2011 Ninth Working IEEE/IFIP Conference on Software Architecture

Assessing Suitability of Cloud Oriented Platforms for Application Development

Balwinder Sodhi and T.V Prabhakar
Department of Computer Science and Engineering
IIT Kanpur, UP 208016 India. Email: {sodhi,tvp}@cse.iitk.ac.in

***Abstract*—The enterprise data centers and software development teams are increasingly embracing the cloud oriented and virtualized computing platforms and technologies. As a result it is no longer straight forward to choose the most suitable platform which may satisfy a given set of Non-Functional Quality Attributes (NFQA) criteria that is significant for an application. Existing methods such as Serial Evaluation and Consequential Choice [1], [2] etc. are inadequate as they fail to capture the objective measurement of various criteria that are important for evaluating the platform alternatives. In practice, these methods are applied in an ad-hoc fashion.**

In this paper we introduce three application development platforms: 1) Traditional non-cloud 2) Virtualized and 3) Cloud Aware. We propose a systematic method that allows the stakeholders to evaluate these platforms so as to select the optimal one by considering important criteria. We apply our evaluation method to these platforms by considering a certain (non-business) set of NFQAs. We show that the pure cloud oriented platforms fare no better than the traditional non-cloud and vanilla virtualized platforms in case of most NFQAs.

***Index Terms*—Cloud computing, software design, non-functional quality attribute, design decision, evaluation scheme, MAUT**

I. Introduction

Enterprises today are increasingly becoming cost conscious and are moving towards data center resource consolidations by embracing virtualization and cloud computing platforms. The enterprises aim to reduce costs by converting capital expenses into operation expenses by adoption of cloud computing platforms for certain types of applications. It is observed that the business drivers are the major contributors that propel that adoption of virtualization and cloud computing platforms in the enterprises. From the standpoint of application design technicalities, the cloud computing is still a young field and lacks mature standards and field proven best practices.

The applications designers need to be cognizant of the increasingly becoming virtualized and cloud oriented data center during the application design and architecture activities. Given a wide and increasing range of virtualization based and cloud oriented platforms and frameworks, it is not always clear that in what different ways can an application be designed to meet the desired NFQAs and how to choose the most suitable platform. Typical questions that one is faced with when starting the development of a new application are:

- If the application is to be deployed on a virtual/cloud platform, what different design approaches are available?
- How to objectively and systematically evaluate the application development platform alternatives by considering criteria of significance?
- How does the choice of a particular platform affect the ability to realize certain NFQAs?

In this paper our focus is to answer these questions. A business problem usually have more than one possible solution. Each solution alternative normally differs in terms of the NFQAs that it can afford to provide. In this work we propose a Multi-Attribute Utility Theory [3] (MAUT) based method for choosing an application development platform which is required to achieve a given set of non-functional quality attributes. Primary motivation behind developing this framework is to allow a systematic decision making that takes into consideration all the significant factors – both favourable and unfavourable – contributing towards the stakeholder concerns and expectations. We then apply this framework to choose an optimal platform for developing an application for deployment on a cloud platform.

Rest of this paper is organized as: section II introduces the application development platforms. Proposed evaluation method is described in section III; and the application of this method to a usecase is discussed in section IV.

II. Application Development Platforms

In the traditional way of application development, the production deployment is typically assumed to be on some dedicated physical machine(s). In such environment, at most, the machine's capacity may be shared by multiple applications, however, there is no notion of virtualized and elastic computing resources (via cloud or direct) involved. For example, a single physical machine may be hosting two different data base server instances each belonging to a different application. The virtualization and cloud based platforms bring with them certain characteristics (both at infrastructure and platform levels) which the application designers can exploit to their advantage or get adversely affected by it. We studied the features of various Infrastructure As a Service (IaaS) and Platform As a Service (PaaS) cloud platforms [4]–[9] and based on that categorize the application design approaches as below:

A. Traditional Non-cloud

This is the the good old traditional platform for application development where virtualization, cloud and the resource elasticity afforded by them are not the first class citizens of the data center. That is, the designer doesn't make any special assumptions about the platform regarding its virtual, elastic or distributed nature. Typically, the data stores are regular SQL based RDBMS engines.

B. Virtualized

This scenario is very similar to the traditional non-cloud platform, except that instead of developing and deploying directly on dedicated physical machines, the application will be deployed on Virtual Machines (VM). Such VMs may be sharing the host server(s) with other VMs. Beyond that the application designer doesn't make any special assumptions, such as of elasticity, about the platform. The chief implication of this kind of *virtualization* is that certain characteristics which resource virtualization brings with it can be exploited at VM level. For instance, it becomes easy and faster to migrate and (re)deploy the VMs. Resource utilization could be enhanced in the data center. Often, most of the benefits with this kind of platform will be for the IT operations personnel.

