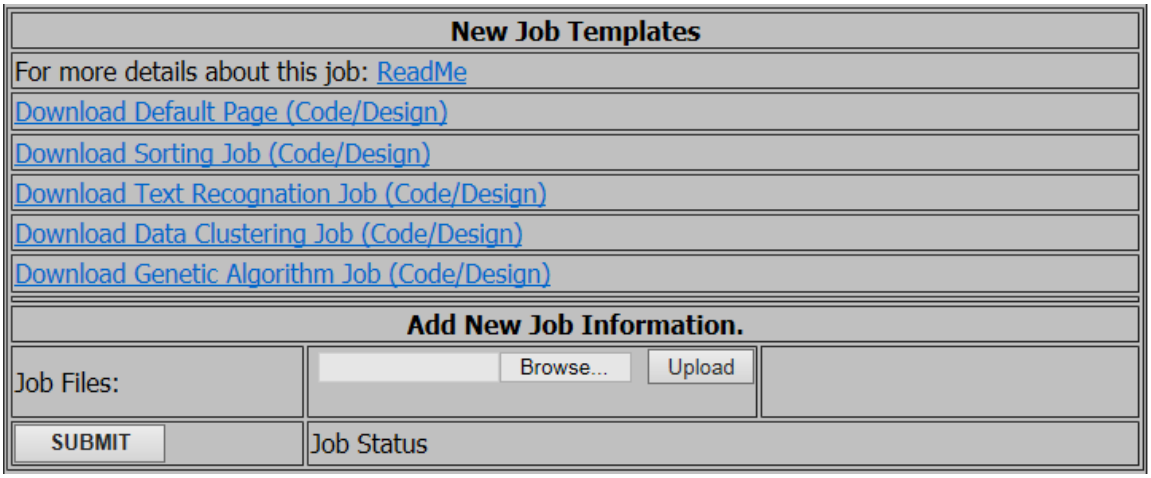


Figure 4.7: Hosting developer algorithm service

4.5.4 Platform Services

The third contribution of this thesis is offering different scalable platforms. Furthermore, providing users and developers with the ability to analyze and interpret data in multiple web server models and configurations allows access to high performance platforms, the ability to compare results, and the opportunity to minimize execution time. A homogeneous environment was implemented between the services provided in the framework and AWS services, which grants users and developers a bridge to easily invoke AWS services. To gain access of these services, clients must provide their AWS account information within the registration process. The platform services are described below.

4.5.4.1 Scale-Up Service

Figure 4.8 shows the first homogeneous service in the framework solution, which is the scale-up service. Amazon EC2 and Amazon AMI were utilized to configure the scale-up service to grant users and developers the possibility to run and/or deploy sequential algorithms on a single high-end web server platform to test performance and examine accuracy. Furthermore, they can create an EC2 instance, which has more CPU cores and memory usage, to examine data using the default solution's image or to create a new solution image to import all new algorithms under their account and scale-up those services using new EC2 instance. As shown in Figure 4.8, users and developers can launch multiple EC2 instances under their AWS account, use the default image of the solution or create their own image, and terminate those instances after jobs are finished.

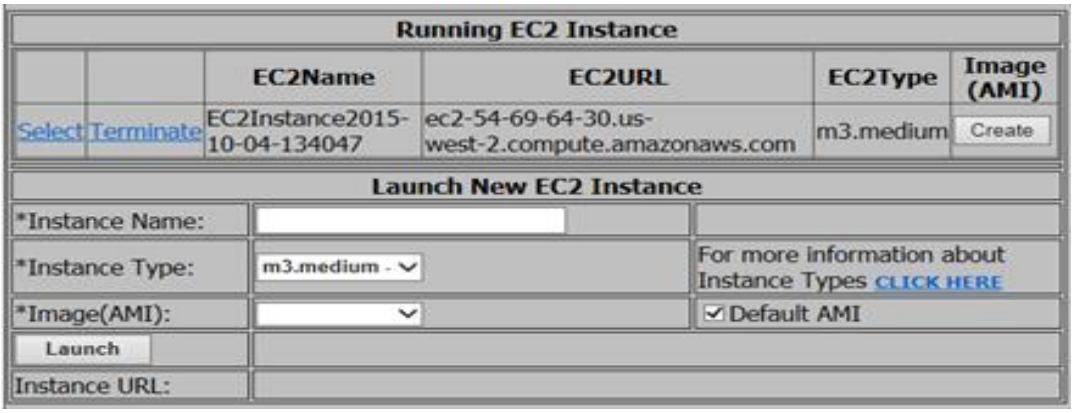


Figure 4.8: Scale-up service interface

4.5.4.2 Scale-Out Service

Figure 4.9 shows the second homogeneous service in the framework solution, which is scale-out. The service utilized two AWS services, which are Amazon S3 and Amazon EMR. This service allows algorithm developers to host parallel algorithm in a more effective and distributed web server platform. Developers can host Map-Reduce algorithms in any of the following supported languages: Ruby, Perl, Python, PHP, R, Bash, and C++ to run over multiple EC2 instances, which will allow analyzing Big Data sets and obtaining results. One of the most important considerations when launching a distributed platform is the number and type of instances to launch. This will determine the processing power and storage capacity of the platform. As shown in Figure 4.9, developers have the ability to select how many EC2 instances they want to launch and the model type for the desired instance. They also can upload multiple data sets to analyze Big Data. Finally, they will upload the job mapper and reducer algorithms. Instead of providing a reducer script, developers can use built-in Hadoop classes, such as aggregate, as a job reducer.

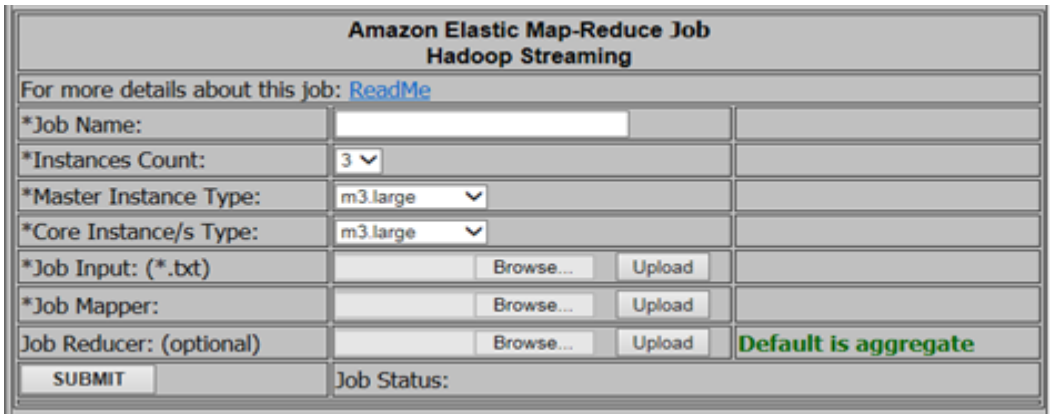


Figure 4.9: Scale-out service interface

4.6 Summary

This chapter presented a detailed description of the prototype implementation. This implementation is divided into three significant steps, namely, selecting the appropriate cloud provider, implementing some common algorithm jobs as services, and assisting clients to obtain their results within a short time period. Although the services used in this prototype were completely implemented in the .NET framework, they can be extended to any other programming platform which offers the Amazon SDK. The experimental evaluation of the prototype will be discussed in the following chapter.

Chapter 5

5. Evaluation Results and Discussion

Evaluation is a necessary component in system design to ensure system operability, availability and performance. This chapter presents evaluation results and analysis of the prototype implementation of the framework for provisioning algorithms as a service, with an emphasis on scalability models, availability and runtime performance. The experimental design and methodology of the evaluation is discussed in Sections 5.1 and 5.2. Sections 5.3, 5.4, 5.5 and 5.6 show the experiments results and analysis.

5.1 Experimental Design

We designed four experiments to evaluate the prototype implementation and investigate how the prototype can offer high algorithm performance and provide an environment that is highly scalable, available, reliable, and comparable. We used algorithms from multiple X-intensives (CPU intensive, Memory intensive, etc.) to help us perform a fair evaluation. A total of five diverse algorithms were used in order to obtain realistic usage scenarios, which are:

1. Bubble Sort algorithm
2. Quick Sort algorithm
3. Word Frequency Counter algorithm
4. K-Means algorithm
5. Genetic algorithm

5.2 Methodology

In order to examine the runtime performance, scalability models, availability, and reliability of the prototype, we generated different dataset groups to use as input for the algorithms.

1. Sorting algorithms: we built a function to generate four dataset groups which have random numbers between 0 and 9,999,999, as shown in Table 5.1

Table 5.1: Sorting algorithm dataset groups

Data Set	Numbers
1	10,000 different numbers
2	200,000 different numbers
3	500,000 different numbers
4	100,000 different numbers

2. Word Frequency Counter algorithm: we used four different sized books, which were downloaded from Project Gutenberg [46] as plain text books, as shown in Table 5.2.

Table 5.2: Word Frequency Counter algorithm dataset groups

Data Set	Book Size (estimate)
1	7.5 MB
2	15 MB
3	544 MB
4	1 GB

3. K-Means algorithm: we built a function to generate three dataset groups which have random numbers between 0 and 9,999,999, as shown in Table 5.3.

Table 5.3: Clustering algorithm dataset groups

Data Set	Numbers
1	300,000 different numbers
2	500,000 different numbers
3	1,000,000 different numbers

4. Genetic algorithm: we built three dataset groups using different US cities from different US states, which were downloaded from realestate3d.com [47], as shown in Table 5.4.

Table 5.4: Genetic algorithm dataset groups

Data Set	City Numbers
1	220 cities from different states
2	291 cities from different states
3	400 cities from different US states

5.3 Default Platform Evaluation

Since the default platform is used as an entrance for the framework and its services, it will be accessible by multiple clients who wish to run available algorithms or launch a new scalability services. It is important to evaluate this platform carefully to ensure that it has the capability to provide the requested services. The first experiment focused on stress testing, as well as availability and reliability, to evaluate Amazon Auto Scaling, Cloud Watch and Amazon ELB, which we utilized in the proposed solution. The motivation behind a stress testing is to find out how easily the framework can recover from overload conditions. The stress testing is an important requirement in evaluating the framework availability and reliability. The availability is the quality aspect and measures whether the framework is obtainable and ready to serve clients [48]. The reliability is the quality aspect of the framework that refers to the capability of maintaining the service and service quality [48].

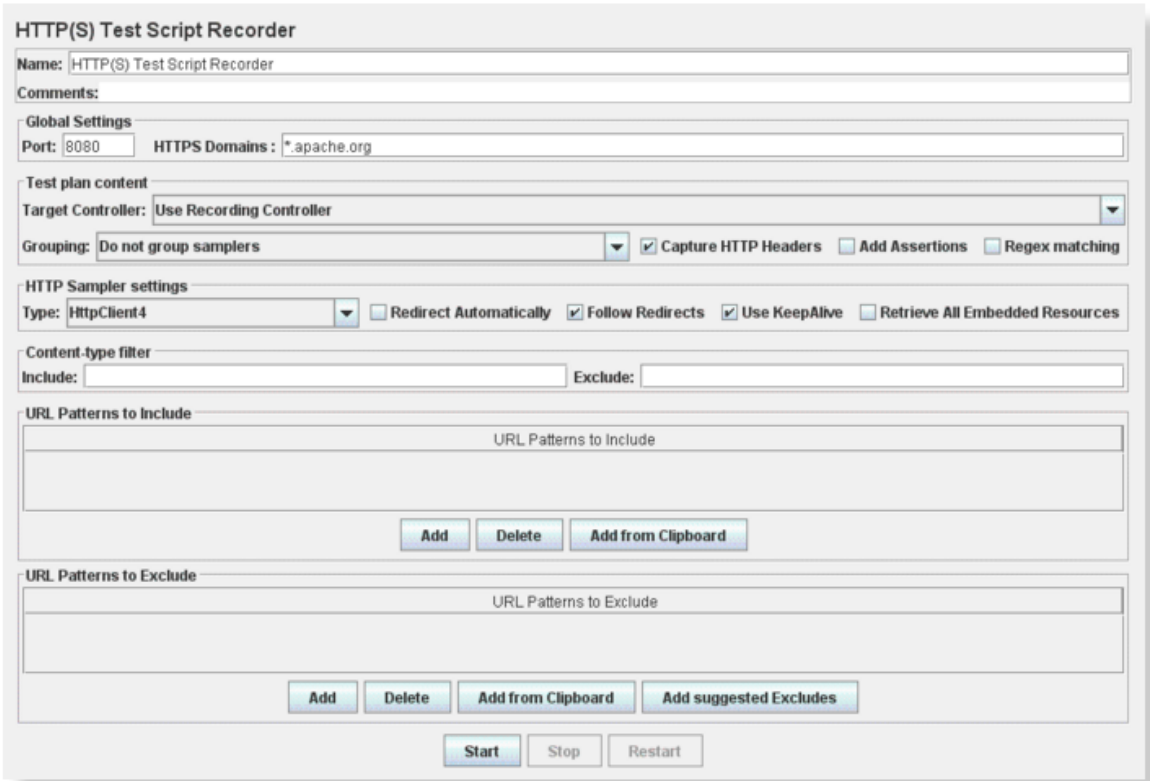


Figure 5.1: JMeter test script recorder

5.3.1 Test Environment

In order to build the test scenario, we used the Apache JMeter application which is a Java open source software designed to load test functional behavior and measure performance [49]. We ran the JMeter on a local machine to record a test scenario using http/s test script recorder service, as shown in Figure 5.1. The http/s test script recorder allows the JMeter to intercept and record actions while we browse the prototype with the normal browser [49]. In order to build a good sample, we used the JMeter to record all services provided by the framework. The JMeter allows options to choose how the test scenario will be run and determines the order in which the samples are processed. The random logic controller was chosen to select one service at random at each pass. To apply the test scenario, we host the default platform over two different hardware environments using AWS.

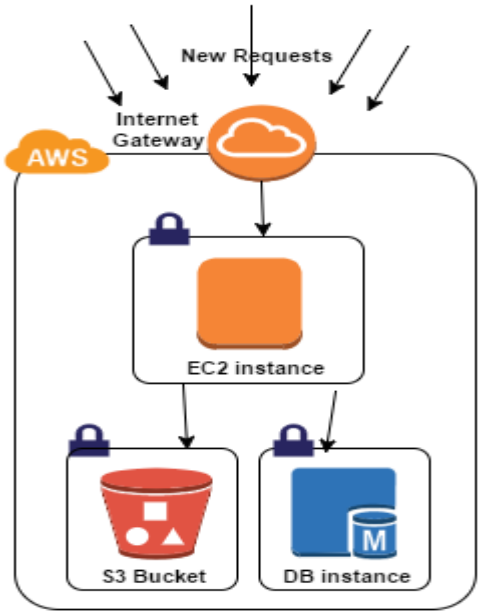


Figure 5.2: First test environment - Architecture

The first environment includes one t2.micro EC2 instance, which has one virtual Intel Xeon CPU with 2.5 GHz as a processor clock speed and 1GB memory as shown in Figure 5.2. **For the second environment**, we applied the Amazon ELB to distribute incoming requests among multiple EC2 instances, the Amazon Cloud Watch to monitor the CPU utilization of the EC2 instances, and the Amazon Auto Scaling service to ensure that the system has a sufficient number of EC2 instances available to handle the load for the requested services as shown in Figure 5.3. In order to set up this test, we created an Auto Scaling group, which has a minimum size of one instance, a desired capacity of two instances, and a maximum size of three instances. A Cloud Watch was applied to monitor CPU utilization. The Auto Scaling group automatically adds one new instance when the CPU utilization is more than 60% for two successive periods of five minutes, to handle any new request using a new EC2 instance. Conversely, the Auto Scaling group removes one

instance when the CPU utilization is less than 10% for six successive periods of five minutes.

