Evaluating Google App Engine for Enterprise Application Development

Comparison of Performance and Development Environment

Kashif Khan and Asar Jan
This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 30 weeks of full time studies.

Contact Information:
Author(s):
Kashif Khan
E-mail: kashifkhankk@gmail.com

Asar Jan
E-mail: asar200790@gmail.com

University advisor(s):
Dr. Håkan Grahn
School of Computing

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

Internet : www.bth.se/com
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
ABSTRACT

Context: Google App Engine (GAE) is a cloud computing platform, which allows developers to develop and deploy web based applications on it. It is composed of hundreds of thousands of commodity servers, distributed globally. GAE scales computing resources automatically, and developers only pay for the amount of time their application uses GAE resources. It imposes some restrictions and limitations on the design and development of applications such as execution time limit, lack of global transactions, way of communication with other applications etc.

Objectives: In this study, we evaluate Google App Engine from enterprise application development point of view. We analyzed the behavior and architecture of Google App Engine to evaluate its feasibility for enterprise application development and compared it with Amazon Elastic Compute 2 (Amazon cloud service). We also studied Google App Engine storage system, Bigtable, which is a distributed non-SQL based data store.

Methods: In this research study, we performed a literature study of the related cloud computing technologies and then a prototype enterprise application was developed for experiment.

Results: Our literature study showed that Google App Engine has some limitations and restrictions on the development tools and environment. It does not provide support for global transaction, lacks strict consistency mechanisms, and provides highly virtualized and abstract view of the platform to the developer. Despite these limitations, the literature study and our experiment showed that Google App Engine is suited for applications with high read, searching and large write-stream operations. Google App Engine showed consistent performance as compared to Amazon Elastic Compute 2 in our experiment.

Conclusions: We conclude that Google App Engine is a scalable platform while maintaining consistent performance but has some limitations due to its architecture and restrictions imposed on the development tools and environment. These limitations make it non-feasible for some types of enterprise applications, such as applications with high and intensive data computation requirement. But in other cases, such as simple data management and non-transactional applications, Google App Engine is an attractive platform.

Keywords: Google App Engine, Enterprise application development, feasibility of application development on cloud, Amazon EC 2
ACKNOWLEDGMENT

First of all we are thankful to ALLAH the Almighty for HIS blessings and for giving us the knowledge and patience to complete this thesis work.

We would like to express our sincere gratitude to our advisor Dr. Håkan Grahn for his motivation, guidance and support throughout the thesis.

We are thankful to our parents & relatives for their prayers and support throughout our studies. We are also thankful to our friends who took part in the experiment.
# Table of Contents

ABSTRACT ................................................................................................................................. 1

ACKNOWLEDGMENT .................................................................................................................... 2

1 INTRODUCTION .......................................................................................................................... 7

1.1 OVERVIEW OF RESEARCH AREA ......................................................................................... 7
1.2 CLOUD COMPUTING .............................................................................................................. 7
1.3 GOOGLE APP ENGINE AND AMAZON ELASTIC COMPUTE 2 ............................................ 9
1.4 PROBLEM DEFINITION ......................................................................................................... 10
1.5 AIMS AND OBJECTIVES ..................................................................................................... 10
1.6 RESEARCH QUESTIONS ..................................................................................................... 10
1.7 MAIN CONTRIBUTION ........................................................................................................ 11
1.8 STRUCTURE OF THESIS ..................................................................................................... 11

2 RESEARCH METHODOLOGY .................................................................................................... 13

2.1 OVERVIEW .......................................................................................................................... 13
2.2 RESEARCH QUESTIONS AND METHODOLOGY .............................................................. 14
2.3 VALIDITY THREATS ........................................................................................................... 14

3 LITERATURE STUDY AND RELATED WORK ............................................................................ 16

3.1 LITERATURE STUDY PLAN ............................................................................................... 16

3.1.1 Keywords and Search String ......................................................................................... 16
3.1.2 Data Sources .................................................................................................................. 17
3.1.3 Criteria for Inclusion and Exclusion ............................................................................. 17
3.1.4 Quality Assessment Criteria ......................................................................................... 17
3.1.5 Bibliography Management ............................................................................................ 18
3.1.6 Search Results ............................................................................................................. 18
3.1.7 Data Extraction Strategy ............................................................................................... 20
3.1.8 Data Synthesis ............................................................................................................. 21

3.2 REVIEW OF MAJOR RELATED WORK ............................................................................... 21

3.2.1 High Performance Scientific and Cloud Computing ...................................................... 21
3.2.2 Open Source Cloud technologies ................................................................................... 22
3.2.3 Cloud Platform Comparison ........................................................................................ 22
3.2.4 Cloud Data Storage Technologies .............................................................................. 22
3.2.5 Migration and Deployment on Cloud Platforms ........................................................... 23

4 REVIEW OF GOOGLE APP ENGINE AND AMAZON CLOUD .................................................. 24

4.1 GOOGLE APP ENGINE ....................................................................................................... 24

4.1.1 NoSQL ............................................................................................................................ 24
4.1.2 Google File system ......................................................................................................... 24
4.1.3 BigTable ....................................................................................................................... 25

4.2 AMAZON CLOUD .............................................................................................................. 26

4.2.1 Amazon Elastic Compute Cloud ................................................................................... 26
4.2.2 Amazon EC 2 Instances ............................................................................................... 26
4.2.3 Amazon EC2 Storage .................................................................................................... 27
4.2.4 Amazon Dynamo .......................................................................................................... 27

4.3 PRICING ............................................................................................................................... 27

5 PROTOTYPE APPLICATION ....................................................................................................... 28

5.1 PURPOSE OF PROTOTYPE APPLICATION ......................................................................... 28
5.2 DEVELOPMENT TOOLS ..................................................................................................... 28
5.3 PROTOTYPE APPLICATION ARCHITECTURE .................................................................... 29
5.4 PROTOTYPE APPLICATION FEATURES ............................................................................. 30
List of Figures
Figure 1 Google search trend about Cloud computing and related technologies [35] ...................... 9
Figure 2 Abstract view of Research Methodology .............................................................................. 13
Figure 3 Number of Studies selected from different Sources .............................................................. 19
Figure 4 Selection of studies and publication years ........................................................................... 19
Figure 5 Selected Studies and Document Types ................................................................................ 20
Figure 6 Abstract View of the Prototype Application ......................................................................... 30
Figure 7 Client Application for invoking web requests ..................................................................... 32
Figure 8 Average Response Time ...................................................................................................... 38
Figure 9 Maximum and Minimum Response Time ............................................................................ 39
Figure 10 Average File Upload Time (10 concurrent requests) .......................................................... 40
Figure 11 Maximum and Minimum File Upload Time (10 concurrent requests) ............................... 41
Figure 12 Average File Read Time .................................................................................................... 42
Figure 13 Maximum and Minimum File Read Time ........................................................................... 43
Figure 14 Average Write Time .......................................................................................................... 43
Figure 15 Maximum and Minimum Write Time ................................................................................... 44
Figure 16 Average Execution Time .................................................................................................... 45
Figure 17 Maximum and Minimum Execution Time ......................................................................... 46
Figure 18 Effect on Execution Time with increase in Data volume .................................................... 47
Figure 19 Effect on Response Time with increase in Data volume ..................................................... 48
List of Tables
Table 1 Research Questions and Research Methodology ................................................................. 14
Table 2 Search Strings ...................................................................................................................... 16
Table 3 Search Results ..................................................................................................................... 18
Table 4 Data Extraction Form ........................................................................................................ 20
Table 5 Memory Requirement ........................................................................................................ 48
Table 6 Source Lines of Code ......................................................................................................... 49
Table 7 Development Time ............................................................................................................. 49
Table 8 Debugging Time .................................................................................................................. 49
Table 9 Recommendations for Platform Selection ........................................................................ 54
1 **INTRODUCTION**

1.1 **Overview of Research Area**

Information technology (IT) is a dynamic field, it changes rapidly and from enterprise point of view, managing information technology is a very complex task [38]. Enterprises have to manage both software and hardware to support their business operations. There are rapid changes in hardware and software technology, and keeping the infrastructure updated to the latest technology on time is a difficult target to achieve in most cases. This is a major issue in many organizations while dealing with IT [38]. Many enterprise organizations have large scale IT infrastructure such as data warehouses, which are both expensive and difficult to maintain [73]. Expanding or changing IT infrastructure takes a lot of time as compared to other resources such as sales, marketing, support etc. In the last decade or so IT has become the main point of latency when business needs to change or move forward [56]. With the fluctuation in economy, the IT infrastructure is also affected. Sometimes more resources are needed, requiring more investment and it also takes time to acquire these resources, while in other cases, resources are considered redundant, which are scrapped in most cases, also causing monetary losses [56].

Customer Relationship Management (CRM) has become an important part of business strategy in the last couple of decades for most organizations [47]. Almost all enterprise organizations have automated their customer relationship processes and different CRM applications are used by them [47]. These applications in many cases require having large data centers with high number of maintenance staff. In this research study, a CRM application is used for comparative analysis of cloud platforms.

The emergence of cloud computing had led enterprise developers to move their applications and data to cloud platforms, in order to achieve reduction in cost, easier maintenance of software and hardware resources, high availability of service etc. Amazon was the first company to properly use the characteristics of cloud computing and to provide their physical resources as virtual resources to the customers [73] followed by others. Now different cloud platforms such as Google App Engine, Windows Azure, and Salesforce.com etc. are available in the market, providing cloud services, which can be utilized by enterprise developers to develop and migrate their application and data, and benefit from cloud computing.

1.2 **Cloud Computing**

The advent of parallel, grid and distributed computing have led to the emergence of a new computing paradigm known as cloud computing. Cloud computing focuses on sharing of computing resources over a scalable network [58]. In cloud computing, a large number of computer resources (data centers) are deployed in order to provide virtually unlimited CPU power, storage capacity etc., the purpose of such a system is to provide computing to the users in the form of commodity, similar to electricity grid. Users/organizations can consume the cloud computing services according to their needs and pay only for what they have consumed [66, 63]. IT organizations can outsource their whole IT infrastructure to cloud service providers in order to get rid of maintenance of their systems [98, 63]. This also helps in cutting the maintenance costs while the cloud service providers perform maintenance, up-gradation and availability of service [98, 73].
Cloud computing is available in different patterns (service types), commonly as Software as a Service (SaaS), Platforms as a Service (PaaS) and Infrastructure as a Service (IaaS). Businesses are using these different patterns of cloud computing to provide services to customers, based on their requirements [58, 74, 29]. Some of the main service types are described below:

- **Software as a Service (SaaS)**
  In this type of service, pre-packaged software is provided publicly to users [72] such as SalesForce.com, Microsoft Dynamics CRM etc. Users have limited ability to modify the functionality of these services [72].

- **Platform as a Service (PaaS)**
  In this type of service, a development platform (including development tools, languages, frameworks and application runtime environment) is provided as service [72]. Examples of such services are Google App Engine, Microsoft Windows Azure etc. The level of customization and flexibility is more than SaaS in this type of service.

- **Infrastructure as a Service (IaaS)**
  This type of service allows users to use raw hardware, storage systems, networking etc. to implement their own computing systems [72]. An example of such service is Amazon EC 2. IaaS provides highest amount of customization and flexibility to the users.

Besides its advantages, cloud computing has many issues and problems to overcome, such as security, privacy of data, availability of service, interoperability among cloud services etc. [43, 88, 61]. Incidents such as T-Mobile Sidekick [77] and outages of different cloud services [60, 8, 89, 4] have made the enterprise community cautious of adopting the cloud [63]. These outages also show the lack of reliability and availability in most of the available cloud platforms. Security and privacy are also some of the major concerns in migration of application and data to cloud computing, for enterprise organization access to data is only restricted to limited people in organization but in cloud computing, data goes out of the boundary of organization [88, 61]. Therefore, cloud provider can access the data and the organization has no or limited control on the privacy and security of data [88, 61]. Another issue is that different cloud providers have different storage systems, programming tools and frameworks, which makes the interoperability and migration of applications to cloud platforms difficult and complex [84, 72].

Cloud computing has its advantages and disadvantages like any other computer technology but still there is a multidimensional research going on in this field such as performance comparison of cloud platforms, cost/benefit analysis, security and privacy issues, and availability and reliability issues in cloud computing. Figure 1 shows the trend of increasing interest in cloud computing technologies.
1.3 Google App Engine and Amazon Elastic Compute 2

Google App Engine is a Platform as a Service (PaaS) and it is Google’s internal infrastructure exposed as a cloud platform. It stands quite different from the rest of the cloud platforms, with many unique attributes [30, 97, 54]. It allows hosting of web applications in Google managed data centers [35, 30, 97, 18], applications are executed virtually across multiple servers and data centers [30, 18]. The architecture of GAE is very complex; it spawns over a million of servers [97, 98], which are distributed geographically across the globe. GAE is discussed in detail later on in Chapter 4.

GAE uses Bigtable, which is a distributed system for storing data [30]. Bigtable is also being used internally at Google for different products such as Google Analytics, Google Finance, Google Earth, Google Search Index etc. [17, 30]. It has been designed to provide sustained performance with scalability and availability [17]. Bigtable is discussed in detail in Chapter 4.

Amazon Elastic Compute 2 (EC2) is an Infrastructure as a Service (IaaS), which provides elastic computation services [3]. Amazon EC2 allows deployment of virtual machine images to the cloud [72]. In this way users can deploy any kind of operating system and software. It is very flexible and customizable cloud platform, users can administer and control the resources allocated to their applications [3]. The Amazon EC2 architecture is discussed in detail in Chapter 4.

Enterprise Applications (EA) are usually transactional, multi-user and distributed applications [39], which are used specifically for managing and controlling organization operation and data, and containing tools to develop applications specific for the organization [26]. They include managing human resources, supply chain, manufacturing, sales, accounting, retail etc. [26]
1.4 **Problem Definition**

Development and migration of application and data to a cloud platform is a non-trivial situation [38]. From the economic point of view, the main concern is the cost of development, migration and maintenance of application in the cloud [38]. From the technical point of view, there can be different factors, such as security and privacy of data, reliability and availability of service, availability of development tools and frameworks etc. [49, 48, 92].

In cloud computing, there is lack of standards for defining cloud services, tools, development environment and frameworks etc. [50]. Different cloud providers have implemented their cloud services and data management systems with different architectures, development tools and languages. Most of the cloud platforms are not interoperable with each other [50, 19]. These differences present a difficult situation for enterprise application developers to select a particular cloud platform for development [19, 72]. The existing platforms also have different level of abstractions [72]. Amazon EC 2 provides a lot of flexibility and customizations to the user, such as hardware, operating system and software selection and customization [3]. While others such as GAE provides a high level of abstraction, in the form of a sandboxed application execution environment, with limited tools and languages [93]. In cases of large enterprise organizations moving their data and application to cloud, require a thorough feasibility study of the target cloud platform. Availability of technical support, security, performance and reliability of the service are the main technical issues that need to be dealt before migrating data and application. In this master thesis we have focused on the development environment and performance of GAE and Amazon EC 2 platforms.

1.5 **Aims and Objectives**

The authors of this study aim to evaluate and compare the architecture and working behavior of GAE and Amazon EC 2 through a literature study and by designing a prototype application. From the development point of view, the authors evaluate the GAE platform as compared to Amazon EC2 in order to study its feasibility for enterprise application development and discuss its advantages and disadvantages.

The main objectives are:
- Understand the behavior and working of the GAE and the Amazon EC2 architecture.
- A comparison of GAE and Amazon EC2 in terms of development environment and tools.
- Evaluating and comparing the architecture, performance and behavior of the two cloud platforms, using a prototype customer relationship management (CRM) system.

1.6 **Research Questions**

The authors formalized four questions, which are answered in this master thesis.

The research questions of this study are;

**RQ1.** What are the limitations and restrictions applied when developing applications for the GAE platform?

This research question aims to understand the limitations of GAE, and which type of tools, applications and requirements cannot be implemented on GAE. This research question is answered by literature study.
RQ2. What are the differences of the GAE platform as compared to Amazon EC 2 in terms of development environment such as languages, tool support and platform (operating system and hardware) features?