The applications designers, however, will use mostly the same design tools and techniques as for developing an application for traditional environment. This platform will often make use of some vanilla virtualization software like hypervisors to run the VMs on a shared hardware. Use of a private or public IaaS cloud service (e.g. Eucalyptus IaaS cloud) could also be classified under this category, as long as the application doesn't depend on any cloud specific features (e.g. automatic scaling) of the IaaS platform.

C. Cloud-Aware

On this kind of a platform, we take into consideration the fact that computing resources exposed to the application are elastic and virtual and can/should be scaled and manipulated in a manner different from the traditional (non-cloud) way of dealing with them. When we say that application is aware of the virtual nature of the underlying platform, it doesn't necessarily mean that the application needs to invoke some special low level virtualization APIs in order to obtain the services from the cloud platform. It just means that a designer is aware of the architecture of the underlying platform which is highly distributed, virtual and elastic; and thus the designer aims to exploit such a platform to achieve application's desired NFQAs.

The tools and techniques needed for effectively designing and developing such applications for the cloud are, therefore, different from the ones used for non-cloud platforms. Distributed, elastic and virtual nature of cloud resources is the key determinant here. Software frameworks used here are geared for simplifying writing application for large distributed clusters of commodity hardware. Similarly the data stores used for such applications often have the following characteristics:

- Massively parallel
- Non-relational
- Distributed
- Schema-free (NoSQL)
- Easy replication support
- Basically available, soft state, eventually consistent - BASE [10] (not ACID [11], [12] as in pure RDBMS world)

Typically, the applications are built by using massively parallel and distributed frameworks like: Hadoop/MapReduce [13], [14], HBase, Cassandra and Hypertable etc. There is a big paradigm shift from the relational way of thinking when it comes to data – principles that were sacrosanct in relational world are no longer so in NoSQL [15] arena. The data access in application using such data stores can happen via existing established APIs such as Java Persistence API (JPA) [16] and Java Data Objects (JDO) [17] etc., but the underlying implementation of such APIs is data store and vendor specific. So from the client perspective the programming semantics may not change in comparison to the non-cloud platforms, however, since the data store architecture is quite different, therefore the organization, lifecycle of data and the working of transactions etc. is often very different in such systems. The bottom line is that the design philosophy, particularly when it comes to data stores, is quite different from the traditional platform. Applications implemented using this platform will often make use of IaaS and PaaS cloud platforms.

III. Evaluation Method

Serial Evaluation and Consequential Choice [1], [2] are the two most commonly used methods for decision making in software development tasks. Former involves serially evaluating the available set of options in a decision. In the latter, the concurrent comparison and trade-off evaluation of more than one option in a decision is performed. These approaches are employed more or less in an ad-hoc fashion by the stakeholders, as such the chance of uncompensated bias and subjectivity in such decision making processes are relatively high. At the same time such methods do not afford the repeatability of the process, that is, with the given set of assumptions and constraints about the design decision alternatives whether a different person can arrive at the same design choice. In this paper we discuss a MAUT (see [3], [18], [19]) based method which allows us to:

- Provide a systematic way to evaluate design decision choices.
- Consider relative importance of all significant factors that affect a design alternative.
- Have the repeatability of the method.
- See the effect of bias on the evaluation results.
- Rigorously apply objective measurement of various criteria to decision making.

In our framework we make use of MAUT, as the basis of evaluation process.

Using MAUT, the evaluation $u(x)$ of an object x can be calculated as the weighted aggregation of x's evaluation with respect each of x's relevant criteria. For example, a laptop may be evaluated on the criteria: battery life, weight, disk space, memory, CPU, price and quality of support. Note that all criteria may not always be objectively measurable, however, a certain *preference ordering* strength may always exist for the criteria amongst competing alternatives. A normalized scoring scale which preserves the ordering may be applied to such cases.

The final evaluation of the object can be calculated as:

$$u(x) = \sum_{\forall c \in C} w_c u_c(x) \tag{1}$$

Here,
$u_c(x)$ = Evaluation of the object on criterion c.
w_c = Weight determining the impact of criterion c on the overall evaluation (also called the relative importance).
C = Set of all different criteria, and $\sum_{\forall c \in C} w_c = 1$.