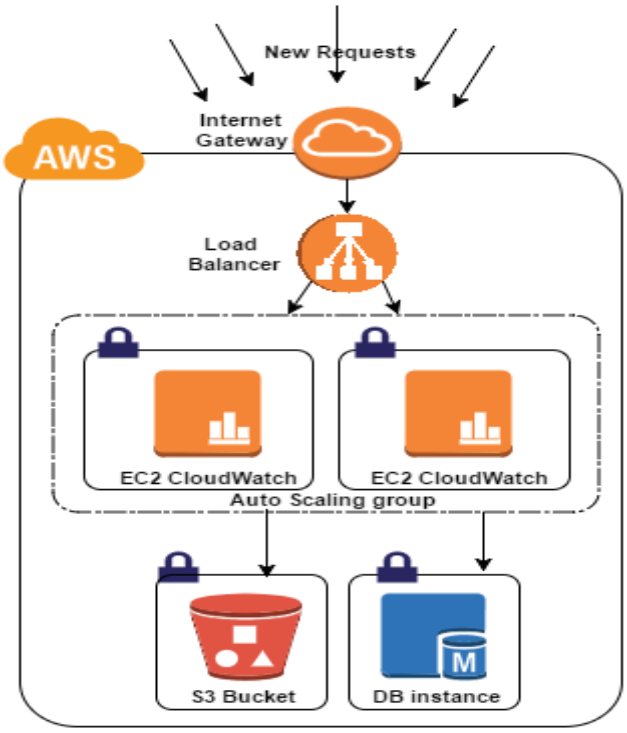


Figure 5.3: Second test environment - Architecture

The Auto Scaling group was applied over t2.micro EC2 instances that offer a single core of Intel Xeon processor with turbo 32.5GHz and 1GB RAM. Since we applied the ELB service, instances that are launched by the Auto Scaling are automatically registered with the ELB, and instances that are terminated by the Auto Scaling are automatically deregistered from the ELB. We want note that, the results could be affected by the network performance, and also some delay is expected as the JMeter runs on a local machine, and the test environments run in the cloud.

5.3.2 Results and Analysis

This section shows the experimental results. The JMeter outputs are presented in Appendix A.1. The results show the evaluation of the Amazon Auto Scaling, Amazon Cloud Watch, and Amazon ELB as applied in the proposed framework. For this evaluation we run the Bubble Sort algorithm to sort a list of 100,000 random numbers.

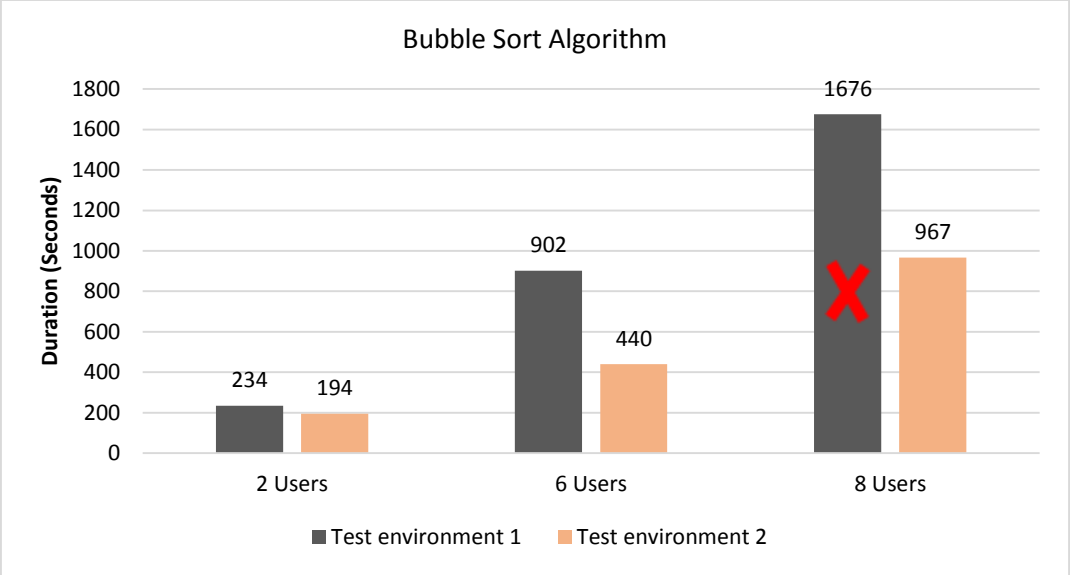


Figure 5.4: Average execution time for Bubble Sort algorithm running over two different test environments and different number of requests

Figure 5.4 presents the average response time and status for the requested service over the two test environments. In the first experiment, when the service was requested by 2 clients in the same time period, the two environments handled the requests properly and the algorithm execution time was convergent. Under the second experiment, where the service was requested by 6 clients in the same time period to sort a list of 100,000 random numbers, the two environments handled the requests properly, but the second environment required less average response time to execute them. This is because the second environment has the ability to scale horizontally and handle the requests using more EC2 instances. When the same job was requested by 8 clients in the same time period, only the

second environment handled them properly, as this environment has the ability to scale automatically to handle a greater number of requests.

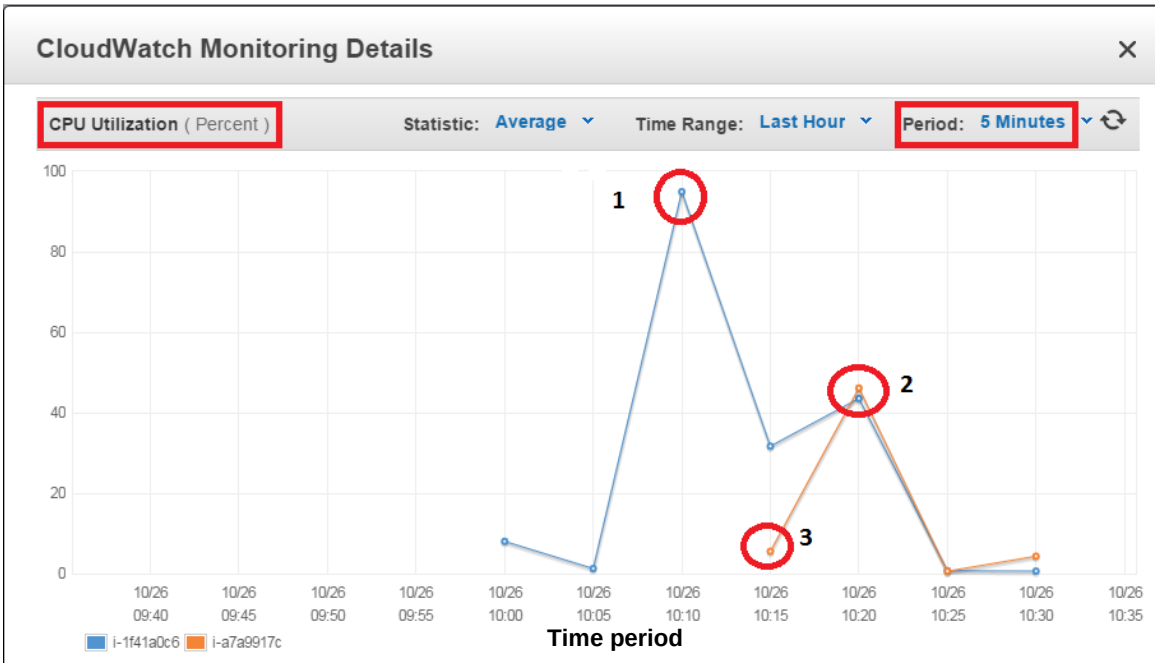


Figure 5.5: Amazon Cloud Watch monitoring details for the Amazon EC2 instances. The vertical data describe CPU utilizations. The horizontal data describe time periods

Figure 5.5 shows the Amazon Cloud Watch monitoring details for the Amazon EC2 instances when the Bubble Sort algorithm was requested by six users in the same time period over the second test environment. As seen, point number 1 in Figure 5.5 describes the CPU utilization for the Amazon EC2 instance when the system started the test scenario (around 90%). This means the CPU utilization exceeded the upper limit, which was defined in the Auto Scaling group (60%). As a result, point number 3 shows the new EC2 instance that was created automatically by the Auto Scaling group to handle the client requests over a new EC2 instance. Point number 2 describes the CPU utilizations for the running EC2 instances after the client requests were distributed amongst them (around 45%). The CPU utilization for the EC2 instances in the second environment was decreased by (around

50%). This decrease was a result of the environment utilizing the Amazon Auto Scaling, which is responsible for adding new Amazon EC2 instances based on the CPU utilization of the running instances.

To conclude, applying the Amazon Auto Scaling, Cloud Watch, and ELB solutions enable the framework prototype the ability to handle any new load by adding new EC2 instances. This will help to achieve high availability and high reliability, as well as increased runtime performance.

5.4 Scale-Up Service Evaluation

Offering a vertical scaling service to re-run the available algorithms, or to develop and deploy new sequential algorithms, is one of the main services in the proposed solution. Evaluating this service over multiple web servers having different models and features, including more CPU cores and high memory capacity, will give users and developers more understanding of this service. The second experiment consisted of a comparison of the performance results received from different web server models that were offered to the clients as options to host the scale-up service.

5.4.1 Test Environment

The second experiment was hosted in two different environments to compare the runtime performance between a limited-resources environment and an unlimited-resources environment. In addition, since our focus was the algorithm execution time, in order to achieve a fair estimation, we used the same database and storage area which are hosted in the cloud environment.

The first environment was hosted in a local host on a Dell Vostro 200 with processor Intel Core 2 Duo CPU at 2.33 GHz, 4 GB random access memory, running Windows 7 Professional, Microsoft .NET Framework Version 4.5.

The second environment was hosted on AWS via different EC2 service models. We selected three different EC2 instance models, namely, general purpose, compute optimized, and memory optimized that we are offered in the current porotype implementation.

The general purpose model provides a balance of compute, memory, and network resources, and is a good choice for many applications such as small and mid-size databases, data processing tasks that require additional memory, caching fleets, and cluster computing [37]. AWS offers multiple generations of the general purpose model, which are T2, M3 and M4. The M4 generation was selected for this evaluation as it is the latest generation of general purpose instances.

The compute optimized model has a higher ratio of CPU to memory than other families, and the lowest cost per CPU among all Amazon EC2 instance models [37]. Compute optimized model is a good choice for compute-intensive applications, including high traffic front-end fleets, on-demand batch processing, distributed analytics, web servers, batch processing, and high performance science and engineering applications [37]. AWS offers two generations namely, the C3 and the C4. The C4 generation is the latest generation of the compute optimized model. C3 and C4 generations were selected for this evaluation.

The memory optimized model has the lowest cost per GB of RAM among Amazon EC2 instance models [37]. Memory optimized instances are a good choice for memory-intensive applications; AWS offers one generation, which is an R3 instance model [37]

which was selected for this evaluation. A total of eight EC2 instances from different models were created, as shown in Table 5.5.

Table 5.5: EC2 instance models and features

Model	EC2 Type	vCPU	Mem (GB)	SSD Storage (GB)	Networking Performance	Physical Processor	Clock Speed (GHz)
General Purpose Model	m4.large	2	8	EBS Only	Moderate	Intel Xeon E5-2676 v3	2.4
	m4.4xlarge	16	64	EBS Only	High	Intel Xeon E5-2676 v3	2.4
Compute Optimized Model	c4.large	2	3.75	EBS Only	Moderate	Intel Xeon E5-2666 v3	2.9
	c4.4xlarge	16	30	EBS Only	High	Intel Xeon E5-2666 v3	2.9
	c3.large	2	3.75	2 x 16 SSD	Moderate	Intel Xeon E5-2680 v2	2.8
	c3.4xlarge	16	30	2 x 160 SSD	High	Intel Xeon E5-2680 v2	2.8
Memory Optimized Model	r3.large	2	15.25	1 x 32 SSD	Moderate	Intel Xeon E5-2670 v2	2.5
	r3.4xlarge	16	122	1 x 320 SSD	High	Intel Xeon E5-2670 v2	2.5

5.4.2 Results and Analysis

We ran the algorithms ten times for each instance and calculated the average of the execution time. A recording of the algorithms execution times for each of the 10 runs under the instances, which evaluates the scale-up service, is summarized in the Appendix A.2.

The following graphs demonstrate a comparison of the received performance results running multiple algorithms on different web server hardware configurations, using several datasets. The local machine has limited resources related to CPU and memory while the cloud web servers can be easily scaled vertically to obtain extra CPU cores and more memory capacity. You will notice that, we used different performance metrics (millisecond, second, and minutes) as suitable.

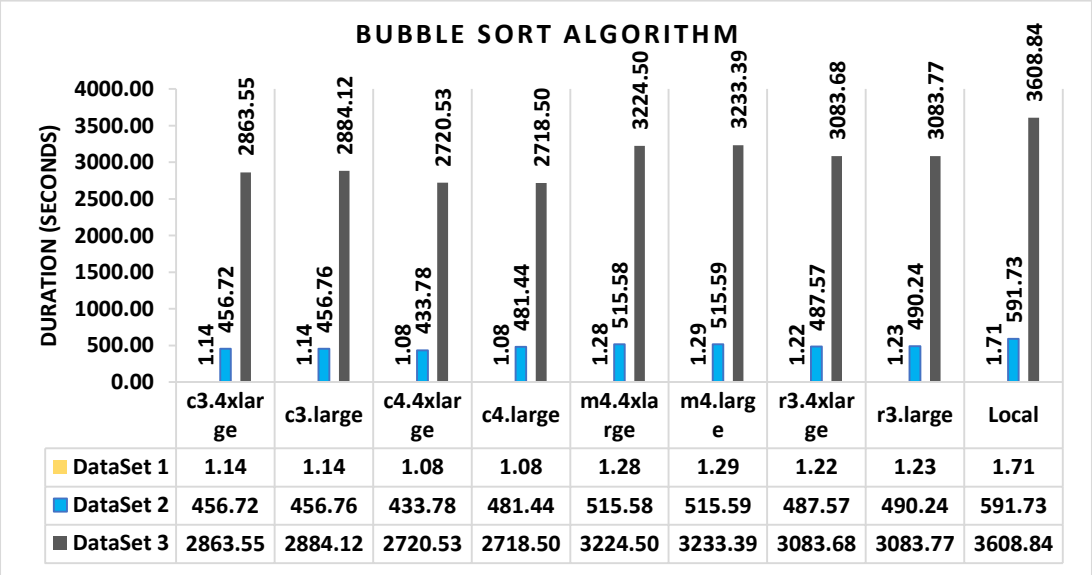


Figure 5.6**:** Average execution times for the Bubble Sort algorithm

Figure 5.6 shows the average execution time (in seconds) for the Bubble Sort algorithm using three different datasets. For the small size dataset (dataset 1) the comparison shows an insignificant difference in the duration of processing time between

the instances. This conclusion can be reached by comparing the highest execution time, which was recorded at 1.71 seconds, and the lowest execution time, which was recorded at 1.08 seconds, which was a difference of only 0.63 seconds. Furthermore, when applying statistical techniques, it was found that the variance between the instance execution times was very small (0.037).

