International Journal of Electrical Electronics & Computer Science Engineering
Special Issue - EEECOS 2014, ISSN : 2348 2273
Available Online at www.ijeecse.com

Meta Cloud Handlers to Thwart Vendor Lock In

Md. Imran, Y. C. Ashok Kumar, S. Asha Varma.
Department of Computer Science and Engineering, Andhra Loyola Institute of Engineering and Technology, Vijayawada.
ycashokkumar@gmail.com,imran02.md@gmail.com

Abstract: As the rise of Cloud Computing is at its peaks one of the most significant factors that hamper cloud users would be service migration to a parallel cloud service provider due to difficulty in the underlying architectures. So this need of the hour calls for a cloud governed clouds termed meta clouds that can support inter operable service migration. We try to introduce the idea of a meta cloud that can incorporates runtime as well as design time components. This meta cloud is different from existing systems technical incompatibilities, and thus handles vendor lock-in problem very well. It helps an user find the right combination of cloud services for a particular task and supports initial deployment and runtime migration issues of an application.

Keywords: Cloud computing, Cloud API, Cloud Provisioning, Load Balancing, Meta clouds.

I. INTRODUCTION

The cloud computing domain has found new growth with widespread adoption throughout the recent years. Despite many benefits of cloud computing, many enterprises hesitate to move data into the cloud, primarily because of issues related to availability and continuity of service,legal uncertainties and data lock-in.If this are the situations, businesses locked into such clouds are essentially and virtually at a halt until the cloud is back online. Vendor Lock-in is an issue because of these reasons. Even though availability and usage of public clouds is usually free and generally high, eventual outages still occur prompting business to close. Hence, a business locked in a cloud has no control over their own IT costs. And hence data locked into such clouds have no guarantees that this cloud will continue to offer the required Quality of Service (QoS) tomorrow. The main reason for the survival and success of cloud computing is the possibility to use the services when required with a pay as you go pricing packages, which proved to be easy and convenient in many aspects. So the user's data is always at the risk of public cloud providers who generally do not and don't need to guarantee or honor particular service level agreements. The terms of service offered by most public cloud service providers (csp) allow the csp to change pricing of their service at any time unilaterally and is beyond the control of the user. Because of cheap costs and high flexibility, migrating to the cloud is deemed necessary to cut costs.

We can understand the need for the enterprises to regularly monitor and exercise control over the date in the cloud they are using, and they should be able to rapidly change horses without bringing their business to a halt if required. Simply saying migration to an alternate cloud with similar architecture if the above monitoring result identifies problems in the future. However, this sort of migration is currently far from reality mainly attributed to the complex underlying architectures. A pool of cloud services providers are storming the market with a confusing service level agreements, For example Amazon Elastic Cloud (EC2) and VMware cloud, Simple Storage Service (S3) are regarded as heavy cloud players. Some of these services are conceptually same and comparable to each other, while others are vastly different, but all of them are technically incompatible with each other and follow proprietary standards of their own. This further complicates the situation, and many businesses are depending on public clouds for their requirements, but they attempt to combine public cloud services with their own enterprise private cloud, leading to the so-called hybrid cloud setups. In this paper we define the concept of a meta cloud derived from meta cloud API's consisting of a pool of design time and runtime components. They abstract away from technical incompatibilities of existing architecture, resulting in vendor lock-in mitigations. It is helpful to explore a compatible combination of cloud services for a particular requirement and initiate support to an applications deployment and runtime migration based on that.

II. RELATED WORK

The technology strategy to operate a single Web portal to web cast the FIFA 2014 world cup is a futile concept considering the huge amount of traffic it can generate. Although a huge number of dedicated hardware resources can sustain the requirement, the question remains after the requirement where the resources are dedicated to be idle. An event of this size and volume requires huge and dedicated infrastructure and the cloud computing domain provides flexible and elastic services and resources that is needed for such a situation of huge proportions. It's capability to handle short-term scaled traffic sizes without the need to have pre dedicated and deployed resources running all the time.