This research question is answered by the literature study performed of the two cloud platforms. This research question provides useful technical information in understanding enterprise application development on these two cloud platforms.

RQ3. What are the response time, execution time, scalability, memory requirement and development time of the prototype CRM application on the GAE and Amazon EC 2 platforms?

We calculated these metrics by developing a web based CRM application prototype. This research question analyzes and compares the two platforms, and provides information in deciding which platform is better in different scenarios.

RQ4. Is GAE suitable for development of enterprise applications, based on the experiment of CRM application and literature study?

This research question is answered after the literature study and the experiment of the prototype application on the cloud platforms. The answers to this question provide guidelines and recommendations which can be helpful to software engineers in evaluating the feasibility of an application for GAE for enterprise application development.

1.7 Main Contribution

The research study has discussed the potential problems being faced by enterprise application developers in performing the feasibility and selection of the appropriate cloud platform in order to develop or move their application and data to a cloud platform. In this study, the authors evaluated the Google App Engine and compared it with Amazon Elastic Compute 2 for its feasibility for enterprise application development. The authors conducted a literature study and performed an experiment on GAE and Amazon EC2 to analyze their architecture, performance and working behavior. Based on the research study, the authors provided guidelines and recommendations for developing enterprise applications on GAE and which platform is more feasible for different kind of enterprise applications. This research work also provides the analysis of Bigtable as compared to RDBMS databases for enterprise applications.

We found out that GAE is more useful for development of non-transactional and non-intensive data applications. Performance of GAE is consistent with increase in the concurrent requests and data load. Lack of support for transactions, limitations on development tools and languages makes the development process difficult for the software developers on GAE platform.

1.8 Structure of Thesis

The thesis is organized in the following chapters;

Chapter 1: Introduction

In this chapter, the authors present the introduction to thesis, the background for different concepts used in the thesis, description of the problem area, research questions and aims and objectives of the thesis.
Chapter 2: Research Methodology
In this chapter authors discusses the research methodology of the thesis and the different validity threats.

Chapter 3: Literature Study and Related Work
In this chapter, authors discuss the literature study plan and its execution. Review of related work and their results are also described in this chapter.

Chapter 4: Literature Review of Google App Engine and Amazon Elastic Compute 2
In this chapter, a detailed explanation of the two cloud platforms is discussed. Their internal working and behavior have been explained in this chapter. This chapter provides the background information for understanding the two cloud platforms.

Chapter 5: Customer Relationship Management Application
In this chapter, the authors discuss the development of the prototype application and its features.

Chapter 6: Experiment Design
In this chapter, the experiment design and environment has been discussed. The performance metrics, parameters and benchmarks have also been discussed. Different experiment scenarios are also explained in this chapter.

Chapter 7: Experiment Results
The results from the experiment are discussed and explained in this chapter. The authors have provided graphs and charts to visualize the experiments results and provide explanation to the results in reference to the literature study which is conducted earlier.

Chapter 8: Discussion and Analysis
In this chapter, the authors provide an analysis of the experiment results and literature study. Limitations of GAE and its evaluation as enterprise application development platform have been discussed in this chapter. The authors also compared GAE and Amazon EC 2 in this chapter. Answers to all research questions are explained as well.

Chapter 9: Conclusion
In this chapter, a summary of the thesis work has been presented. Research questions are also answered shortly here as well.

Chapter 10: Future Work
In this chapter, the authors provide the possibility of future work related to this thesis.
2  RESEARCH METHODOLOGY

2.1  Overview

The authors of this study have used a mixed research method to achieve the aims and objectives of the research study. The research methodology is divided into two phases. In the first phase the authors perform a literature study of the current state of research in the research area, reviewing related work and studying the research work on related technologies (Bigtable, GFS, NoSQL etc.). The authors also used the literature study to understand the existing technologies used in cloud computing. In the second phase, the authors designed a prototype CRM application to analyze the behavior, performance and working behavior of GAE and Amazon EC2 by using different performance metrics. Figure 2 shows an overview of our research methodology.

Figure 2 Abstract view of Research Methodology
2.2 Research Questions and Methodology

Table 1 shows the research methods used to answer the different questions.

<table>
<thead>
<tr>
<th>Research Questions</th>
<th>Research Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>Literature study</td>
</tr>
<tr>
<td>RQ2</td>
<td>Literature study</td>
</tr>
<tr>
<td>RQ3</td>
<td>Prototype application experiment</td>
</tr>
<tr>
<td>RQ4</td>
<td>Literature study and prototype application experiment</td>
</tr>
</tbody>
</table>

RQ1 and RQ2 are answered by the literature study. For RQ1, the authors searched through the literature and web resources to identify limitations on developing applications for GAE. In RQ2, the differences of GAE and Amazon EC2 were studied in terms of development environment, tools, frameworks, and platform features.

RQ3 answers the performance comparison and working behavior of GAE and Amazon EC2 on the basis of experiment of prototype application. The SLOC, development time, debugging time and memory requirement of the two platforms are also explained in this question.

RQ4 is answered by both the literature study and the prototype application experiment. The authors analyzed the knowledge gathered from the literature study and the experiment results to answer this question.

The literature study is discussed in detail in Chapter 3 and experiment design for prototype application is explained in Chapter 6.

2.3 Validity Threats

The main concern about the experiment results and data gathered through literature study is that they should draw valid conclusions [94]. Therefore, validity threats should be addressed before starting the research work and should be monitored throughout the research work. The main validity threats related to our master thesis are discussed in the following sections;

**Internal Validity**

There was a shortage of literature availability on the internal architecture and design of GAE, Bigtable, GFS and Amazon EC2. The authors used official web resources, white papers, manuals, and documentations to overcome the shortage. Moreover, the research studies used in the literature study are from the last 5 years only (with the exception of 1 research paper), in order to base the thesis work on latest knowledge and research.

The authors were used to software development in environments similar to Amazon EC2 but not as much on GAE. Therefore, 1 month was spent on developing a sample social network application for GAE in order to get used to its development environment and tools. This helped in bringing the skills of the developers to the same level for both the platforms. Since only two developers worked in the development of prototype application, our results need to be verified by an experiment where a large group of developers
a. **Literature Study and Selection of Studies**
   In order to base the literature study on the most relevant research work, we formalized search strings (listed in section 3.1.1) based on our research questions. The most widely used computer related scientific publications (listed in 3.1.2) were used to find related research work. We used inclusion/exclusion criteria (section 3.1.3) to include or exclude research work.

**External Validity**
The experiment was conducted in a BTH laboratory environment. During experiment unnecessary applications and services were stopped on the client machines in order to achieve the correct results, which can be validated in other environments as well.

a. **Selection of Development Tools**
   We have used the same development tools which were available on both the platforms in order to have same difficulty level in development and debugging of the prototype application. Some tools such as Spring Security and Hibernate which were used on Amazon EC 2 but could not be used on GAE, were replaced by developing our own but similar code files.

**Conclusion Validity**
In order to minimize the chances of incorrect conclusion, the conclusion and results of the study are based on literature study, related work, experience gathered from prototype application development and experiment results.
3 LITERATURE STUDY AND RELATED WORK
In this chapter, the authors present the literature study plan and review of related work.

3.1 Literature Study Plan
A literature study plan was used to search the research material. The literature study was needed to help answer RQ1, RQ2 and RQ4. First of all, keywords were identified from the research questions, and then alternate words were identified for these keywords [51]. Alternate words are joined by Boolean OR and keywords are joined by using Boolean AND or Boolean OR to form search string [51].

3.1.1 Keywords and Search String
Following the guidelines outlined in [51], we identified keywords and alternate words from our research area and questions to form search string. The search strings are listed in table 2, which were used to search the different databases. In some cases the search strings were modified based on the searching techniques available in a particular database.

<table>
<thead>
<tr>
<th>Search String #</th>
<th>Search String</th>
<th>OR</th>
<th>Design OR Development</th>
<th>OR</th>
<th>Google App Engine OR Amazon EC2 OR Cloud Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Enterprise OR Business AND Application OR Software OR Design OR Development OR Google App Engine OR Amazon EC2 OR Cloud Platform</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Limit* OR Restrict* AND Google App Engine OR Amazon EC2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Compar* OR Performance Analysis OR Evaluat* OR Migrat* AND Cloud Platform OR Amazon EC2 OR Google App Engine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Compar* OR (Pros OR Cons) OR (Advantages OR Disadvantages) AND RDBMS OR Relational database OR NoSQL OR Bigtable OR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
* It is a wild card. It is used to search out unknown number of characters following the words.

3.1.2 Data Sources
Kitchenham et al. [51, 14] has mentioned the main sources of literature for software engineering, which are listed below. Google web search was also used to find literature missing from the identified sources.

1. IEEE Explore
2. ACM Digital Library
3. Springer Link
4. Science Direct
5. Inspec (Ei Village 2)
6. SCOPUS
7. CiteseerX
8. Google Scholar
9. Google web search

3.1.3 Criteria for Inclusion and Exclusion
The authors defined criteria for excluding and including the search results related to the research work on the bases of relevance. Any search result containing research work based on the following criteria were included and the rest were excluded;

1. Articles related to performance analysis or working behavior or comparison of GAE or Amazon EC 2 or other cloud platform
2. Cloud data storage comparison or analysis or working behavior (Such as Bigtable)
3. RDBMS, NoSQL, Bigtable advantages or disadvantages.
4. Case studies related to migration to cloud platform
5. Results published in the last 5 years (since 2007). Older studies were included only if there was no newer study available.

Document Type = Books, conference papers, journal papers, white papers.
Language = English

We applied these criteria by studying the titles, abstract and in some case the conclusion to decide about the relevancy of literature.

3.1.4 Quality Assessment Criteria
In order to check the quality of selected articles, we defined quality assessment criteria following Kitchenham guidelines [51]. Different criteria’s were identified for quality assessment and were ranked on a scale from 1 to 5 (where 1 is poor, 2 is fair, 3 is good, 4 is very good and 5 is best quality) by both the authors.
### Bibliography Management

Zotero was used for reference management while Mendeley was used for managing duplicates.

### Search Results

Based on the identified search strings, we searched the 9 identified databases and search engines. After applying the inclusion/exclusion criteria, a total of 35 studies were selected. Table 3 shows the total results found in each database/search engine.

#### Table 3 Search Results

<table>
<thead>
<tr>
<th>Database/Search engine</th>
<th>Number of Raw Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE</td>
<td>3454</td>
</tr>
<tr>
<td>ACM</td>
<td>1018</td>
</tr>
<tr>
<td>SpringerLink</td>
<td>247</td>
</tr>
<tr>
<td>ScienceDirect</td>
<td>2291</td>
</tr>
<tr>
<td>Inspec</td>
<td>885</td>
</tr>
<tr>
<td>Scopus</td>
<td>1224</td>
</tr>
<tr>
<td>CiteseerX</td>
<td>180</td>
</tr>
<tr>
<td>Google Scholar</td>
<td>888</td>
</tr>
</tbody>
</table>

Primary studies are provided in Appendix B. Search results did not yield enough information about the internal architecture and working behavior of GAE and Amazon EC2. Therefore, we also used Google web search to search other internet resources using our main keywords. The selected internet resources are also provided in Appendix B Internet Resources section.
Figure 3 shows the number of studies selected from the different databases and search engines. 29 out of 35 studies are selected from IEEE and ACM, which are the most widely used publishers for research work related to the fields of computing.

![Number of Studies selected from different Sources](image)

Figure 3 Number of Studies selected from different Sources

Figure 4 shows the number of studies selected from each year. 1 study is included from the year 2003 which is an exception to our inclusion criteria. But it is the only research work available which explains the Google file system (GFS). 33 studies are from the last 4 years. Therefore most of the literature found, contains the latest research work in the field of cloud computing.

![Selection of studies and publication years](image)

Figure 4 Selection of studies and publication years
Figure 5 shows the selected studies based on document type. Most of the studies are conference papers, but there are some journal articles and books included as well.

![Figure 5 Selected Studies and Document Types](image)

3.1.7 Data Extraction Strategy

We used Kitchenham guidelines to formulate our data extraction strategy. Data extraction forms were used to gather data from the selected primary studies. The contents of our data extractions form are listed in Table 4.

<table>
<thead>
<tr>
<th>Table 4 Data Extraction Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of Reviewer:</td>
</tr>
<tr>
<td>Data of Data Extraction:</td>
</tr>
<tr>
<td>Study Title:</td>
</tr>
<tr>
<td>Study Authors:</td>
</tr>
<tr>
<td>Publication Details:</td>
</tr>
<tr>
<td>Document Type: Journal/Book/Conference Paper/Report etc.</td>
</tr>
<tr>
<td>Reference</td>
</tr>
<tr>
<td>Relevancy to problem area</td>
</tr>
<tr>
<td>Abstract</td>
</tr>
<tr>
<td>Description of Research methodology</td>
</tr>
<tr>
<td>Summary of Results</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>
3.1.8 Data Synthesis

Data gathered in literature review is discussed in Section 3.2 Related Work and Chapter 4. Section 3.2 provides descriptive synthesis of gathered data from different research work while chapter 4 explains the architecture of GAE and Amazon EC 2. Gathered data from literature study is also used to answer RQ 1 and RQ 2 in Chapter 8. The gathered data consists of both quantitative and qualitative results. Our focus is on the conclusion of these studies in order to understand the working behavior of GAE and Amazon EC 2 and draw conclusions from our experiment in comparison with this data.

3.2 Review of Major Related Work

In this section we summarize the related work into different categories. Most of the related work is around scientific computing on cloud platforms, cloud technologies comparison, migration/deployment of application on cloud platforms and evaluation of open source cloud technologies.

3.2.1 High Performance Scientific and Cloud Computing

Walker [91] used macro and micro benchmarks to study the performance of Amazon EC 2. The authors compared the performance of Amazon EC 2 instances with high-performance computing clusters, which were used by scientists at National Center for Supercomputing Applications (NCSE) [91]. NASA Advanced Supercomputing (NAS) parallel benchmarks (NPB) (based on Message Passing Interface (MPI) and Open Multithreading Processing (OpenMP)) were used to measure the performance [91]. Results of this study showed that Amazon EC2 performance degraded by 7-21% as compared to NCSE cluster by using NPB-OpenMP. Similarly using NPB-MPI, Amazon EC 2 performance degraded by 40 – 1000 % [91]. Hill et al. [42] also performed Message Passing Interface (MPI) benchmarks on Amazon EC 2 in order to compare its performance with small size scientific clusters. The experiment is conducted on a local high performance cluster of University of Virginia and different instance types of Amazon EC 2 [42]. The results in general are similar to Walker, Amazon EC 2 is not suitable for high performance computing due to its high latency for frequent small amount of communication among nodes, and the virtualization technology degrades the actual performance of physical resources [42]. Ostermann et al. [65] also evaluated Amazon EC 2 platform for scientific computing. Their study used micro benchmarks, kernels and e-Science workloads to measure the performance of Amazon EC 2 and alternative scientific grids available to scientists. The study showed that Amazon EC 2 performance was inferior and unreliable in terms of scientific computing but is a viable alternative to meet short deadlines and for small scientific experiments [65]. Stantchev [81] performed an experiment to evaluate the nonfunctional properties (NFP) of particular instances of Amazon EC 2. The author used WTest benchmark (web services benchmark) to evaluate the performance of NFP. Two NFP metrics, transaction rate and response times were used to measure the performance of NFP of Amazon EC 2 instances. The results showed that performance improves by 65% with each increase in number of instances in case of transaction rate while there is an increase of 100% in the case of response time [81].

Vecchiola et al. [90] performed a study of effectiveness of cloud computing (Amazon EC 2) in terms of cost and performance for the scientific community. Their study showed a considerable amount of reduction in cost and improved usage of resources for scientific computing [90]. Hazelhurst [40] also performed a case study of Amazon EC 2 for its cost
effectiveness in terms of scientific experiments. His findings also support usage of cloud computing in scientific computing from the view point of cost reduction [40].