As the size of the data in the dataset increased, the difference in the execution time between the instances significantly increased, as shown when using dataset 3. Under this dataset, the difference between the highest execution time and the lowest execution time was around 890 seconds. This also can be supported statistically, since the variance between the instances was very large (82411.2).

In addition, when comparing the average execution time between the cloud instance models with the average execution time calculated using the local instance, we found the following: the compute optimized model instances decreased the execution time by approximately 24%, the memory optimized model instances decreased the execution time by approximately 14%, and the general purpose model instances decreased the execution time by approximately 10%. This comparison shows that instances under the compute optimized model have the best performance results. This is further supported by observing the average execution time, which, under the compute optimized model, was less than the average deviation from the mean.

Finally, when comparing the different types of instances under the same model, we found that there was no significant difference in the duration of processing time between them.

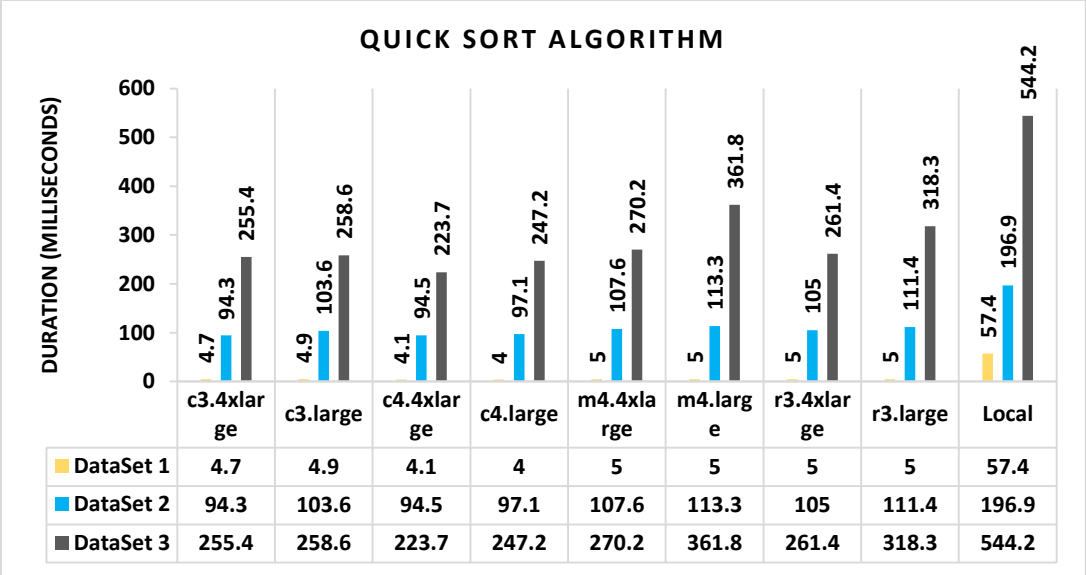


Figure 5.7: Average execution times for the Quick Sort algorithm

Figure 5.7 shows the average execution time (in milliseconds) for the Quick Sort algorithm using three different datasets. This comparison shows that, as the size of data in a dataset increases, the difference in the duration of processing time between the instances increases significantly, as shown when using dataset 3. Under this dataset, the difference between the highest execution time and the lowest execution time was approximately 320 milliseconds. In addition, when comparing the average execution time between the cloud instance models and the average execution time calculated using the local instance, we found the following: the compute optimized model instances decreased the execution time by approximately 58%, the memory optimized model instances decreased the execution time by approximately 51%, and the general purpose model instances decreased the execution time by approximately 50%. This comparison also shows that the compute optimized model instances have the best performance results. Finally, when comparing the different types of instances under the same model, we found that there is a significant difference in the duration of processing time between the memory optimized model

instances and between the general purpose model instances. The r3.4xlarge memory optimized instance required approximately 10% less execution time in comparison to the r3.large instance. Furthermore, the m4.4xlarge general purpose instance required approximately 17% less execution time in comparison to the m4.large instance. This is due to the instances having more memory capacity than other instances.

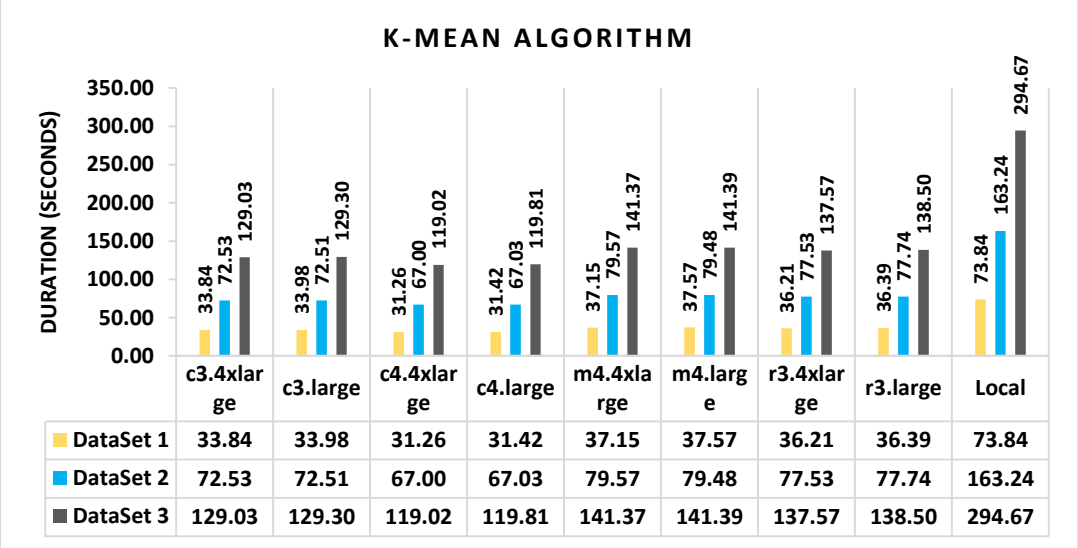
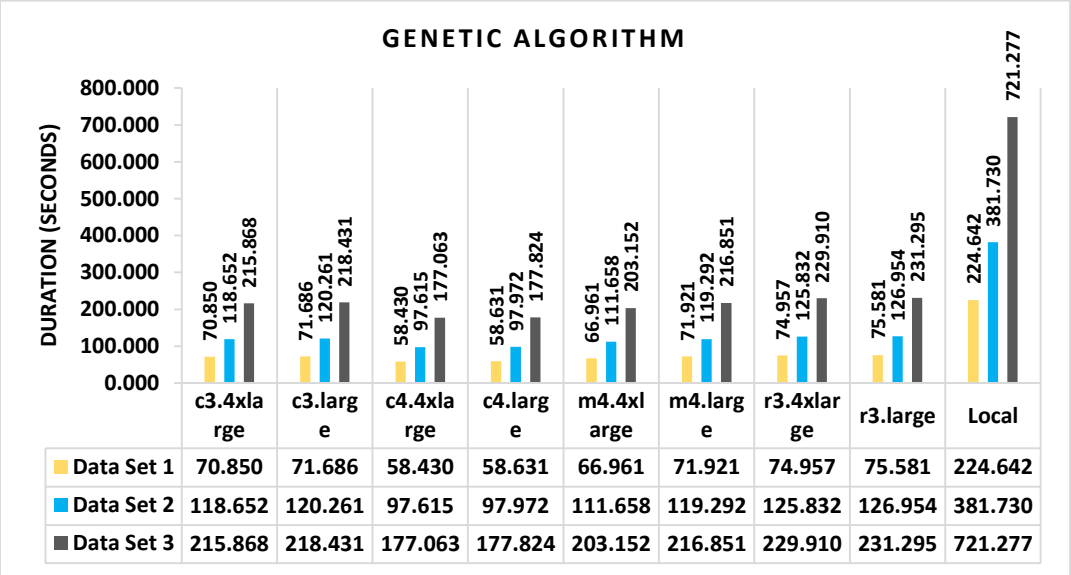


Figure 5.8: Average execution times for the K-Mean algorithm

Figure 5.8 shows the average execution time (in seconds) for the K-Means algorithm using three different datasets. This comparison shows that, as the size of data in a dataset increases, the difference in the duration of processing time between the instances increases significantly, as shown when using dataset 3. Under this dataset, the difference between the highest execution time and the lowest execution time was approximately 174 seconds. In addition, when comparing the average execution time between the cloud instance models and the average execution time calculated using the local instance, we found the following: the compute optimized model instances decreased the execution time by approximately 59%, the memory optimized model instances decreased the execution time by

approximately 53%, and the general purpose model instances decreased the execution time by approximately 52%. This comparison also shows that the compute optimized model instances have the best performance results. Finally, when comparing the different types of instances under the same model, we found that there was no significant difference in the duration of processing time between them.



	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarge	m4.large	r3.4xlarge	r3.large	Local
Data Set 1	70.850	71.686	58.430	58.631	66.961	71.921	74.957	75.581	224.642
Data Set 2	118.652	120.261	97.615	97.972	111.658	119.292	125.832	126.954	381.730
Data Set 3	215.868	218.431	177.063	177.824	203.152	216.851	229.910	231.295	721.277

Figure 5.9: Average execution times for the Genetic algorithm

Figure 5.9 shows the average execution time (in seconds) for the Genetic algorithm to solve the traveling salesman problem to identify the most efficient route that enables the salesman to visit every city once, using three different datasets. This comparison shows that, as the size of data in a dataset increases, the difference in the duration of processing time between the instances increases significantly, as shown when using dataset 3. Under this dataset, the difference between the highest execution time and the lowest execution time was approximately 544 seconds. This also can be supported statistically, since the variance between the instances was very large (29576.2). In addition, when comparing the

average execution time between the cloud instance models and the average execution time calculated using the local instance, we found the following: the compute optimized model instances decreased the execution time by approximately 75%, the memory optimized model instances decreased the execution time by approximately 68%, and the general purpose model instances decreased the execution time by approximately 71%. Finally, when comparing the different types of instances under the same model, we found that there was no significant difference in the duration of processing time between them.

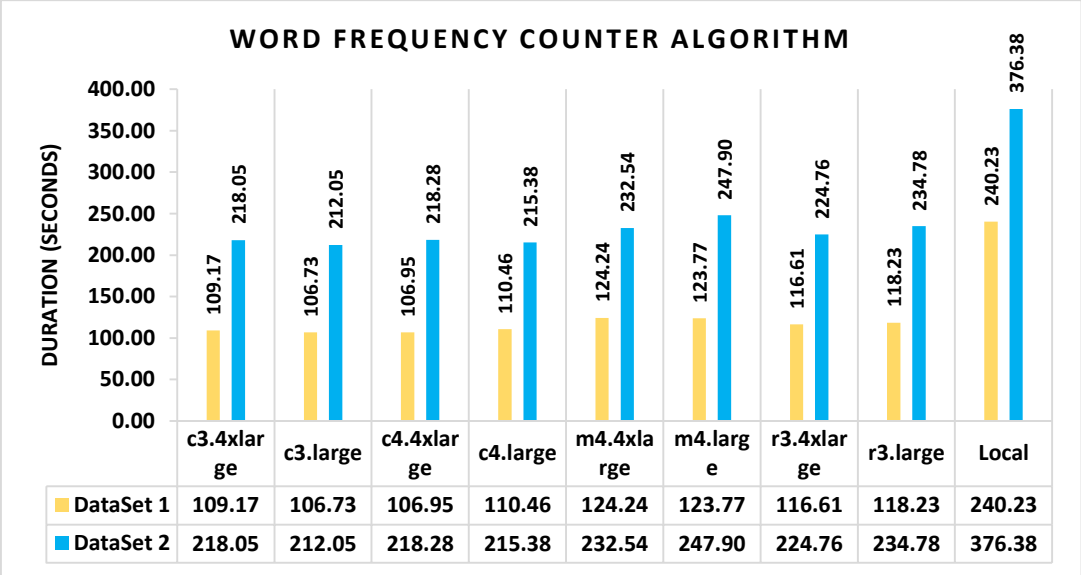


Figure 5.10: Datasets 1&2 - Average execution times for the Word Frequency Counter algorithm

Figure 5.10 shows the average execution time (in seconds) for the Word Frequency Counter algorithm using different datasets. This comparison shows that, as the size of data in a dataset increases, the difference in the duration of processing time between the instances increases significantly. Furthermore, when we attempted to run the third dataset, which has approximately 519 MB over the experimental machines as shown in Figure 5.11, we observed that only machines that have more CPU cores and high memory can process the data. In other words, the instances having high features (16 virtual CPU

cores and more than 30 GB of memory) can run the algorithm job and extract the results. These experimental results show the extent of the benefit of running algorithm services within high-end resource environment.

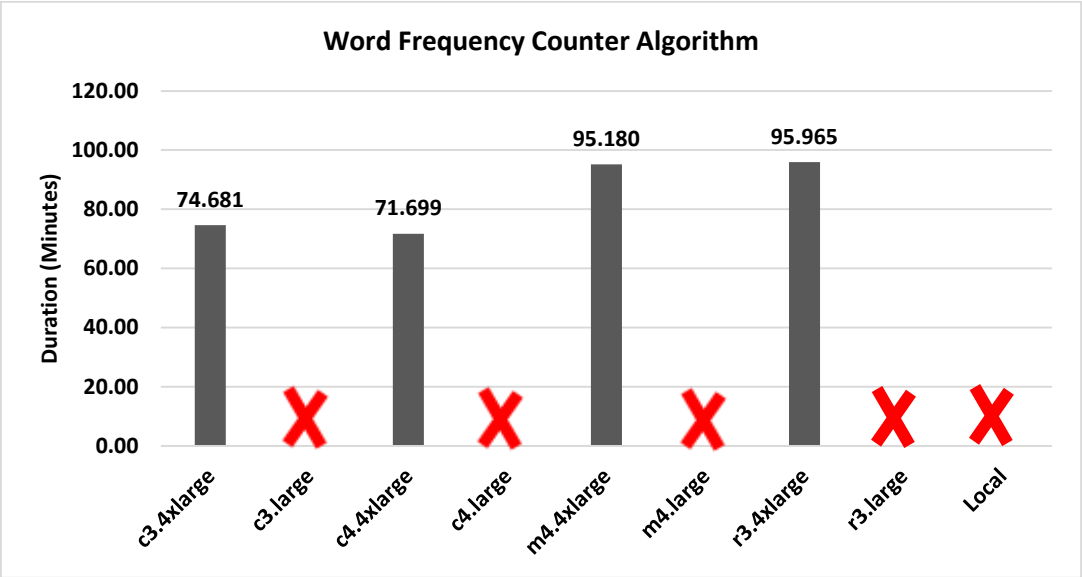


Figure 5.11: Dataset 3 - Average execution times for the Word Frequency Counter algorithm

To conclude, offering Algorithms as a Service within an easily scalable framework and a configurable high-performance environment helps clients to analyze data using variety of web server models and types and grants higher runtime performance. Furthermore, with the ability to launch new instances that have more CPU cores and high memory capacity, clients can analyze the data, minimize the execution time, improve the system performance, and avoid runtime errors related to resource limitations.

