A Tale of Migration to Cloud Computing for Sharing Experiences and Observations

Muhammad Ali Babar
IT University of Copenhagen
Rued Langgaards, Vej 7
2300-DK Copenhagen, Denmark
+45 7218 5107
malibaba@itu.dk

Muhammad Aufeef Chauhan
School of Innovation, Design, and Engineering
Mälardalen University
721 78 Västerås, Sweden
+46 733 907353
mcn10003@student.mdh.se

ABSTRACT

Cloud computing is an emerging paradigm, which promises to make the utility computing model comprehensively implemented by using virtualization technologies. An increasing number of enterprises have started providing and using Cloud-enabled infrastructures and services. However, the advancement of cloud computing poses several new challenges to existing methods and approaches to develop and evolve software intensive systems. This paper reports our experiences and observations gained from migrating an Open Source Software (OSS), Hackystat, to cloud computing. We expect that our description of Hackystat's architecture prior and after migration and design decisions can provide some guidance about modifying architecture of a service-based system for cloud computing. Moreover, we also hope that our experiences reported in this paper can contribute to the identification of some research questions for improving software engineering support for developing and evolving cloud-enabled systems.

Categories and Subject Descriptors
D.2.13: Reusable Software – *Reuse model*

General Terms: Management, Theory

Keywords: Cloud Computing, Software Migration, Evolution, Open Source Software, Software Engineering.

1. INTRODUCTION

Cloud Computing has been gaining tremendous attentions as it promises unparallel levels of scalability and flexibility without making upfront investment in setting up and running large scale computing infrastructures and data centers. This relatively new paradigm is expected to enable organizations to scale up or scale down their consumption of information technology based on their consumption patterns [2, 13]. Cloud computing service models can generally be classified into three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [13]. Scalability is a prominent quality attribute for all these categories of systems [6, 19]. The IaaS systems offer on demand storage resources like Amazon's Simple Storage Service (S3) and elastic computing resources like Amazon Compute Elastic (EC2).

Nowadays, any user of Web applications may have been utilizing the platforms or applications deployed on a cloud computing infrastructure as it has many facets and forms. For example, most of the state-of-the-art social networking applications such as Facebook, Twitter, YouTube, and Flickr are reported to be based on high-performance cloud platforms and application engines provided by some of the leading public cloud providers such Google and Amazon. From this situation, it is clear that many companies have started providing and using cloud-enabled services and hundreds of others are seriously considering entering in this foray. However, there are several confusions about this emerging paradigm, which appears to have taken computing and software engineering practitioners in general and researchers in particular by surprise. One of the commonly observed confusion is about what cloud computing means as different people may talk about different things when using the word cloud computing. There have been several definitions of cloud computing [2, 16, 18]. Like many practitioners and researchers, we also use the following definition of cloud computing provided by the US National institute of standards and technology (NIST).

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [16].

Like any emerging paradigm of information technology, cloud computing present several research challenges that needs to be addressed [19]. While the literature reporting research challenges in cloud computing [2, 15, 19] is growing quite fast, there has not been much attention paid to identify and report software engineering related challenges for designing, developing, and deploying solutions with or for cloud computing. We have observed that while several dozens of cloud based services are being offered, the landscape is still fragmented and there are no comprehensive guidelines for designing, developing and deploying solutions to leveraging cloud computing. Moreover, we have also observed that there is hardly any guidance available for migrating existing systems to cloud computing in terms of software engineering aspects.

