Towards Process Support for Migrating Applications to Cloud Computing

Muhammad Aufeef Chauhan
Software & Systems Group
IT University of Copenhagen
DK-2300 Copenhagen S, Denmark
muac@itu.dk

Muhammad Ali Babar
Software & Systems Group
IT University of Copenhagen
DK-2300 Copenhagen S, Denmark
malibaba@itu.dk

***Abstract* — Cloud computing is an active area of research for industry and academia. There are a large number of organizations providing cloud computing infrastructure and services. In order to utilize these infrastructure resources and services, existing applications need to be migrated to clouds. However, a successful migration effort needs well-defined process support. It does not only help to identify and address challenges associated with migration but also provides a strategy to evaluate different platforms in relation to application and domain specific requirements. This paper present a process framework for supporting migration to cloud computing based on our experiences from migrating an Open Source System (OSS), Hackystat, to two different cloud computing platforms. We explained the process by performing a comparative analysis of our efforts to migrate Hackystate to Amazon Web Services and Google App Engine. We also report the potential challenges, suitable solutions, and lesson learned to support the presented process framework. We expect that the reported experiences can serve guidelines for those who intend to migrate software applications to cloud computing.**

Keywords: Cloud Computing, Software Migration, Open Source Software, Service Oriented Architecture, Amazon Elastic Cloud (AEC), Google App Engine (GAE).

I. INTRODUCTION

Cloud computing has become an active area of practice and research over the last few years. One of the most attractive aspects of Cloud computing is on demand scalability of resources without upfront investments [1-3]. Individuals and organizations can access a large pool of shared resources based on pay per use model [4]. Cloud computing paradigm is broadly classified into service and deployment models. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) are the commonly known service models; whereas public, private, hybrid, community and virtual clouds are the categories of the deployment models. Along with its benefits, cloud computing also has associated challenges. Many of those are related to its adoption. One of these challenges is migration of existing application to cloud computing. There are few studies as reported in [4-5] that address the evaluation of different cloud platforms for performance indicators. However, there is not enough literature available on process support for migrating existing applications to cloud environment.

In an effort to fill this gap, we have been incrementally developing and assessing a framework for supporting migration to cloud computing. The migration process framework presented in this paper has been incrementally developed by formalizing the migration steps that we reported in [6-7], extending the work for different types of cloud platforms and verifying the process on multiple flavours of cloud environments offered by different vendors. We have also introduced additional activities associated with evaluation of cloud environments against quality characteristics. The development of the presented process framework has also sought guidance from methodologies for migrating applications to Service Oriented Architectures (SOA) [8-11]. We illustrate different activities involved in the migration process by reporting a comparative analysis of our efforts to migrate an open source software to two different cloud platforms: Amazon Web Services (AWS) and Google App Engine (GAE). We assert that the reported migration process can be followed as a meta process for streamlining the migration activities. We have also described how existing architecture analysis methodologies (such as [8-11]) can be leveraged during the analysis phase. Our observations from experimentations enable us to provide a set of generic guidelines to support the migration process described in this paper; however, these guidelines can also be used independent of the presented process.

This paper makes two important contributions to the growing body of knowledge that can be leveraged to support the efforts aimed at migrating legacy applications to cloud computing infrastructures.

- It reports a process framework for supporting migration to cloud computing and illustrates the use of the presented process by reporting a comparative analysis of migrating an application to AWS vs. GAE.
- It reports some useful guidelines based on our observations and lessons learned from migrating an application to two different cloud infrastructures.

II. RELATED WORK

Cloud computing embraces many SOA concepts and technologies; including Service Architecture Engineering (SAE), Service Oriented Analysis and Design (SOAD), Service Oriented Development of Applications (SODA) and Service Oriented Modeling and Architecture (SOMA) [5]. Mohagheghi and Sæther [5] have reported challenges associated with migrating applications to service cloud platforms. The reported challenges include identifying the advantages of migrating to cloud, enhancing software architecture, modifying data management schemas, addressing quality of service (QoS) and extra functional

requirements, verifying the cloud-based solution and redefining business models for pay per use pricing scheme.

Razavian and Lago [12-13] have presented a framework for migrating legacy systems to SOA called SOA-MF. The migration process begins with recovering the lost abstractions and eliciting the legacy fragments. The abstraction of a legacy system is transformed into service-based abstraction. Finally the target system is modified using service based abstractions and new requirements. They have also presented an overview of SOA migration approaches in [12] addressing *code translation* focusing on wrapping the whole code inside a web service without decomposing; *service identification* focusing on analyzing code and architecture analysis for services; *business model transformation* focusing on meta process for migration and business process reengineering approaches; *design composition element transformation* focusing on basic and composite design elements; *pattern based composite transformation* focusing on altering system architecture into the service based architecture; and *forward engineering with gap analysis* focusing on basic and composite design elements followed by analyzing and designing services.