5.5 Scale-Out Service Evaluation

The third experiment was designed to evaluate the horizontal scaling (scale-out) service that provides a distributed environment to host and run distributed and parallel algorithms. In this experiment we ran the Word Frequency Counter algorithm over multiple web server types and numbers, which we offer in the prototype implementation to host this service, to evaluate how this service can improve the runtime performance.

5.5.1 Test Environment

The implementation was hosted on AWS via Amazon EMR service. This service provides a managed Hadoop framework for distributing and processing Big Data across horizontally scalable Amazon EC2 instances. We selected two different EC2 instance types under the M3 instance model which provides a balance of compute, memory, and network resources [37]. The instance types and models are shown in Table 5.6.

Table 5.6: EC2 instance models and features for the scale-out evaluation

Model	EC2 Type	vCPU	Mem (GB)	SSD Storage (GB)	Networking Performance	Physical Processor	Clock Speed (GHz)
General Purpose Model	m3.xlarge	4	15	2 x 40 SSD	High	Intel Xeon E5-2670 v2	2.5
	m3.2xlarge	8	30	2 x 80 SSD	High	Intel Xeon E5-2670 v2	2.5

5.5.2 Results and Analysis

All ten algorithm execution times for the Word Frequency Counter algorithm, which evaluates the scale-out platform, are summarized in Appendix A.3. The following figures demonstrate a comparison of the performance results running the Word Frequency Counter algorithms on the Amazon EMR service under multiple instance types and numbers. The results show the total execution time, which represents the total time required to launch the EC2 instances and to execute the algorithm in that instances. It also shows the job execution time, which represents the total time required to execute the algorithm on the EC2 instances. We decided to calculate the job execution time separately since the total execution time will be affected by the number of instances.

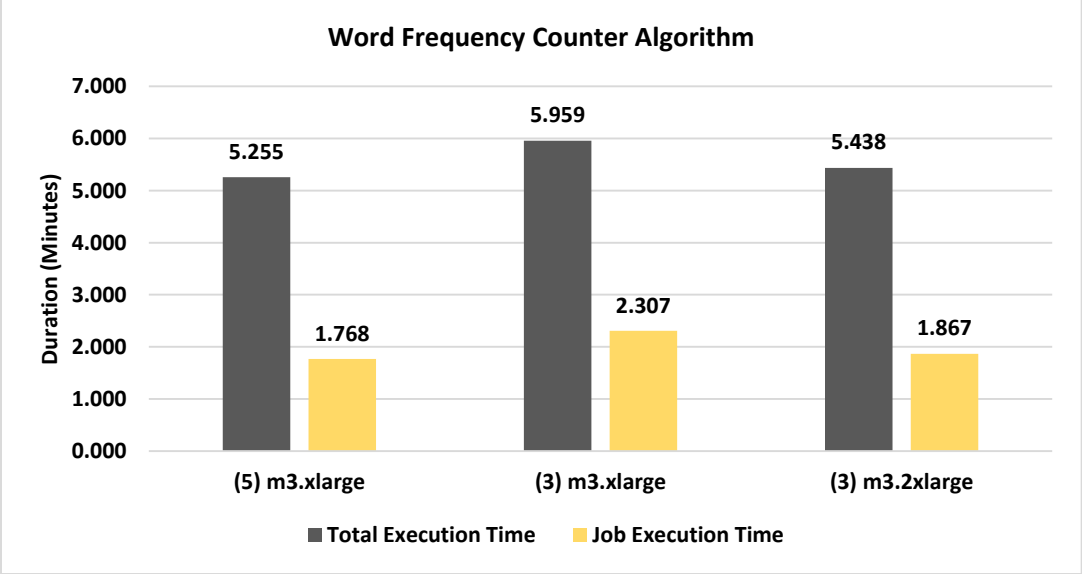


Figure 5.12: Average execution times for Word Frequency Counter algorithm running over three different Hadoop Map-Reduce environments

Figure 5.12 shows the average execution time (in minutes) for the run and the job execution time, using three different cases, which involved running Hadoop Map-Reduce Word Frequency Counter algorithm over five m3.xlarge instances, three m3.xlarge

instances, and three m3.2xlarge instances using more than 500 MB sized book. The experiment shows that running Hadoop Map-Reduce Word Frequency Counter algorithm over **five m3.xlarge** instances decreased the algorithm execution time by approximately 23% in comparison to the algorithm execution time for the **three m3.xlarge** instances (same instance types but with more instance numbers). It also shows that running Hadoop Map-Reduce Word Frequency Counter algorithm over **three m3.2xlarge** instances decreased the algorithm execution time by approximately 19% in comparison to the algorithm execution time for the **three m3.xlarge** instances (same instance numbers but with high resource usage).

In summary, distributing the task among more number of machines is another way to minimize the execution time and improve the runtime performance. Another benefit of this service is that it gives the framework the ability to be scaled horizontally by increasing the number of instances, and, at the same time, to be scaled vertically by selecting high resources usage instances to distribute an algorithm among them.

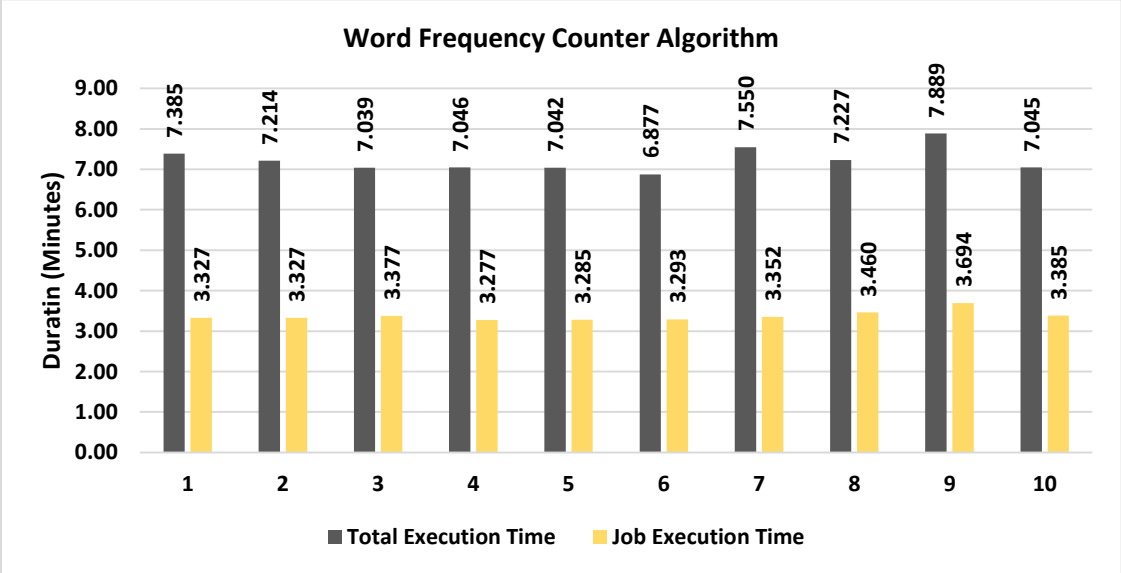


Figure 5.13: Execution times for Word Frequency Counter algorithm running over three m3.xlarge instances

Figure 5.13 displays the execution times of running the Word Frequency Counter Map-Reduce algorithm using 1 GB sized book over **three m3.xlarge** EC2 instances. As we can see, we doubled the book size to investigate by how much the job execution time will increase in relation to the new size. The job execution time in the Figure 5.13 shows an acceptable execution time related to the large amount of data. Furthermore, the job execution time for the Words Frequency Counter algorithm, using big amounts of data through horizontal scaling, is less than the execution time as compared to the vertical scaling.

To summarize, providing scale-out service in the proposed framework grants users and developers the capability to host and run Map-Reduce algorithms within a horizontally, as well as a vertically scalable environment. This service helps to analyze big data within acceptable execution time using variety of web server configurations.

5.6 Comparing Algorithms Evaluation

The last experiment was designed to compare the runtime performance for different algorithms under the same job category using the default template that is available to algorithm developers to deploy new sequential algorithms. Some clients may decide to do this in order to comparing the performance or the result for different algorithms, and determining the most appropriate one. We selected different sorting algorithms under a sorting job category to demonstrate how the proposed framework can provide a comparable environment. A total of six sorting algorithms were used which included the Bubble Sort, the Insertion Sort, the Merge Sort, the Quick Sort, the Selection Sort, and the Shell Sort.

5.6.1 Test Environment

The algorithm services were hosted on Amazon EC2 service using the scale-up service which is provided via the prototype. We launched one EC2 instance under the general purpose model. The instance details are shown in Table 5.7.

Table 5.7: Comparing algorithms - EC2 instance model and feature

Model	vCPU	Mem (GB)	SSD Storage (GB)	Networking Performance	Physical Processor	Clock Speed (GHz)
m4.large	2	8	EBS Only	Moderate	Intel Xeon E5-2676 v3	2.4

5.6.2 Results and Analysis

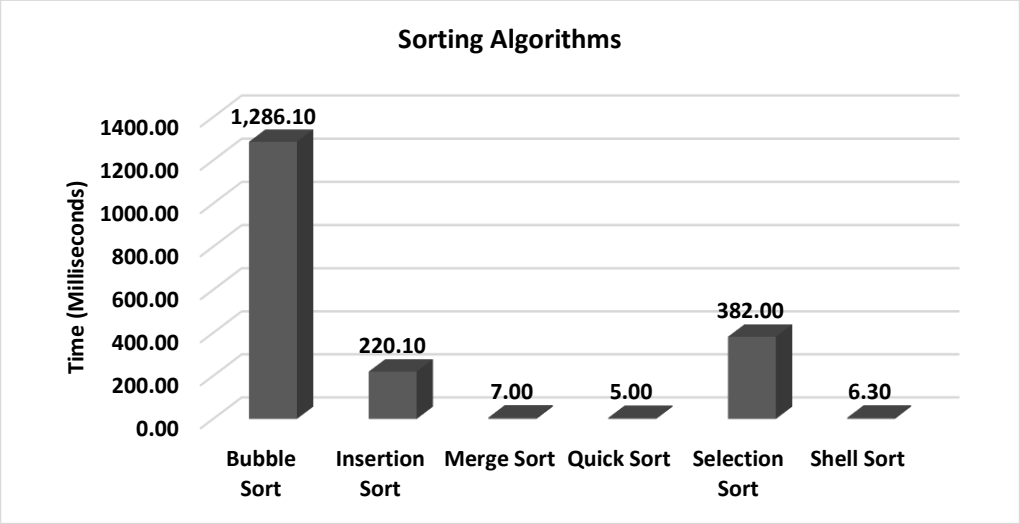


Figure 5.14: Comparing average execution times for all sorting algorithms

The summary and the average execution time (in milliseconds) for all sorting algorithms are shown in Figure 5.14. The experiment has shown that the Bubble Sort algorithm presents the highest execution time and the Quick Sort algorithm presents the

lowest execution time. The template allows algorithm developers to develop and deploy, then run and compare their new algorithms within the same algorithm job category. Furthermore, they can launch different high-end web servers and examine the algorithm performance among them.

5.7 Lessons Learned

There are some lessons learned from the previous experiments and the evaluation results which are listed below:

- Applying the Auto Scaling as well as Load Balancing solutions grant the proposed framework the ability to achieve availability and reliability by adding new web servers to handle the new load;
- Executing algorithms within a vertically scalable environment using a high-end web server, which has more CPU cores and high memory capacity, minimize the execution time and improve the system performance;
- Analyzing Big Data within a horizontally scalable environment using Map-Reduce algorithms provides a higher performance than vertical scalable using sequential algorithms; and
- Offering algorithms as a service over an easily scalable framework and a high-end resource environment enables clients to achieve a higher algorithm performance and avoids runtime errors related to resource limitations.

5.8 Summary

This chapter presented the experimental prototype evaluation related to prototype scalability models, availability, and runtime performance. We evaluated the runtime performance from different aspects to investigate how the prototype provided value added services. Providing algorithm services over cloud computing, which has many advantages and solutions as well as virtually unlimited resources, grants the service more scalability, availability, reliability, and high performance.

Chapter 6

6. Conclusion and Future Work

6.1 Conclusion

Cloud computing is an environment which provides many services in order to achieve client satisfaction. It offers compatible and on-demand hardware and software services for different computing resources. Building a framework that utilizes these high-end services and providing them in a friendly interface enables clients to execute algorithms easily, and improve the efficiency of algorithms.

This thesis presented the architecture, prototype, and evaluation of a framework for a system that provides Algorithms as a Service. This framework solves various issues, including finding an appropriate algorithm to process data, recognizing algorithm code, installing and configuring a software to run the algorithm, and increasing runtime performance. The framework offers different services in the cloud to help clients test data, and extract results within an appropriate execution time frame.

The contributions of this thesis are:

- Designing a framework for provisioning sequential and parallel algorithms as a service;
- A prototype implementation using Amazon Web Services;
- Providing multiple scalability models to help public users and data researchers analyze data and compare results over different web server configurations; and

- Evaluation of the prototype implementation, with an emphasis on the runtime performance based on scalability models, as well as availability performance.

Collecting different varieties of algorithms in one environment will motivate clients to utilize the framework services. Furthermore, researchers from several research fields have the ability to study data and extract useful information. The framework allows algorithm developers the ability to deploy new algorithm services. This service gives developers the capability to add new algorithms to the default solution image, execute them, and examine their results. Clients can compare algorithm performance by running the algorithms over various web servers having differing resources configuration using the scalability services. This services allow clients to scale the framework via one of the two available scalability models. Scale-up allows clients to re-execute the algorithm job over any chosen high-end web servers. Scale-out allows clients to distribute an algorithm job among multiple web servers. The prototype evaluation focused on the algorithm performance and how the provided services can improve the runtime performance as well as the quality of the framework by achieving availability and reliability.

6.2 Future Work

While this thesis was successful in providing evidence that cloud computing is a valid platform for implementing a high performance framework for provisioning algorithms as a service, there are still many interesting open research areas, as listed below.

- **Comparing scalability models performance**

In considering future work, one area that could be researched further is restructuring sequential algorithms to work as parallel algorithms to evaluate the runtime performance

for these algorithms over different scalability models. This researches will help to study the relationship between a scale model and an algorithm job.

- **Applying internal load balancer**

This idea refers to the building of an internal load balancer algorithm to automatically select preferred scalability models based on historical execution times for the previous running algorithm jobs. Creating a Database table to save the execution time related to an algorithm job, data input size, and scalability model, and using this data as experimental data for the load balancing algorithm, is the basis for this work.

- **Building a multi-language environment**

The current solution allows algorithm developers to develop and deploy parallel algorithms in several programming languages as mentioned in Section 4.5.4.2. Building a multi-language environment to develop and deploy new sequential algorithms using different programming languages is considered as one of the main future work to give algorithm developers the ability to implement and host new sequential algorithms using different programming languages

- **Comparing different cloud service providers**

As discussed in Section 4.2, we selected AWS to implement the framework prototype and we evaluated the framework services based on the AWS provider and their services. Implementing the framework prototype using other cloud providers and evaluating the framework services based on the cloud providers services, is an important point for future work.