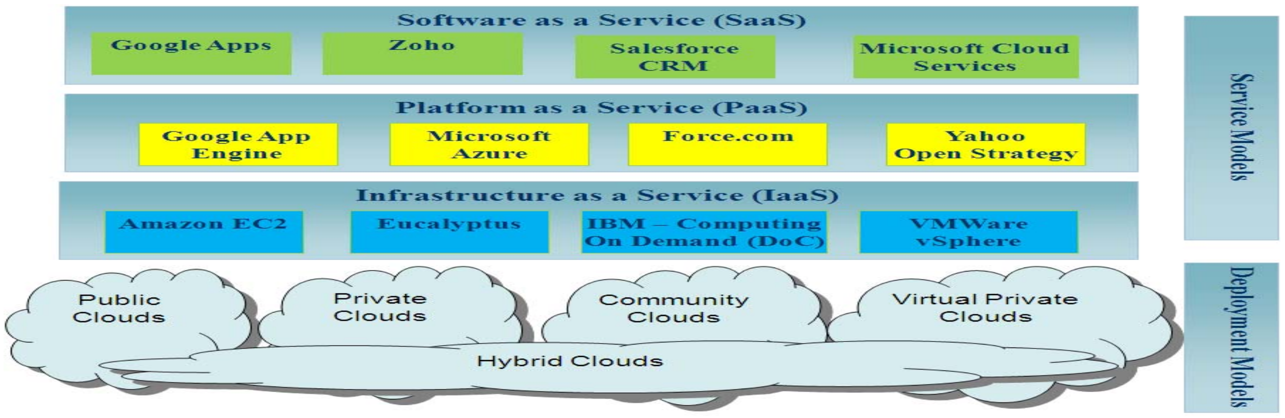

Figure 1: Most commonly known service and deployment models of cloud computing

We experienced and observed the challenges to be reported in this paper when we undertook a research and development project aimed at modernizing an Open Source Software (OSS) framework, Hackystat, for leveraging the flexibility and scalability of using cloud computing paradigm. Since we could not find any specific guideline on how to migrate a software system to cloud computing when we started our project almost a year ago, we decided to support the migration effort through our knowledge and experiences with different methods and approaches (such as [4, 7, 10, 14, 17] developed by software architecture community as we intended to learn whether or not these approaches can be easily applicable to migration projects for cloud computing and what kinds of methodological and technical challenges software engineering researchers can focus on for providing supportive methodologies and techniques for cloud computing. This paper presents the insights we gained while migrating Hackystat to cloud computing.

Organization of the paper is as follows. Second section provides the overview of service and deployment models of cloud computing. Section 3 details our migration project. Section 4 reports our experiences and observations that are expected to contribute to the identifying of software engineering challenges raised by cloud computing. Section 5 finishes the paper.

2. OVERVIEW OF CLOUD COMPUTING

It has been mentioned that one of the key promises of cloud computing is an enormous level of flexibility for scaling up or scaling down software and hardware infrastructures without huge upfront investments. Hence, it is expected that any cloud-based infrastructure should have three characteristics: ability to acquire transactional resource on demand, resource publication through a single provider, and mechanisms to bill users on the basis of resource utilization [19]. In this section, we briefly discuss the commonly known service and deployment models of cloud computing in order to provide a context to report our effort to migrate an OSS framework. A brief description of the service and deployment models of cloud computing will also help to steer our discussion about the challenges that software engineering methods and approaches in general and software architecture approaches in particular need to address. For a details discussion about the service and deployment of cloud computing, please see these publications [1-3, 11, 13, 17, 19].

Taking advantage of the cloud infrastructure, organizations offer different types of services. Software as a Service (SaaS) infrastructure offers software applications to its customers. In this type of services, developers usually do not have any option to customize the applications. However, a user of the systems can modify the respective software according to their needs by using the available customization option. Google email service is an example of SaaS model. In Platform as a Service (PaaS) infrastructure, a developer has access to a development platform through its APIs. These platforms support a specific set of programming languages. Google Application Engine is an example of such platform. Applications that are built for PaaS clouds do have benefits from the underlying cloud computing infrastructure. Infrastructure as a Service (IaaS) model offers infrastructure for computing and storage resources. This infrastructure is used to host applications. IaaS models often provide automatic support for on demand scalability of computing and storage resources [2].

There are mainly three deployment models of cloud computing: private clouds, public clouds and hybrid clouds [2, 6]. Private cloud is a collection of computing resources, storage resources and cloud technologies owned by an organization for its private use. The organization has full control of all the resources and technologies. The owner organization is also responsible for the maintenance of the infrastructure. Public cloud infrastructure is a collection of resources maintained by different organizations and the resources are offered to external users. A hybrid cloud is a category of cloud resources in which part of the infrastructure is maintained by the organization itself where as it also acquires the services from public clouds. These different models of the cloud environments have their respective advantages and disadvantages. The advantage for maintaining the private cloud is that an organization can have full control of all the resources related to a cloud infrastructure. The disadvantage of such a deployment model is that an organization not only has to invest in computing and storage resources but also on the related software and maintenance activities. The advantage of using the public cloud is that an organizations itself does not have to take care of the cloud computing infrastructure and operational activities. The disadvantage of utilizing the services from a public cloud provider is that it is completely dependent upon another business entity that is offering resources through public cloud. One of the key concerns of using public cloud is the security of the data that is stored on the public cloud. In such cases the tradeoff between public and private cloud is made and a hybrid approach can be adopted. In such cases, an organization can choose to maintain their critical resources as private cloud and rely on the public clouds for non critical business operations [2, 19].