Further, $u_c(x)$ can be calculated as the evaluation of the relevant contributing sub-criteria/attributes for the criteria c:

$$u_c(x) = \sum_{\forall a \in A_c} w_{ac} u_{ac} \tag{2}$$

Here,
A_c = Set of all contributing attributes for criterion c.
u_{ac} = Evaluation of attribute a of c.
w_{ac} = Weight determining the impact of the evaluation of attribute a on c.
Also, $\sum_{\forall a \in A_c} w_{ac} = 1$ for each criterion c, where $c \in C$

The optimal[1] value of the object (i.e. the design decision alternatives in our case) from the set of possible alternatives is determined as:

$$\max_{x \in X} u(x) \tag{3}$$

Here, X is the set of all possible alternatives that we want to evaluate. $u(x)$ is the evaluation of alternative x as calculated using equation 1.

A. Method details

The process followed by the proposed method to determine the optimal design decision is outlined as below:

1) Identify design decision alternatives (i.e. the set X).
2) Identify and rank desired NFQAs (i.e. the set C and $w_c \forall c \in C$).
3) Identify and rank contributing factors for each NFQA (i.e. set $A_c, w_{ac} \forall c \in C$)
4) For each design decision alternative:
    a) Evaluate each contributing factor (i.e. determine $u_{ac} \forall a, c$).
    b) Evaluate each NFQA (i.e. determine $u_c(x) \forall c$ by using equation 2).
    c) Calculate the overall scores (i.e. determine $u(x) \forall x \in X$ by using equation 1).
5) Select the highest scoring alternative (i.e. evaluate equation 3).

The use of rankings brings a certain amount of subjectivity into this process. However, since the relative importance of various NFQAs (and NFQA-contributors of design alternatives) has to be allocated based on the decision makers' perception and understanding of the problem at hand, therefore ranking steps become a necessary tool for allowing the needed flexibility. In order to see the effect of subjectivity, a dual approach can be followed where in one case each ranked element can be given equal weightage/rank thus making it free from any subjectivity, and second in which the rankings are as per the stakeholders' perceptions.

[1]Can be maximum or minimum depending on the context of the problem. For our case it is the maximum.

IV. Use-case discussion

Primary motivation is to be able to systematically decide about the optimal application development platform to achieve a given set of NFQAs. A framework which is easy to use and understand, and allows for repeatable decision making results while taking into consideration the specific constraints and assumptions concerning the decision alternatives is needed.

For any new application development we had to choose from one of the three application development platform alternatives we introduced earlier. These alternatives form the set X from which we want to determine the optimal choice of $u(x) \mid x \in X$. These alternatives are denoted as:

- x_1: Cloud-Aware
- x_2: Virtualized
- x_3: Traditional Non-coud

Our general observation is that the adoption of cloud oriented platform by the enterprises is driven by either the cost economics or by their desire to harness the unique technical capabilities such as massive elastic scalability that such platform may offer. When trying to implement cloud platform for achieving cost economics goals, the primary question is of deciding about a provisioning model for computing resources (portion A of Fig. 1). On the other hand, when the goal is to harness the unique technical capabilities of the cloud oriented platforms then the decision is not quite as straight forward (portion B of Fig. 1).

In order to remove any ambiguity about what we are focussing on, a concept map depicting the architecture-process centric view of various aspects of the problem is shown in Fig. 1. Our interest is in dealing with the portion B of this diagram.

Non-functional Quality Attributes

The NFQAs, which are of most interest to us, form the set C of the criteria on which we want to evaluate the application development platform alternatives from X. These NFQAs are denoted as:

- c_1: Maintainability
- c_2: Security
- c_3: Portability
- c_4: Scalability
- c_5: Availability

We are not considering any monitory/economic/business constraints and drivers for our evaluation. This is because we want to focus on the situation where the adoption of a cloud oriented platform by the enterprises is driven mainly by their desire to harness the unique technical capabilities that such