Cetin and colleagues have presented six step roadmap for migrating systems to services referred as MASHUP [14]. The six steps are: modeling of business requirements, analysis of existing legacy systems, identification of services by mapping business requirements to system components, services composition for satisfying business requirements, defining service level agreements and finally deploying services after implementation.

Service-Oriented Migration and Reuse Technique (SMART) approach has been proposed by SEI for analyzing legacy systems to determine their capability to be exposed as services [11]. This approach establishes the context for migration of services. The key activities in this stage are: identification of system stakeholders, identification of components and determining their feasibility for migration to SOA, and identifying migration concerns. In next stage, the capabilities of an existing system are determined by making a list of components and migration issues related with them. Finally a list of potential services is prepared for gap analysis that takes migration issues into consideration. Bianco et al. have proposed a SOA evaluation methodology in terms of number of design decision [8], which focus on target platforms, communication between services, service granularity (coarse- or fine-grained), exceptional handling and fault recovery strategy, service authentication and authorization, service discovery, service integration with other systems and approach for service versioning.

All the methodologies supporting migration from legacy to SOA discussed in this section provide some guidance that can also be useful for designing systems and services for cloud. However they are not specifically addressing some of the key issues involved in migrating to cloud computing such as analysis of target cloud environments against specific application's requirements, evaluation of platforms for cloud specific quality attributes of SaaS applications, the impact of a target cloud platform on each of the migration activities, and the potential influence of different services offered by cloud environments on migration activities and on the new architecture of a system to be migrated. The migration process support proposed in this paper aims at addressing these issues and provides comprehensive guidelines that support evaluation of application architecture with respect to features of a target cloud environment.

III. MIGRATION PROCESS

In this section, we describe the main steps of the process for migrating to cloud computing. This process is aimed at providing guidelines for migrating systems to clouds.

A. Requirements Identification

First activity is associated with identification of the business requirements that initiate the migration process. This step aims at elaborating the main motivation for migration and the objectives to be achieved. It also provides a foundation for the changes required in a system and describes system's behavior specific to cloud computing environment. The business requirements are then broken down into more specific functional and/or extra functional requirements that can be analyzed qualitatively or quantitatively.

Artifacts: A set of requirements is produced as a result of this activity.

Actors: Business analysts are main actors of this activity.

B. Identification of Potential Cloud Hosting Environments

This activity purports to identify a set of potential cloud computing platforms based on a project's nature, data confidentiality and sensitivity requirements, budget constraints and long-term organizational objectives. The selection of the potential clouds is also influenced by a company's long-term commitment with specific cloud providers. The features of the potential cloud platforms are also explored in this activity.

Artifacts: A list of potential cloud environments is produced along with their respective features.

Actors: Project managers and system architects are main actors in this activity.

C. Analyzing Applications' Compatibility with Potential Cloud Environments

During this activity, an application is analyzed to assess its compatibility with the potential cloud computing environments. This activity purports to identify the changes required to resolve incompatibility issues between a system and a target platform. For example, it may be the case that a target PaaS cloud does not support frameworks or specific technologies being used by an application. If such issues are identified, then these need to be resolved first.

Artifacts: This activity produces trade-off analysis documents specifying pros and cons of the environments identified during the previous activity and their suitability with applications from business and technology perspectives.

Actors: Systems analysts and architects are main contributors of this activity.

D. *Identification of Potential Architecture Solutions*

After the identification of the requirements and selecting the potential cloud computing platforms, the requirements are analyzed against each of the potential cloud environments. The application components that are sensitive to different quality attributes (such as security and privacy) are also identified. The proposed solutions are then analyzed for their advantages and disadvantages with respect to the quality attributes. At the end of this activity, a solution is selected that best satisfies the functional as well as quality requirements. In some cases, it may not be possible that one solution satisfies all the requirements. Then a tradeoff analysis is performed and an optimal solution is devised that satisfies the most important requirements.

Artifacts: This activity results in high-level design documents of potential architecture solution.

Actors: System architects are main participants of this activity.

E. *Evaluation of Cloud Environments for Cloud Specific Quality Attributes*

This activity aims at analysing the potential cloud platforms for specific features of a provided service, e.g., SaaS that needs to be supported by a hosting platform. Few examples of these quality requirements are multi-tenancy, decentralized deployment of components on hybrid clouds, interoperability with alternative hosting clouds, and support for programming languages and frameworks. A cloud environment satisfying most important quality requirements is selected for implementation.

Artifacts: This activity results in artifacts providing a map between proposed solutions and support for quality requirements by potential cloud environments.

Actors: System architects are major actors in this activity.