3.2.2 Open Source Cloud technologies

You et al. [96] worked on Hadoop (open source implementation of GFS and MapReduce) in order to develop a distributed and high performance data ware house (HDW) for online analytical processing (OLAP). The HDW was implemented on 18 nodes with 36 cores. The authors did not provide any performance metrics but only provided the implementation details [96]. Carstoiu et al. [16] also performed an experiment on Hadoop-based HBase database. The authors focused on random writes and reads performance in their experiment. The results showed that random write performance was better than random reads, because for random reads there was more probability of reading the data from disk as compared to memory, while in case of writes the HBase (similar to Bigtable) uses in-memory storage of data and later on moves the data to disk [16].

3.2.3 Cloud Platform Comparison

Li et al. [54] provides a comparison of 4 different cloud platforms (Amazon Web Services, GAE, Microsoft Windows Azure, Rackspace Cloud servers) by measuring the different properties of cloud platforms such as computation, network and storage performance. Their study shows that Amazon EC 2 has highest intra-cloud bandwidth, Windows Azure has most powerful virtual machine and GAE has lowest network latency [54].

3.2.4 Cloud Data Storage Technologies

Cooper et al. [19] at Yahoo, developed a benchmarking tool Yahoo! Cloud Serving Benchmarks (YCSB) to measure the performance of different cloud based storage systems, such as Cassandra (based on Google Bigtable), HBase (based on Google Bigtable), Yahoo!’s PNUTS and a sharded implementation of MySQL. The results showed that Cassandra and HBase have higher read latency for large stream of read operation as compared to PNUTS and MySQL while latency of Cassnadra and HBase decreased in terms of update operations. PNUTS and Cassnadra scaled well while HBase had an erratic behavior during scalability [19]. Shi et al. [76] have performed a thorough study of different data management systems used in cloud systems such as Hive, HBase, Cassandra, HadoopDB. The authors performed many different set of experiments on these data management systems and compared their performances [76]. In terms of data loading Hive took least time. In range query and grep operation MapReduce based data management systems (Hive, HBase and HadoopDB) performed better than others. In aggregation operation only Hive and HadoopDB were evaluated, the rest does not support this operation. Hive performed better in this operation [76]. In fault tolerance, MapReduce based data management systems again showed better performance in fault tolerance than others [76]. Bunch et al. [15] developed an API based on Google App Engine data store API in order to provide a single API to communicate with different distributed database technologies, such as MongoDB, MemcacheDB, HBase, Hypertable and Cassandra. The authors combined their API with AppScale cloud platform (an opensource implementation of Google Appengine) to evaluate the performance of different distributed database technologies [15]. The results show that Master-Slave databases (MySQL, HBase, Hypertable, MemcacheDB, MongoDB) have better response time in general as compared to peer-to-peer databases (Cassandra, Voldemort). In query response time MySQL performs better than other databases, while in get and put operations, the response
time of Cassandra was better than other databases [15]. The authors also performed this experiment on Google Appengine with Bigtable and found the result similar to that of MySQL/AppScale but the author notes that GAE showed higher variance of access time [15].

Leavitt [53] performs a deep study of NoSQL databases, with a focus on its advantages and disadvantages. The author explains that NoSQL performs better than RDBMS in terms of speed, since there is no ACID and consistency mechanism [53]. NoSQL has better speed and scalability than RDBMS but the lack of consistency and integrity mechanism makes application and data management more complex [53].

3.2.5 Migration and Deployment on Cloud Platforms

Fu et al. [27] showed deployment of an enterprise application on Amazon EC 2 to demonstrate a reliable, highly available and secure Amazon EC 2 based configuration for enterprise applications. The authors used Amazon Machine Images (pre-configured installation of operating system) to deploy the virtual machine in Amazon EC 2 [27]. Two server instances with heartbeat configuration and a database server with Amazon Elastic Block storage were deployed. One server was the primary server while the other server was backup server; both were connected by heartbeat mechanism [27]. The configuration used in this research study was limited to two servers but the basic principal can be applied to any number of servers in Amazon EC 2 [27]. Marshall et al. [57] has proposed architecture for elastic provisioning of resources in cloud systems. The authors used Nimbus to build their own resource manager and evaluate its performance on Nimbus based cloud at University of Chicago and Amazon EC 2. The architecture was implemented on Linux clusters, it contains an elastic site manager which monitors jobs in the job queue and allocates resources based on three policies such as on-demand, steady stream and bursts [57]. The experiment demonstrates the reactiveness of the resource manager to different policies on the Amazon EC 2 and University of Chicago cloud [57].

Li [55] worked on RDBMS and NoSQL databases in order to formalize a method for transforming RDBMS into HBase (NoSQL) to help in migrating application data to cloud databases. The author proposed 3 rules for the transformation; [55]

a) Grouping of co-related data
b) Adding foreign key references to each side of relationship
c) Merging of attached tables

Khajeh-Hosseini et al. [50] conducted a case study of an enterprise organization by moving their IT infrastructure to Amazon EC2 from financial and technical perspectives. Their study shows that in case of Amazon EC 2 the organization can reduce infrastructure cost by 37% in the next 5 years and support calls cost can be reduced by 21% [50]. Misra et al. [62] presented a suitability index which analyzes the characteristics of a company business and identifies its favorability in terms of migration to cloud architecture. Their study focuses on financial and managerial aspects of migration to cloud computing [62].
4 REVIEW OF GOOGLE APP ENGINE AND AMAZON CLOUD

4.1 Google App Engine

Google App Engine is a platform for developing and deploying web-based applications in Google infrastructure [75]. GAE is free for use up to a certain level of usage, after this it charges users [13, 70]. Developers can use Python and Java as the two primary development languages, although the availability of Java virtual machine (JVM) allows the use of all JVM based languages on GAE [33]. Java support is not fully available and some of the features are omitted because of security and the web-based nature of the platform [85, 45]. Some of Python modules written in C language are also disabled due to security concern [33, 34].

4.1.1 NoSQL

NoSQL refers to non-relational based databases, these type of databases do not support full relational model, structure query language and lacks consistency [11]. These type of databases focuses on performance and scalability but as a tradeoff lack consistency and integrity [11, 82]. Relational database performance decreases with increase in data, and when data size reaches to gigabytes and terabytes, it’s difficult to manage such large size of data; many websites which uses large volume of data faced these problems, such as Facebook and YouTube [22]. Facebook moved its database to Cassandra [53, 22] and YouTube started using Bigtable [22] as its storage mechanism, both of which are NoSQL databases. There are many other NoSQL databases available such as Bigtable, Amazon SimpleDB, Hive, HBase, MongoDB, CouchDB, Cassandra etc.

4.1.2 Google File system

It is a scalable, distributed and data intensive file system, like all traditional distributed file system it provides reliability, availability, fault tolerance, scalability and performance [28]. It was designed by Google engineers to meet the data storage needs of their applications. There are many open source implementations of GFS, most notably Hadoop Distributed File System (HDFS) which is used by Yahoo [22]. GFS design decisions were based on taking many factors into considerations, such as; [28]

- Component failures are common in large distributed systems
- Traditionally files are huge and GB size files are common.
- Large files are appended mostly rather than modifying the existing data.
- Co-designing of application and file system API is more beneficial.

GFS design has the following features; [28]

- It is composed of many inexpensive commodity servers.
- It commonly stores file with sizes in GB.
- It is designed for large streaming reads or small random reads and large stream of writes (usually appending).
- It allows concurrent appending of files by multiple clients while guaranteeing atomicity.
- It maintains high bandwidth for data transfer.

A GFS system is composed of a single master server and many chunk servers. Files are composed of many chunks and chunks are stored on multiple chunk servers in order to provide
reliability and high availability [28]. Master maintains metadata information about all files. Master and chunk servers communicate continuously using heartbeats [28].

Single server makes the design simple but allows for bottle neck when traffic load is high. To avoid this problem, client reads and writes data directly through chunk server [28]. For the first time, client asks the master about locations of chunk servers and after that for the subsequent requests client communicates directly with the chunk servers for some period of time [28]. All metadata information is stored by the master and is served from the memory to provide high performance. GFS uses log for backup and recovery, the log is replicated on many remote servers for reliability and availability [28].

GFS divides data into chunks which are 64 MB in size, and it’s the GFS block size. This size is very large as compared to other file system, but it has many advantages [28]. GFS was designed to support large streams of read and write, thus having large block size allows the client to get most of the data from the same chunk server and even the same block [28]. Large block size also allows a smaller metadata, which enables the GFS to store all metadata in memory for high performance [28].

4.1.3 BigTable

It is a distributed storage system for managing structured data spanning across thousands of systems [17, 87]. It was developed to support scalability, high availability and reliability while still maintaining optimum performance [17, 22]. Bigtable is trade-off between full support for relational data model and scalability [17], it provides simple data model with dynamic control over format and layout of data [17]. There has been a widespread use of Bigtable inside Google and also outside the Google world in the form of open source implementations of Bigtable [22], such as HBase [44], which is an open source implementation of Google Bigtable [22].

A set of processes runs the Bigtable software, which is called Bigtable cluster. In Bigtable data is stored in tables like relational databases but these tables are multidimensional sorted map, organized as row, column and timestamp and are distributed [17]. The data referenced by the row, column and timestamp is called cell. Consecutive rows with similar row keys are stored together known as tablets while similar columns keys are grouped together known as column families [17]. The third dimension, timestamp allows storage of multiple versions of data.

Bigtable implementation contains multiple components. It contains a single master server and many tablets servers. A master server allocates tablets to different tablet servers and a single tablet server can serve many tablets [17]. Communication between tablet server and client is direct and master does not take any role in it, therefore avoiding the single master server from over loading [17].

MetaData tablets are used to store the locations of all the tablets while root tablets store the locations of metadata tablets [17]. A chubby file stores the location of root tablet. Chubby is also used by master server to keep track of all tablet servers [17]. Bigtable also stores the table schema in chubby. Distributed lock mechanism is provided by the chubby. Chubby contains 5 replicas but only one is active and is used by the master [17].

Tablet servers serve the tablets to clients and store the tablet data in GFS. All changes in the tablets are stored in logs with redo records. Latest changes in the tablets are stored in memtable which is sorted buffer in memory while the older updates are written to SSTables which are written to GFS [17].

Bigtable and similar implementations such as HBase are being used in many projects such as Youtube, FaceBook and Flicker, which have seen millions of records in their RDBMS.
per day and it became difficult to maintain and to provide optimum performance to users [22, 53].

4.2 Amazon Cloud

4.2.1 Amazon Elastic Compute Cloud

Amazon web services are a suite of cloud based services allowing data storage, networking, content delivery, database and computation services [3]. Amazon EC2 is the Amazon elastic compute cloud service, which is an infrastructure as a service (IaaS). EC2 is a virtual computing environment much like a traditional computing system; users can control the processing power, primary memory, networking, operating system, application software etc [3].

Prominent features of Amazon EC2; [3]

- Amazon EC2 allows full configuration of the infrastructure by the user, thus allowing users to use many types of programming tools, languages and applications [3].
- EC2 provides persistent storage through Amazon elastic block storage; users can create volumes from 1 GB to 1 TB [3].
- It allows the user to place computation instances in different locations [3].
- It provides elastic IP addresses which maps to user’s instances and remaps in case of failure to other instances [3].
- It provides virtual private network (VPN) between amazon EC2 and an organization existing IT infrastructure, called as Amazon virtual private cloud [6].
- Amazon EC2 allows adding or removing instance according to criteria’s defined by the user, known as auto scaling [10].
- Elastic load balancing automatically distributes the load among all instances [25].
- Amazon EC2 allows usage of machine images, which are pre-configured installation of operating system and application software. This feature allows easy setup of Amazon EC2 instances and user can easily migrate their current infrastructure to the EC2 [3].

4.2.2 Amazon EC2 Instances

Amazon offers the user different computation instances on Amazon EC2 [3]. It offers virtual processing unites known as EC2 compute unit [3]. A EC2 compute unit provides equivalent to 1.0 – 1.2 Ghz 2007 Opteron or 2007 Xeon processor, these instances have memory ranges from 613 MB up to 68.4 GB [3]. EC2 also offers GPU instances which are based on NVIDIA Tesla “Fermi” M2050 and Intel Xeon X5570, quad-core “Nehalem” processors [3]. Users can use many different combinations of these instances to meet their requirements. All major operating systems are available to users for installation on these instances in the form of Amazon machine images (AMI). Amazon have pre-configured AMI containing almost all Windows and Linux versions along with most popular enterprise applications such as Oracle database, IBM DB2, Microsoft SQL Server etc. [3].

4.2.3 Amazon EC2 Storage

Amazon EC2 does not provide persistent storage, so when an instance is terminated; all of its data is lost [2]. To add persistent storage to Amazon EC2 instances, there is another service known as Amazon Elastic Block store (EBS) [2]. EBS is a raw storage service and user can format it with any file system [2]. EBS is automatically replicated on to Amazon Simple
Storage Service (S3) [5], which is highly available, scalable, distributed and reliable storage system based on Amazon Dynamo [24].

4.2.4 Amazon Dynamo

Amazon storage system is based upon a highly available, reliable, scalable and distributed storage system, known as Dynamo [24]. Dynamo powers all the web service of Amazon.com including Simple Storage Service (S3) [24]. Dynamo provides ACID properties while keeping high availability [24].

Dynamo is a distributed key-value store, a decentralized storage system comprising of thousands of nodes [24]. The data is stored across all nodes and a consistent hashing mechanism is used to partition data incrementally across these nodes [24]. All nodes communicate a routing table with each other which allows them to know the data hosted on each node. Data is replicated across many nodes to provide high availability and reliability [24]. Dynamo provides versioning of data, this feature ensures that data is never lost [24].

In dynamo, each storage node uses a persistence storage unit. There are three persistence units in use i.e. MySQL, BDB Java Edition and Berkeley Database Transactional data store [24]. An application can choose the type of local persistence engine based on its requirements [24].

4.3 Pricing

Amazon EC 2 and GAE provide users with free usage up to a specified usage levels and after that it charges user on the amount of service consumed [3, 13, 70]. GAE provides users with free service throughout the life time of the user account while Amazon EC 2 provide only for the first year [13]. Both Amazon EC 2 and GAE calculate computation usages in terms of hours and data storage in terms of GB [3, 13].
5 PROTOTYPE APPLICATION

In this chapter the authors explain the details of developing the prototype application and its purpose, which is a customer relationship management (CRM) application.

5.1 Purpose of Prototype Application

The purpose of developing a prototype application on GAE and Amazon EC2 was to explore the difficulties and complexities in designing and developing for the two platforms, and use it in the experiment to measure the different behavior and performance metrics. Developing the prototype application also helped in understanding the features and requirements which can or cannot be implemented. It also helped in developing the solutions to the different restrictions imposed by GAE.

5.2 Development Tools

Windows Server 2008 (WS) was selected as the development and deployment environment for Amazon EC2. WS is the most widely used server in enterprise [12], therefore it was selected as the operating system on Amazon EC 2.

Microsoft .Net languages and Java is the most widely used programming language for enterprise application development [79]. .Net languages are not supported on GAE; therefore we selected Java as the development language for the prototype application. Netbeans and Eclipse are two commonly used and freely available development tools for Java. GAE provides an official development plugin and support for Eclipse; therefore we selected Eclipse as the development tool for the two platforms.

GAE provides only usage of Jetty Web Server for development and deployment, so it was used as the only option. While Apache Webserver with Tomcat was used on Amazon EC 2 because it’s the most widely used webserver [9] and it supports all the features and libraries of Java language [68]. MySQL community edition was selected as the database server on Amazon EC 2 since it is available for free and it has limited resource consumption.

In the development of prototype application, we used Java frameworks and libraries such as Apache Struts [7], SpringSecurity [80] and Redhat Hibernate [41]. Hibernate is commonly used to manage the data from database in Java applications. Hibernate is compatible with SpringSecurity and Struts. Spring Security was used for managing the user authentication and management while Struts framework was used to manage the business logic and flow of execution. Hibernate and SpringSecurity does not support Bigtable, therefore these two frameworks were not used in the prototype application for GAE [45].