More recently, some other deployment models have also been reported. Two of them are Virtual Private Cloud (VPC) [19] and community cloud [3]. A VPC runs on top of the public cloud with the additional benefit to have customized network topology and security settings. Applications running on a VPC can take benefit of cost effective infrastructure of public clouds and security mechanism of private clouds. In some cases, multiple collaborative organizations can have some common requirements of a set of software systems and can share their infrastructure to have more reliable and cost effective infrastructure. This kind of arrangement is called community cloud [3]. For a community cloud, all participating organizations share the cost of the infrastructure that means having a community cloud can be more cost effective as compared to maintaining a private cloud. A community cloud can also provide much higher level of security and privacy as it is available to the members of a small community.

Our experimentation with the cloud computing has focused on SaaS and PaaS as service model and public cloud as a deployment model. Though we planned to have a private cloud for the cloud-enabled Hackystat, lack of financial resources has not enabled us to materialized this plan so far. However, the methodological and technological challenges and experiences reported in this paper are related to developing and/or migrating software systems for PaaS and SaaS service models and public and private deployment models. In the following sections, we provide key aspects of our efforts to migrate Hackystat to cloud computing.

3. MIGRATING TO CLOUDS

In this section, we provide some details about our project, which aimed at migrating an OSS, Hackystat [8], for process and product measurements to cloud computing. A detailed description of the project can be found in [1].

3.1 System Details

Hackystat is an open source framework used for automated collection of process and product data [8]. It is under development since 2001 and has undergone eight releases. Most significant enhancement was done in release 8. In that release, Hackystat framework was migrated from a client-server web application to a service-oriented architecture following Representational State Transfer (REST) design principles [9].

Figure 2 shows the high level service-oriented architecture of Hackystat. This framework collects process and product data with the help of Plugins installed on Integrated Development Environment (IDE) of developers. These plugins are called Sensors. Another type of client used for data collection is called SensorShell. It can be used by a java based Hackystat client. SensorsShell can also send information to Hackystat server but it also has additional features like buffering data instead of immediately sending it, automated data transmission at user defined intervals, a command line interactive interface for manual data transmission and offline data storage to be transmitted later.

These Sensor and SensorShell send information to SensorBase service. SensorBase service saves this information in to the database. Whenever other services of Hackystat framework require this information, it is retrieved through SensorBase service. The DailyProjectData is another service that provides daily summary of the project's process and product metrics. All the information through database is retrieved through SensorBase service. Telemetry service provides project and process data that is displayed by Hackystat user interface services. This service uses DailyProjectData service to produce process and product trends over time.

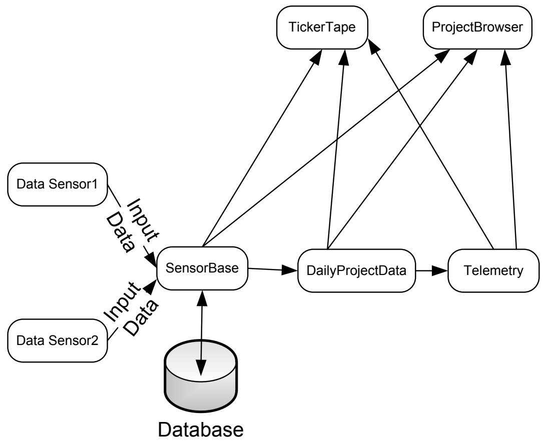

Figure 2: Hackystat Version 8

Hackystat framework has different client services that interact with its main services for the purpose of data presentation and analysis. TickerTape is one of these services. It interacts with SensorBase and DailyProjectData services to get information about changes made to a specific project, and then generates a status report that can be sent to different external applications like Nabaztag Rabbit and Twitter. ProjectBrowser in another client service, that provides a web interface to Hackystat services. It also provides services for managing projects.