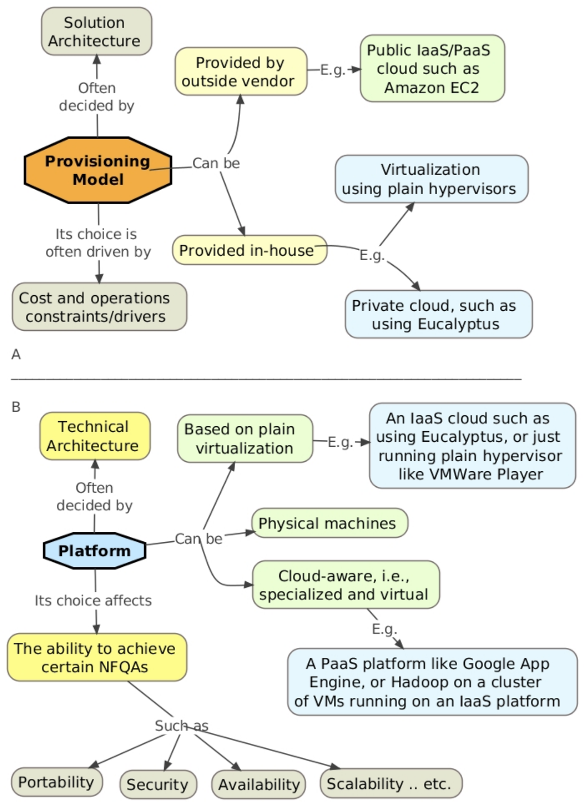
Solution Architecture
Often decided by
Provisioning Model
Can be
Provided by outside vendor
E.g.
Public IaaS/PaaS cloud such as Amazon EC2
Virtualization using plain hypervisors
Provided in-house
E.g.
Private cloud, such as using Eucalyptus
Its choice is often driven by
Cost and operations constraints/drivers
A
B
Technical Architecture
Often decided by
Platform
Can be
Based on plain virtualization
E.g.
An IaaS cloud such as using Eucalyptus, or just running plain hypervisor like VMWare Player
Physical machines
Cloud-aware, i.e., specialized and virtual
E.g.
A PaaS platform like Google App Engine, or Hadoop on a cluster of VMs running on an IaaS platform
Its choice affects
The ability to achieve certain NFQAs
Such as
Portability
Security
Availability
Scalability .. etc.

Fig. 1. Architecture-process centric view

platform may offer, and not driven by the cost economics. That is, we want to find out that purely from the technical standpoint which of the platforms is best able to afford us a given set of NFQAs.

In order to compare the platforms, we identified the important attributes (i.e. $a \mid a \in A_c$) which affect the realization of a particular NFQA. For example, NFQA of *Deployability* is affected by availability of good quality automation tools/APIs for deployment. Such attributes were identified by studying [20]–[23] and are discussed in subsequent sub-sections.

Scores (i.e. u_{ac}) of these contributing attributes for each design approach were assigned by studying the information available from various types of platform vendors' documentation (e.g. [7]–[9]). For instance, looking at [8] we find that Google App Engine (GAE) supports only two programming languages: Java and Python; and that it supports only a partial set of Java's core APIs. We also looked at the issues filed (e.g. [24]–[28]) against each of the stated features/APIs that a platform vendor claims to be providing.

For example, a cloud platform vendor may claim that it provides deployment automation tools/APIs; but whether the tools/APIs are effectively working for the users can be gauged, to reasonable extent, by looking at the bugs filed against that feature for the platform.

TABLE I
NORMALIZED SCORE MAPPING

Qualitative Score	Normalized Score
Better	1.25
Same	1.00
Worse	0.75

Keeping the above points in mind, a given platform was assigned the score of *worse, same* or *better* with respect to the traditional non-cloud platform.

Applying The Method To The Use-case

The scores of various NFQA attributes are w.r.t the baseline platform x_3, that is, whether the given platform x is better, same or worse than the baseline. The normalized scores are assigned as shown in Table-I. The variation from baseline value of 1.0 is chosen to be small (i.e. ± 0.25) to be on the conservative side. This choice is also in accordance with the differences which were observed for various contributing attributes of NFQAs for each design alternative x_i. We would like to mention that these scores have been arrived at by following Dephi process [29] by a group of software development practitioners.

For determining the optimal design approach we consider the following scenarios for assigning weights to the NFQAs:

1) When all the NFQAs are assigned equal weight. That is, $w_{ci} = \frac{1}{n}$ where $i \in (1...n)$, n being the number of different NFQAs to consider.
2) When one NFQA is assigned higher weight than the rest of them all of which have equal weight. For example, $w_{c1} = 0.3$ and $w_{ci} = 0.175$ where $i \in (2...5)$

Following subsections discuss the details of arriving at scores for individual NFQAs.

A. *Maintainability*

Most factors influencing maintainability of a software application are related to how mature the software design and development processes are [30]. Equally important is the quality and maturity of upstream dependencies i.e. the systems, frameworks and APIs on which the application depends. Achieving high maintainability in cloud-aware platform is difficult. This platform requires an entirely new way of solving the problems, particularly how the data modelling and concurrency are handled in highly distributed virtual environments [8], [31], [32]. The programming model is different (e.g. [8]) from the well established traditional non-cloud platform. There has been limited experience with cloud-aware platform itself and the APIs/frameworks which one uses in this case. Best practices have not been matured and field proven yet. Very few standards exist for the application frameworks. Table-II shows the scores assigned to various contributing attributes for this NFQA.