Development tools used on GAE

1. Java Language
2. Eclipse Helios IDE
3. Google App Engine SDK
4. Java Development Kit (JDK) v6
5. Apache Struts v2.1.6
6. Jetty Web Server
Development tools on Amazon EC2
1. Java Language
2. Eclipse Helios IDE
3. Java Development Kit (JDK) v6
4. Apace Struts v2.1.6
5. SpringSecurity v2
6. Redhat Hibernate 3.2.6
7. Apache Webserver with Tomcat v6
8. MySQL v5

5.3 Prototype Application Architecture
The prototype CRM application is a web based application. We have designed the prototype application for both platforms using the Model-View-Controller (MVC) architecture style. MVC is the most widely used architecture style for Java based web applications [67]. Figure 6 shows the abstract view of the prototype application. MVC architecture provides a modular approach to designing and developing web-based applications [67]. The class diagrams for the model are provided in the Appendix B.

SpringSecurity is used to authenticate users, which can directly access the model layer to look up username/passwords in the database. In case of GAE, SpringSecurity is not supported. Therefore we used our own code (custom designed classes) to authenticate users. The Struts framework provides the basic skeleton for our MVC style. It provides the controller which handles all user requests and fetches the data from model. Struts also render the fetched data from the model to the view. View of the application for both platforms is designed using HTML and JSP. Model layer is designed using Hibernate. Hibernate uses MySQL to stores data. In case of GAE, Hibernate is not supported. So we have used Plain Old Java Objects (POJO) for storage of data and Java Data Objects (JDO) for managing data in Bigtable.
5.4 Prototype Application Features

In developing the prototype CRM application, requirements were gathered from existing CRM applications, such as Salesforce.com [21] and Microsoft Dynamics CRM [59]. We selected the most common set of features in the existing CRM applications. The features implemented in the prototype application are:

- Management of Users and Roles

  There are two set of roles defined in the prototype as Administrator and User. Administrator creates user account and assigns roles to them. Each role can be assigned
set of authorities, which allows the user to perform different operations. This feature is used to measure the response time of the two platforms in the experiment.

- Issue Tracker
  This feature allows user to start a ticket about an issue or ask questions from sales staff or administrator. Administrator/moderators answer the ticket by writing comments.

- Sales Management
  In this feature, sales of products and services are managed. Sales staff records sales figures, revenue, profit and no. of unit’s sales in this section. In experiment, this feature is used to measure the small write-stream performance of the two cloud platforms.

- Management of business opportunities
  This feature allows managing business opportunities such as offering discounts, discussing ideas for new products or services etc. This feature is used in experiment to measure the execution time in general of the two platforms.

- Management of business events
  This feature provides the functionality of managing business events such as special sales events, product display etc.

- Contacts Management
  In this feature, administrators manage contact lists for sales staff and customers.

- Searching
  Searching facility is provided to search contacts in the database. This feature is used to increase the number of contacts in the database and perform searching on the records. In experiment, the effect of increase in number of data records on response and execution time is measured.

- Upload/View documents
  This functionality is provided to upload and view documents. From the experiment point of view, this functionality is used to measure the file read (large stream) and write time (large stream) of the two cloud platforms.

Development was done by both the authors and waterfall development life cycle was used. During development, time was recorded, which was spent on each development activity during the development process. After development, the application was tested and the debugging time was also recorded.

### 5.5 Client Application and Scripts

A client application in C# was developed for measuring the response time during experiment. This application sends data to the Prototype CRM application in the form of HTTP GET and POST method and records the response time. A separate set of scripts was written to measure the execution time during the different experiment scenarios.

Figure 7 shows the client application used to generate workload. This application contains the URL of the web pages each related to specific test case scenario. Number of concurrent requests is also specified through this application.
Figure 7 Client Application for invoking web requests

The workload is generated through the client application by invoking the web requests as shown in the following code snippet (C#). This code snippet measures the response time.

```csharp
StringBuilder sb = new StringBuilder();
byte[] buf = new byte[8192];

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
request.Method = "POST";
request.ContentType = "application/x-www-form-urlencoded";

ASCIIEncoding encoding = new ASCIIEncoding();
byte[] data = encoding.GetBytes(parameters);
request.ContentLength = data.Length;

Stream newStream = request.GetRequestStream();
DateTime starttime = DateTime.Now;
newStream.Write(data, 0, data.Length);//sending parameters
newStream.Close();
```
HttpWebResponse response = (HttpWebResponse) request.GetResponse();
Stream resStream = response.GetResponseStream();
if(resStream.StatusCode == HttpStatusCode.OK)
{
    int count = 0;
    count = resStream.Read(buf, 0, 1);//read first byte only
    DateTime endtime = DateTime.Now;
    TimeSpan ts = endtime.Subtract(starttime);
}

The execution time is measured on the server side. The following code snippet is of a Java Filter which is executed for all web requests that are invoked. It measures the execution time.

try
{
    PersistenceManager manager = getManager();
    Date starttime = new Date();
    chain.doFilter(req, res);
    Date stoptime = new Date();
    long executiontime = stoptime.getTime()-starttime.getTime();
    Stats stat = new Stats();
    stat.setOperation(req.getParameter("operation_id"));
    stat.setTime(executiontime);
    manager.makePersistent(stat);
} finally
{
    if (manager != null && !manager.isClosed())
    {
        manager.flush();
        manager.close();
    }
}
6 EXPERIMENT DESIGN
In this chapter, the authors present the experiment design and environment. Two experiment environments were setup for the experiment.

6.1 Developers Background
Developer 1: Kashif Khan
Developer 1 has done Bachelor of Science in Information Technology and is studying Master of Science in Software Engineering. He has experience of programming both desktop, web and mobile based application in different tools and languages. The developer has developed applications for small and medium size organizations and mostly in group 2 or 3 programmers. He has been programming for the last 7 years.

Developer 1 has experience in the following tools;
1. Java, C#, C++, PHP
2. Eclipse, Netbeans, Visual Studio, QT
3. Windows and Linux platforms
4. Databases: Oracle, MS SQL Server, MySQL

Developer 2: Asar Jan
Developer 2 has done Bachelor in Information Technology and is studying Mast of science in Software Engineering. He has experience of developing web based applications in Java and PHP languages. He has been programming for the last 3 years.

Developer 1 has experience in the following tools;
1. Java, PHP
2. Eclipse
3. Windows platform
4. Databases: MySQL and MS Access

Before the start of prototype application development, both the developers created a social-network based web application on GAE platform. This improved their skills on GAE platform, and brought their skills at the same level for GAE and Amazon EC 2 platform.

6.2 Experiment Environment
The experiment was conducted on two different platforms i.e. Google App Engine and Amazon EC 2.

Google App Engine
The GAE platform is already setup with the following built-in configuration;
1. Java 6 Runtime
2. Jetty web server
3. Google Bigtable storage (accessed through Java Data Objects (JDO))
There was no configuration performed on GAE, since it configures and allocates resources automatically.
Amazon EC2
In Amazon EC2 configuration, 2 servers were used, 1 Large Amazon EC 2 server each for the web server and the MySQL database server.

Webserver Hardware Configuration
- Amazon Standard Extra Large Instance
- 8 EC2 Compute Unit 64bit
- 15 GB memory

Webserver Software
- Windows Server 2008 64bit
- Java 6 Runtime
- Apache Tomcat v6

Database Server Hardware Configuration
- Amazon Standard Extra Large Instance
- 8 EC2 Compute Unit 64bit
- 15 GB memory

Database Server Software Configuration
- Windows Server 2008 64bit
- MySQL v5

6.3 Experiment Metrics (Dependent Variables)
The experiment was conducted to gather data for the following metrics.

6.3.1 Response Time
The response time measures the time it takes for the application to respond to the user request. This time is calculated from the start of web request to the start of web response (when the first byte of server response reaches the client) from the server. Separate client application was used to record this metric.

6.3.2 Execution Time
The execution metric measures the time it takes the application to complete the user request. It is calculated as time elapsed since start of execution to end of execution of request. A Java Filter was used to record start time and end time of the web request during execution to measure this metric.

6.3.3 Scalability
The scalability metric measures the effect on response time, execution time and memory requirement with the increase in volume of stored and processed data. Response and execution time are measured the same way as described above.

Average, maximum and minimum memory requirements were measured for both platforms. In case of memory requirements, it is calculated by running the application and monitoring the memory usage of the application on Amazon EC 2. We used JavaMelody [46] web based tool to measure the memory requirement for Amazon EC 2. JavaMelody measures the memory consumption of the incoming requests, Apache Tomcat process and MySQL process. While in case of GAE, an administrative interface provides the statistics about memory consumption of the application.
6.3.4 Development and Debugging Time
Time spent on the development and debugging of prototype applications on the two different platforms was also measured.

6.3.5 Number of Source Lines of Code (SLOC)
LOC was measured for the prototype applications for each platform. Only logical lines of code were measured. User interface code was not included in SLOC.

6.3.6 Function Points with Effort
Function points (FP) of the prototype application were measured; prototype requirements were used to calculate the FP. The development time for each module was also recorded.

6.4 Parameters and Benchmarks (Independent Variables)
In the experiment we used three parameters:
- Concurrent Requests
- File Size
- Data Size

The experiment was conducted with 10 client computers. Each computer was installed with client application to invoke concurrent requests. The parameter values for Data Size were based on studying the data entry forms in two commercially available CRM applications i.e. Salesforce.com [21] and Microsoft Dynamics CRM [59].

The parameter value for File Size was restricted to a maximum of 1 MB since GAE does not support more than 1 MB file size.

The parameter values for Concurrent Requests were based on the number of users visiting YouTube per second. YouTube receives 500 visitors per second [95]; therefore the concurrent request in our experiment was ranged from 10 to 500, but in some experiment scenarios a small number of concurrent requests were used in order to avoid overloading of the server.

6.5 Experiment Scenarios
In this section, different experiment scenarios are explained. A total of 6 different scenarios are used in the experiment.

6.5.1 Response Time Scenario
In this scenario, User login operation was performed. In the experiment, username and password were passed on to the application for user authentication. There were 1000 user records in the database. These records were searched to find a match. After successful match, an object of Session class was created. The time is noted, which is spent in passing of the user credentials till the first byte is received by the user. The total number of concurrent requests used in this scenario ranges from 10 to 500. Average, maximum and minimum response time was noted during this scenario for different number of concurrent request generated. Object size for both User and Session Class is around 1KB.

6.5.2 File Upload Scenario
In this scenario, photos (JPEG) are uploaded to the application, in order to measure the performance in terms of large stream of write data. The file size of the photos range from
100KB to 1MB (1MB is the upper limit for uploading files in GAE). The concurrent requests used were 10. Average, maximum and minimum upload time were noted for the different file sizes.

6.5.3 File Read Scenario
In this scenario, photos (JPEG) are read from the database/data store to the application, in order to measure the performance in terms of large stream of reading data. The file size of the photos range from 100KB to 1MB. The concurrent requests used were 10. Average, maximum and minimum file read time were noted for the different file sizes.

6.5.4 Small Write-Stream Scenario
In this scenario, a small stream of data is generated by creating an object of Sales class. An object of Sales class is 10KB in size. The concurrent requests ranged from 10 to 500. Average, maximum and minimum write time were noted for different number of concurrent requests generated.

6.5.5 Execution Time Scenario
In this scenario, an object of Opportunity class is updated. First of all, an Opportunity class object is read randomly from the database and then its values are changed and stored again in the database. An object of Opportunity class is 3KB. Concurrent requests ranged from 10 to 500. Average, maximum and minimum execution times were noted for the different number of concurrent requests generated.

6.5.6 Scalability Scenario
In this scenario, the performance of the prototype application is measured by increasing the number of records in database. An object of Contact class is searched in database. Concurrent requests are 100 in this scenario. The records in database are increased from 1,000 to 10,000. In this scenario, execution and response time are recorded for the different number of records in database.

Class diagrams and source code for the classes used in the experiment are given in Appendix C and D respectively.
7 EXPERIMENT RESULTS

In this chapter, the authors present the experiment results for all the experiment scenarios which were discussed in experiment design chapter.

7.1 Experiment Scenario Results

7.1.1 Response Time Scenario

Prototype CRM application on both the platforms were accessed through client application to record response time by using concurrent threads. 8 different variations of concurrent requests were generated and their response times were recorded.

Figure 8 show the average response time of the prototype application on the two platforms. In the figure 8, Google App Engine response time decreases with the increase of concurrent requests while it’s the vice versa in case of Amazon EC 2. At 10 concurrent requests GAE average response time is 582ms while Amazon EC 2 is at 479ms but at 500ms concurrent requests the GAE average response time decreases to 491ms while Amazon EC 2 average response time increases to 584ms.

GAE uses automatic resource allocation, therefore with the increase of workload, it allocates more resource to the application and GAE also has lower network latency [54]. Therefore, in this experiment, with the increase of concurrent requests from 50 to onwards, a decrease of considerable amount of response time is visible. While in case of Amazon EC 2, for less than 50 concurrent requests, the response time is better than GAE, but with increase in concurrent requests of more than 50, the Amazon EC 2 response time starts increasing. Amazon
EC 2 requires a manual configuration of resources [3], which can then be utilized by it in case of increasing workload.

![Figure 9 Maximum and Minimum Response Time](image)

Figure 9 shows the maximum and minimum response time of the prototype application on the two platforms. GAE maximum response time is 997ms while for Amazon EC 2 it is 956ms. GAE minimum response time is 8ms while for Amazon EC 2 it is 96ms. GAE shows high variance in maximum and minimum response time as compared to Amazon EC 2.
7.1.2 File Upload Scenario

Figure 10 shows the average file upload time. GAE performance remained consistent during the experiment of uploading the files with size ranging from 100KB to 1MB (GAE does not allow uploading of file larger than 1MB size, therefore the upload file size was restricted to 1MB in the experiment). In case of Amazon EC2 the performance degraded after the file size increased more than 300 KB, and with each subsequent increase in file size, the upload time increased as well. Amazon EC2 upload time for 100 KB of file size is 311ms but it reaches to 429ms for 1MB file size while for GAE it remains around 423ms to 436ms.

GAE file upload performance decrease after file size is greater than 300 KB and for Amazon EC2 it starts increasing after 300 KB. Bigtable uses in-memory writing of data into memtable, when it sizes reaches to a threshold then it is written to GFS [17], therefore in this experiment, most of the write operations were performed in memory only. While in Amazon EC2, MySQL stores every uploaded file to the disk [64], therefore its performance starts decreasing with increase of file size.
Figure 11 Maximum and Minimum File Upload Time (10 concurrent requests)

Figure 11 shows the maximum and minimum file upload time on both the platforms. GAE maximum upload time is 991ms while for Amazon EC 2 it is 813ms. GAE minimum upload time is 23ms while for Amazon EC 2 it is 37ms. GAE shows high variance in the maximum and minimum file upload time as compared to Amazon EC 2.
7.1.3 File Read Scenario

Figure 12 shows the average file read time. GAE performance remained consistent during the experiment of reading the file from Bigtable with size ranging from 100KB to 1MB. In case of Amazon EC2 the performance degraded continuously with the increase in file size. Amazon EC2 file read time was 264ms for 100 KB of file size but it reached to 387ms for 1MB file size while for GAE it remained between 321ms and 339ms.

The results are similar to the file upload performance; GAE performance improves with the increase in file size by auto allocating resources and it shows that Bigtable is optimized for large stream of reads. Since Bigtable uses Block caching for reading data and on the backend GFS loads the whole chunk into memory for fast reading, therefore, the performance improved in sequential reads [17, 28].
Figure 13 Maximum and Minimum File Read Time

Figure 13 shows the maximum and minimum file reading time on both the platforms. GAE maximum upload time is 847ms while for Amazon EC 2 it is 743ms. GAE minimum read time is 39ms while for Amazon EC 2 it is 56ms. GAE again shows higher variance in the maximum and minimum file upload time as compared to Amazon EC 2.