The project data can either be viewed using ProjectBrowser or be posted to external applications using TickerTape. ProjectBrowser and TicketTape access methods of other services to present or post information. For example, a function call to produce a single telemetry operation may invoke many operations on DailyProjectData service which in turn invoke required operations from SensorBase service to produce the required information. In this way, different services of Hackystat act like a pipeline.

3.2 Requirements Identification

The objective of our project was to modify Hackystat as an SaaS model to be deployed on an IaaS cloud. While we had a goal of migrating Hackystat so it can be provided as SaaS to globally distributed teams whose manager may be managing several dozens of projects simultaneously, we did not have specific customers which cloud have been involved in eliciting the requirements for cloud-enabled Hackystat. Hence, the approaches like quality attribute workshop [4] may not be of any help in these kinds of situations but quality attribute scenarios [10] can be quite useful for describing quality attributes. We decided to start the identification of the requirements by analyzing the properties of SaaS and IaaS. Through this exercise, we identified two characteristics of SaaS: SaaS cloud platforms do have the capability to add or remove additional computing resources and storage resources on demand; and SaaS offers scalability in a transparent fashion and a user can access the services in seamless way. IaaS clouds provide environments to host applications to utilize computing resources. These clouds also offer the storage resources so that storage capacity of the clouds can be utilized. Based on this analysis, we identified four key requirements for cloud-enabled Hackystat as described below:

R1: The system should be able to scale up or scale down to meet the performance requirements for several dozens of projects each of which may have dozens of developers dispersed in different parts of the world.

R2: System components should be able to be deployed on public or private IaaS clouds.

R3: Components of the system that are responsible for persistence handling should be able to take advantage of storage resources provided by IaaS clouds in future.

R4: End users of the system and external clients should have access to the system through same interfaces as provided by existing systems.

3.3 Requirements Analysis for Architecture Decisions

Having identified the key requirements for the cloud-enabled Hackystat, our next task was to analyze those requirements in the context of the existing architecture of Hackystat in order to gain a good understanding of the kinds of architectural change that needed to be made.

According to the first requirement (R1), the system should be able to scale up or down according to the performance requirements. We used this requirement for identifying two required quality attributes: modularity and scalability. For scalability, different components of the system should be able to be replicated so that these can be deployed on multiple physical or logical nodes in numbers that best suits the performance criteria. These components can only be deployed on IaaS clouds and not on PaaS because in case of PaaS, different components have to be re-factored according to the API of a specific PaaS cloud. There was not any immediate requirement for PaaS service deployment so the older components with modified capability to have multiple deployments on nodes were selected.

The second requirement (R2) was associated with deployments on the public and private IaaS clouds. We identified portability and changeability quality attribute associated with this requirement. As components are expected to have deployment on IaaS clouds, hence, any IaaS that supports the technology on which Hackystat is build would be able to host them.

The third requirement (R3) was that the components that were responsible for data persistence should be able to take use of IaaS storage. We identified portability as the quality attribute associated with this requirement. We decided to meet this requirement by making a separate component to handle data persistence tasks. For the use of IaaS storage, persistence handler component would require platform specific modification for portability in future. Modification to storage component would take care of all portability issues related to persistence and other parts would need any modification.

The fourth requirement (R4) required a consistent view of the system for external client and components. This arised the need of a new component for each group of replicated services (for example four running instances of SensorBase component) to provide a single and uniform view to other components and client services. This component should also handle the replicated live nodes of components transparent from the end user. There were two potential solutions for this wrapper component. One is to have a REST based wrapper component service that provides the same API as that of replicated service component it is responsible to handle (for example SensorBase controller service for replicated SensorBase components). Other way was to have a simple web component that routes the requests to the target service without processing it. The second solution was adopted because its advantage to be independent from the interfaces of the clustered services. Hence, it required no modification if the interfaces of the service in cluster changes and could also be replaced by some more sophisticated load balancing tools provided by a public cloud provider.

3.4 Modified Architecture

In this section, we briefly describe the main changes made to modify the Hackystat's SOA architecture and the modified architecture. A detailed description about the rationale for different modifications, implantation level modifications, and migration steps followed have been reported in [1].