F. *Evaluation of Proposed Solutions and Effected Components Against Target Platforms*

This activity purports to analyze the changes that are proposed to assess how a modified system will be deployed on clouds. This is a very important activity to identify any conflict between a proposed solution and a chosen cloud environment. Other architecture evaluation techniques (such as [10-11]) can be used to support this activity. In case of discrepancies, the activities D and E can be performed again.

Artifacts: This activity results in artifacts describing compatibility map of a proposed solution against a target cloud environment as well as a set of quality characteristics to be supported by a cloud environment.

Actors: System architects are the main actors.

G. *Implementation*

The final activity of this process aims at implementing (and/or re-factoring) the designed solutions for migrating a system to be deployed on a target cloud environment.

Artifacts: This activity results in a working system deployable on selected cloud or combination of clouds.

Actors: System architects and developers are the main contributors to this activity.

Figure 1 shows the activities, their sequence, artifacts produced, artifacts taken as input and relations between the activities of the proposed migration process.

IV. A Case Study of Migration To Amazon EC2 and Google App Engine

This section provides a detailed description of the presented process framework by reporting a comparative case study of migrating an open source software, Hackystat, to two cloud computing environments, Amazon EC2 (AEC) an IaaS cloud and Google App Engine (GAE) a PaaS cloud. This section also reports our experiences and observations from our migration efforts.

Hackystat is an open source framework developed to collect process and product metrics. It also supports the analysis and visualization of these metrics at different levels of abstraction. Figure 2 shows high-level architecture and services of Hackystat version 8 [15]. As shown in figure, the data are sent to Hackystat services using plugins installed on data sources. The data are received

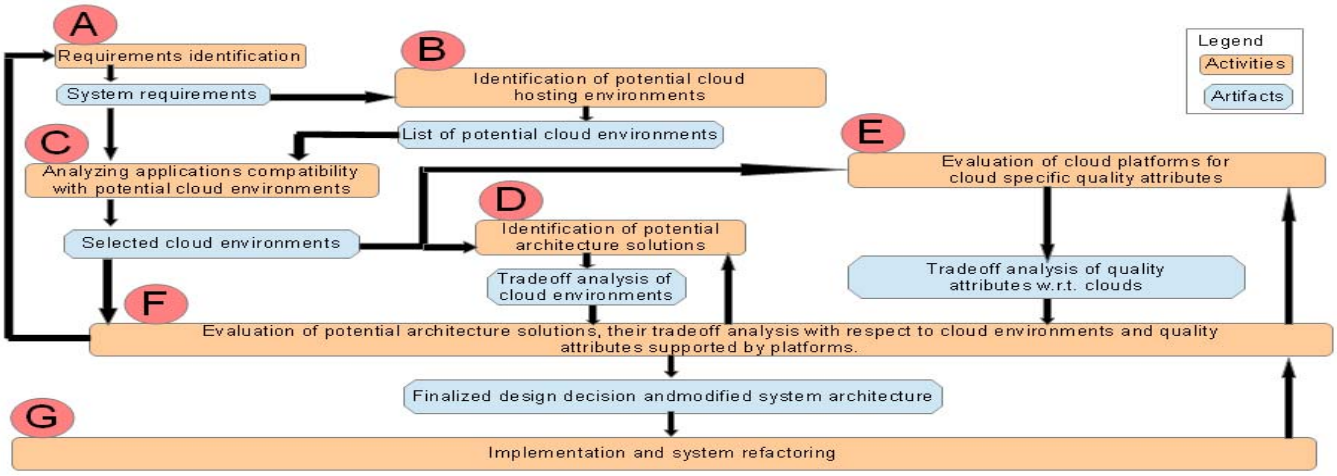

Figure 1: Activities of Migration Process and Generated Artifacts

by Sensorbase, root service of framework; responsible for persistence management and handling of configurations along with notification operations. The other services, DailyProjectData and Telemetry works on top of Sensorbase. These are used to compute daily, weekly, monthly and yearly abstractions of data. ProjectBrowser and TickerTape are client components used to present metrics through graphical user interface and posts information on external applications like Nabaztag Rabbit [16] and Twitter [17]. More detail on Hackystat can be found in [7, 15].