- **Comparing between the public and private cloud environments**

Implementing the framework prototype on a private cloud environment should also be considered for future work to compare the cost and the quality of private cloud services with public cloud providers.

Finally, adding multiple algorithms under the same job category will support and improve an algorithm comparative environment and will also allow clients to analyze data and compare results between different algorithms. Such a comparison would give more evidence for the results and support higher data analysis.

References

[1] R. Buyya, J. Broberg, and A. M. Goscinski, “Cloud computing: Principles and paradigms,” *John Wiley Sons*, vol. 87, pp. 391–411, 2010.

[2] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, “A view of cloud computing,” *Commun. ACM 53.4*, vol. 53, no. 4, pp. 50–58, 2010.

[3] M. S. Jassas, A. A. Qasem, and Q. H. Mahmoud, “A Smart System Connecting e-Health Sensors and the Cloud,” *Electr. Comput. Eng. (CCECE), 2015 IEEE 28th Can. Conf. on. IEEE*, pp. 712–716, 2015.

[4] NIST, “The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology,” *Natl. Inst. Stand. Technol.*, vol. 145, p. 7, 2011.

[5] G. Eugene, “Cloud Computing Models,” 2013.

[6] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, “Cloud computing,” *IBM Corp.*, vol. 1, 2007.

[7] Amazon Web Services, “Getting Started with AWS,” 2014. [Online]. Available: http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/gsg-aws-intro.html.

[8] M. Godse and S. Mulik, “An approach for selecting Software-as-a-Service (SaaS) product,” *IEEE Int. Conf. Cloud Comput.*, pp. 155–158, 2009.

[9] D. Pop, “Machine Learning and Cloud Computing: Survey of Distributed and SaaS Solutions,” *Inst. e-Austria Timisoara, Tech. Rep 1*, 2012.

[10] “BigML.com is Machine Learning for everyone.” [Online]. Available: https://bigml.com/. [Accessed: 05-Dec-2015].

[11] “BitYota - Data Warehouse as a Service.” [Online]. Available: http://www.bityota.com/. [Accessed: 05-Dec-2015].

[12] “Prediction API — Google Cloud Platform.” [Online]. Available: https://cloud.google.com/prediction/docs. [Accessed: 05-Dec-2015].

[13] K. U. . Jaseena and J. M. David., “ISSUES, CHALLENGES, AND SOLUTIONS: BIG DATA MINING,” *Comput. Sci. Inf. Technol.*, pp. 131–140, 2014.

[14] A. Sharma and P. Gulia, “Analysis of big data 1 1 2,” *Int. J. Comput. Sci. Mob. Comput.*, vol. 3, no. 9, pp. 56–68, 2014.

[15] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li, “Big Data Processing in Cloud Computing Environments,” *12th Int. Symp. Pervasive Syst. Algorithms Networks*, pp. 17–23, 2012.

[16] L. Ismail, M. M. Masud, and L. Khan, “FSBD: A Framework for Scheduling of Big Data Mining in Cloud Computing,” *IEEE Int. Congr. Big Data*, pp. 514–521, 2014.

[17] W.-T. Tsai, Y. Huang, X. Bai, and J. Gao, “Scalable Architectures for SaaS,” *IEEE 15th Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput. Work.*, pp. 112–117, 2012.

[18] S. Sharma, S. Singh, and M. Sharma, “Performance analysis of load balancing algorithms,” *Int. J. Comput. Electr. Autom. Control Inf. Eng.*, vol. 2, no. 2, pp. 269–272, 2008.

[19] A. Agarwal and G. Manisha, “Performance Analysis of Cloud Based Load Balancing Techniques,” *Parallel, Distrib. Grid Comput. (PDGC), Int. Conf. on. IEEE*, pp. 49–52, 2014.

[20] A. Hanieh, L. Yan, and H.-L. Abdelwahab, “Analyzing Auto-scaling Issues in Cloud Environments,” *Proc. 24th Annu. Int. Conf. Comput. Sci. Softw. Eng. IBM Corp.*, pp. 75–89, 2014.

[21] N. N. Kumbhar, V. V. Chaudhari, and M. A. Badhe, “The Comprehensive Approach for Data Security in Cloud Computing: A Survey,” *Int. J. Comput. Appl.*, vol. 39, no. 18, pp. 23–29, 2012.

[22] D. S. Raghuwanshi, “MS2 : Practical Data Privacy and Security Framework for Data at Rest in Cloud,” *Comput. Appl. Inf. Syst. (WCCAIS), IEEE*, 2014.

[23] Y. N. Moschovakis, “What is an algorithm?,” *Math. Unltd. — 2001 beyond*, pp. 919 – 936, 2001.

[24] T. L. Naps, J. R. Eagan, and L. L. Norton, “JHAVI -- An Environment to Actively Engage Students in Web- based Algorithm Visualizations,” *ACM SIGCSE Bull.*, vol. 32, no. 1, pp. 109–113, 2000.

[25] I. Reif and T. Orehovacki, “ViSA: Visualization of sorting algorithms,” *MIPRO, 2012 Proc. 35th Int. Conv. IEEE*, pp. 1146–1151, 2012.

[26] C. Ordonez, J. García-García, C. Garcia-Alvarado, W. Cabrera, V. Baladandayuthapani, and M. S. Quraishi, “Data mining algorithms as a service in the cloud exploiting relational database systems,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, p. 1001.

[27] N. Ankita and M. M. R, “Realisation of Resourceful Data Mining Services Using Cloud Computing,” *Int. J. Comput. Sci. Eng.*, vol. 5, no. 3, pp. 625–632, 2013.

[28] T. Chen, J. Chen, and B. Zhou, “A system for parallel data mining service on cloud,” *Second Int. Conf. Cloud Green Comput.*, pp. 329–330, 2012.

[29] A. Karadimce, “Model of Cloud-Based Services for Data Mining Analysis,” *Comput. Inf. Sci.*, vol. 8, no. 4, p. 40, 2015.

[30] “Algorithms-as-a-Service - Cloud’N'Sci.fi.” [Online]. Available: https://cloudnsci.fi/. [Accessed: 29-Oct-2015].

[31] “Algorithmia - Open Marketplace for Algorithms.” [Online]. Available: https://algorithmia.com/. [Accessed: 29-Oct-2015].

[32] M. S. Bhiogade, “Secure Socket Layer,” *Comput. Sci. Inf. Technol. Educ. Conf.*, no. June, pp. 85–90, 2002.

[33] Amazon Web Services, “Amazon Elastic Load Balancing,” 2012. [Online]. Available: http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elb-dg.pdf.

[34] Amazon Web Services, “Amazon Auto Scaling.” [Online]. Available: http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-dg.pdf.

[35] S. Ramamoorthy and S. Rajalakshmi, “Optimized data analysis in cloud using BigData analytics techniques,” *4th Int. Conf. Comput. Commun. Netw. Technol. ICCCNT 2013*, pp. 4–8, 2013.

[36] Amazon Web Services, “Amazon Elastic Compute Cloud (Amazon EC2),” *Amazon Web Services LLC*, 2011. [Online]. Available: http://aws.amazon.com/ec2/.

[37] “EC2 Instance Types – Amazon Web Services (AWS).” [Online]. Available: https://aws.amazon.com/ec2/instance-types/. [Accessed: 05-Dec-2015].

[38] Amazon Web Services, “Amazon Simple Storage Service,” 2012. [Online].

Available: http://docs.aws.amazon.com/AmazonS3/latest/gsg/s3-gsg.pdf.

[39] Amazon Web Services, “Amazon Relational Database Service,” 2013. [Online]. Available: http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/rds-ug.pdf.

[40] Amazon Web Services, “Amazon Elastic MapReduce,” 2009. [Online]. Available: http://awsmedia.s3.amazonaws.com/pdf/introduction-to-amazon-elastic-mapreduce.pdf.

[41] “Comparison Sorting Algorithms in C# Explained - CodeProject.” [Online]. Available: http://www.codeproject.com/Articles/80546/Comparison-Sorting-Algorithms-in-C-Explained. [Accessed: 12-Oct-2015].

[42] “List Words By Frequency.” [Online]. Available: http://nlpdotnet.com/SampleCode/ListWordsByFrequency.aspx. [Accessed: 12-Oct-2015].

[43] B. Everitt, “Unresolved Problems in Cluster Analysis,” *Int. Biometric Soc.*, vol. 35, pp. 169–181, 1979.

[44] “Implementing Genetic Algorithms in C# - CodeProject.” [Online]. Available: http://www.codeproject.com/Articles/873559/Implementing-Genetic-Algorithms-in-Csharp. [Accessed: 12-Oct-2015].

[45] H. Riesel, “What is a Prime number?,” in *Prime numbers and computer methods for factorization*, Basel, Switzerland: Springer Science & Business Media, 2012.

[46] “Free ebooks by Project Gutenberg - Gutenberg.” [Online]. Available: https://www.gutenberg.org/. [Accessed: 06-Dec-2015].

[47] “Latitude-Longitude of US Cities (www.realestate3d.com).” [Online]. Available: http://www.realestate3d.com/gps/latlong.htm. [Accessed: 06-Dec-2015].

[48] E. Al-Masri and Q. H. Mahmoud, “QoS-based discovery and ranking of Web services,” *Proc. - Int. Conf. Comput. Commun. Networks, ICCCN*, pp. 529–534, 2007.

[49] “Apache JMeter - Apache JMeter™.” [Online]. Available: http://jmeter.apache.org/. [Accessed: 12-Oct-2015].

Appendix A: Experimental Data

This appendix provides a more details of the results of the experiments mentioned in Chapter 5.

A.1 Default Platform Evaluation

This section presents JMeter outputs for the default platform evaluation.

Table A.1: JMeter report of response times and status for two requests running over the first test environment. Time is measured in milliseconds

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
21 /Login.aspx	2	468	222	715	246.50	0.00%	2.4/min	0.31	7987.0
22 /Login.aspx	2	306	77	535	229.00	0.00%	2.4/min	0.02	454.0
23 /MainPage.as...	2	242	77	407	165.00	0.00%	2.3/min	0.21	5393.0
24 /MainPage.as...	2	342	78	606	264.00	0.00%	2.3/min	0.01	378.0
25 /Default.aspx	2	248	83	414	165.50	0.00%	2.3/min	0.60	16087.0
26 /SortingAlgorit...	2	345	83	608	262.50	0.00%	2.3/min	0.85	22828.0
27 /SortingAlgorit...	2	233576	207243	259909	26333.00	0.00%	27.7/hour	0.17	23162.0
28 /SortingAlgorit...	2	77	75	80	2.50	0.00%	23.0/sec	8.40	374.0
29 /Login.aspx	2	76	73	79	3.00	0.00%	21.1/sec	164.21	7987.0
TOTAL	18	26186	73	259909	73846.98	0.00%	4.1/min	0.63	9405.6

Table A.2: JMeter report of response times and status for two requests running over the second test environment. Time is measured in milliseconds

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
21 /Login.aspx	2	524	211	837	313.00	0.00%	2.4/min	0.31	8011.0
22 /Login.aspx	2	352	102	602	250.00	0.00%	2.3/min	0.02	478.0
23 /MainPage.aspx	2	353	101	605	252.00	0.00%	2.3/min	0.20	5417.0
24 /MainPage.aspx	2	251	99	403	152.00	0.00%	2.3/min	0.02	402.0
25 /Default.aspx	2	451	197	705	254.00	0.00%	2.3/min	0.60	16111.0
26 /SortingAlgorithms.aspx	2	514	217	811	297.00	0.00%	2.3/min	0.84	22852.0
27 /SortingAlgorithms.aspx	2	193947	188121	199774	5826.50	0.00%	28.5/hour	0.18	23186.0
28 /SortingAlgorithms.aspx	2	367	99	635	268.00	0.00%	1.9/min	0.01	398.0
29 /Login.aspx	2	258	103	413	155.00	0.00%	1.9/min	0.24	8011.0
TOTAL	18	21890	99	199774	60862.74	0.00%	4.3/min	0.65	9429.6

Table A.3: JMeter report of response times and status for six requests running over the first test environment. Time is measured in milliseconds

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
21 /Login.as...	6	1046	288	1711	473.35	0.00%	4.2/min	0.55	7987.0
22 /Login.as...	6	816	96	1534	454.12	0.00%	4.2/min	0.03	454.0
23 /MainPag...	6	932	111	1629	544.11	0.00%	4.1/min	0.36	5393.0
24 /MainPag...	6	905	103	1497	516.42	0.00%	4.1/min	0.03	378.0
25 /Default.a...	6	835	110	1427	484.24	0.00%	4.0/min	1.05	16087.0
26 /SortingAl...	6	965	103	1424	474.76	0.00%	3.9/min	1.47	22828.0
27 /SortingAl...	6	902291	492216	1121967	213329.08	0.00%	17.8/hour	0.11	23162.0
28 /SortingAl...	6	1209	94	2124	751.22	0.00%	30.0/hour	0.00	374.0
29 /Login.as...	6	1050	96	2033	644.37	0.00%	30.0/hour	0.07	7987.0
TOTAL	54	101116	94	1121967	292047.74	0.00%	2.7/min	0.41	9405.6

Table A.4: JMeter report of response times and status for six requests running over the second test environment. Time is measured in milliseconds

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
21 /Login.aspx	6	889	190	2227	722.54	0.00%	4.2/min	0.55	8011.0
22 /Login.aspx	6	775	98	1816	608.25	0.00%	4.1/min	0.03	478.0
23 /MainPag...	6	982	93	2236	736.74	0.00%	4.1/min	0.30	4580.5
24 /MainPag...	6	715	98	1419	497.04	0.00%	4.0/min	0.03	402.0
25 /Default.a...	6	686	92	1311	452.52	0.00%	4.0/min	0.87	13492.2
26 /SortingAl...	6	771	190	1421	449.83	0.00%	3.9/min	1.45	22852.0
27 /SortingAl...	6	439852	1079	562044	201639.49	0.00%	33.7/hour	0.18	19388.0
28 /SortingAl...	6	854	95	2199	721.75	0.00%	33.7/hour	0.00	398.0
29 /Login.aspx	6	917	97	2439	861.84	0.00%	33.7/hour	0.07	8011.0
TOTAL	54	49604	92	562044	153475.16	0.00%	5.0/min	0.71	8623.6

Table A.5: JMeter report of response times and status for eight requests running over the first test environment. Time is measured in milliseconds

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
21 /Login.as...	8	1151	225	2486	618.35	0.00%	5.3/min	0.69	7987.0
22 /Login.as...	8	991	91	2306	652.56	0.00%	5.2/min	0.04	454.0
23 /MainPag...	8	950	84	1797	578.06	0.00%	5.1/min	0.45	5393.0
24 /MainPag...	8	926	82	1794	553.16	0.00%	5.0/min	0.03	378.0
25 /Default.a...	8	962	87	1804	552.11	0.00%	4.9/min	1.29	16087.0
26 /SortingAl...	8	937	83	1583	497.44	0.00%	4.9/min	1.81	22828.0
27 /SortingAl...	8	1676390	1626794	1725408	31930.74	100.00%	16.7/hour	0.01	2164.0
TOTAL	56	240330	82	1725408	586393.81	14.29%	1.9/min	0.25	7898.7

Table A.6: JMeter report of response times and status for eight requests running over the second test environment. Time is measured in milliseconds

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
21 /Login.aspx	8	3894	165	13908	4821.89	0.00%	5.4/min	0.70	8011.0
22 /Login.aspx	8	1003	242	1823	549.00	0.00%	5.9/min	0.05	478.0
23 /MainPag...	8	799	83	1626	579.51	0.00%	5.9/min	0.52	5417.0
24 /MainPag...	8	869	77	1832	720.27	0.00%	5.8/min	0.04	402.0
25 /Default.a...	8	1076	159	1698	504.45	0.00%	5.7/min	1.50	16111.0
26 /SortingAl...	8	2835	152	14077	4331.99	0.00%	5.6/min	2.10	22852.0
27 /SortingAl...	8	967311	924068	995049	26015.11	0.00%	28.9/hour	0.18	23186.0
28 /SortingAl...	8	7015	76	27833	12016.46	0.00%	17.2/min	0.11	398.0
29 /Login.aspx	8	132	76	515	144.44	0.00%	17.0/min	2.21	8011.0
TOTAL	72	109437	76	995049	303469.22	0.00%	4.1/min	0.63	9429.6

A.2 Scale-Up Service Evaluation

This section presents the total execution times for multiple algorithms to evaluate the scale-up service.