The modification of the architecture tasks revealed that the overall SOA based Hackystat system needed only two main modification to the previous architecture:

- We had to separate the database layer from the SensorBase component and;
- We had to introduce an additional layer to provide orchestration when component are replicated on different physical or virtual machines.

After these two main modifications, the system was expected to work as it did previously except that now every request to different services is passed through newly introduced controller layer. This layer has a separate service for each cluster of the SensorBase, DailyProjectData and Telemetry components.

Figure 3 shows the cloud-enabled architecture of Hackystat. According to the modified architecture, different sensors send data for process and product metrics to SensorBase Controller. This service is responsible for delegating a request to any of the available instances of SensorBase service depending upon routing algorithm. In our current implementation, we have only implemented round robin algorithm to equally distribute the load among all available instances of a service. Different instances of the SensorBase communicate with the database service to interact with the persistent repository system. The database services provides abstraction for data management operations provided by the data management system being used by a cloud provided. Whenever information is required to be

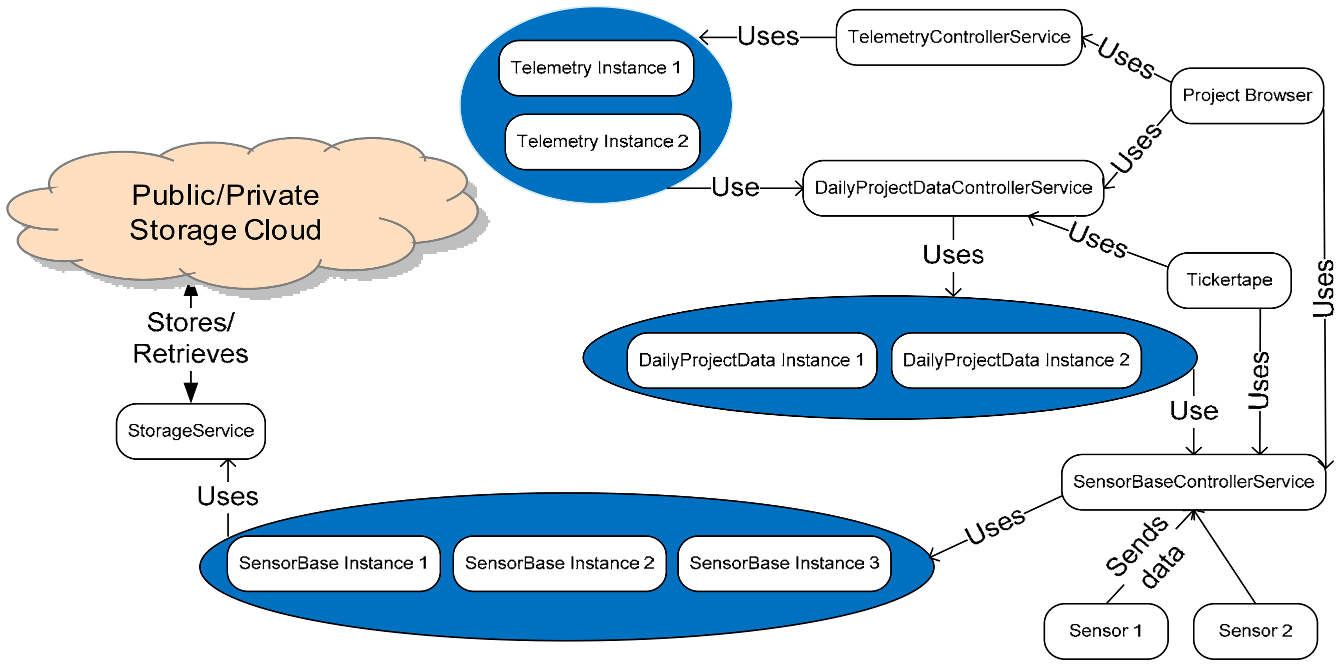

Figure 3: Hackystat Architecture - Enabled for cloud deployment

fetched from or stored to the persistent data repository, it is performed through the Database service.