7.1.4 Small Write-Stream Scenario

Figure 14 Average Write Time

Figure 14 shows the average time of storing data. GAE took more time to store data as compared to Amazon EC 2 but GAE performance was consistent when concurrent requests
increased. GAE write time was 521ms and 513ms for 10 and 500 concurrent requests respectively. While Amazon EC2 performance degraded with increase in concurrent requests, from 10 concurrent requests to 500 current requests the time increased from 287 to 429ms.

GAE write performance shows ups and downs in the experiment. For 10 concurrent requests, there are minimum resources allocated to GAE application and it takes more time to write the data. But with increase in concurrent requests to 20 and beyond GAE allocates resources to this application and the write time drops suddenly and then starts increasing again until 200 concurrent requests. When the concurrent requests increases more than 200, there is again decrease in write time and the performance improves. Amazon EC2 performed well for small number of requests (up to 75) but after that there is a considerable amount of increase in write time. Again, it is due to the internal working of Bigtable that GAE performance improved for higher number of writes requests, since all write operations are performed in memory and later on they are stored in to GFS [17].

Figure 15 shows the maximum and minimum time of storing data on the two platforms. GAE maximum write time is 990 while for Amazon EC 2 it is 922. GAE minimum write time is 21 while for Amazon EC 2 it is 43. GAE shows high variance in the maximum and minimum write time as compared to Amazon EC2.
7.1.5 Execution Time Scenario

Figure 16 Average Execution Time

Figure 16 shows the execution time on the two platforms. Amazon EC2 performance decreases with the increase of concurrent requests. GAE has smaller execution time than Amazon EC2; it maintains a consistent level of execution time with increase in concurrent requests. Amazon EC2 took 146ms as an average for execution to handle 10 concurrent requests but it took 476ms for 500 concurrent requests while GAE took 285ms for 10 concurrent requests and 472ms for 500 concurrent requests.

Amazon EC2 shows better performance for small and medium size workload but shows a degraded performance when concurrent requests increase more than 200. GAE showed an inconsistent performance in the beginning of this experiment. Since there is random read and write operations in this experiment, for most of the reads there was block-cache miss in Bigtable and GAE has to fetch it from GFS [17], but in other cases when there was block-cache hit, the performance was better. This behavior is shown throughout in this experiment by GAE.
Figure 17 Maximum and Minimum Execution Time

Figure 17 shows the maximum and minimum execution time on both the platforms. GAE maximum execution time is 910ms while for Amazon EC 2 it is 867ms. GAE minimum execution time is 23ms while Amazon EC 2 it is 46ms. GAE shows larger difference between maximum and minimum execution time as compared to Amazon EC 2.
7.1.6 Scalability Scenario

Figure 18 Effect on Execution Time with increase in Data volume

Figure 18 shows the effect on execution performance of the two platforms while increasing the records in database/data store. In case of GAE, the search performance was consistent and lower than Amazon EC2. Amazon EC2 execution time increased over 5 times of GAE when the numbers of records in the data store/database reach to 10,000 from 1000 records. GAE scaled well with the increase in records in the data store. In the start it took 238ms to search the records, but with increase in records, the performance remained consistent. MySQL performance decreased with increase in number of records showing the limitations of RDBMS during increase in data volume [82].
Figure 19 shows the effect on response time when the size of data store/database was increased to 10,000 records. Amazon EC2 response time increased rapidly with increase in database records while GAE response time remained at almost consistent level with the increase in records in data store. For 10,000 records in data store, GAE took 293ms while Amazon EC2 took 1585ms. This experiment is same as the one shown in Fig 17, but here the data shows that GAE has faster response time than Amazon EC2, which was also observable in the experiment conducted by Li et al. [54].

### 7.2 Memory Requirement

<table>
<thead>
<tr>
<th>Memory Requirement</th>
<th>Google App Engine</th>
<th>Amazon EC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXIMUM</td>
<td>153MB</td>
<td>109MB</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>76MB</td>
<td>54MB</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>113MB</td>
<td>74MB</td>
</tr>
</tbody>
</table>

Table 5 shows memory requirement for the prototype application. Memory consumption increased on both platforms with the increase of data in data store/database. When the no. of records reached 10,000 (initial no. of records 1000), both systems recorded their maximum memory usage i.e. GAE was consuming 153 MB and Amazon EC2 memory usage was at 109 MB.

Bigtable stores the data in memory until it reaches a certain level of threshold then it moves it to GFS [17]. In case of read operations, Bigtable moves in the whole SSTable (which
stores Bigtable data) into memory [17], this leads to more memory consumption in GAE application as compared to MySQL, which only fetches the data into memory when it is requested by application query [63].

7.3 Function Points
Function points were calculated from the requirements using COSMIC function point method [20]. The calculated function points were 167.

7.4 Source Lines of Code

<table>
<thead>
<tr>
<th></th>
<th>Google App Engine</th>
<th>Amazon EC 2</th>
<th>Difference</th>
<th>% Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>3793</td>
<td>3319</td>
<td>474</td>
<td>12.5</td>
</tr>
</tbody>
</table>

Table 6 shows the Source Lines of Code (SLOC) for both the platforms and the difference. SLOC only include business logic code and do not include user interface and other code. SLOC for GAE is more than Amazon EC 2, mostly because there was extra coding involved to work around the limitations and restrictions of GAE platforms. Such as file uploading, working around to make the Java frameworks and libraries compatible with GAE and writing extra code for data consistency and integrity.

7.5 Development Time

<table>
<thead>
<tr>
<th>Module</th>
<th>Google App Engine (Person-hours)</th>
<th>Amazon EC 2 (Person-hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounts Module</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>Contacts Module</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Issues Module</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>User Module</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>Opportunity Module</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>Sales Module</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>Document Module</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Events Module</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Other</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>83</td>
<td>59</td>
</tr>
</tbody>
</table>

Table 7 shows that GAE took more time in development as compared to Amazon EC 2. It was difficult to find help and reference for programming on GAE. We have to post problems and questions on Google help forums and other websites to get help for the problems we encountered during development.

7.6 Debugging Time

<table>
<thead>
<tr>
<th></th>
<th>Google App Engine (Person-hours)</th>
<th>Amazon EC 2 (Person-hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>21</td>
<td>13</td>
</tr>
</tbody>
</table>
Table 8 shows the debugging time of the prototype application. Development and debugging time includes business logic development/debugging time only and do not include user interface and other development/debugging time. Database design time is also not included, since the same database design is used for both the applications.

Debugging time for GAE was also more than Amazon EC 2 similar to development time. Fixing bugs and re-coding them was time consuming in two ways, first there was lack of clear cut solutions available online (Google help forums, other websites) and secondly the solutions which we came up with took more time in understanding and coding. We also noticed strange behavior in the development tools provide by Google for GAE. In some cases, the code will execute properly on local machine but would fail to execute on GAE. Although we could not come up with proper solution to this problem but this affected the debugging time on GAE.
8 ANALYSIS AND DISCUSSION

8.1 Analysis

In this section we discuss and analyze the results gathered through the experiment and literature study. During the experiment, we made observations about the behavior of the two cloud platforms. These are listed below.

- GAE has poor response time for a small number of requests. But with the increase of the number of concurrent requests, GAE response time improved considerably as compared to Amazon EC2. This shows the efficient resource utilization and allocation in GAE. It also shows that GAE network bandwidth and latency performance is better than Amazon EC2 [54].

- Bigtable is optimized for large streams of write/read operations, and the experiment showed that when upload and read file size were increased up to 1MB, GAE upload and read file time start to decrease and the performance improved. Amazon EC2 showed better performance for small file size uploads but performance degraded for large file sizes.

- In short write streams, Amazon EC2 performed better than GAE, but GAE had higher performance when the concurrent requests reached more than 200. Amazon performance degraded with the increase in concurrent requests.

- GAE showed ups and downs when a combination of random read and write operations were performed. One reason for this behavior can be attributed to the block-cache miss of data. Amazon EC2 performed better than GAE for a small number of concurrent requests in terms of random read and write operations, but performance decreased for large number of concurrent requests.

- In terms of scalability, GAE performed in a consistent manner while Amazon EC2 performance degraded rapidly with the increase in data volume and the number of concurrent requests.

- In all experiment scenarios except the scalability scenario, GAE performance showed higher variance as compared to Amazon EC2 [15].

The experiment results provide interesting behavior patterns for GAE and Amazon EC2 platform. The higher variance in response and execution time of GAE shows that their physical resources are diverse in terms of processing speed, while Amazon EC2 has homogeneous hardware resources [54, 72], and also GAE uses automatic resource allocation which can sometimes cause variable performance.

Bigtable has been designed to provide consistent performance with scalability [17], and the results show the same behavior. In cases of large streams of write/read data, a high number of concurrent requests and searching large number of records, GAE performed with consistent performance while Amazon EC2 performance showed gradual decrease with a higher number of concurrent requests. GAE uses automatic resource allocation, which helped in providing consistent performance with scalability but this feature is not present in Amazon EC2, and its performance was degraded while scaling in terms of data volume.

Applications developed for GAE are incompatible with other platforms and cannot be ported to other platforms in their existing forms. It provides email and authentication system based on
Gmail and Bigtable storage system which are Google proprietary tools and software; this creates a lock-in for the developers and organizations into GAE platform.

In the development of prototype application, it took more time and SLOC to develop the GAE prototype application as compared to Amazon EC2. The main reasons are the limited availability of development resources for GAE, development frameworks and technologies, help resources etc.

Some of the available Java development frameworks and libraries require extra coding and modification before they can be compatible with GAE [45]. Similarly, in debugging of these applications, there is shortage of help and solutions availability in case of GAE, which we also encountered during prototype application development.

8.2 Limitations of the Google App Engine

Google has imposed many restrictions on GAE, which limits the development of applications on it. Since GAE is a platform as a service, it is offered as an application runtime environment. GAE is highly abstracted and virtualized as compared to Amazon EC 2. The following are the main limitations of GAE:

- GAE only allows web-based applications [30]. This limits its usability in some development contexts.
- Only two languages are supported; Python and Java. JVM allows many other JVM-based languages to be deployed on GAE [33], but other mainstream languages such as C++, .Net Languages, PHP etc. cannot be used directly.
- Applications are executed in a sandbox environment; therefore they cannot access the operating system and hardware layer [33]. Although sandboxing ensures security, it also makes enterprise level application development a troublesome process. Applications cannot access the file system for writing, no network layer access (no socket access), no shared memory etc. The only way an application can make communication with other applications or the outside world is through URL fetch or email [30].
- There is a 30 second limit on the execution time of a web request. This limit is not flexible [93].
- No multithreading or execution of sub-process are allowed in the GAE [93].
- Data is stored in a data store which is based on Bigtable. Bigtable is a schema-less storage system, as compared to RDBMs, which are widely used in enterprise applications [31]. The Bigtable storage mechanism is a tradeoff between consistency and scalability. It does not enforce the level of consistency and integrity as compared to RDBMS in order to provide scalability [17]. Data will lose consistency and integrity, which is a very critical issue in enterprise applications.
- Web request/response size, code files, user files cannot be greater than 10 MB [71].
- The Data store does not allow objects larger than 1 MB [32].
- Unlike RDBMS, GAE data store uses Google Query Language (GQL), which is a language similar to SQL but with limited features. GQL is specifically designed for use with the GAE data store and is not fully compatible with existing SQL code. [37]
- Developer has a GQL learning curve before coding. GQL has some restrictions on queries.
- In a SELECT query, a limited number of objects can be retrieved as a result of the 30 second execution time limit [93].
- In a single query, an inequality filter (>, <, >=, <=, !) can be used only once [69].
In retrieving entities from the data store, there is a limit on the “offset” value to 1000 [18].

GQL does not implement UPDATE, DELETE and JOIN SQL statements. [18]

Any kind of data store operation is dependent on the 30 second time limit [93].

Transactions are only supported for single entities or entity groups (entities with relationships), other than that transactions are not supported globally [87].

GAE does not have the functionality to query all items in the data store [18].

GAE is not compatible with some existing programming frameworks, libraries and tools [45]. In some cases it requires extra coding and workarounds, while in some cases it’s not compatible completely [45].

8.3 Amazon EC 2 versus Google App Engine

In this section we discuss the main differences between GAE and Amazon EC 2. Both cloud platforms are virtualized and abstracted at different levels. GAE provides a higher level of abstraction and a tightly controlled development environment while Amazon EC 2 provides lower level of abstraction with flexibility and customization of hardware and software resources.

- EC 2 is an IaaS, allowing use of any kind of operating system and software while GAE is a PaaS, providing a sandbox environment for the execution of applications.
- EC 2 can be used to deploy any kind of applications (web based and standalone applications) while GAE only allows web based applications.
- GAE only supports Java (and JVM based languages) and Python language. While any language can be used for developing applications on EC 2.
- GAE provides plugins for development on Eclipse and NetBeans IDE. On the other hand, any development tool can be used to develop applications for EC 2.
- GAE allows deployment of applications through its GAE plugin for Java or Python only. EC 2 allows setup of the platform through Amazon plugins for Eclipse, NetBeans etc. and also the deployed platform can be controlled directly to deploy applications by using SSH clients.
- GAE supports Bigtable as the only storage mechanism, which is a schema-less, NoSQL [31] storage system, while EC 2 provides both NoSQL and SQL based support. Amazon Relational Database service (RDS) provides MYSQL based database service to users. Users can deploy MYSQL database instances and use it as storage for applications running in Amazon EC 2. Amazon SimpleDB is a NoSQL storage service, which is similar to Bigtable in functionality. It provides both SQL and NoSQL based storage services, which adds to the advantage of Amazon EC 2.
- GAE does not allow files larger than 10 MB while Amazon web services allow hosting of files from 1 GB to 1 TB.
- GAE supports completely automatic scaling and allocation of resources and the user has no control on the resources assigned to their applications [18]. On the other hand, Amazon EC2 is completely configurable. It allows the user to run the required number of instances each with their own user-selectable operating system and application software. User can also configure networking among these instances and allocate IP addresses to the instances.
- Amazon allows the selection of geographical locations where a user can store data, such as United States, Ireland and Singapore data centers [3], while GAE does not provide this level of administration. This feature allows users of Amazon EC 2 to place their
application and data in those data centers which are closer to their target customers or conform to their country laws [1].

8.4 NoSQL vs. RDBMS Database

In this section we discuss the advantages and disadvantages of NoSQL as compared to RDBMS, which we identified during literature study and the prototype application experiment.

- NoSQL are schema less storage system. This feature can both be an advantage and a disadvantage. Lack of schema improves speed, because the data store does not have to lookup metadata and no lockup of data to enforce consistency and integrity of data [82, 53].
- The main disadvantage of NoSQL data stores is lack of full support for ACID properties [82].
- NoSQL are distributed databases systems which improve availability and reliability but can also make maintenance difficult [82].
- Most NoSQL implementations don’t support full SQL, which causes problems in migrating data and applications to NoSQL based systems [55]. Moreover, NoSQL databases such as HBase, Cassandra, Bigtable, Amazon Dynamo have different implementations with different features, which make the data difficult to port between them [53].
- NoSQL databases can handle unstructured data such as documents, multimedia files, email, social media etc. efficiently than RDBMS [53].