B. *Portability*

Most PaaS providers have their proprietary APIs for the services they provide. For example, applications written for

TABLE II
ATTRIBUTE SCORES FOR *Maintainability*

Attribute	Cloud Aware	Virtualized
Availability of modular programming support.	Same	Same
Experience in developing the software on the given platform	Worse	Same
Maturity of the APIs and their implementations.	Worse	Same
Complexity of programming model.	Worse	Same
Module coupling.	Same	Same
Scaling model (horizontal vs. vertical)	Worse	Same

TABLE III
ATTRIBUTE SCORES FOR *Portability*

Attribute	Cloud Aware	Virtualized
Support for popular programming platforms. More platforms are better.	Worse	Same
Availability of standardized APIs for basic services (persistence, messaging etc.)	Worse	Same
Availability of multiple implementations of standard APIs.	Worse	Same
Adoption and reach of standard APIs.	Worse	Same
Maturity of the APIs and their implementations.	Worse	Same
Release cycle length of APIs and their implementations.	Worse	Same

GAE cannot be executed on AWS platform without significant re-design. The GAE Software Development Kit (SDK) for Java doesn't support all the Java runtime features [33]. Relying on platform specific APIs, such as in cloud-aware approach, makes the portability difficult. A cursory look at the open bugs lists [24] for GAE or for AWS [25] would highlight various kinds of issues faced in this respect. Secondly, since the cloud-aware approach makes use of the techniques and frameworks (e.g. Hadoop/MapReduce, NoSQL data stores etc.) that are designed to work best on highly distributed environments such as large VM clusters on a cloud, therefore it is essentially a redesigning of an application if one needs to switch to/from the non-cloud environment. The traditional non-cloud and virtualized platforms, on the other hand, don't suffer from as serious portability issues. The traditional non-cloud and virtualized platforms rely on existing APIs and enjoy the availability of widely adopted standards for various application level services such as persistence, security, data querying etc. Moving an application built on a virtualized platform between cloud and non-cloud platforms doesn't require any reworking of the application.

C. Security

Security in cloud broadly falls into two major areas: 1) related to the trust guarantees that a cloud tenant requires from the cloud vendor 2) related to the safety of cloud software itself – both the infrastructure level as well as application level software. In the former case, the main goal is to ensure the confidentiality of tenant's data. In the latter case the main goal is to ensure the safety of entire software stack on cloud. For example, we do not want to allow any vulnerability (e.g. [34]–[36]) in the virtualization software to result in breaking the isolation of various VMs running on the underlying shared hardware fabric. With the above mentioned two broad impact areas, we can compare the overall system security for traditional versus the cloud oriented platforms.

TABLE IV
ATTRIBUTE SCORES FOR *Security*

Attribute	Cloud Aware	Virtualized
Multi-tenancy isolation	Worse	Worse
Maturity and stability of virtualization platform	Worse	Worse
Encryption of data-at-rest	Same	Same
Ability to choose legal jurisdiction of hosted data	Worse	Worse

1) Tenant's Trust on The Platform: This parameter is easier to compare between the traditional non-cloud and cloud platforms. Normally, in case of the traditional non-cloud platform, the datacenter is owned by the tenant organization. Therefore, the tenant organization is always in possession of its data. Trust, in this case, is implicit. Situation on the cloud is very different. Cloud provider needs to ensure that the tenant's data is always protected from unauthorized (i.e. without consent of the owner of data) access as well as destruction. There's more to it than just employing appropriate encryption and authentication mechanisms. Laws and regulations of the land have important roles to play. Laws in various countries give certain rights to the government agencies to gain access to data, under some circumstances, without explicit consent from the owner. Since cloud providers usually host data and services in diverse geographical areas, so the issue of data security becomes complicated and difficult to analyse and manage.

2) Platform Software Safety: Clouds essentially have introduced two key entities into the datacenter's ecosystem: a) virtualization of datacenter resources and b) the concept of hardware level multitenancy, i.e., multiple VMs sharing the same hardware. In a multitenant environment like cloud, isolation amongst tenants is dependent on the virtualization layer's (i.e. hypervisor's) capabilities and robustness. Potential vulnerabilities in the virtualization software increase the attack surface area for the entire cloud. On a public cloud, one has to worry about keeping both the hypervisor as well as the guest OS running the VM secure. In a traditional datacenter, however, one is mainly concerned about keeping only the OS properly patched up for security vulnerabilities. In addition to this, the *elasticity* of the cloud platforms makes it complicated to manage the scaling of infrastructure artifacts like the VM specific firewalls etc. and the system policies at VM level.