The functionality of the Hackystat is invoked in a top down approach. According to process of the invoking Hackystat functionality, when a client or a service at an upper level of a pipeline has to invoke functionality from its low level services it is always done thorough the controller services. These controller services are responsible for routing the requests to an appropriate instance of the required service at a lower level deployed on a cloud computing infrastructure. For example, if the project browser of the Hackystat needs to present telemetry information to a project manager, it sends request for the required information through the Telemetry controller. Having received the request form the project browser, the Telemetry controller decided which instance of the Telemetry service from a cloud cluster can handle the request and then the Telemetry controller delegates the request to that particular instance. Since the Hackystat's services work in a top down manner, a Telemetry service requires the services of the DailyProjectData service in order to fetching the required information. The Telemetry service requests the DailyProjectData controller which subsequently delegates the request to an service from the cluster of the DailyProjectData instances. When the DailyProjectData requires information from the SensorBase, it also needs to request the required information in the same manner (i.e., via the SensorBase controller). The Architecture of the cloud enabled Hackystat framework shown in Figure 3 has been deployed on Amazon EC2 windows cloud instance. Different components can be deployed on separate clouds on the basis of scalability requirements.

4. EXPERIENCES AND OBSERVATIONS

In this section, we share our experiences and insights from modifying Hackystat to be provided as SaaS and deployed a public IaaS. We hope that our experiences and observations reported here can provide some discussion points for identifying the areas that need attention in order to address the software engineering related challenges of developing and evolving software systems for cloud computing.

Scalability is the most important functionality of the SaaS platforms as these are expected to support large number of users. Hence, such systems should be designed to utilize computing resources from different varieties of IaaS clouds to have less dependency on a specific platform provider. Applications that are constructed from components with very less dependency on each other and capability to communicate over a variety of networks are easy to migrate to cloud-based environment.

If an application is implemented as a SaaS model, then data persistence related functionality should not be made part of the business logic components. Different IaaS clouds offer different data persistence solutions, for example, different types and versions of database systems. If underlying database needs to be changed for a business or technical reason, separate components to handle data persistence makes it easy to migrate. It also saves development and testing cost. If the components are able to communicate over HTTP protocols, separate data persistence components also provide the opportunity to move them to some other cloud that suits the database requirements while keeping the business components on the same cloud. Moreover, software system that use database only for data persistence and do not store business logic in databases in form of triggers and store procedures can use cloud database with less re-factoring cost.

It may not be possible to port existing application components to any flavor of cloud computing. This issue is more critical when applications are intended to use PaaS clouds. PaaS clouds offer a specific version of technologies and a specific set of corresponding APIs. To port existing applications and components may require major refactoring efforts that are likely to reduce the benefits of using cloud computing. In those scenarios, a tradeoff needs to be made. A specific set of components that are necessary to be migrated to clouds should be re-factored and ported, while the remaining components can be deployed on existing infrastructure.

Software systems that consist of multiple flavors of technologies are hard to migrate to cloud environments and may require to re-factor the complete applications. This is because of the reason that both IaaS and SaaS clouds offer support only for a specific set of tools, software and technologies. However, the systems using stateless components are easy to migrate to IaaS clouds. Such systems are also easy to have the scalability features.

Heavy reliance on cloud infrastructure providers: when SaaS system are developed by using APIs provided by PaaS software systems, such systems heavily rely on the cloud provider and portability of applications on other environments becomes doubtful. As portability is considered a very important quality attribute for long term evolution of the software systems, so the evolution of such systems would heavily depend on the evolution of the PaaS cloud environments.

Evaluation of system complexity: In traditional software development, members of the software development team have access to all the code resources and different software components that are being used in a software system. This is easy for the system analysts to compute system complexities. Systems built using the API provided by the PaaS cloud infrastructure cannot be evaluated for complexities because actual implementation of the APIs is hidden from clients.

Testing: testing the software systems deployed on cloud infrastructure is different from testing the traditional systems from many perspectives. In case of cloud environments, hardware resources have virtual representation and application access the virtual hardware to perform the task. Testing of systems deployed on virtual environments is quiet challenging specially from performance aspect. Performance of the application also depend on how efficiently virtual recourses are manages by a cloud provider. In order to test applications from performance perspective, testing is required to be performed on regular basis. Any change in the underlying infrastructure that is managing the virtual resources may decrease the performance of the application and cause a system failure. This may also result in increased software maintenance cost without actually any modification or new enhancement.