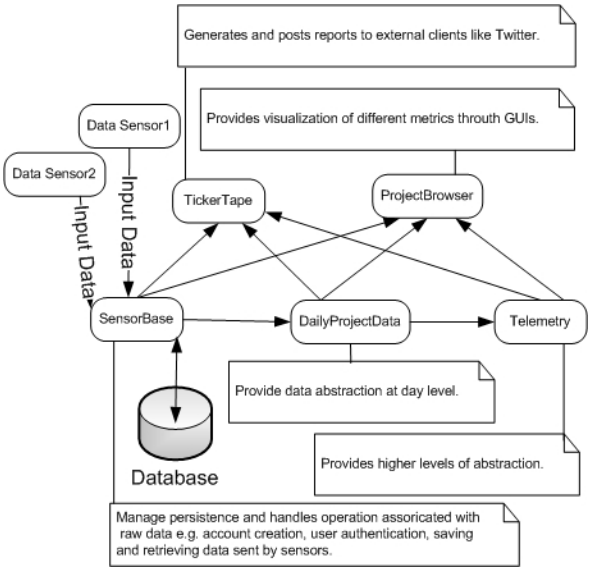

Figure 2: Hackystat version 8

A. *Requirements Identification*

For Hackystat to serve as SaaS, it should have the capability to scale for the required computing and storage resources. This section enlists the requirements of Hakcystat SaaS. Further detail about the requirements analysis and elicitation process can be found in [6-7].

RQ1: System should be able to scale according to specific quality requirements (for example, efficiency, throughput, and response time).

RQ2: System components should be deployable on commercial clouds providers (public cloud) as well as on organizations' private clouds.

RQ3: System should be able to handle unexpected increase in storage requirements by utilizing storage resources and storage services of clouds.

RQ4: End users of the system should have consolidated view of system through unified service interfaces, irrespective of actual deployment configuration.

RQ5: End user operations and service requests should not be affected during system scalability.

RQ6: System should efficiently acquire resources when needed and release those when not required in order to have more cost effective and green solution.

B. *Identification of Potential Cloud Hosting Environments*

The criterion for initial selection of potential cloud hosting environments is driven by business objectives. In the beginning, the cloud providers that satisfy the budgetary constraints of a project, support the respective business domain of the project and meet the legal requirements are selected. As our effort was aimed at exploring the challenges associated with migration efforts, we chose a SaaS cloud i.e., AEC and one PaaS cloud i.e., GAE. Choosing one SaaS and one PaaS provided us with an opportunity to explore the challenges associated with both types of service models.

This step also incorporates analyzing features of potential cloud platforms. We analyzed features of both platforms. AEC provides scalable computing resources [18]. These are associated with *Amazon Elastic Block Storage (EBS)* for persistence. Amazon provides ability to place AEC instances in different supported regions and time zones to deal with system failures and legal constraints on data storage locations. Amazon supports organizations to connect their existing infrastructure with Amazon's resources using *Amazon Virtual Private Cloud*. *Amazon CloudWatch* service can be used to monitor resources consumed by instances and applications deployed on the instances. Metrics for monitoring CPU utilization, disk reads/writes, and network traffic are supported. These metrics can be accessed using web service APIs or command line tools. *Elastic load Balancer* is used to distribute clients' requests among different instances. This service is also used for detecting unhealthy instances and consequently routing requests to healthy ones. There can be one Elastic Load Balancer for whole application or can be separate for each deployment zone. It maintains its operational metrics that can be used by CloudWatch services. Amazon also supports *import and export* of virtual machine images from its cloud.

GAE on the other hand supports application development via its APIs [19-20]. As it is a PaaS, so applications do not have direct access to underlying resources. It supports Java Servlet [21] and Python [22] runtime environments. Its underlying platform takes care of dynamic application scalability and load balancing, sports persistence storage, provides access to email APIs, and supports task scheduling and triggering events at specified times.

C. *Analyzing Applications' Compatibility with Potential Cloud Environments*

We need to evaluate the compatibility of a system with target platforms with respect to technologies being used. During our analysis, we found that configuring AEC instances for supporting Hackystat JDK and Restlet framework requirements was convenient. AEC provide option for configuring resources of virtual instances and install pre-requisite software. We also found out that Hackystat does not have any compatibility issues with AEC and does not need any modification for making it compatible with host platform.

Subsequent, we analyzed GAE for evaluating compatibility of Hackystat with it. During that analysis it was figured out that GAE was not supporting the version of the framework required by Hackystat as Hackystat is

implemented using Reslet 1.0 framework specification, whereas GAE supports a customized version of Restlet 2.0 framework. Moreover, Hackystat has been implemented for deployment on native machines outside application servers. GAE does not support this kind of deployment. GAE also does not support specifying system configuration on properties file. Hence, there was a need for refactoring Hackystat to make it compatible with GAE on three dimensions.

- Refactoring of whole system to make it compatible with Restlet 2.0 specification customized for GAE.
- Move all system configurtion from propertied file into database.
- Change server related implementation of Hackystat so that it can be deployed inside application servers rathar than launching its own server.

D. *Identification of Potential Architecture Solutions*

We have followed guidelines presented in [8, 23] for analyzing requirements and their potential solutions for IaaS and PaaS. This section is focusing on requirements analysis of GAE only and requirements analysis for AEC is discussed only if there have been new featured introduces since we reported our previous studies [6-7].