Table A.7: Data Set 1 - Execution times for Bubble Sort algorithm. Time is measured in seconds

	Sorting Algorithms - Bubble Sort - Data Set 1 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarge	m4.large	r3.4xlarge	r3.large	Local
1	1.139	1.14	1.082	1.076	1.283	1.287	1.223	1.235	1.745
2	1.143	1.143	1.082	1.077	1.284	1.285	1.227	1.223	1.747
3	1.148	1.137	1.084	1.077	1.284	1.286	1.223	1.225	1.629
4	1.14	1.147	1.084	1.077	1.284	1.287	1.223	1.226	1.654
5	1.14	1.144	1.082	1.077	1.283	1.287	1.224	1.222	1.769
6	1.139	1.138	1.082	1.077	1.284	1.285	1.224	1.228	1.921
7	1.141	1.138	1.082	1.077	1.284	1.287	1.223	1.224	1.892
8	1.14	1.143	1.083	1.077	1.283	1.285	1.224	1.226	1.65
9	1.15	1.14	1.082	1.078	1.284	1.286	1.227	1.226	1.539
10	1.151	1.141	1.082	1.075	1.284	1.286	1.227	1.224	1.584
Average	**1.1431**	**1.1411**	**1.0825**	**1.0768**	**1.2837**	**1.2861**	**1.2245**	**1.2259**	**1.713**
Variance	**0.03734791**								
Max	**1.151**	**1.147**	**1.084**	**1.078**	**1.284**	**1.287**	**1.227**	**1.235**	**1.921**
Min	**1.139**	**1.137**	**1.082**	**1.075**	**1.283**	**1.285**	**1.223**	**1.222**	**1.539**

Table A.8: Data Set 2 - Execution times for Bubble Sort algorithm. Time is measured in seconds

	Sorting Algorithms - Bubble Sort - Data Set 2 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarg	m4.large	r3.4xlarge	r3.large	Local
1	457.222	456.960	433.979	706.922	515.661	515.436	486.811	488.575	555.939
2	457.184	456.932	433.705	633.840	515.625	515.447	486.761	489.827	605.385
3	456.083	456.479	433.778	434.162	515.745	515.462	486.771	489.373	596.225
4	457.307	457.064	433.659	434.231	515.707	515.593	486.982	490.172	596.719
5	457.577	455.854	433.812	434.376	515.667	515.594	486.790	492.687	597.688
6	457.319	457.051	433.674	433.745	515.785	515.577	488.544	489.049	590.241
7	457.198	456.703	433.667	434.254	515.565	515.549	488.433	491.980	596.926
8	456.082	455.860	433.643	434.230	515.606	515.469	488.564	490.687	578.709
9	456.431	457.469	433.679	434.320	515.742	515.456	487.088	490.049	600.339
10	454.807	457.258	434.201	434.327	514.680	516.263	488.891	489.980	599.119
Average	**456.721**	**456.763**	**433.780**	**481.441**	**515.578**	**515.585**	**487.564**	**490.238**	**591.729**
Variance	**2133.496**								
Max	**457.577**	**457.469**	**434.201**	**706.922**	**515.785**	**516.263**	**488.891**	**492.687**	**605.385**
Min	**454.807**	**455.854**	**433.643**	**433.745**	**514.680**	**515.436**	**486.761**	**488.575**	**555.939**

Table A.9: Data Set 3 - Execution times for Bubble Sort algorithm. Time is measured in seconds

	Sorting Algorithms - Bubble Sort - Data Set 3 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarg	m4.large	r3.4xlarge	r3.large	Local
1	2846.10	2854.837	2719.741	2717.768	3219.252	3347.495	3167.483	3081.708	3460.983
2	2850.84	2858.741	2719.815	2705.574	3219.009	3222.234	3077.970	3091.650	3500.814
3	2885.72	2894.060	2725.060	2720.594	3217.096	3222.607	3092.170	3099.800	3493.985
4	2879.43	2917.048	2724.982	2723.118	3218.940	3221.563	3092.646	3098.167	3502.623
5	2879.61	2887.993	2725.307	2725.711	3217.928	3216.110	3061.760	3075.318	3480.950
6	2854.50	2857.829	2713.362	2724.421	3223.855	3219.990	3092.273	3130.129	3616.207
7	2863.88	2862.294	2724.785	2713.416	3218.554	3220.915	3076.216	3062.816	3932.044
8	2864.64	2903.185	2717.144	2717.516	3218.546	3220.040	3061.997	3066.598	3914.052
9	2854.93	2903.667	2722.131	2719.988	3219.096	3217.382	3066.670	3054.873	3497.995
10	2855.82	2901.562	2712.937	2716.961	3272.673	3225.527	3047.615	3076.598	3688.783
Average	**2863.55**	**2884.122**	**2720.526**	**2718.507**	**3224.495**	**3233.386**	**3083.680**	**3083.766**	**3608.844**
Variance	**82410.82**								
Max	**2885.72**	**2917.048**	**2725.307**	**2725.711**	**3272.673**	**3347.495**	**3167.483**	**3130.129**	**3932.044**
Min	**2846.10**	**2854.837**	**2712.937**	**2705.574**	**3217.096**	**3216.110**	**3047.615**	**3054.873**	**3460.983**

Table A.10: Data Set 1 - Execution times for Quick Sort algorithm. Time is measured in milliseconds

	Sorting Algorithms - Quick Sort - Data Set 1 (Milliseconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarge	m4.large	r3.4xlarge	r3.large	Local
1	5.000	5.000	5.000	4.000	5.000	5.000	5.000	5.000	25.000
2	5.000	5.000	4.000	4.000	5.000	5.000	5.000	5.000	26.000
3	5.000	5.000	4.000	4.000	5.000	5.000	5.000	5.000	29.000
4	4.000	5.000	4.000	4.000	5.000	5.000	5.000	5.000	203.000
5	5.000	5.000	4.000	4.000	5.000	5.000	5.000	5.000	25.000
6	5.000	5.000	4.000	4.000	5.000	5.000	5.000	5.000	24.000
7	4.000	5.000	4.000	4.000	5.000	5.000	5.000	5.000	22.000
8	4.000	5.000	4.000	4.000	5.000	5.000	5.000	5.000	133.000
9	5.000	4.000	4.000	4.000	5.000	5.000	5.000	5.000	53.000
10	5.000	5.000	4.000	4.000	5.000	5.000	5.000	5.000	34.000
Average	**4.700**	**4.900**	**4.100**	**4.000**	**5.000**	**5.000**	**5.000**	**5.000**	**57.400**
Variance	**308.598**								
Max	**5.000**	**5.000**	**5.000**	**4.000**	**5.000**	**5.000**	**5.000**	**5.000**	**203.000**
Min	**4.000**	**4.000**	**4.000**	**4.000**	**5.000**	**5.000**	**5.000**	**5.000**	**22.000**

Table A.11: Data Set 2 - Execution times for Quick Sort algorithm. Time is measured in milliseconds

	Sorting Algorithms - Quick Sort - Data Set 2 (Milliseconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarg	m4.large	r3.4xlarge	r3.large	Local
1	95.000	93.000	87.000	138.000	101.000	101.000	101.000	99.000	203.000
2	95.000	93.000	84.000	85.000	102.000	101.000	140.000	148.000	188.000
3	93.000	91.000	175.000	84.000	161.000	97.000	103.000	98.000	197.000
4	94.000	91.000	83.000	84.000	102.000	126.000	99.000	127.000	189.000
5	95.000	117.000	86.000	158.000	103.000	100.000	102.000	98.000	199.000
6	95.000	92.000	85.000	85.000	99.000	152.000	101.000	99.000	192.000
7	95.000	92.000	86.000	87.000	103.000	100.000	98.000	101.000	200.000
8	92.000	177.000	86.000	83.000	104.000	100.000	101.000	142.000	204.000
9	93.000	94.000	86.000	82.000	99.000	99.000	101.000	100.000	203.000
10	96.000	96.000	87.000	85.000	102.000	157.000	104.000	102.000	194.000
Average	**94.300**	**103.600**	**94.500**	**97.100**	**107.600**	**113.300**	**105.000**	**111.400**	**196.900**
Variance	**1020.393**								
Max	**96.000**	**177.000**	**175.000**	**158.000**	**161.000**	**157.000**	**140.000**	**148.000**	**204.000**
Min	**92.000**	**91.000**	**83.000**	**82.000**	**99.000**	**97.000**	**98.000**	**98.000**	**188.000**

Table A.12: Data Set 3 - Execution times for Quick Sort algorithm. Time is measured in milliseconds

	Sorting Algorithms - Quick Sort - Data Set 3 (Milliseconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarg	m4.large	r3.4xlarge	r3.large	Local
1	241.000	243.000	222.000	218.000	264.000	257.000	260.000	269.000	554.000
2	246.000	247.000	227.000	226.000	278.000	972.000	267.000	266.000	554.000
3	238.000	249.000	223.000	220.000	265.000	484.000	264.000	256.000	546.000
4	352.000	244.000	220.000	225.000	329.000	254.000	255.000	296.000	840.000
5	249.000	239.000	224.000	331.000	250.000	257.000	256.000	264.000	437.000
6	247.000	247.000	226.000	359.000	262.000	263.000	266.000	267.000	496.000
7	247.000	245.000	225.000	221.000	268.000	261.000	265.000	356.000	524.000
8	248.000	249.000	226.000	224.000	254.000	252.000	257.000	418.000	566.000
9	239.000	387.000	219.000	224.000	264.000	266.000	260.000	400.000	439.000
10	247.000	236.000	225.000	224.000	268.000	352.000	264.000	391.000	486.000
Average	**255.400**	**258.600**	**223.700**	**247.200**	**270.200**	**361.800**	**261.400**	**318.300**	**544.200**
Variance	**9786.683**								
Max	**352.000**	**387.000**	**227.000**	**359.000**	**329.000**	**972.000**	**267.000**	**418.000**	**840.000**
Min	**238.000**	**236.000**	**219.000**	**218.000**	**250.000**	**252.000**	**255.000**	**256.000**	**437.000**

Table A.13: Data Set 1 - Execution times for K-Mean algorithm. Time is measured in seconds

	K-Mean Algorithm - Data Set 1 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarge	m4.large	r3.4xlarge	r3.large	Local
1	33.835	33.836	31.166	31.394	37.880	37.404	36.114	36.566	77.771
2	33.836	33.811	31.212	31.332	37.097	37.761	36.218	36.544	77.094
3	33.903	33.880	31.242	31.486	37.014	37.673	36.279	36.334	76.817
4	33.963	34.514	31.178	31.442	37.069	37.780	36.081	36.245	71.438
5	33.756	33.848	31.677	31.581	36.977	37.700	36.207	36.485	71.014
6	33.777	33.914	31.154	31.504	37.139	37.775	36.295	36.438	71.211
7	33.833	33.810	31.197	31.353	37.134	37.786	36.090	36.356	71.163
8	33.898	33.818	31.160	31.306	36.956	37.503	36.131	36.252	72.181
9	33.807	34.481	31.480	31.319	37.123	37.139	36.170	36.357	71.652
10	33.812	33.894	31.138	31.448	37.076	37.175	36.492	36.344	78.010
Average	**33.842**	**33.981**	**31.260**	**31.417**	**37.147**	**37.570**	**36.208**	**36.392**	**73.835**
Variance	**175.341**								
Max	**33.963**	**34.514**	**31.677**	**31.581**	**37.880**	**37.786**	**36.492**	**36.566**	**78.010**
Min	**33.756**	**33.810**	**31.138**	**31.306**	**36.956**	**37.139**	**36.081**	**36.245**	**71.014**

Table A.14: Data Set 2 - Execution times for K-Mean algorithm. Time is measured in seconds

	K-Mean Algorithm - Data Set 2 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarg	m4.large	r3.4xlarge	r3.large	Local
1	72.687	72.486	66.719	67.188	79.329	79.355	77.027	77.419	170.096
2	72.423	72.519	67.580	67.059	79.191	79.277	78.224	77.621	154.512
3	72.630	72.703	66.623	66.955	80.487	79.264	77.943	77.658	158.753
4	72.336	72.443	67.947	67.142	79.202	79.554	77.357	77.746	157.118
5	72.147	73.008	67.107	67.144	79.391	79.487	77.442	77.723	159.573
6	72.549	72.428	66.750	66.920	79.354	79.638	77.322	77.761	160.597
7	72.787	72.456	66.805	66.935	80.924	79.802	77.462	77.855	159.026
8	72.499	72.538	66.754	66.796	79.229	79.592	77.541	77.791	178.343
9	72.802	72.132	66.967	66.996	79.321	79.532	77.377	78.130	174.342
10	72.466	72.419	66.750	67.199	79.229	79.318	77.623	77.725	160.012
Average	**72.533**	**72.513**	**67.000**	**67.033**	**79.566**	**79.482**	**77.532**	**77.743**	**163.237**
Variance	**904.983**								
Max	**72.802**	**73.008**	**67.947**	**67.199**	**80.924**	**79.802**	**78.224**	**78.130**	**178.343**
Min	**72.147**	**72.132**	**66.623**	**66.796**	**79.191**	**79.264**	**77.027**	**77.419**	**154.512**

Table A.15: Data Set 3 - Execution times for K-Mean algorithm. Time is measured in seconds