Technology enhancements are bound with technology support provided by cloud vendors: software systems often require technology upgrades to meet the new requirements and marketing challenges. For application deployed on cloud infrastructure, technology upgrades cannot be accomplished unless it is supported by a respective cloud vendor. Similarly if different components of the applications are built using different types of technologies, the support of all the technologies is required by vendors. These limitations make it difficult for complex software systems to be deployed on public cloud infrastructures. However, such complex systems can take benefit from private cloud infrastructures.

Tradeoff between business and system requirements: Motivation to move existing systems on to cloud aware infrastructure comes from business stakeholders. Software systems that are already in service are built according to the business objectives that were important at the time when they were developed. Some of the requirements of service-based system may be in conflict with new business objectives. This conflict motivates to perform the tradeoff analysis once again before initiating the migrations activity so that stakeholders' requirements can be evaluated.

Tooling support: existing software design tools, modeling languages and testing frameworks have focused on only software related aspects of the system and do not consider the underlying platforms. These is a vital need for software engineering tools that can help design systems and perform testing aligned with the underline virtualized environment provided by cloud infrastructures.

Architecture evaluation: software architecture evaluation is a key activity to assess the architecture design decisions before implementing them. Our experience is that the current architecture evaluation methods (such as [5, 10]) may not provide full support to addresses the quality attribute evaluation challenges for cloud infrastructures. The existing architecture evaluation methods do not effectively support cloud-centric quality aspects such as scalability and accessibility. We assert that quality attributes need to be evaluated more frequently in the context of cloud computing. Hence, we need to scale the existing or develop new architecture evaluation approaches, which can provide more robust support for factoring unknown users of a cloud-enabled system and to be deployed on a cloud infrastructure not known at the design time.

Security: security of the data deployed on public clouds is a big concern. To protect the sensitive data from any accidental access more secure means need to be investigated rather than encryption. Although encryption can provide the security to a certain extent but this may directly affect some other key quality attributes of a systems like efficiency. To ensure security, efficient methods need to be investigated that allow the distribution of data on clouds in such a way that other than the system to which the data belong, no one can access it.

Hard to ensure certain constraints: there are some extra functional requirements that are related to laws imposed by regulatory authorities. One example of such requirement is constrains on the geographic locations of data storage places. For some applications, it is required that data should not store outside a particular region. In IaaS clouds, data is maintained at different location for reliability and cost effectiveness. Users of the cloud services do not have information about where exactly on the cloud application data are stored. There needs to be middleware support for enabling users to manage the privacy and storage of their data.

Energy efficiency: one of the big challenges being faced by cloud computing infrastructure providers is enormous consumption of energy for operating large scale data center [19]. This challenge present an opportunity for the software engineering community to develop new and novel methods, techniques and tools for designing, developing, and evolving carbon sensitive software intensive services for cloud computing and data centre managements. Such services should incorporate smart mechanisms for automatic monitoring and energy consumption for corrective measures aligned with carbon emission reduction goals of cloud computing infrastructure providers and operators of large scale data centers [12]. There is also an impending need of intelligent approaches for supporting various decisions during the whole lifecycle of cloud-enabled software services in terms of their carbon footprint.

Interoperability: public cloud providers tend to use proprietary platforms, which usually do not support interoperability. Hence, there is a need of reference architecture and standards to support interoperability of different vertically integrated clouds, design common service descriptions for meeting required quality requirements between different vendors in various layers of the cloud stack, and supporting service and cloud infrastructure configuration on individual user basis.

5. CONCLUSION

Cloud computing offers several opportunities of using IT infrastructures as a utility with the possibility of scaling up or scaling down depending upon the needs an organization. However, like most of the emerging IT technologies, cloud computing has raised several technical and socio-technical challenges [2, 11, 15, 19]. However, we have not observed much literature on software engineering aspects of developing and evolving cloud-enabled systems or on software engineering related challenges that needs to be addressed for designing, developing, and deploying solutions with or for cloud computing. Our pursuit for finding suitable guidelines for migrating software systems to cloud computing was motivated by an effort to make an OSS framework cloud-enabled.