8.5 Recommendations for Platform Selection

Enterprise application development requires the target platform to be able to meet certain requirements. These requirements vary from business-to-business. We have identified a set of requirements based on our literature study [38, 39], studying existing enterprise applications and the experience gained from prototype application development. Table 9 shows grouping of enterprise applications into 4 categories such as Transactional or Non-Transactional and Data Intensive or Data Non-Intensive applications.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Application Requirements</th>
<th>Application Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon EC 2</td>
<td>Data Analysis Application</td>
<td>Transactional Application</td>
</tr>
<tr>
<td></td>
<td>(e.g. Banking, Finance)</td>
<td></td>
</tr>
<tr>
<td>Google App Engine</td>
<td>Data Management Applications</td>
<td>Non-Transactional Application</td>
</tr>
<tr>
<td></td>
<td>(e.g. Customer Support, Feedback, Issue Tracker)</td>
<td></td>
</tr>
</tbody>
</table>

In table 9, we recommend GAE as suitable for non-transactional applications and those enterprise applications that do not perform heavy analysis on large set of data since GAE lacks global transaction support and processing performance of GAE is low as compared to Amazon EC 2 [54]. Although following the recommendations mentioned in section 8.5, other types of enterprise applications can also be implemented in GAE but it requires changes (discussed later) in the data model and design of applications. Enterprise applications which require a high
amount of computation are not suitable for GAE, since GAE does not allow manual allocation of resources, while Amazon EC2 allows this feature and high number of resources (with powerful processing capabilities, even GPU) can be allocated to provide high computation performance [3].

GAE provides a quick setup of development and production environment as compared to Amazon EC2. In case of Amazon EC2, configuration of resources is required, such as networking, installation of operating system and software, which can be complex and difficult to maintain in some cases. GAE eliminates the problems of operating system maintenance, upgrade, installing patches, fault recovery etc. which exists in Amazon EC2.

Development and debugging time on GAE is longer than Amazon EC2. This was evident in the development of the prototype CRM application, which took 474 (12.5 %) more SLOC to develop for GAE. One of the reasons for this increase in time and SLOC is the lack of compatibility and support of programming libraries and frameworks on GAE. Moreover, there is limited amount of help and support available for developers.

In order to develop application for GAE, some changes are required during design and development, in order to make them compatible with GAE platform and take advantage of the platform, which are mentioned in the following section.

8.6 Recommendations for Development

In this section we have suggested some solutions to the limitations imposed by GAE. These solutions came through the literature study, online help forums, Google groups, and the example applications for GAE platform by Google.

- Data objects which are reading and writing data to/from other objects should be grouped together by creating relationships or merging the data objects; this will allow GAE to perform transactions on this data and to achieve integrity and consistency.
- Grouped data also allow GAE to store this data in a single or few SSTables, which will make reading and writing of data faster.
- Data grouping will also allow reduction in random reads and random writes. Random reads is a weak point of Bigtable performance [17].
- In some cases, the relationship creation is not possible and may cause loss of integrity and consistency. In such cases GAE is not the suitable development environment.
- GAE cannot count all records in the data store if counting task exceeds the 30 second limit; one way to overcome this restriction is to use counting procedure at the time of writing data. Another way to solve it is to break down the task into less than 30 seconds of sub task and then these sub tasks can be executed serially.
- A file system can be implemented on top of Bigtable to allow features such as writing of files, which is missing in GAE.
- Files larger than 1MB should be divided into smaller size chunks which can then be stored in Bigtable, but this will make the development and maintenance of the application difficult.
9 CONCLUSION

In this research study we evaluated the feasibility of Google App Engine from the enterprise application development point of view. We explored the GAE architecture, its behavior and the limitations by using literature study and developing a prototype application. We also compared the architecture and behavior of GAE with Amazon EC 2, and analyzed the type of applications that can be built for these platforms.

GAE allows deployment of web-based applications only while Amazon supports web-based applications as well as many other types of applications. In our literature study we found a lack of support for multi-threading, global transactions, write access to file system, access to socket, lack of development languages and tools etc. on the GAE platform which greatly reduces and affects the development of wide range of applications. Most of the available Java tools and libraries are incompatible with GAE. These libraries require extra coding and workarounds. Only one Integrated Development Environment (IDE) is available for development officially by Google, while other IDE’s are difficult to setup and use with GAE.

In the prototype application experiment, GAE showed consistent performance. In high read, searching and large-write operations GAE outperformed Amazon EC 2 in all scenarios. However, in random read and random write scenarios, GAE performance was inconsistent. Usually for the first web request, GAE took more time than its average response or execution time. Because GAE uses cheap hardware resources as compared to Amazon EC 2, and also GAE loads and unloads the application resources automatically depending on the user traffic.

Lack of global transactions in the GAE platform makes development of enterprise applications difficult. The commodity based nature of GAE resources provides scalability but lacks high performance which is critical for data intensive applications. Data sensitive or intensive applications are not feasible to develop on GAE. But for non-critical data GAE provides an attractive platform, which can scale without compromising too much on performance. Deploying application on GAE is simpler as compared to Amazon EC 2. GAE also provides automatic resource allocation, which is difficult to setup on Amazon EC 2.

GAE as a platform has not yet matured and is being constantly improved by Google. Amazon EC 2 as an infrastructure is very flexible and customizable, and can be used for a wide range of application development. Our results can be used to analyze the feasibility of enterprise applications for GAE and Amazon EC 2. Our results can also be helpful in developing and designing applications for GAE.
10 **Future Work**

In this research study, we tried to cover the problem area with two aspects, by using literature and working with cloud platform by developing applications for them in order to understand their behavior and architecture. We analyzed and discussed the results in order to explain the behavior and architecture of these platforms in the contexts of related work and CRM application. But further research work can also be done to analyze the behavior in the contexts of other enterprise applications and different experiment scenarios can be used to explore the architecture and behavior in further detail.

Other cloud platforms such as Microsoft Windows Azure can also be used in the future research work to analyze the most suitable platform for the different enterprise application types. Only two developers worked on developing the prototype application for the two cloud platforms, and future research work can be conducted to validate the results by using large number of developers to see the effect on development and debugging time and SLOC. We compared only single MySQL instance with Amazon EC 2 configuration, in future different configurations can be used to compare with GAE.
REFERENCES


[89] Update From Amazon Regarding Friday’s S3 Downtime: http://www.centernetworks.com/amazon-s3-downtime-update.


# Appendix A: Terminologies

<table>
<thead>
<tr>
<th>Terminology</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoSQL</td>
<td>It refers to non-SQL based storage system, which do not use SQL as their query language</td>
</tr>
<tr>
<td>Cassandra, MongoDB, Hbase, Hive</td>
<td>These are NoSQL based databases</td>
</tr>
<tr>
<td>GAE</td>
<td>Google App Engine</td>
</tr>
<tr>
<td>EC 2</td>
<td>Amazon Elastic Compute 2</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management Systems</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>GFS</td>
<td>Google File System</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JDO</td>
<td>Java Data Objects</td>
</tr>
<tr>
<td>POJO</td>
<td>Plain Old Java Objects</td>
</tr>
<tr>
<td>SLOC</td>
<td>Source Lines of Code</td>
</tr>
</tbody>
</table>


APPENDIX B: PRIMARY STUDIES


International Conference on Software Engineering and Service Sciences, ICSESS 2010 (Beijing, Jul. 2010), 683-687.


[P33] Zeng, S.Q. and Xu, J.B. 2010. The improvement of PaaS platform. 1st International


Internet Resources

[23] The Toposphere - Meterology for the cloud computing world: http://itknowledgeexchange.techtarget.com/cloud-computing/amazons-early-efforts-at-
cloud-computing-partly-accidental/.
APPENDIX C: DATA MODEL CLASS DIAGRAMS

Opportunity
- dateAdded: String
- daysOnStage: Short
- description: String
- estimatedCloseDate: String
- estimatedRevenue: String
- expectedSales: String
- id: Key
- notes: String
- opportunityName: String
- updatedDays: String
- userName: String

Sales
- accountName: String
- id: Key
- netProfit: String
- saleAmount: String
- saleDiscount: String
- saleItem: String
- saleName: String
- saleQuantity: String
- saleRep: String
- saleStatus: String
- saleTime: Timestamp
- timeAdded: Timestamp
- userName: String

Contacts
- accountId: Key
- contactCellphone: String
- contactCompany: String
- contactEmail: String
- contactHomePhone: String
- contactName: String
- contactOfficePhone: String
- contactRole: String
- dateAdded: String
- id: Key
- userName: String

Meetings
- confirmCancel: Integer
- date: String
- duration: String
- id: Key
- location: String
- opportunityId: Integer
- priority: Integer
- responsible: String
- status: Integer
- title: String
- userName: String

Events
- accountName: String
- contactName: String
- eventDescription: String
- eventName: String
- eventEndTime: Timestamp
- eventStart: Timestamp
- eventType: String
- id: Key
- userName: String

ContactAddresses
- addressName: String
- addressOne: String
- addressTwo: String
- city: String
- country: String
- id: Key
- state: String
- userContactId: Key
- userName: String
- zipCode: String

LogCalls
- callAgenda: String
- callDuration: String
- callTime: Date
- id: Key
- opportunityId: Integer
- userName: String
APPENDIX D: DATA MODEL SOURCE CODE

Account.java

```java
package gcrm.model;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import com.google.appengine.api.datastore.Key;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

/**
 * @author Kashif Khan
 */
@PersistenceCapable(detachable = "true", identityType = IdentityType.APPLICATION)
public class Accounts implements java.io.Serializable{

    /**
     * @uml.property name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;

    /**
     * @uml.property name="userName"
     */
    @Persistent private String userName;

    /**
     * @uml.property name="accountName"
     */
    @Persistent private String accountName;

    /**
     * @uml.property name="dateAdded"
     */
    @Persistent private String dateAdded;

    /**
     * @uml.property name="isActive"
     */
    @Persistent private Integer isActive;

    /**
     * @return
     * @uml.property name="id"
     */
    public Key getId() {
        return id;
    }

    /**
     * @param id
     * @uml.property name="id"
     */
    public void setId(Key id) {
        this.id = id;
    }
```

/
* @return
* @uml.property name="userName"
*
public String getUserName() {
    return userName;
}
/**
* @param userName
* @uml.property name="userName"
*/
public void setUserName(String userName) {
    this.userName = userName;
}
/**
* @return
* @uml.property name="accountName"
*
public String getAccountName() {
    return accountName;
}
/**
* @param accountName
* @uml.property name="accountName"
*/
public void setAccountName(String accountName) {
    this.accountName = accountName;
}
/**
* @return
* @uml.property name="dateAdded"
*
public String getDateAdded() {
    return dateAdded;
}
/**
* @param dateAdded
* @uml.property name="dateAdded"
*/
public void setDateAdded(String dateAdded) {
    this.dateAdded = dateAdded;
}
/**
* @return
* @uml.property name="isActive"
*
public Integer getIsActive() {
    return isActive;
}
/**
* @param isActive
* @uml.property name="isActive"
*/
public void setActive(Integer isActive) {
    this.isActive = isActive;
}
AuthGroupAuthorities.java
package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
 * @author  Kashif Khan
 */
@SuppressWarnings("serial")
@PersistenceCapable(detachable = "true", identityType = IdentityType.APPLICATION)
public class AuthGroupAuthorities implements java.io.Serializable{
    /**
     * @uml.property  name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property  name="groupId"
     */
    @Persistent private Key groupId;//auth group
    /**
     * @uml.property  name="authority"
     */
    @Persistent private String authority;
    /**
     * @return
     * @uml.property  name="id"
     */
    public Key getId() {
        return id;
    }
    /**
     * @param id
     * @uml.property  name="id"
     */
    public void setId(Key id) {
        this.id = id;
    }
    /**
     * @return
     * @uml.property  name="groupId"
     */
    public Key getGroupId() {
        return groupId;
    }
    /**
     * @param groupId
     * @uml.property  name="groupId"
     */
    public void setGroupId(Key groupId) {
        this.groupId = groupId;
    }
}
public String getAuthority() {
    return authority;
}
/**
 * @param authority
 * @uml.property name="authority"
 */
public void setAuthority(String authority) {
    this.authority = authority;
}

AuthGroupMembers.java
package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
 * @author Kashif Khan
 */
@SuppressWarnings("serial")
@PersistenceCapable(detachable = "true", identityType = IdentityType.APPLICATION)
public class AuthGroupMembers implements java.io.Serializable {
    /**
     * @uml.property name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property name="userName"
     */
    @Persistent private String userName;
    /**
     * @uml.property name="groupId"
     */
    @Persistent private Key groupId;
    /**
     * @return
     * @uml.property name="id"
     */
    public Key getId() {
        return id;
    }
    /**
     * @param id
     * @uml.property name="id"
     */
    public void setId(Key id) {
        this.id = id;
    }
    /**
     * @return
     * @uml.property name="groupId"
     */
    public Key getGroupId() {
        return groupId;
AuthGroups.java
package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
 * @author Kashif Khan
 */
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class AuthGroups implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    /**
     * @uml.property name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property name="groupName"
     */
    @Persistent private String groupName;
    /**
     * @return
     * @uml.property name="id"
     */
    public Key getId() {
        return id;
    }
    /**
     * @param id
     * @uml.property name="id"
     */
    public void setId(Key id) {
        this.id = id;
    }
    /**
     * @param groupId
     * @uml.property name="groupId"
     */
    public void setGroupId(Key groupId) {
        this.groupId = groupId;
    }
    /**
     * @return
     * @uml.property name="userName"
     */
    public String getUserName() {
        return userName;
    }
    /**
     * @param userName
     * @uml.property name="userName"
     */
    public void setUserName(String userName) {
        this.userName = userName;
    }
}
public String getGroupName() {
    return groupName;
}

public void setGroupName(String groupName) {
    this.groupName = groupName;
}

@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Comment {

    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;

    @Persistent private String description;

    @Persistent private String userName;

    @Persistent private Date dateAdded;

    @Persistent private Key issueId;

    public Key getId() {
        return id;
    }

}
* @param id
* @uml.property name="id"
*/
public void setId(Key id) {
    this.id = id;
}
/**
* @return
* @uml.property name="description"
*/
public String getDescription() {
    return description;
}
/**
* @param description
* @uml.property name="description"
*/
public void setDescription(String description) {
    this.description = description;
}
/**
* @return
* @uml.property name="userName"
*/
public String getUserName() {
    return userName;
}
/**
* @param userName
* @uml.property name="userName"
*/
public void setUserName(String userName) {
    this.userName = userName;
}
/**
* @return
* @uml.property name="dateAdded"
*/
public Date getDateAdded() {
    return dateAdded;
}
/**
* @param dateAdded
* @uml.property name="dateAdded"
*/
public void setDateAdded(Date dateAdded) {
    this.dateAdded = dateAdded;
}
/**
* @return
* @uml.property name="issueId"
*/
public Key getIssueId() {
    return issueId;
}
public void setId(Key issueId) {
    this.issueId = issueId;
}

Competitors.java
package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
 * @author  Kashif Khan
 */
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Competitors implements java.io.Serializable{
    /**
     * private static final long serialVersionUID = 1L;
    */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    @Persistent private String userName;
    @Persistent private String name;
    @Persistent private String description;
    @Persistent private Integer opportunityId;