RQ1 deals with scalability of system with respect to specific quality attributes. As in PaaS, it is the responsibility of the platform for allocating resources to applications according to demand. There is no need to replicate components when utilizing PaaS.

RQ2 addresses portability of components on public and private clouds. It is not easy and sometimes impossible to achieve portability when dealing with PaaS clouds. Each PaaS service platform may have its own specific APIs of frameworks and tools in order to provide seamless scalability of resources. It requires customization of application components before migrating them from one PaaS platform to another. Application migration from one PaaS platform to other is almost impossible without refactoring.

RQ3 focuses on handling scalability and utilization of storage capacity. For GAE, the components that are hendling persistence functionality need to be modified according to persistence management mechanism of platform. GAE supports Datastore [24] and Google Cloud SQL [25]; two different mechanisms for handling data storage. Datastore is schema-less data storage and has a support for atomic transactions. It also has a query engine that is used for information retrieval from storage. Datastore is accessed using customized version of Java Data Objects (JDO) [26] APIs. Applications using Datastore do not need to bother about schema creations. Relations among objects is established when these are persisted. Datastore does not support joins. Depending on configuration, database indexes can be implicitly or explicitly defined. Choosing this option for Hackystat requires refactoring of persistence layer for achieving scalability of date storage. Cloud SQL is Google version of MySQL [27] hosted on Google cloud. It is a relational database and supports all operations associated with relational database management.

TABLE I. REQUIREMENTS VS. ARCHITECTURE DECISIONS

Quality Attributes	Req ID	Architecture Decision	
		Amazon EC2 & S3	***Google App Engine***
Scalability	RQ1	Replication of system services to meet performance requirments.	No action required. Scalability is handled by platform.
	RQ3	Separation of database layer into a new service that utilize platform specific persistence features.	Refactoring of persistence components to make it compatible with Google Datastore persistence.
Portability	RQ2	A wrapper layer is added to ensure platform independent scalability. A separate database service to provide seamless transfer of database layer.	Portability to other platforms is not possible.
Compatibility	RQ4	System features are exposed through origonal REST API. A wrapper layer is added to provice abstraction to services cluster and their deployment configuration.	System features are exposed through origonal REST API.
Reliability and Autonomous Scalability	RQ5	Façade/Waper layer to provide abstraction. Amazon's Elastic Load Balancer ensures autonomous scalability.	Ensured by platform.
Efficient and cost effective deployments	RQ6	Amazon Elastic Load Balancer ensures auto scaling as well as efficient and cost effective deployment configuration.	Deployment of application components on cloud is managed by platform.

Datastore provides more efficient solution on cloud, so we choose this for persistence.

RQ4 addresses importance of compatibility of the system services with external components and clients. In PaaS, scalability is handled by platform itself and there is no replication of components on cloud so this façade is not required. The system's clients can access its features through REST based APIs of components.

RQ5 highlights the importance of continuous system availability during resource provisioning. Extending functionality of Façade introduced to satisfy RQ4 handle this requirement as well. There is also no need to explicitly address this feature for PaaS because scalability and system availability is seamlessly ensured by the platform.

RQ6 seeks a cost effective and green solution so that resources can be efficiently released when no more required in order to have cost effective solution for pay per use model of cloud computing. We chose Amazon's Elastic Load Balancer and Auto Scaling to acquires and release instances according to predefined parameters because it does not require modification in application components. Scaling of applications on GAE is maintained by platform and pricing scheme is based on the number of request received by

application and CPU cycles consumed. This results in an efficient as well as cost effective solution without implementing any additional features in application.

Table I provides a summary of relation between requirements, quality attributes and architecture decisions made to achieve requirements for AEC [6-7] and GAE.

E. Evaluation of Cloud Environments for Cloud Specific Quality Attributes

Other than quality attributes of application addressing extra functional requirements, there are some specific features of SaaS solutions that can leverage advantage of hosting platforms. These features are also referred as architecture requirements of SaaS system. This section describes evaluation of AEC and GAE for these features.

1) Multi-tenancy

This characteristic required application to have isolation between components that perform processing on data sensitive data [28]. AEC provides control over virtual instances and these can be configured according to specific requirements of multi-tenancy. Components that do processing on tenant specific data can be configured by having logical isolations on the same instance as well as by deploying sensitive REST components on separate virtual instances. Amazon S3 storage services provide options to create separate virtual storage units and database instances. This feature can be used to store sensitive data on separate logical databases. GAE Datastore and cloud SQL manages data storage transparent to applications. Applications do not have any control over how data is actually persisted in storage units and make it hard to achieve segregation of sensitive data.