During this project, we studied the Hackystat framework for identifying the steps that needed to be taken to migrate an SOA based applications to cloud aware systems. This paper aims at sharing our experiences and observations gained through this project and critical analysis of the literature with those who intend to migrate software systems in general and SOA based system in particular to cloud computing. Through this paper, we also hope to contribute to the identification of the software engineering research challenges that need to be addressed in order to provide sufficient support for engineering software systems for clod computing.

6. ACKNOWLEDGMENT

Muhammad Aufeef Chauhan acknowledges the support of EURECA project (www.mrtc.mdh.se/eureca) funded by the European Commission; and Philip Johnson, founder of Hackystat project from University of Hawaii for his help in understanding Hackystat.

7. REFERENCES

[1] Ali Babar, M. and Chauhan, A., Toward a Framework for Migrating Software Systems to Cloud Computing, Tech Report TR-2011-139, IT University of Copenhagen, Denmark, 2011.

[2] Armbrust, M., et al., A View of Cloud Computing, Communication of ACM, 2010. 53(4): pp. 50-58.

[3] Baiardi, F. and Sgandurra, D., Secuing a Community Cloud, in IEEE 30th International Conference on Distributed Computing Systems Workshops. 2010, IEEE Computer Society. pp. 32-41.

[4] Barbacci, M.R., et al., Quality Attribute Workshops (QAWs), Tech Report CMU/SEI-2003-TR-016, SEI, Carnegie Mellon University, USA., 2003.

[5] Bianco, P., Kotermanski, R., and Merson, P., Evaluating a Service-Oriented Architecture, Tech Report ESC-TR-2007-015, Softwar Engineering Institute, Carnegie Mellon University, 2007.

[6] Grossman, R., The Case for Cloud Computing, IEEE Computer, 2009. 11(2): pp. 23-27.

[7] Hofmeister, C., et al., A general model of software architecture design derived from five industrial approaches, Journal of System and Software, 2007. 80(1): pp. 106-126.

[8] Johnson, P.M., Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System, in First International Symposium on Empirical Software Engineering and Measurement. 2007, IEEE Computer Society: Madrid, Spain. pp. 81-90.

[9] Johnson, P.M., Zhang, S., and Senin, P., Experiences with Hackystat as a service-oriented architecture, Tech Report Department of Information and Computer Sciences, University of Hawaii, USA, 2009.

[10] Kazman, R., Klein, M., and Clements, P., ATAM: Method for Architecture Evaluation, Tech Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University, 2000.

[11] Khajeh-Hosseini A, Greenwood D, and 2010., S.I., Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS, in IEEE 3rd International Conference on Cloud Computing. 2010, IEEE Computer Society: Miami, USA. pp. 450-457.

[12] Lago, P. and Jansen, T., Creating Environmental Awareness in Service Oriented Software Engineering in First International Workshop on Services, Energy, & Ecosystem, colocated with ICSOC 2010, San Francisco, USA. . 2010.

[13] Lenk, A., et al., What is Inside the Cloud? An Architectural Map of the Cloud Landscape, in Workshop on Software Engineering Challenges of Cloud Computing, Collocated with ICSE 2009 2009, IEEE Computer Society: Vancouver, Canada. pp. 23-31.

[14] Lewis, G., et al., SMART: The Service-Oriented Migration and Reuse Technique, Tech Report CMU/SEI-2005-TN-029, Softwar Engineering Institute, Carnegie Mellon University, 2005.

[15] Mei, L., Chen, W.K., and Tse, T.H., A Tale of Clouds: Paradigm Comparison and Some Thoughts on Research Issues, in Asia-Pacific Services Computing Conference. 2008: Yilan, Taiwan.

[16] Mell, P. and Grance, T., The NIST Definition of Cloud Computing, Tech Report National Institute of Standards and Technology, USA, 2009.

[17] Strowd, D.H. and Lewis, G., T-Check in System-of-Systems Technologies: Cloud Computing, Tech Report CMU/SEI-2010-TN-009, Softwar Engineering Institute, Carnegie Mellon University, 2009.

[18] Vaquero, L., Merino, L., Caceres, J., and Lindner, M., A break in the clouds: towards a cloud definition, SIGCOMM Computer Communications Review, 2009. 39(1): pp. 50-55.

[19] Zhang, Q., Cheng, L., and Boutaba, R., Cloud computing: state-of-the-art and research challenges, Journal of Internet Services and Applications, 2010. 1(1): pp. 7-18.