    /**
     * @return
     */
    public Key getId() {
        return id;
    }
    /**
     * @param id
     */
    public void setId(Key id) {
        this.id = id;
    }
}
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public Integer getOpportunityId() {
    return opportunityId;
}

public void setOpportunityId(Integer opportunityId) {
    this.opportunityId = opportunityId;
}

public String getUserName() {
    return userName;
}

public void setUserName(String userName) {
    this.userName = userName;
}
package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
 * @author Kashif Khan
 */
@SuppressWarnings("serial")
@PersistenceCapable(detachable = "true", identityType = IdentityType.APPLICATION)
public class ContactAddresses implements java.io.Serializable{
  /**
   * @uml.property name="id"
   */
  @PrimaryKey
  @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
  private Key id;
  /**
   * @uml.property name="userName"
   */
  @Persistent String userName;
  /**
   * @uml.property name="userContactId"
   */
  @Persistent private Key userContactId; //contacts
  /**
   * @uml.property name="addressName"
   */
  @Persistent private String addressName;
  /**
   * @uml.property name="addressOne"
   */
  @Persistent private String addressOne;
  /**
   * @uml.property name="addressTwo"
   */
  @Persistent private String addressTwo;
  /**
   * @uml.property name="zipCode"
   */
  @Persistent private String zipCode;
  /**
   * @uml.property name="city"
   */
  @Persistent private String city;
  /**
   * @uml.property name="state"
   */
  @Persistent private String state;
  /**
   * @uml.property name="country"
   */
  @Persistent private String country;
  /**
   * @return
public Key getId() {
    return id;
}
/**
 * @param id
 * @uml.property name="id"
 */
public void setId(Key id) {
    this.id = id;
}
/**
 * @return
 * @uml.property name="userContactId"
 */
public Key getUserContactId() {
    return userContactId;
}
/**
 * @param userContactId
 * @uml.property name="userContactId"
 */
public void setUserContactId(Key userContactId) {
    this.userContactId = userContactId;
}
/**
 * @return
 * @uml.property name="addressName"
 */
public String getAddressName() {
    return addressName;
}
/**
 * @param addressName
 * @uml.property name="addressName"
 */
public void setAddressName(String addressName) {
    this.addressName = addressName;
}
/**
 * @return
 * @uml.property name="addressOne"
 */
public String getAddressOne() {
    return addressOne;
}
/**
 * @param addressOne
 * @uml.property name="addressOne"
 */
public void setAddressOne(String addressOne) {
    this.addressOne = addressOne;
}
public String getAddressTwo() {
    return addressTwo;
}

public void setAddressTwo(String addressTwo) {
    this.addressTwo = addressTwo;
}

public String getZipCode() {
    return zipCode;
}

public void setZipCode(String zipCode) {
    this.zipCode = zipCode;
}

public String getCity() {
    return city;
}

public void setCity(String city) {
    this.city = city;
}

public String getState() {
    return state;
}

public void setState(String state) {
    this.state = state;
}
public String getCountry() {
    return country;
}

public void setCountry(String country) {
    this.country = country;
}

public String getUserName() {
    return userName;
}

public void setUserName(String userName) {
    this.userName = userName;
}
* @uml.property name="contactOfficePhone"
*/
@Persistent private String contactOfficePhone;
/**
* @uml.property name="contactCellphone"
*/
@Persistent private String contactCellphone;
/**
* @uml.property name="contactCompany"
*/
@Persistent private String contactCompany;
/**
* @uml.property name="contactName"
*/
@Persistent private String contactName;
/**
* @uml.property name="contactRole"
*/
@Persistent private String contactRole;
/**
* @uml.property name="dateAdded"
*/
@Persistent private String dateAdded;
/**
* @uml.property name="accountId"
*/
@Persistent private Key accountId;

/**
* @return
* @uml.property name="id"
*/
public Key getId() {
    return id;
}
/**
* @param id
* @uml.property name="id"
*/
public void setId(Key id) {
    this.id = id;
}
/**
* @return
* @uml.property name="userName"
*/
public String getUserName() {
    return userName;
}
/**
* @param userName
* @uml.property name="userName"
*/
public void setUserName(String userName) {
    this.userName = userName;
}
/**
* @return
* @uml.property name="contactEmail"
*/
public String getContactEmail() {
    return contactEmail;
}
/**
* @param contactEmail
* @uml.property name="contactEmail"
*/
public void setContactEmail(String contactEmail) {
    this.contactEmail = contactEmail;
}
/**
* @return
* @uml.property name="contactHomePhone"
*/
public String getContactHomePhone() {
    return contactHomePhone;
}
/**
* @param contactHomePhone
* @uml.property name="contactHomePhone"
*/
public void setContactHomePhone(String contactHomePhone) {
    this.contactHomePhone = contactHomePhone;
}
/**
* @return
* @uml.property name="contactOfficePhone"
*/
public String getContactOfficePhone() {
    return contactOfficePhone;
}
/**
* @param contactOfficePhone
* @uml.property name="contactOfficePhone"
*/
public void setContactOfficePhone(String contactOfficePhone) {
    this.contactOfficePhone = contactOfficePhone;
}
/**
* @return
* @uml.property name="contactCellphone"
*/
public String getContactCellphone() {
    return contactCellphone;
}
/**
* @param contactCellphone
* @uml.property name="contactCellphone"
*/
public void setContactCellphone(String contactCellphone) {
    this.contactCellphone = contactCellphone;
}
/**
 * @return
 * @uml.property name="contactCompany"
 */
public String getContactCompany() {
    return contactCompany;
}
/**
 * @param contactCompany
 * @uml.property name="contactCompany"
 */
public void setContactCompany(String contactCompany) {
    this.contactCompany = contactCompany;
}
/**
 * @return
 * @uml.property name="contactRole"
 */
public String getContactRole() {
    return contactRole;
}
/**
 * @param contactRole
 * @uml.property name="contactRole"
 */
public void setContactRole(String contactRole) {
    this.contactRole = contactRole;
}
/**
 * @return
 * @uml.property name="contactName"
 */
public String getContactName() {
    return contactName;
}
/**
 * @param contactName
 * @uml.property name="contactName"
 */
public void setContactName(String contactName) {
    this.contactName = contactName;
}
/**
 * @return
 * @uml.property name="dateAdded"
 */
public String getDateAdded() {
    return dateAdded;
}
/**
 * @param dateAdded
 * @uml.property name="dateAdded"
 */
public void setDateAdded(String dateAdded) {
    this.dateAdded = dateAdded;
}
```java
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class DocumentFiles implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    private Key id;
    private String userName;
    private String fileName;
    private Blob fileData;
    private String updatedTime;

    /**
     * @return
     * @uml.property name="id"
     */
    public Key getId() {
        return id;
    }

    /**
     * @param accountId
     * @uml.property name="accountId"
     */
    public void setAccountId(Key accountId) {
        this.accountId = accountId;
    }
}
```
public void setId(Key id) {
    this.id = id;
}

public String getUserName() {
    return userName;
}

public void setUserName(String userName) {
    this.userName = userName;
}

public String getFileName() {
    return fileName;
}

public void setFileName(String fileName) {
    this.fileName = fileName;
}

public Blob getFileData() {
    return fileData;
}

public void setFileData(Blob fileData) {
    this.fileData = fileData;
}

public String getUpdatedTime() {
    return updatedTime;
public void setUpdatedTime(String updatedTime) {
    this.updatedTime = updatedTime;
}

Events.java
package gcrm.model;
import java.sql.Timestamp;
import javax.jdo.
import com.google.appengine.api.datastore.Key;
/**
 * @author  Kashif Khan
 */
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Events implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    /**
     * @uml.property  name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property  name="userName"
     */
    @Persistent private String userName;
    /**
     * @uml.property  name="eventName"
     */
    @Persistent private String eventName;
    /**
     * @uml.property  name="eventDescription"
     */
    @Persistent private String eventDescription;
    /**
     * @uml.property  name="eventStartTime"
     */
    @Persistent private Timestamp eventStartTime;
    /**
     * @uml.property  name="eventEndTime"
     */
    @Persistent private Timestamp eventEndTime;
    /**
     * @uml.property  name="accountName"
     */
    @Persistent private String accountName;
    /**
     * @uml.property  name="contactName"
     */
    @Persistent private String contactName;
}
/**
 * @return
 * @uml.property name="id"
 */
public Key getId() {
    return id;
}
/**
 * @param id
 * @uml.property name="id"
 */
public void setId(Key id) {
    this.id = id;
}
/**
 * @return
 * @uml.property name="userName"
 */
public String getUserName() {
    return userName;
}
/**
 * @param userName
 * @uml.property name="userName"
 */
public void setUserName(String userName) {
    this.userName = userName;
}
/**
 * @return
 * @uml.property name="eventName"
 */
public String getEventName() {
    return eventName;
}
/**
 * @param eventName
 * @uml.property name="eventName"
 */
public void setEventName(String eventName) {
    this.eventName = eventName;
}
/**
 * @return
 * @uml.property name="eventDescription"
 */
public String getEventDescription() {
    return eventDescription;
}
/**
 * @param eventDescription
 * @uml.property name="eventDescription"
 */
public void setEventDescription(String eventDescription) {
    this.eventDescription = eventDescription;
}
public void setEventDescription(String eventDescription) {
    this.eventDescription = eventDescription;
}

public Timestamp getEventStartTime() {
    return eventStartTime;
}

public void setEventStartTime(Timestamp eventStartTime) {
    this.eventStartTime = eventStartTime;
}

public Timestamp getEventEndTime() {
    return eventEndTime;
}

public void setEventEndTime(Timestamp eventEndTime) {
    this.eventEndTime = eventEndTime;
}

public String getAccountName() {
    return accountName;
}

public void setAccountName(String accountName) {
    this.accountName = accountName;
}

public String getContactName() {
    return contactName;
}

public void setContactName(String contactName) {
    this.contactName = contactName;
}
public void setContactName(String contactName) {
    this.contactName = contactName;
}
/**
 * @return
 * @uml.property  name="eventTypeId"
 */
public Integer getEventTypeId() {
    return eventTypeId;
}
/**
 * @param eventTypeId
 * @uml.property  name="eventTypeId"
 */
public void setEventTypeId(Integer eventTypeId) {
    this.eventTypeId = eventTypeId;
}

EventTypes.java
package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
 * @author  Kashif Khan
 */
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class EventsTypes implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    /**
     * @uml.property  name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property  name="userName"
     */
    @Persistent private String userName;
    /**
     * @uml.property  name="name"
     */
    @Persistent private String name;
    /**
     * @uml.property  name="description"
     */
    @Persistent private String description;
    /**
     * @return
     * @uml.property  name="id"
     */
    public Key getId() {
        return id;
    }
    /**
* @param id
* @uml.property name="id"
*/
public void setId(Key id) {
    this.id = id;
}
/**
* @return
* @uml.property name="name"
*/
public String getName() {
    return name;
}
/**
* @param name
* @uml.property name="name"
*/
public void setName(String name) {
    this.name = name;
}
/**
* @return
* @uml.property name="description"
*/
public String getDescription() {
    return description;
}
/**
* @param description
* @uml.property name="description"
*/
public void setDescription(String description) {
    this.description = description;
}
/**
* @return
* @uml.property name="userName"
*/
public String getUserName() {
    return userName;
}
/**
* @param userName
* @uml.property name="userName"
*/
public void setUserName(String userName) {
    this.userName = userName;
}
public class ForgetPassword implements java.io.Serializable {

    private static final long serialVersionUID = 1L;
    /**
     * @uml.property  name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property  name="userName"
     */
    @Persistent private String userName;
    /**
     * @uml.property  name="activationId"
     */
    @Persistent private String activationId;
    /**
     * @uml.property  name="timeStamp"
     */
    @Persistent private Timestamp timeStamp;

    /**
     * @return
     * @uml.property  name="id"
     */
    public Key getId() {
        return id;
    }
    /**
     * @param id
     * @uml.property  name="id"
     */
    public void setId(Key id) {
        this.id = id;
    }
    /**
     * @return
     * @uml.property  name="userName"
     */
    public String getUserName() {
        return userName;
    }
    /**
     * @param userName
     * @uml.property  name="userName"
     */
    public void setUserName(String userName) {
        this.userName = userName;
    }
    /**
     * @return
     * @uml.property  name="activationId"
     */
    public String getActivationId() {
        return activationId;
    }
}

return activationId;
}
/**
 * @param activationId
 * @uml.property name="activationId"
 */
public void setActivationId(String activationId) {
    this.activationId = activationId;
}
/**
 * @return
 * @uml.property name="timeStamp"
 */
public Timestamp getTimeStamp() {
    return timeStamp;
}
/**
 * @param timeStamp
 * @uml.property name="timeStamp"
 */
public void setTimeStamp(Timestamp timeStamp) {
    this.timeStamp = timeStamp;
}

InvolvedContacts.java
package gcrm.model;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.Key;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;
/**
 * @author Kashif Khan
 */
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class InvolvedContacts implements java.io.Serializable{
    private static final long serialVersionUID = 1L;
    /**
     * @uml.property name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property name="userName"
     */
    @Persistent private String userName;
    /**
     * @uml.property name="opportunityId"
     */
    @Persistent private Integer opportunityId;
    /**
     * @uml.property name="contactsId"
     */
    @Persistent private Integer contactsId;
/**
 * @return
 * @uml.property name="id"
 */
public Key getId() {
    return id;
}
/**
 * @param id
 * @uml.property name="id"
 */
public void setId(Key id) {
    this.id = id;
}
/**
 * @return
 * @uml.property name="opportunityId"
 */
public Integer getOpportunityId() {
    return opportunityId;
}
/**
 * @param opportunityId
 * @uml.property name="opportunityId"
 */
public void setOpportunityId(Integer opportunityId) {
    this.opportunityId = opportunityId;
}
/**
 * @return
 * @uml.property name="contactsId"
 */
public Integer getContactsId() {
    return contactsId;
}
/**
 * @param contactsId
 * @uml.property name="contactsId"
 */
public void setContactsId(Integer contactsId) {
    this.contactsId = contactsId;
}
/**
 * @return
 * @uml.property name="userName"
 */
public String getUserName() {
    return userName;
}
/**
 * @param userName
 * @uml.property name="userName"
 */
public void setUserName(String userName) {
    this.userName = userName;
}
package gcrm.model;
import java.util.Date;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
 * @author Kashif Khan
 */
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Issue {
    /**
     * @uml.property name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property name="title"
     */
    @Persistent private String title;
    /**
     * @uml.property name="description"
     */
    @Persistent private String description;
    /**
     * @uml.property name="userName"
     */
    @Persistent private String userName;
    /**
     * @uml.property name="dateAdded"
     */
    @Persistent private Date dateAdded;
    /**
     * @uml.property name="status"
     */
    @Persistent private String status;

    /**
     * @return
     * @uml.property name="id"
     */
    public Key getId() {
        return id;
    }
    /**
     * @param id
     * @uml.property name="id"
     */
    public void setId(Key id) {
        this.id = id;
    }
    /**
     * @return
     * @uml.property name="title"
     */
public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public String getUserName() {
    return userName;
}

public void setUserName(String userName) {
    this.userName = userName;
}

public Date getDateAdded() {
    return dateAdded;
}

public void setDateAdded(Date dateAdded) {
    this.dateAdded = dateAdded;
}
public String getStatus() {
    return status;
}

public void setStatus(String status) {
    this.status = status;
}

package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;

@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class LeadSource implements java.io.Serializable {
    private static final long serialVersionUID = 1L;

    private Key id;

    private String userName;

    private String name;

    public Key getId() {
        return id;
    }

    public void setId(Key id) {
        this.id = id;
    }

    public String getName() {
    }

    public String getUserName() {
    }

    public String setName(String name) {
    }

    public String getName() {
    }

    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;

    @Persistent
    @Property(name = "id")
    @PrimaryKey
    private String userName;

    @Persistent
    @Property(name = "name")
    private String name;

    public Key getId() {
        return id;
    }

    public void setId(Key id) {
        this.id = id;
    }

    public String getName() {
    }

    public String getUserName() {
    }

    public String setName(String name) {
    }

    public String getName() {
    }
return name;

/**
 * @param name
 * @uml.property name="name"
 */
public void setName(String name) {
    this.name = name;
}

/**
 * @return
 * @uml.property name="userName"
 */
public String getUserName() {
    return userName;
}

/**
 * @param userName
 * @uml.property name="userName"
 */
public void setUserName(String userName) {
    this.userName = userName;
}

package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
 * @author  Kashif Khan
 */
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class LeadSourceSales implements java.io.Serializable{
    private static final long serialVersionUID = 1L;
    /**
     * @uml.property name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property name="leadSrcId"
     */
    @Persistent private Integer leadSrcId;
    /**
     * @uml.property name="saleId"
     */
    @Persistent private Integer saleId;

    /**
     * @return
     * @uml.property name="id"
     */
    public Key getId() {
        return id;
    }
}
LogCalls.java
package gcrm.model;
import java.util.Date;
import com.google.appengine.api.datastore.Key;
import javax.jdo.annotations.*;

@SuppressWarnings("serial")
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class LogCalls implements java.io.Serializable{
    /**
     * @author Kashif Khan
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property name="leadSrcId"
     */
    private Integer leadSrcId;
    /**
     * @uml.property name="saleId"
     */
    private Integer saleId;