2) De-centralized arrangement of data storage/processing components and Interoperability with alternative host solutions

Support for de-centralized arrangement of data storage and processing components enable application to store and process sensitive data on private cloud and offloading storage and processing of non-sensitive data on public clouds [29]. Interoperability of platform is also an important factor to consider while selecting a cloud platform for software expected to evolve over a long period of time [30]. There are few open source PaaS solutions like Eucalyptus [31] that provides compatible alternative to AEC on private cloud. Availability of these solutions makes AEC a good choice for applications requiring de-centralized arrangements of components and hybrid cloud deployment models. On the alternate side, GAE does not have any alternative available that can host parts of application on private cloud and fails to satisfy such arrangements.

3) Support for programming languages and application development frameworks

Platform support for programming languages and frameworks is an important factor to consider. AEC is much flexible and provides flexibility to have customized configuration for supporting multiple programming languages and frameworks. On the other hand, GAE has only support for two existing programming languages: Java and Python; and platform specific Go programming language. It only supports specific set of frameworks with customized APIs (e.g. Restlet).

4) Resource customization with respect to Service Level Agreements (SLA)

Customization and provisioning of resources according to SLAs is helpful in achieving resources according to specific application's requirements [32]. AEC provides option to have customized CPU, storage and network resources. GAE provides support for autonomous scalability; however, there is no explicit support for specifying scalability parameters and customizing network topologies.

F. Evaluation of Proposed Solutions and Effected Components against Target Platforms

Once system is compatible with target cloud platform, different components of system are evaluated to achieve quality attributes described in Table I.

Replication of components is proposed as a medium to achieve scalability on AEC. GAE platform handles scalability of components itself without any change in application. GAE is more suitable cloud platform in terms of scalability.

The wrapper layer for AEC solution ensures portability and seamless access by client components. A separate database layer makes it possible to utilize different services offered by Amazon S3 storage and database services as needed, and supports portability. Components developed for GAE are not portable to other platforms.

Compatibility with external system components is positively addressed by both AEC and GAE because system interfaces are exposed through same REST interfaces. Reliability needs to be handled by application in AEC with combination of AEC Load Balancer. In GAE, platform is totally responsible for providing reliability and autonomous scalability. To handle scalability, AEC has a load balancer that provides flexibility to define scaling rules based on parameters related to CPU usage, disk access space and access frequency, and network bandwidth utilized by application. This provides more control over scaling schemes. GAE does not provide support for this kind of customization.

TABLE II. IMPACT OF PLATFORM ON QUALITY ATTRIBUTES

Quality Attribute	**Platform**	
	Amazon Web Services	***Google App Engine***
Scalability	O	+
Portability	+	-
Compatibility	+	+
Reliability and Autonomous Scaling	+	+
Customization of Efficient and Cost Effective Deployment Configurations	+	-

Table II shows list of quality attributes along with if these are positively or negatively addressed in AEC and GAE. Plus sign (+) represent if platform has a positive impact on quality attribute, negative sign (-) shows if platform has negative impact on quality attribute and circle (O) shows if

platform does not have any impact and it is responsibility of application to address respective characteristic.

G. Implementation and Architecture Overview

Hackystat migration on GAE requires more implementation relate challenges including refactoring of whole data base layer in Sensorbase components to incorporate GAE's datastore feature. GAE also required refactoring of application to make it compatible with Restlert 2.0 version supported by GAE as well as changing the Mailer component (for emails) inside Sensorbase to utilize email services or Gmail. We also made some minor modification into application structure because of limitations imposed by GAE framework. For example, GAE framework does not support applications to write files on server (e.g. properties files). We have implemented selected use cases to verify our findings from architecture analysis of the application.

Figure 3 presents architecture of the modified system on AEC [7] and GAE. The system on AEC cloud requires less refactoring. Dividing system into more modules and adding an additional layer for orchestration can make it ready for deployment on IaaS. Implementation details for AEC can be found in [7]. For GAE, which is a specific example of PaaS, it requires few architecture level modification and more challenges associated with implementation and refactoring of the application.

V. CONCLUSIONS

We have presented and explained a process for migrating software applications to cloud computing. This process has been incrementally developed through the experiences and insights gained by migrating an open source SOA based application, Hackystat, to two types of cloud infrastructures, Amazon an IaaS and Google App Engine a PaaS. The presented process has been developed with specific focus on migrating applications to SaaS. Main activities involved in the migration process include identification of requirements and potential cloud platforms, analyzing application compatibility with potential cloud environments, identification of potential architecture solutions, evaluation of cloud environments for cloud specific quality attributes, tradeoff analysis of potential architecture solutions, selection of architecture modifications to be incorporated and refactoring of the system to incorporate new architecture modifications.

Following are our key observations from our migration efforts. We expect that these observations can serve as guidelines for those who intend to migrating software intensive applications to cloud computing infrastructures.