	K-Mean Algorithm - Data Set 3 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarg	m4.large	r3.4xlarge	r3.large	Local
1	129.921	128.711	118.830	120.182	140.621	140.944	137.207	138.416	285.972
2	129.222	129.589	119.443	119.459	141.292	141.095	137.336	138.411	286.803
3	129.537	129.332	119.732	119.230	141.715	141.061	138.550	138.835	305.745
4	128.397	129.364	118.420	119.201	140.642	141.344	137.662	138.377	309.059
5	128.433	129.237	119.255	119.566	140.988	141.469	136.919	138.035	284.976
6	129.187	129.787	121.194	121.717	141.115	140.862	137.081	138.835	286.520
7	129.092	129.505	118.340	121.778	142.197	142.382	139.356	139.359	283.721
8	129.015	129.661	118.287	118.988	143.606	142.568	137.006	138.749	287.033
9	128.713	128.902	118.400	119.098	140.706	141.192	137.184	138.266	307.719
10	128.779	128.864	118.287	118.930	140.785	140.978	137.374	137.687	309.129
Average	**129.030**	**129.295**	**119.019**	**119.815**	**141.367**	**141.390**	**137.568**	**138.497**	**294.668**
Variance	**3012.960**								
Max	**129.921**	**129.787**	**121.194**	**121.778**	**143.606**	**142.568**	**139.356**	**139.359**	**309.129**
Min	**128.397**	**128.711**	**118.287**	**118.930**	**140.621**	**140.862**	**136.919**	**137.687**	**283.721**

Table A.16: Data Set 1 - Execution times for Genetic algorithm. Time is measured in seconds

	Genetic Algorithm - Data Set 1 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarge	m4.large	r3.4xlarge	r3.large	Local
1	70.842	71.594	58.438	58.730	67.059	71.872	74.890	75.507	226.134
2	70.770	71.812	58.572	58.733	66.901	71.734	74.720	75.784	227.188
3	70.921	71.553	58.331	58.400	66.639	72.029	74.976	75.455	228.312
4	70.780	71.607	58.268	58.667	67.073	72.037	75.013	75.637	221.451
5	71.022	71.562	58.358	58.813	67.211	71.737	74.791	75.615	222.096
6	70.632	71.674	58.610	58.228	67.055	72.123	75.126	75.397	225.796
7	70.904	71.865	58.500	58.728	66.771	71.820	74.811	76.095	222.241
8	71.071	71.586	58.203	58.569	67.151	72.107	75.395	75.578	220.568
9	70.794	71.726	58.583	58.780	66.882	71.664	75.081	75.695	229.761
10	70.765	71.880	58.441	58.661	66.867	72.087	74.767	75.046	222.870
Average	**70.850**	**71.686**	**58.430**	**58.631**	**66.961**	**71.921**	**74.957**	**75.581**	**224.642**
Variance	**2744.535**								
Max	**71.071**	**71.880**	**58.610**	**58.813**	**67.211**	**72.123**	**75.395**	**76.095**	**229.761**
Min	**70.632**	**71.553**	**58.203**	**58.228**	**66.639**	**71.664**	**74.720**	**75.046**	**220.568**

Table A.17: Data Set 2 - Execution times for Genetic algorithm. Time is measured in seconds

	Genetic Algorithm - Data Set 2 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarg	m4.large	r3.4xlarge	r3.large	Local
1	118.867	119.217	98.706	100.716	113.949	115.142	130.004	128.940	382.724
2	118.415	120.083	96.708	97.096	110.951	119.441	125.571	126.674	388.824
3	118.074	120.065	97.640	97.523	111.076	119.832	124.714	127.043	381.790
4	118.781	120.774	97.346	97.497	111.047	120.251	125.276	126.140	383.344
5	118.622	119.827	97.033	97.356	112.577	118.933	126.202	126.897	381.574
6	119.657	122.228	97.620	97.176	111.423	121.044	125.278	127.720	383.909
7	118.489	119.755	98.309	99.061	111.769	119.133	125.459	126.427	384.261
8	117.866	120.176	97.388	97.382	111.536	120.095	125.486	126.694	386.908
9	118.881	120.372	97.809	97.829	111.442	119.242	124.665	126.233	379.239
10	118.867	120.114	97.586	98.082	110.814	119.809	125.669	126.768	364.729
Average	**118.652**	**120.261**	**97.615**	**97.972**	**111.658**	**119.292**	**125.832**	**126.954**	**381.730**
Variance	**8033.408**								
Max	**119.657**	**122.228**	**98.706**	**100.716**	**113.949**	**121.044**	**130.004**	**128.940**	**388.824**
Min	**117.866**	**119.217**	**96.708**	**97.096**	**110.814**	**115.142**	**124.665**	**126.140**	**364.729**

Table A.18: Data Set 3 - Execution times for Genetic algorithm. Time is measured in seconds

	Genetic Algorithm - Data Set 3 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarg	m4.large	r3.4xlarge	r3.large	Local
1	214.214	215.632	177.351	181.183	205.356	206.361	234.136	232.633	723.448
2	216.314	219.367	178.014	177.952	201.702	217.276	230.201	231.785	724.808
3	216.177	218.277	176.426	177.065	204.553	217.489	228.809	231.745	704.315
4	216.197	218.574	176.891	176.697	202.170	217.279	229.653	230.744	689.299
5	215.564	218.656	176.820	176.965	203.028	218.746	229.823	231.286	715.358
6	216.007	218.542	177.006	177.389	203.257	218.519	229.190	230.245	712.665
7	216.051	218.148	177.242	177.163	203.401	218.630	228.352	230.839	720.651
8	216.218	218.859	177.113	178.292	203.226	218.231	230.029	231.413	725.674
9	216.706	219.529	177.203	177.163	201.931	217.437	230.232	231.636	744.217
10	215.227	218.728	176.566	178.374	202.891	218.546	228.675	230.621	752.331
Average	**215.868**	**218.431**	**177.063**	**177.824**	**203.152**	**216.851**	**229.910**	**231.295**	**721.277**
Variance	**29576.176**								
Max	**216.706**	**219.529**	**178.014**	**181.183**	**205.356**	**218.746**	**234.136**	**232.633**	**752.331**
Min	**214.214**	**215.632**	**176.426**	**176.697**	**201.702**	**206.361**	**228.352**	**230.245**	**689.299**

Table A.19: Data Set 1 - Execution times for Word Count Algorithm. Time is measured in seconds

	Word Count Algorithm - Data Set 1 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarge	m4.large	r3.4xlarge	r3.large	Local
1	109.114	109.217	109.397	109.407	123.735	124.945	117.162	117.325	291.402
2	109.674	103.196	108.840	110.043	125.264	121.512	116.764	118.092	363.805
3	109.186	108.546	108.398	109.964	119.834	124.199	117.069	117.932	199.014
4	107.946	103.008	110.535	110.038	122.898	124.292	117.164	118.376	204.073
5	109.202	102.636	110.362	111.887	117.758	124.457	114.174	118.076	256.207
6	109.539	100.153	108.982	110.407	120.480	124.073	116.995	118.117	212.341
7	109.252	111.320	110.738	110.243	125.425	123.541	116.626	120.226	262.358
8	109.095	109.181	110.501	110.964	125.564	124.174	116.965	118.614	172.736
9	109.179	109.925	78.495	110.538	122.278	122.481	115.303	117.611	174.301
10	109.527	110.101	113.268	111.087	139.182	124.020	117.842	117.915	266.029
Average	**109.171**	**106.728**	**106.952**	**110.458**	**124.242**	**123.769**	**116.606**	**118.228**	**240.227**
Variance	**1800.970**								
Max	**109.674**	**111.320**	**113.268**	**111.887**	**139.182**	**124.945**	**117.842**	**120.226**	**363.805**
Min	**107.946**	**100.153**	**78.495**	**109.407**	**117.758**	**121.512**	**114.174**	**117.325**	**172.736**

Table A.20: Data Set 2 - Execution times for Word Count Algorithm. Time is measured in seconds

	Word Count Algorithm - Data Set 2 (Seconds)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarg	m4.large	r3.4xlarge	r3.large	Local
1	218.428	218.064	221.280	221.294	234.571	249.869	235.281	234.992	336.395
2	218.856	219.832	219.062	215.947	178.398	242.737	234.351	234.821	440.384
3	214.144	213.144	218.332	215.076	245.084	247.766	233.827	234.924	314.834
4	218.466	213.726	218.908	220.526	248.396	247.904	234.328	234.711	364.036
5	218.526	220.007	219.448	220.299	244.731	247.628	234.118	235.816	435.928
6	218.483	220.011	210.243	220.763	241.656	248.754	185.967	235.843	348.534
7	218.435	218.717	217.692	214.600	187.002	248.126	234.637	234.658	380.858
8	218.310	212.555	216.232	220.647	249.770	248.099	185.685	235.292	435.918
9	217.972	172.754	219.649	185.082	250.881	248.076	234.808	230.818	365.665
10	218.875	211.689	221.934	219.547	244.953	250.063	234.591	235.892	341.200
Average	**218.050**	**212.050**	**218.278**	**215.378**	**232.544**	**247.902**	**224.759**	**234.777**	**376.375**
Variance	**2658.995**								
Max	**218.875**	**220.011**	**221.934**	**221.294**	**250.881**	**250.063**	**235.281**	**235.892**	**440.384**
Min	**214.144**	**172.754**	**210.243**	**185.082**	**178.398**	**242.737**	**185.685**	**230.818**	**314.834**

Table A.21: Data Set 3 - Execution times for Word Count Algorithm. Time is measured in Minutes

	Word Count Algorithm - Data Set 3 (Mins)								
	Compute Optimized				General Purpose		Memory Optimized		
	c3.4xlarge	c3.large	c4.4xlarge	c4.large	m4.4xlarg	m4.large	r3.4xlarge	r3.large	Local
1	76.201		69.989		110.600		98.021		
2	76.090		72.884		85.171		85.624		
3	73.449		71.037		89.360		106.787		
4	73.743		70.301		95.018		91.595		
5	74.889		71.261		85.978		91.591		
6	74.583		71.438		106.823		89.931		
7	74.071		72.973		110.824		91.981		
8	73.762		72.467		83.559		113.677		
9	76.623		72.383		86.881		95.745		
10	73.394		72.253		97.587		94.701		
Average	**74.681**		**71.699**		**95.180**		**95.965**		
Variance	**168.586**								
Max	**76.623**		**72.973**		**110.824**		**113.677**		
Min	**73.394**		**69.989**		**83.559**		**85.624**		

A.3 Scale-Out Service Evaluation

This section presents the total ten times of the Word Frequency Counter algorithm execution time for evaluating the scale-out service.

Table A.22: Execution times for the Hadoop Word Count algorithm. Time is measured in minutes

	Hadoop - Word Count Algorithm - Data Set 3 (Mins)					
	m3.xlarge				m3.2xlarge	
	5 EC2 - Total	5 EC2 - Job	3 EC2 - Total	3 EC2 - Job	3 EC2 - Total	3 EC2 - Job
1	6.056	1.718	5.897	2.268	5.204	1.884
2	5.369	1.785	5.705	2.376	5.040	1.810
3	5.706	1.777	6.207	2.285	5.708	1.860
4	5.033	1.726	5.702	2.376	5.367	1.826
5	5.371	1.776	5.707	2.318	6.208	2.018
6	4.871	1.776	5.873	2.268	5.368	1.818
7	5.038	1.818	5.871	2.293	5.542	1.918
8	5.032	1.793	5.705	2.318	5.373	1.826
9	5.033	1.784	6.374	2.285	5.032	1.843
10	5.043	1.726	6.550	2.285	5.536	1.868
Average	5.255	1.768	5.959	2.307	5.438	1.867
Max	6.056	1.818	6.550	2.376	6.208	2.018
Min	4.871	1.718	5.702	2.268	5.032	1.810

Appendix B: Selected Source Code

This section presents selected snippets of the prototype implementation source code that were discussed in the thesis.

B.1 Sample Template

This section presents an example of design and source code templates that are offered for algorithm developers to host new algorithms. As seen in the section the source code template has all standard source code to provide the new algorithm using the same workflow process. Developers only need to enter the algorithm source code and any helper functions in their sections as shown in the sample source code template, lines 120 and 192.

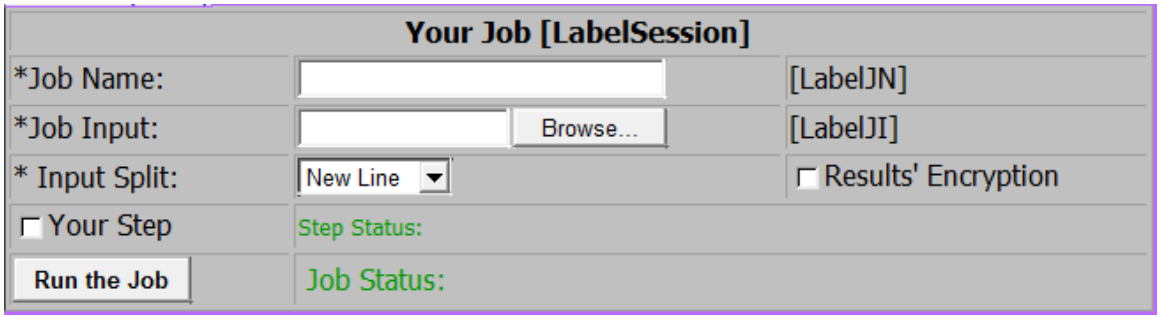


Figure B.1: Adding new algorithms – Default template design

```
            // start Job Step
            if (CheckBoxStepName.Checked == true)
            {
                // Calculate algorithm execution time - start
                ts = (DateTime.UtcNow - new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc));
                millis = (long)ts.TotalMilliseconds;


                /* Start: user algorithm
                *
                * Write your code Here
                * Write your code Here
                * Write your code Here
                * Write your code Here
                * Write your code Here
                *
                *
                */
                //End: user algorithm


                /*  Save Output function to Storage
                  * Use functions in HelpingFunctions Class for help
                  * For example..
                */
                // encreption
                if (CheckBoxEncr.Checked == true)
                {
                    byte[] Key = Encoding.UTF8.GetBytes(db.Encryption_KEY(LabelSession.Text));
                    if (Key.Length != 0)
                    {
                        f.EncryptStringToBytes(ResultPath, Key, Key, ResultName);
                        Encryption = "1";
                    }
```

Figure B.2: Default template: new algorithm source code section

```
                //Final status
                LabelStepStatus.Text = "Step status: Executed. Execution Time is " + ExecutionTime + " Milliseconds";
                LabelStepStatus.ForeColor = Color.Green;
            } // end job step

            LabelJobStatus.Text = "Job status: Completed.";
        }
    }

    // Start Job Helping Functions
    /*
    *
    * Write youe helping function
    * Which you need to execute your code
    * here.
    *
    *
    */
    // End Job Helping Functions
}
```

Figure B.3: Default template: new algorithm helper functions section

B.2 Utility Services

This section presents utility classes which are responsible for building a bridge between the framework services and AWS services.