D. Scalability

An application's scalability is primarily determined by its design. The massive scalability afforded by the cloud is attributed to certain design decisions about the application's architecture. Only a specific class of applications can be massively scaled on cloud. The application scalability on cloud comes mainly from:

- By using BASE transactions instead of ACID [11], [12]

TABLE V
ATTRIBUTE SCORES FOR *Scalability*

Attribute	**Cloud Aware**	**Virtualized**
Horizontal scaling	Better	Same
Vertical scaling	Better	Better
Ability to exploit NoSQL based data stores	Better	Same
Data replication	Better	Better
Inter-tier I/O strategy (collocated vs. distributed tiers).	Better	Better

- By leveraging shared-nothing designs for state and data
- By exploiting parallelism present in the application's logic so as to utilize MapReduce kind of tools

To be able to be massively scalable, an application's logic needs to be suitable for exploiting at least one of the above. As such, there's nothing in the *cloud itself* that makes an application more scalable than what it would have been on a traditional platform. Of course, the elasticity afforded to cloud by virtualization makes it easy to dynamically add and remove resources in response to the application's scalability needs.

E. Availability

In software engineering literature there are varying definitions of availability for software systems, however, the central theme of most of them is that the system's availability is typically determined from its ability to consistently meet the *performance* and *correctness* Service Level Agreements (SLA) [37], [38].

Conceptually, the availability A of the system is calculated as:

$$A = \frac{MTBF}{MTBF + MTTR} \tag{4}$$

Here, $MTBF =$ Mean time between failure; and $MTTR =$ Mean time to repair.

The attributes that contribute towards the software system's availability are, therefore, the ones that affect both $MTBF$ and $MTTR$. These factors are listed in the Table-VI. Most of the factors that affect the $MTTR$ score better on cloud based approach. This is the natural outcome of the resource virtualization and use of VMs on cloud platforms. For instance, restoring a failed virtual machine is much faster than restoring a physical machine. Also, the elastic nature of the cloud based platforms makes it easier and faster to react to changing load patterns. Thus the degradation of availability arising out of the *performance SLA* related violations can be handled much faster in the case of cloud platforms. Another aspect is that on cloud platforms, due to economies of scale, the cloud vendor is able to offer better turn around times for certain kind of service requests. For instance, Amazon or Google will be in a position to provide better infrastructure support than the in-house.

Results Discussion

Individual NFQA scores for design approach alternatives and the overall scores for design approach alternatives are depicted in Fig-2 and Fig-3 respectively. It is interesting to see

TABLE VI
ATTRIBUTE SCORES FOR *Availability*

Attribute	**Cloud Aware**	**Virtualized**
Complexity of the software	Worse	Same
Size of the software	Same	Same
Experience in developing the software on the given platform	Worse	Same
Stability and maturity of the dependency software libraries	Worse	Same
Ability to automatically detect failures	Better	Better
Ability to automatically reboot the OS	Better	Better
Live migration ability	Better	Better
Automatic snapshot and restore abilities	Better	Better

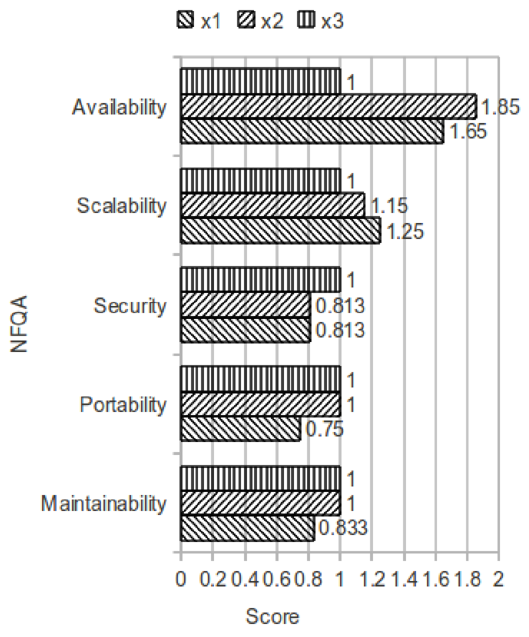

Fig. 2. NFQA scores for design approach alternatives

that the baseline approach (traditional non-cloud) fares better or same as that of the cloud based approaches in all except for availability and scalability NFQAs. When all NFQAs are given equal importance, the *Virtualized* approach performs better than the other two options.

The trend is in accordance with the observed adoption and deployment statistics [39]–[42] of cloud platforms, which indicates that business drivers are the most important criteria for majority of the cloud adoption cases. None of these sources (and we couldn't find any other credible source) indicate the *technical* NFQAs, that concern an application's design and architecture, as the significant driver for adoption of cloud oriented design approach.