- The systems that consist of stateless components e.g. implemented using REST architectures style can easily be scalable without significant refactoring as scalability can be achieved at application level by replicating components. It makes Amazon IaaS the best candidate for such systems. In case of applications with stateful components, Google App Engine can be more suitable platform because it handles scalability in a manner that is transparent to applications.
- The suitability of potential cloud environments for security and sensitivity requirements of application and data to form hybrid clouds can play a vital role in success of application hosted on cloud. For PaaS environments like Google App Engine, it is not possible to have hybrid deployment; whereas IaaS like Amazon EC2 provides solutions for hybrid models because of availability of alternative solutions for private clouds and customization of network topologies.

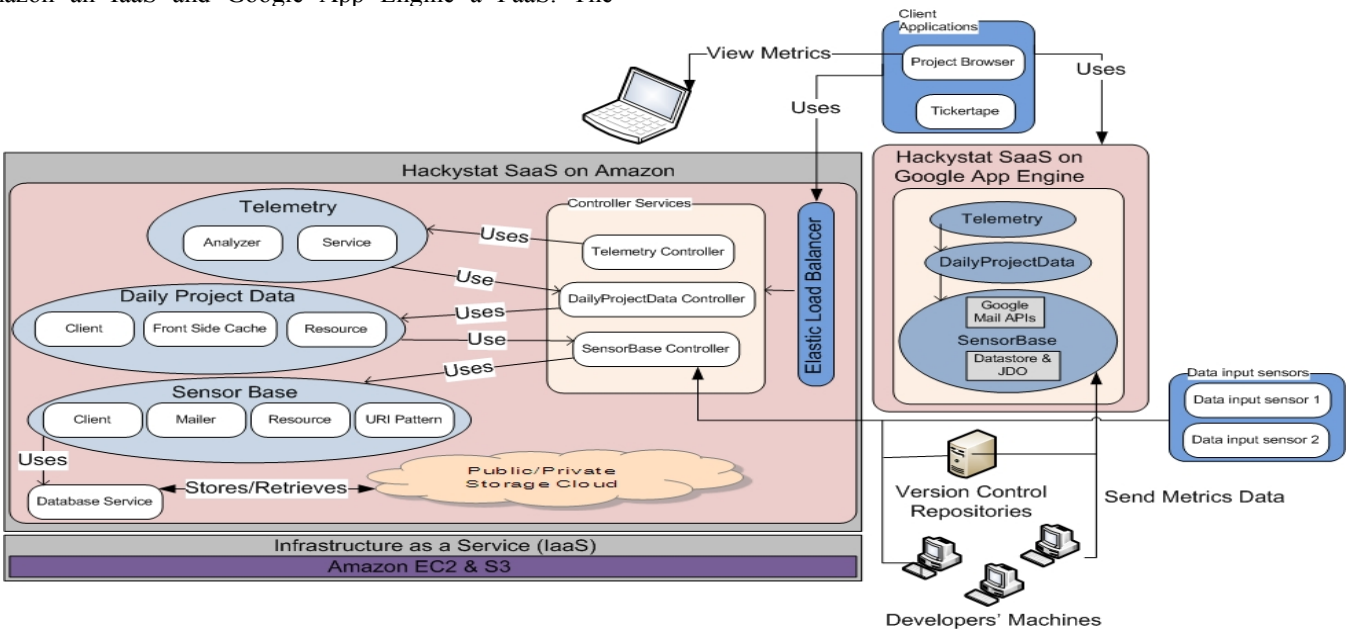

Figure 3: Modified Architecture for Amazon EC2 and Google App Engine

- Every cloud environment addresses a specific set of cloud quality attributes more than others. Aligning quality attributes of application with quality attributes of a particular cloud environment can make the migration process easy. It can also help to avoid anomalies and issues related to the satisfaction of SLAs.
- A widespread adoption of PaaS environments for migrating existing system on them would highly depend upon the available support for programming languages and frameworks by PaaS providers.
- The organizations seeking advantages of PaaS clouds would require long term commitments with platforms because of unavailability of alternative solutions. It will not only tie PaaS clients with it but also make PaaS cloud a crutial factor for the survival of clients organizations.
- The advantages of cloud computing brings associated challenges. One of the challenges is meeting quality of service requirements when applications are using third party infrastructures and services. To meet these challenges efficiently and effectively; a cloud environment should provide support for defining SLA specific quality attributes.

Hackystat, the system we chose for migration, is used for data collection from a large number and different types of data sources as well as data visualization in different forms by varying number of end users at different levels of abstractions. As migrated system is utilizing scalability offered by cloud computing for both storage and computing resources; we assert that the presented migration process can be effectively followed for migration of large-scale industrial systems. In future, we plan to explore the socio-technical aspects of migration to cloud computing to enhance our process framework as well as to identify the requirements for appropriate tools to support the migration to cloud computing. We also intend to experiment with more cloud platforms in order to provide a broader set of guidelines for supporting the migration process.