B.2.1 DB Manager Class

This section presents samples of the DB manager class functions which is responsible for managing data transaction between the web servers and the DB server.

```
73  public bool InsertResultsSummaryToDB(string JobName, string ResultPath, string UserName, string Encryption)
74  {
75      try
76      {
77          SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString);
78          conn.Open();
79          string InsertData =
80              "insert into Results(ResultPath,JobName,UserName,Encryption) values (@ResultPath,@JobName,@UserName,@Encryption)";
81          SqlCommand com = new SqlCommand(InsertData, conn);
82
83          com.Parameters.AddWithValue("@ResultPath", ResultPath);
84          com.Parameters.AddWithValue("@JobName", JobName);
85          com.Parameters.AddWithValue("@UserName", UserName);
86          com.Parameters.AddWithValue("@Encryption", Encryption);
87
88          com.ExecuteNonQuery();
89          conn.Close();
90          return true;
91      }
92      catch (Exception ex)
93      {
94          return false;
95      }
96
97  }
```

```
125 public string InsertEC2Images(string ImageId, string ImageName,string EC2ID, string UserName)
126 {
127     try
128     {
129         SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString);
130         conn.Open();
131         string InsertData =
132             "insert into EC2Images(ImageId,ImageName,EC2Id,UserName) values (@ImageId,@ImageName,@EC2Id,@UserName)";
133         SqlCommand com = new SqlCommand(InsertData, conn);
134
135         com.Parameters.AddWithValue("@ImageId", ImageId);
136         com.Parameters.AddWithValue("@ImageName", ImageName);
137         com.Parameters.AddWithValue("@EC2Id", EC2ID);
138         com.Parameters.AddWithValue("@UserName", UserName);
139
140         com.ExecuteNonQuery();
141         conn.Close();
142         return "Done";
143     }
144     catch (Exception ex)
145     {
146         return ex.Message;
147     }
148
149 }
```

Figure B.4: Samples of the DB manager class functions

B.2.2 EC2 Manager Class

This section presents samples of the EC2 manager class functions which is responsible for managing transaction between the framework and Amazon EC2.

```
24      public string[] LanuchEC2Instance(string AWS_ACCESS_KEY, string AWS_SECRET_KEY,string Instance_Type, string Image_Id, int Min_Count, int Max_Count)
25      {
26          //const string AWS_ACCESS_KEY = "AKIAJEP6U4CWBVWU5DCA";
27          //const string AWS_SECRET_KEY = "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";
28          try
29          {
30              IAmazonEC2 ec2Client = AWSClientFactory.CreateAmazonEC2Client(AWS_ACCESS_KEY, AWS_SECRET_KEY, RegionEndpoint.USWest2);
31
32              //To launch EC2 instances
33              var runInstancesRequest = new RunInstancesRequest()
34              {
35                  ImageId = Image_Id,
36                  InstanceType = Instance_Type,
37                  MinCount = 1,
38                  MaxCount = 1,
39                  //SecurityGroupIds = new List<string> { "sg-52288537" },
40                  SecurityGroups = new List<string> { "AaaS_SecurityGroup" }
41                  //KeyName = "FinalProject-WinKey"
42              };
43              RunInstancesResponse runResponse = ec2Client.RunInstances(runInstancesRequest);
44
45
46              //CreateImageRequest
47              ReservedInstances instanceRe = new ReservedInstances();
48              string dd = instanceRe.ReservedInstancesId;
49
50
51
52
53              List<Instance> instances = runResponse.Reservation.Instances;
54              List<String> instanceIDs = new List<string>();
55              foreach (Instance item in instances)
56              {
```

```
221     public string CreateAMIS(string AWS_ACCESS_KEY, string AWS_SECRET_KEY, string InstanceId, string ImageName)
222     {
223         try
224         {
225             IAmazonEC2 ec2Client = AWSClientFactory.CreateAmazonEC2Client(AWS_ACCESS_KEY, AWS_SECRET_KEY, RegionEndpoint.USWest2);
226
227             var AMISRequest = new CreateImageRequest();
228             AMISRequest.InstanceId = InstanceId;
229             AMISRequest.Name = ImageName;
230             CreateImageResponse AMISResponse = ec2Client.CreateImage(AMISRequest);
231             string ImageId = AMISResponse.ImageId;
232
233             Thread.Sleep(100000);
234             return "Complete:"+ImageId;
235         }
236         catch (AmazonEC2Exception EC2Exception)
237         {
238             return "Error:" + EC2Exception.Message;
239         }
240     }
241
```

Figure B.5: Samples of the EC2 manager class functions

Appendix C: Sample Services

This appendix provides some of the framework prototype services source code. Out of the many services that were implemented in the prototype, a select few have been featured in this Appendix.

C.1 Scale-Up Service

This section presents samples of the scale-up service source code which was discussed in Section 4.5.4.1.

```
45          Image_Id = DropDownListAMI.SelectedValue;
46      }
47      if (Image_Id.Length == 0)
48      {
49          LabelLunchStatus.Text = "Please choose an image (AMI)";
50          LabelLunchStatus.ForeColor = Color.Red;
51          return;
52      }
53      int Min_Count = 1;
54      int Max_Count = 1;
55      DBManager db = new DBManager();
56      string AWS_ACCESS_KEY = db.AWS_ACCESS_KEY(LabelSession.Text);
57      string AWS_SECRET_KEY = db.AWS_SECRET_KEY(LabelSession.Text);
58      if (AWS_ACCESS_KEY.Length == 0 || AWS_SECRET_KEY.Length == 0)
59      {
60          LabelLunchStatus.Text = "Sorry : This user don't have AWS account information";
61          LabelLunchStatus.ForeColor = Color.Red;
62          return;
63      }
64      EC2Manager ec2 = new EC2Manager();
65      string[] RunEC2Instance = ec2.LanuchEC2Instance(AWS_ACCESS_KEY, AWS_SECRET_KEY, Instance_Type, Image_Id, Min_Count, Max_Count);
66      if (RunEC2Instance[0] == "running")
67      {
68          //LabelInstanceID.Text = RunEC2Instance[3];
69          HelpingFunctions f = new HelpingFunctions();
70          bool status = db.InsertInstanceInfo(RunEC2Instance[3], RunEC2Instance[2],RunEC2Instance[1],TextBoxIN.Text, LabelSession.Text);
71          LabelLunchStatus.Text = "Instance Status is " + RunEC2Instance[0];
72          LabelLunchStatus.ForeColor = Color.Green;
73          LabelEC2URL.Text = RunEC2Instance[2];
74      }
75  }
```

```
125         EC2Manager ec2 = new EC2Manager();
126         string result = ec2.CreateAMIS(AWS_ACCESS_KEY, AWS_SECRET_KEY, EC2ID.Text, EC2Name);
127         string[] resultinfo = result.Split(':');
128         if (resultinfo[0] == "Complete")
129         {
130             string status = db.InsertEC2Images(resultinfo[1], EC2Name, EC2ID.Text, LabelSession.Text);
131             LabelSDStatus.Text = "Image has been Created:" + resultinfo[1];
132             LabelSDStatus.ForeColor = Color.Green;
133             return;
134         }
```

Figure C.1: Samples of the scale-up service source code

C.2 Scale-Out Service

This section presents a sample of the scale-up service source code which was discussed in Section 4.5.4.2.

```
            //upload Jop Mapper
            S3JobMapper = bucketName + "/" + JobName + "/Job";
            if (LabelMapperPath.Text.Length == 0)
            {
                LabelMU.Text = "Please Upload a Mapper Code File";
                LabelMU.ForeColor = Color.Red;
                return;
            }
            s3.UploadFile(AWS_ACCESS_KEY, AWS_SECRET_KEY, LabelMapperPath.Text, S3JobMapper);
            filename = Path.GetFileName(LabelMapperPath.Text);
            EMRJobMapper = "s3://" + S3JobMapper + "/" + filename;
            File.Delete(LabelMapperPath.Text);

            //upload Jop Reducer
            if (LabelReducerPath.Text.Length != 0)
            {
                S3JobReducer = bucketName + "/" + JobName + "/Job";
                s3.UploadFile(AWS_ACCESS_KEY, AWS_SECRET_KEY, LabelReducerPath.Text, S3JobReducer);
                filename = Path.GetFileName(LabelReducerPath.Text);
                EMRJobMapper = "s3://" + S3JobReducer + "/" + filename;
                File.Delete(LabelReducerPath.Text);
            }
            EMRJobReducer = "aggregate";
            Directory.Delete(Path.Combine(Server.MapPath("~/UploadedFiles/"), LabelSession.Text), true);
            // Calculate  Upload Data Time - end
            TimeSpan t = sw.Elapsed;
            string UploadDataTime = sw.Elapsed.ToString();
```

```
        EMRManager emr = new EMRManager();
        string status = emr.RuStreamingStep(AWS_ACCESS_KEY, AWS_SECRET_KEY, TextBoxJN.Text, DropDownListMIT.SelectedValue.ToString(),
            DropDownListSIT.SelectedValue.ToString(), Int32.Parse(DropDownListIC.SelectedValue.ToString()),
            JobInput, JobResult, EMRJobMapper, EMRJobReducer, LogFile);

        string[] statusinfo = status.Split('/');
        if (statusinfo[0] != "completed")
        {
            bool RSStatus = db.InsertResultsSummaryToDB(TextBoxJN.Text, JobOutput, LabelSession.Text, Encryption);
            bool InsertStatus = db.InsertResultsDetailToDB(JobName, JobName, JobOutput, LabelSession.Text, "Error");
            if (InsertStatus == false)
            {
                LabelJStatus.Text = "Error: Can't Insert output files to DB.";
                LabelJStatus.ForeColor = Color.Red;
                return;
            }
            LabelJStatus.Text = "Error: " + statusinfo[1] + ".. Check Log File for Details";
            LabelJStatus.ForeColor = Color.Red;
            return;
        }
```

Figure C.2: Samples of the scale-out service source code

C.3 Sorting Algorithms

This section presents a sample of the sorting algorithm source code which was discussed in Section 4.5.2.1.

```
119             // start Bubble Algorithm Step
120             if (CheckBoxBS.Checked == true)
121             {
122                 // Calculate algorithm execution time - start
123                 sw.Restart();
124 
125                 long[] numbers = new long[numbers_org.Length];
126                 Array.Copy(numbers_org, numbers, numbers_org.Length);
127                 BubbleSort(numbers);
128                 string[] result = new string[numbers.Length];
129 
130                 if (RadioButtonITypes.SelectedValue == "File")
131                 {
132                     for (int i = 0; i < result.Length; i++)
133                     {
134                         result[i] = numbers[i].ToString();
135                     }
136                     string ResultPath = Path.Combine(Server.MapPath("~/Results"), LabelSession.Text, TextBoxJN.Text);
137                     if (!System.IO.Directory.Exists(ResultPath))
138                     {
139                         System.IO.Directory.CreateDirectory(ResultPath);
140                     }
141                     string ResultName = Path.Combine(ResultPath, CheckBoxBS.Text);
142                     File.WriteAllLines(ResultName, result);
143 
144                     // encreption
145                     if (CheckBoxEncr.Checked == true)
146                     {
147                         byte[] Key = Encoding.UTF8.GetBytes(db.Encryption_KEY(LabelSession.Text));
148                         if (Key.Length != 0)
149                         {
150                             f.EncryptStringToBytes(ResultName, Key, Key, ResultName);
151                             Encryption = "1";
152                         }
```

```
190                     //Final status
191                     LabelStatusBS.Text = "Step status: Executed. Execution Time is " + ExecutionTime;
192                     LabelStatusBS.ForeColor = Color.Green;
193                 }
194                 else //if (RadioButtonITypes.SelectedValue == "Text")
195                 {
196                     sw.Restart();
197                     TextBoxManual.Text += "\r\n";
198                     TextBoxManual.Text += "*** Bubble Sort ***";
199                     TextBoxManual.Text += "\r\n";
200                     for (int i = 0; i < result.Length; i++)
201                     {
202                         result[i] = numbers[i].ToString();
203                         TextBoxManual.Text += result[i];
204                         TextBoxManual.Text += " , ";
205                     }
206 
207                     // Calculate algorithm execution time - end
208                     TimeSpan t = sw.Elapsed;
209                     // double ExecutionTime = sw.Elapsed.TotalMilliseconds;
210                     string ExecutionTime = sw.Elapsed.ToString();
211 
212                     //Final status
213                     LabelStatusBS.Text = "Step status: Executed. Execution Time is " + ExecutionTime;
214                     LabelStatusBS.ForeColor = Color.Green;
215                 }
216             } // end bubble algorithm
```

Figure C.3: Samples of the sorting algorithm source code

C.4 K-Mean Algorithm

This section presents a sample of the K-Mean algorithm screenshot and source code which was discussed in Section 4.5.2.3.

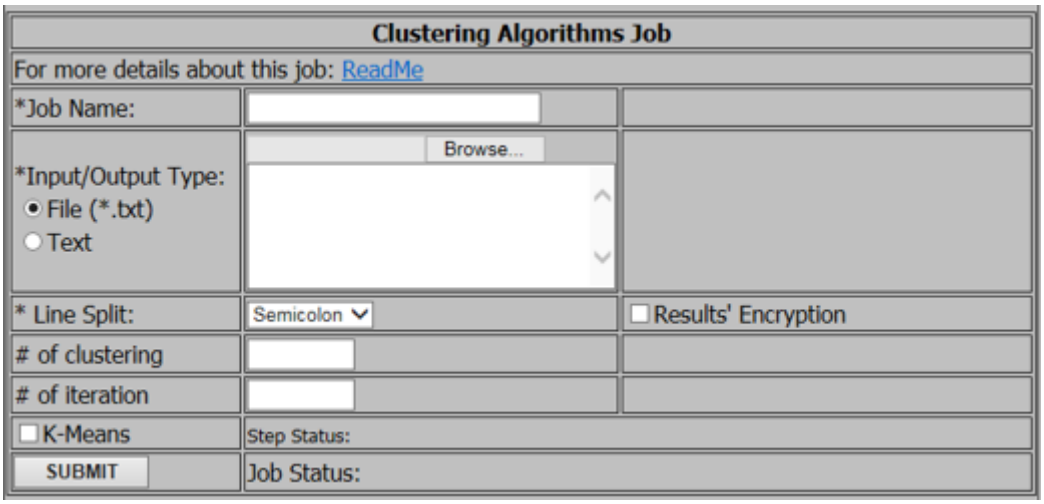


Figure C.4: K-Mean algorithm interface

```
263     public int[] Cluster(double[][] rawData, int numClusters)
264     {
265         // k-means clustering
266         // index of return is tuple ID, cell is cluster ID
267         // ex: [2 1 0 0 2 2] means tuple 0 is cluster 2, tuple 1 is cluster 1, tuple 2 is cluster 0, tuple 3 is cluster 0,
268         // an alternative clustering DS to save space is to use the .NET BitArray class
269         double[][] data = Normalized(rawData); // so large values don't dominate
270 
271         bool changed = true; // was there a change in at least one cluster assignment?
272         bool success = true; // were all means able to be computed? (no zero-count clusters)
273 
274         // init clustering[] to get things started
275         // an alternative is to initialize means to randomly selected tuples
276         // then the processing loop is
277         // loop
278         //    update clustering
279         //    update means
280         // end loop
281         int[] clustering = InitClustering(data.Length, numClusters, 0); // semi-random initialization
282         double[][] means = Allocate(numClusters, data[0].Length); // small convenience
283         int ite = Int32.Parse(TextBoxIteration.Text);
284         int maxCount = data.Length * ite; // sanity check
285         int ct = 0;
286         while (changed == true && success == true && ct < maxCount)
287         {
288             ++ct; // k-means typically converges very quickly
289             success = UpdateMeans(data, clustering, means); // compute new cluster means if possible. no effect if fail
290             changed = UpdateClustering(data, clustering, means); // (re)assign tuples to clusters. no effect if fail
291         }
```

Figure C.5: Samples of the K-Mean algorithm source code