V. CONCLUSIONS

It is observed that often it is the business reasons that drive the adoption of cloud oriented platforms for applications development, however, the technical factors like the need to achieve

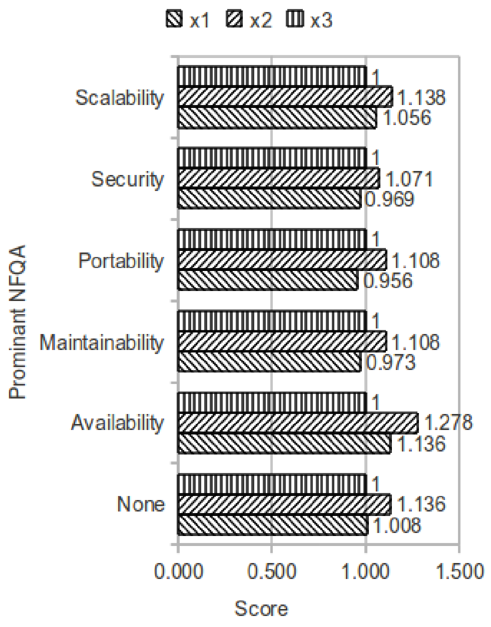

Fig. 3. Overall scores for design approach alternatives

certain non-functional quality attributes from the application also play important role. In this work we introduced and evaluated three application development platforms by considering a set of significant NFQAs. In the evaluation we find that cloud oriented platforms are not always a compelling choice when certain NFQAs are to be ensured in technical design of a business application. Purely from the technical NFQA standpoint, the cloud oriented platforms and design approaches work best for achieving high scalability and availability. Other than that, the cloud oriented design approaches seem no better than the traditional non-cloud and the vanilla virtual platforms. This is a very important observation because even though a cloud oriented platform and design approach may be more beneficial at the *overall solution architecture* level (due to business/economic reasons), but at the level of technical design of an application that may not always be the best choice.

As a future work we would like to include a wider set of NFQAs into the evaluation. We would like to categorize NFQA into domain or significant usecase based groups and then determine how various design options fare. Such analysis can then be incorporated into the the solution architecture process. At this stage, we also have excluded from our evaluation the criteria that assess the plausibility, price, complexity and ease of adopting a given platform for an application. Major reason being that such an evaluation is highly dependent on the application's domain and functionality.

Acknowledgement

This work is partially supported by the the Dept. of Science and Technology, Govt. of India. We'd also like to thank our colleague Ashish Agrawal for his valuable inputs.

References

[1] P. K. Lawlis, K. E. Mark, D. A. Thomas, and T. Courtheyn, "A formal process for evaluating cots software products," *Computer*, vol. 34, pp. 58–63, 2001.

[2] J. Dean, "An evaluation method for cots software products," in *Proceedings of the Twelfth Annual Software Technology Conference*, May 2000.

[3] J. Figueira, S. Greco, M. Ehrogott, and J. Dyer, "Maut - multiattribute utility theory," in *Multiple Criteria Decision Analysis: State of the Art Surveys*, ser. International Series in Operations Research Management Science, F. S. Hillier, Ed. Springer New York, 2005, vol. 78, pp. 265–292.

[4] OpenNebula, "Opennebula 2.0 features," http://www.opennebula.org/documentation:features, OpenNebula Project, retrieved: January 2011.

[5] Eucalyptus, "Features of eucalyptus," http://open.eucalyptus.com/wiki/EucalyptusFeatures_v2.0, Eucalyptus Project, retrieved: January 2011.

[6] Nimbus, "Nimbus documentation: Major features," http://www.nimbusproject.org/docs/2.6/features.html, Nimbus Project, retrieved: January 2011.

[7] D. Chappell, "Introducing the windows azure platform," http://go.microsoft.com/fwlink/?LinkId=158011, David Chappell and Associates, December 2009.

[8] Google, "Google app engine developer's guide," http://code.google.com/appengine/docs/, Google Inc., July 2010.

[9] Amazon, "Amazon web services documentation," https://aws.amazon.com/documentation/, Amazon Inc., July 2010.

[10] D. Pritchett, "Base: An acid alternative," *Queue*, vol. 6, no. 3, pp. 48–55, 2008.

[11] J. Gray, "The transaction concept: Virtues and limitations (invited paper)," in *VLDB*, 1981, pp. 144–154.

[12] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, 1983.

[13] A. S. Foundation, "Apache hadoop," http://hadoop.apache.org/, Apache Software Foundation, retrieved: August 2010.

[14] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[15] N. Leavitt, "Will nosql databases live up to their promise?" *IEEE Computer*, vol. 43, no. 2, pp. 12–14, 2010.

[16] R. Biswas and E. Ort, "The java persistence api - a simpler programming model for entity persistence," http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html, Sun Microsystems, retrieved: August 2010.