REFERENCES

[1] P. Louridas, "Up in the Air: Moving Your Applications to the Cloud," *Software, IEEE,* vol. 27, pp. 6-11, 2010.

[2] R. L. Grossman, "The Case for Cloud Computing," *IT Professional,* vol. 11, pp. 23-27, 2009.

[3] Q. Zhang*, et al.*, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications,* vol. 1, pp. 7-18, 2010.

[4] Z. Liang*, et al.*, "Evaluating Cloud Platform Architecture with the CARE Framework," in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, 2010, pp. 60-69.

[5] P. Mohagheghi*, et al.*, "Software Engineering Challenges for Migration to the Service Cloud Paradigm: Ongoing Work in the REMICS Project," in *Services (SERVICES), 2011 IEEE World Congress on*, 2011, pp. 507-514.

[6] M. A. Babar and M. A. Chauhan, "A tale of migration to cloud computing for sharing experiences and observations," presented at the Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing, Waikiki, Honolulu, HI, USA, 2011.

[7] M. A. Chauhan and M. A. Babar, "Migrating Service-Oriented System to Cloud Computing: An Experience Report," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 404-411.

[8] P. Bianco*, et al.*, "Evaluating a Service-Oriented Architecture," *Engineering,* pp. 1-91, 2007.

[9] H. D. Strowd and G. A. Lewis, "T-Check in System-of-Systems Technologies: Cloud Computing," *Software Engineering Institute, Carnegie Mellon University,* vol. Tech Report CMU/SEI-2010-TN-009, 2010.

[10] R. Kazman*, et al.*, "ATAM: Method for Architecture Evaluation," *Software Engineering Institute, Carnegie Mellon University,* vol. TECHNICAL REPORT CMU/SEI-2000-TR-004 ESC-TR-2000-004, 2000.

[11] G. Lewis*, et al.*, "SMART: The Service-Oriented Migration and Reuse Technique," *Software Engineering Institute, Carnegie Mellon University,* vol. CMU/SEI-2005-TN-029, 2005.

[12] M. Razavian and P. Lago, "A Frame of Reference for SOA Migration Towards a Service-Based Internet." vol. 6481, E. Di Nitto and R. Yahyapour, Eds., ed: Springer Berlin / Heidelberg, 2010, pp. 150-162.

[13] M. Razavian and P. Lago, "Towards a conceptual framework for legacy to SOA migration," presented at the Proceedings of the 2009 international conference on Service-oriented computing, Stockholm, Sweden, 2009.

[14] S. Cetin*, et al.*, "Legacy Migration to Service-Oriented Computing with Mashups," in *Software Engineering Advances, 2007. ICSEA 2007. International Conference on*, 2007, pp. 21-21.

[15] "Hackystat, http://code.google.com/p/hackystat/ [August, 2012]."

[16] "Nabaztag, http://www.nabaztag.com/ [August, 2012]."

[17] "Twitter, http://twitter.com/ [August, 2012]."

[18] "Amazon EC2, http://aws.amazon.com/ec2/ [August, 2012]."

[19] "Google App Engine, http://code.google.com/appengine/ [August, 2012]."

[20] "What is Google App Engine?, https://developers.google.com/appengine/docs/whatisgoogleappengine [August, 2012]."

[21] "The Java Servlet Environment, https://developers.google.com/appengine/docs/java/runtime [August, 2012]."

[22] "Python Runtime Environment, https://developers.google.com/appengine/docs/python/runtime [August, 2012]."

[23] J. Yen and W. A. Tiao, "A systematic tradeoff analysis for conflicting imprecise requirements," in *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*, 1997, pp. 87-96.

[24] "Google Datastores, http://code.google.com/appengine/docs/java/datastore/ [August, 2012]."

[25] "Google MySQL, http://code.google.com/edu/tools101/mysql.html [August, 2012]."

[26] "Java Data Objects, http://www.oracle.com/technetwork/java/index-jsp-135919.html [August, 2012]."

[27] "MySQL, http://www.mysql.com/ [August, 2012]."

[28] A. Azeez*, et al.*, "Multi-tenant SOA Middleware for Cloud Computing," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, ed: IEEE, 2010, pp. 458-465 %@ 978-1-4244-8207-8.

[29] H. Ludwig*, et al.*, "REST-based management of loosely coupled services," presented at the Proceedings of the 18th international conference on World wide web, Madrid, Spain, 2009.

[30] E. M. Maximilien*, et al.*, "IBM altocumulus: a cross-cloud middleware and platform," presented at the Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, Orlando, Florida, USA, 2009.

[31] "Eucalyptus, http://www.eucalyptus.com/ [August, 2012]."

[32] C. Chapman*, et al.*, "Software architecture definition for on-demand cloud provisioning," *Cluster Computing,* pp. 1-22, 2011.