    /**
     * @param id
     * @uml.property name="id"
     */
    public void setId(Key id) {
        this.id = id;
    }
    /**
     * @return
     * @uml.property name="leadSrcId"
     */
    public Integer getLeadSrcId() { return leadSrcId; }
    /**
     * @param leadSrcId
     * @uml.property name="leadSrcId"
     */
    public void setLeadSrcId(Integer leadSrcId) {
        this.leadSrcId = leadSrcId;
    }
    /**
     * @return
     * @uml.property name="saleId"
     */
    public Integer getSaleId() { return saleId; }
    /**
     * @param saleId
     * @uml.property name="saleId"
     */
    public void setSaleId(Integer saleId) {
        this.saleId = saleId;
    }
}
@Persistent private String userName;
/**
 * @uml.property name="callAgenda"
 */
@Persistent private String callAgenda;
/**
 * @uml.property name="callTime"
 */
@Persistent private Date callTime;
/**
 * @uml.property name="callDuration"
 */
@Persistent private String callDuration;
/**
 * @uml.property name="opportunityId"
 */
@Persistent private Integer opportunityId;

/**
 * @return
 * @uml.property name="id"
 */
public Key getId() {
    return id;
}
/**
 * @param id
 * @uml.property name="id"
 */
public void setId(Key id) {
    this.id = id;
}
/**
 * @return
 * @uml.property name="callAgenda"
 */
public String getCallAgenda() {
    return callAgenda;
}
/**
 * @param callAgenda
 * @uml.property name="callAgenda"
 */
public void setCallAgenda(String callAgenda) {
    this.callAgenda = callAgenda;
}
/**
 * @return
 * @uml.property name="callTime"
 */
public Date getCallTime() {
    return callTime;
}
/**
 * @param callTime
 * @uml.property name="callTime"
 */
public void setCallTime(Date callTime) {
    this.callTime = callTime;
}

/**
 * @return
 * @uml.property  name="callDuration"
 */
public String getCallDuration() {
    return callDuration;
}

/**
 * @param callDuration
 * @uml.property  name="callDuration"
 */
public void setCallDuration(String callDuration) {
    this.callDuration = callDuration;
}

/**
 * @return
 * @uml.property  name="opportunityId"
 */
public Integer getOpportunityId() {
    return opportunityId;
}

/**
 * @param opportunityId
 * @uml.property  name="opportunityId"
 */
public void setOpportunityId(Integer opportunityId) {
    this.opportunityId = opportunityId;
}

/**
 * @return
 * @uml.property  name="userName"
 */
public String getUserName() {
    return userName;
}

/**
 * @param userName
 * @uml.property  name="userName"
 */
public void setUserName(String userName) {
    this.userName = userName;
}

Meetings.java
package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
 * @author  Kashif Khan
 */
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Meetings implements java.io.Serializable{

}
private static final long serialVersionUID = 1L;
/**
 * @uml.property name="id"
 */
@PrimaryKey
@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
private Key id;
/**
 * @uml.property name="userName"
 */
@Persistent private String userName;
/**
 * @uml.property name="title"
 */
@Persistent private String title;
/**
 * @uml.property name="responsible"
 */
@Persistent private String responsible;
/**
 * @uml.property name="date"
 */
@Persistent private String date;
/**
 * @uml.property name="duration"
 */
@Persistent private String duration;
/**
 * @uml.property name="location"
 */
@Persistent private String location;
/**
 * @uml.property name="status"
 */
@Persistent private Integer status;
/**
 * @uml.property name="priority"
 */
@Persistent private Integer priority;
/**
 * @uml.property name="confirmCancel"
 */
@Persistent private Integer confirmCancel;
/**
 * @uml.property name="opportunityId"
 */
@Persistent private Integer opportunityId;
/**
 * @return
 * @uml.property name="id"
 */
public Key getId() {
    return id;
}
/**
 * @param id
 */
return id;
public void setId(Key id) {
    this.id = id;
}

public String getUserName() {
    return userName;
}

public void setUserName(String userName) {
    this.userName = userName;
}

public String getResponsible() {
    return responsible;
}

public void setResponsible(String responsible) {
    this.responsible = responsible;
}

public String getDate() {
    return date;
}

public void setDate(String date) {
    this.date = date;
}

public String getDuration() {
    return duration;
}
public void setDuration(String duration) {
    this.duration = duration;
}

public String getLocation() {
    return location;
}

public void setLocation(String location) {
    this.location = location;
}

public Integer getPriority() {
    return priority;
}

public void setPriority(Integer priority) {
    this.priority = priority;
}

public Integer getConfirmCancel() {
    return confirmCancel;
}

public void setConfirmCancel(Integer confirmCancel) {
    this.confirmCancel = confirmCancel;
}

public Integer getOpportunityId() {
    return opportunityId;
}
```java
* @uml.property name="opportunityId"
*/
public void setOpportunityId(Integer opportunityId) {
    this.opportunityId = opportunityId;
}
/**
* @return
* @uml.property name="title"
*/
public String getTitle() {
    return title;
}
/**
* @param title
* @uml.property name="title"
*/
public void setTitle(String title) {
    this.title = title;
}
/**
* @return
* @uml.property name="status"
*/
public Integer getStatus() {
    return status;
}
/**
* @param status
* @uml.property name="status"
*/
public void setStatus(Integer status) {
    this.status = status;
}
}

Opportunity.java
package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
* @author Kashif Khan
*/
@SuppressWarnings("serial")
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Opportunity implements java.io.Serializable {
    /**
     * @uml.property name="id"
     */
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    /**
     * @uml.property name="userName"
     */
    @Persistent private String userName;
    /**
     * @uml.property name="opportunityName"
     */
    @Persistent private String opportunityName;
```
/**
 * @Persistent private String opportunityName;
 /**
 * @uml.property name="dateAdded"
 */
 @Persistent private String dateAdded;
 /**
 * @uml.property name="expectedSales"
 */
 @Persistent private String expectedSales;
 /**
 * @uml.property name="daysOnStage"
 */
 @Persistent private Short daysOnStage;
 /**
 * @uml.property name="updatedDays"
 */
 @Persistent private String updatedDays;
 /**
 * @uml.property name="description"
 */
 @Persistent private String description;
 /**
 * @uml.property name="notes"
 */
 @Persistent private String notes;
 /**
 * @uml.property name="estimatedCloseDate"
 */
 @Persistent private String estimatedCloseDate;
 /**
 * @uml.property name="estimatedRevenue"
 */
 @Persistent private String estimatedRevenue;

/**
 * @return
 * @uml.property name="id"
 */
 public Key getId() {
     return id;
 }

/**
 * @param id
 * @uml.property name="id"
 */
 public void setId(Key id) {
     this.id = id;
 }

/**
 * @return
 * @uml.property name="userName"
 */
 public String getUserName() {
     return userName;
 }
/**
 * @param userName
 * @uml.property  name="userName"
 */
public void setUserName(String userName) {
    this.userName = userName;
}
/**
 * @return
 * @uml.property  name="opportunityName"
 */
public String getOpportunityName() {
    return opportunityName;
}
/**
 * @param opportunityName
 * @uml.property  name="opportunityName"
 */
public void setOpportunityName(String opportunityName) {
    this.opportunityName = opportunityName;
}
/**
 * @return
 * @uml.property  name="dateAdded"
 */
public String getDateAdded() {
    return dateAdded;
}
public void setTimeAdded(String dateAdded) {
    this.dateAdded = dateAdded;
}
/**
 * @param expectedSales
 * @uml.property  name="expectedSales"
 */
public void setExpectedSales(String expectedSales) {
    this.expectedSales = expectedSales;
}
/**
 * @return
 * @uml.property  name="expectedSales"
 */
public String getExpectedSales() {
    return expectedSales;
}
/**
 * @return
 * @uml.property  name="updatedDays"
 */
public String getUpdatedDays() {
    return updatedDays;
}
* @param updatedDays
* @uml.property  name="updatedDays"
*/
public void setUpdatedDays(String updatedDays) {
    this.updatedDays = updatedDays;
}
/**
 * @return
 * @uml.property  name="daysOnStage"
*/
public Short getDaysOnStage() {
    return daysOnStage;
}
/**
 * @param daysOnStage
 * @uml.property  name="daysOnStage"
*/
public void setDaysOnStage(Short daysOnStage) {
    this.daysOnStage = daysOnStage;
}
/**
 * @return
 * @uml.property  name="description"
*/
public String getDescription() {
    return description;
}
/**
 * @param description
 * @uml.property  name="description"
*/
public void setDescription(String description) {
    this.description = description;
}
/**
 * @return
 * @uml.property  name="notes"
*/
public String getNotes() {
    return notes;
}
/**
 * @param notes
 * @uml.property  name="notes"
*/
public void setNotes(String notes) {
    this.notes = notes;
}
/**
 * @return
 * @uml.property  name="estimatedRevenue"
*/
public String getEstimatedRevenue() {
    return estimatedRevenue;
}
public void setEstimatedRevenue(String estimatedRevenue) {
    this.estimatedRevenue = estimatedRevenue;
}

public String getEstimatedCloseDate() {
    return estimatedCloseDate;
}

public void setEstimatedCloseDate(String estimatedCloseDate) {
    this.estimatedCloseDate = estimatedCloseDate;
}

Sales.java
package gcrm.model;
import java.sql.Timestamp;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;

public class Sales implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    private Key id;
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key id;
    @Persistent private String userName;
    @Persistent private String saleName;
    @Persistent private String saleAmount;
    @Persistent private Timestamp saleTime;
    @Persiste...
@Persistent private Timestamp timeAdded;
/**
 * @uml.property name="saleItem"
 *
 */
@Persistent private String saleItem;
/**
 * @uml.property name="saleQuantity"
 *
 */
@Persistent private String saleQuantity;
/**
 * @uml.property name="saleDiscount"
 *
 */
@Persistent private String saleDiscount;
/**
 * @uml.property name="accountName"
 *
 */
@Persistent private String accountName;
/**
 * @uml.property name="saleRep"
 *
 */
@Persistent private String saleRep;
/**
 * @uml.property name="saleStatus"
 *
 */
@Persistent private String saleStatus;
/**
 * @uml.property name="netProfit"
 *
 */
@Persistent private String netProfit;

/**
 * @return
 * @uml.property name="id"
 */
public Key getId() {
    return id;
}
/**
 * @param id
 * @uml.property name="id"
 */
public void setId(Key id) {
    this.id = id;
}
/**
 * @return
 * @uml.property name="userName"
 */
public String getUsername() {
    return userName;
}
/**
 * @param userName
 * @uml.property name="userName"
 */
public void setUserName(String userName) {
    this.userName = userName;
}
/**
 * @return
 * @uml.property name="saleName"
 */
public String getSaleName() {
    return saleName;
}
/**
 * @param saleName
 * @uml.property name="saleName"
 */
public void setSaleName(String saleName) {
    this.saleName = saleName;
}
/**
 * @return
 * @uml.property name="saleAmount"
 */
public String getSaleAmount() {
    return saleAmount;
}
/**
 * @param saleAmount
 * @uml.property name="saleAmount"
 */
public void setSaleAmount(String saleAmount) {
    this.saleAmount = saleAmount;
}
/**
 * @return
 * @uml.property name="saleTime"
 */
public Timestamp getSaleTime() {
    return saleTime;
}
/**
 * @param saleTime
 * @uml.property name="saleTime"
 */
public void setSaleTime(Timestamp saleTime) {
    this.saleTime = saleTime;
}
/**
 * @return
 * @uml.property name="timeAdded"
 */
public Timestamp getTimeAdded() {
    return timeAdded;
}
/**
 * @param timeAdded
 * @uml.property name="timeAdded"
 */
public void setTimeAdded(Timestamp timeAdded) {
    this.timeAdded = timeAdded;
}
/**
 * @return
 * @uml.property name="saleItem"
 */
public String getSaleItem() {
    return saleItem;
}
/**
 * @param saleItem
 * @uml.property name="saleItem"
 */
public void setSaleItem(String saleItem) {
    this.saleItem = saleItem;
}
/**
 * @return
 * @uml.property name="saleQuantity"
 */
public String getSaleQuantity() {
    return saleQuantity;
}
/**
 * @param saleQuantity
 * @uml.property name="saleQuantity"
 */
public void setSaleQuantity(String saleQuantity) {
    this.saleQuantity = saleQuantity;
}
/**
 * @return
 * @uml.property name="saleDiscount"
 */
public String getSaleDiscount() {
    return saleDiscount;
}
/**
 * @param saleDiscount
 * @uml.property name="saleDiscount"
 */
public void setSaleDiscount(String saleDiscount) {
    this.saleDiscount = saleDiscount;
}
/**
 * @return
 * @uml.property name="accountName"
 */
public String getAccountName() {
    return accountName;
}
/**
 * @param accountName
 * @uml.property name="accountName"
 */
public void setAccountName(String accountName) {
public void setAccountName(String accountName) {
    this.accountName = accountName;
}

/**
 * @return
 * @uml.property  name="saleRep"
 */
public String getSaleRep() {
    return saleRep;
}

/**
 * @param saleRep
 * @uml.property  name="saleRep"
 */
public void setSaleRep(String saleRep) {
    this.saleRep = saleRep;
}

/**
 * @return
 * @uml.property  name="saleStatus"
 */
public String getSaleStatus() {
    return saleStatus;
}

/**
 * @param saleStatus
 * @uml.property  name="saleStatus"
 */
public void setSaleStatus(String saleStatus) {
    this.saleStatus = saleStatus;
}

/**
 * @return
 * @uml.property  name="netProfit"
 */
public String getNetProfit() {
    return netProfit;
}

/**
 * @param netProfit
 * @uml.property  name="netProfit"
 */
public void setNetProfit(String netProfit) {
    this.netProfit = netProfit;
}

SalesRep.java
package gcrm.model;
import javax.jdo.annotations.*;
import com.google.appengine.api.datastore.Key;
/**
 * @author  Kashif Khan
 */
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class SalesRep implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
}
/**
 * @uml.property name="id"
 */
@PrimaryKey
@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
private Key id;
/**
 * @uml.property name="userName"
 */
@Persistent private String userName;
/**
 * @uml.property name="name"
 */
@Persistent private String name;
/**
 * @uml.property name="phone"
 */
@Persistent private String phone;

/**
 * @return
 * @uml.property name="id"
 */
public Key getId() {
    return id;
}
/**
 * @param id
 * @uml.property name="id"
 */
public void setId(Key id) {
    this.id = id;
}
/**
 * @return
 * @uml.property name="name"
 */
public String getName() {
    return name;
}
/**
 * @param name
 * @uml.property name="name"
 */
public void setName(String name) {
    this.name = name;
}
/**
 * @return
 * @uml.property name="phone"
 */
public String getPhone() {
    return phone;
}
public void setPhone(String phone) {
    this.phone = phone;
}
/**
 * @return
 * @uml.property  name="userName"
 */
public String getUserName() {
    return userName;
}
/**
 * @param userName
 * @uml.property  name="userName"
 */
public void setUserName(String userName) {
    this.userName = userName;
}
/**
 * @return
 * @uml.property name="userName"
 */
public String getUserName() {
    return userName;
}

/**
 * @param userName
 * @uml.property name="userName"
 */
public void setUserName(String userName) {
    this.userName = userName;
}

/**
 * @return
 * @uml.property name="password"
 */
public String getPassword() {
    return password;
}

/**
 * @param password
 * @uml.property name="password"
 */
public void setPassword(String password) {
    this.password = password;
}

/**
 * @return
 * @uml.property name="name"
 */
public String getName() {
    return name;
}

/**
 * @param name
 * @uml.property name="name"
 */
public void setName(String name) {
    this.name = name;
}

/**
 * @return
 * @uml.property name="enabled"
 */
public Integer getEnabled() {
    return enabled;
}

/**
 * @param enabled
 * @uml.property name="enabled"
 */
public void setEnabled(Integer enabled) {
    this.enabled = enabled;
}
/**
 * @return
 * @uml.property  name="id"
 */
public Key getId() {
    return id;
}
/**
 * @param id
 * @uml.property  name="id"
 */
public void setId(Key id) {
    this.id = id;
}