[17] S. Microsystems, "Java data objects," http://www.oracle.com/technetwork/java/index-jsp-135919.html, Sun Microsystems, retrieved: August 2010.

[18] D. v. Winterfeld and W. Edwards, *Decision Analysis and Behavioral Research*. Cambridge, England: Cambridge UniversityPress, 1986.

[19] J. Wallenius, J. S. Dyer, P. C. Fishburn, R. E. Steuer, S. Zionts, and K. Deb, "Multiple criteria decision making, multiattribute utility theory: Recent accomplishments and what lies ahead," *Manage. Sci.*, vol. 54, pp. 1336–1349, July 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1527849.1527860

[20] D. Spinellis, *Code Quality: The Open Source Perspective*. Addison Wesley, 2006.

[21] D. Coleman, B. Lowther, and P. Oman, "The application of software maintainability models in industrial software systems," *Journal of Systems and Software*, vol. 29, no. 1, pp. 3 – 16, 1995, oregon Metric Workshop. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0N-3YGV4XF-J/2/4c27230ff284fa2fe8fc716babc866cd

[22] C. U. Smith and L. G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, Boston, MA, 2002.

[23] D. M. Deepak Alur and J. Crupi, *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall, Englewood Cliffs, NJ, 2003.

[24] G. Code, "Google app engine issues list," http://code.google.com/p/googleappengine/issues/list, Google Inc., December 2010.

[25] Amazon, "Aws discussion forums," http://developer.amazonwebservices.com/connect/forumindex.jspa, Amazon Inc., December 2010.

[26] Nimbus, "Nimbus community resources: Discussion and support," http://lists.globus.org/pipermail/workspace-user/, Nimbus Project, retrieved: December 2010.

[27] OpenNebula, "Opennebula: Recent issues tracker," http://lists.opennebula.org/pipermail/users-opennebula.org/, OpenNebula Project, retrieved: December 2010.

[28] Eucalyptus, "Eucalyptus: Recent issues tracker," http://open.eucalyptus.com/tracker, Eucalyptus Project, retrieved: December 2010.

[29] H. A. Linstone and M. Turoff, *The Delphi Method: Techniques and Applications*. Adison-Wesley, Reading, Mass., 1975.

[30] C. C. Mann, "Why software is so bad," *MIT Technology Review*, JULY 2002.

[31] Microsoft, "Msdn code gallery: Azure platform code samples," http://code.msdn.microsoft.com/Project/ProjectDirectory.aspx?TagName=azure, Microsoft Inc., December 2010.

[32] Amazon, "Amazon web services documentation: Sample code libraries," http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=293, Amazon Inc., January 2011.

[33] G. Groups, "Will it play in app engine," http://groups.google.com/group/google-appengine-java/web/will-it-play-in-app-engine?pli=1, Google Inc., retrieved: August 2010.

[34] S. Connect, "Vmware products directory traversal vulnerability," http://www.securityfocus.com/bid/36842/info, securityfocus.com, Oct 2009.

[35] VMware, "Critical vmware security alert for windows-hosted vmware workstation, vmware player, and vmware ace," http://knova-prod-kss-vip.vmware.com:8080/selfservice/dynamickc.do?cmd=show&forward=nonthreadedKC&docType=kc&externalId=1004034&sliceId=1&stateId=, securityfocus.com, Aug 2009.

[36] Secunia, "Secunia advisory sa26986. xen multiple vulnerabilities," http://secunia.com/advisories/26986/, Secunia Inc., Sept 2007.

[37] V. Mainkar, "Availability analysis of transaction processing systems based on user-perceived performance," in *Proceedings of the 16th Symposium on Reliable Distributed Systems*, ser. SRDS '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 10–. [Online]. Available: http://portal.acm.org/citation.cfm?id=829522.830918

[38] S. H. Kan, *Metrics and Models in Software Quality Engineering*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[39] IBM, "The benefits of cloud computing," ftp://public.dhe.ibm.com/common/ssi/ecm/en/diw03004usen/DIW03004USEN.PDF, IBM Corporation, Somers, NY 10589 USA, 2009, retrieved: January 2011.

[40] F. Gens, "Idc survey: What it is likely to move to the cloud?" http://blogs.idc.com/ie/?p=843, IDC, posted: January 2010.

[41] ——, "Idc on the cloud?" http://blogs.idc.com/ie/?p=189, IDC, posted: September 2008.

[42] K. Fogarty, "Which apps should you move to the cloud? 5 guidelines," http://www.cio.com/article/603702/Which_Apps_Should_You_Move_to_the_Cloud_5_Guidelines, CIO.com, posted: August 2010.