The issue is, however not the technology but the provider itself considering that once the application has been developed using cloud services of one particular cloud service provider(csp) using their specific API, the application is bound to the provider; migrating the application to a different cloud with similar architecture would usually require a complete redesign and redeployment. This sort of mess is defined as vendor lock-in which leads to strong reliance on the cloud service provider.

In the example of the FIFA portal, besides the ability to scale the application by dynamically allocating/releasing resources,

additional aspects such as resource costs which are again might be growing/shrinking and regional latencies and communication bandwidths have to be considered. Let us assume the fifa streaming portal application is based on the load balancer that forwards HTTP requests to a number of streaming servers hosting the same web application, which allows users to submit a request. Request records are put into a service delivery queue and subsequently stored into a repository. Using Elastic cloud services, this scenario can be realized to host applications. Instead of being bound to one cloud instance, the streaming application should be hosted on a better cloud environment that is void of vendor lock in. In order to leverage this sort of diverse cloud landscape that offers better flexibility, and that can avoid vendor lock-in, the following constraints were need to be achieved by the meta cloud.

1) Find the optimal strategic combination of cloud services for a certain streaming application with respect to QoS and SLAs for the users and the price for hosting.

2) Develop and Deploy a cloud-based application once, then run it anywhere, more importantly including the support for runtime migration which happens to be scope of the current project context.

Lately, the meta cloud idea has received some much needed attention and several approaches were designed and put forward to try to tackle at least parts of the problem. In the next section we will briefly highlight our proposed solutions and their implications.

III. PROPOSED SCHEME

Firstly, standardized kernel programming APIs is used to enable the development of cloud-neutral platform applications, which are not specifically hardwired to any cloud service or single provider. The cloud service provider's abstraction libraries such as libcloud provides unified APIs for accessing and manipulating cloud products of different cloud vendors. Using this open source cross platform libraries, developers are independent of technological cloud vendor locks, as they can switch back and forth between different cloud providers for their deployed applications with relatively low or negligible overhead.

Further the meta cloud api makes use of resource service templates to determine hardwired features that the application requires from the cloud set up. For example, an application requires a certain number of computing resources, database storages, web servers, internet access etc. Some current tools and initiatives (e.g., Amazon's EC2 or the TOSCA specifications) are achieving it as we speak, and can be adapted and upgraded to fit these required features for the meta cloud.

Besides the resource service templates, the automated setup, formation and resource provisioning of cloud applications are expected in the meta cloud. Predictable outcomes and controlled automated smooth application deployment is a central issue for cost-effective and efficient deployments in normal cloud, and even more so for a meta cloud. Several application resource provisioning solutions that are used today (examples include Opscode. Chef, Puppet, and juju) can be extended to Meta clouds too.

At runtime, one important criteria expected of the meta cloud is application monitoring. Because it enables meta cloud to decide whether new instance of the application should be queued and provisioned, or to initiate entire application migration. However, the meta cloud requires more advanced monitoring techniques, especially for making automated provisioning decision at runtime based on context and total number of clients using the applications currently.

Resource Monitoring. The resource monitoring component is responsible for receiving data collected by meta cloud proxies about the resources they are using, on application's request. These data are altered and preprocessed, and then stored to the knowledge base for further processing. This helps to generate comprehensive QoS information of cloud service providers and the particular services they are providing, including response time, availability, and more service specific quality statements.

Provisioning Strategy. The main task of the provisioning strategy component is to match an application's cloud service requirements to actual cloud service providers. It is able to find and rank cloud services based on data in the knowledge base. The initial deployment decision is based on the resource templates, specifying the resource requirements of an application, together with QoS and pricing information about service providers. The result is a ranked list of possible combinations of cloud services regarding expected QoS and costs. At runtime, the component is able to reason about whether migration of a resource to another resource provider is beneficial based on new insights into the application's behavior and updated cloud provider QoS or pricing data. Reasoning about migrating additionally involves calculating migration costs. Decisions of the provisioning strategy result in executing customers defined deployment or migration scripts.

Knowledge Base. The knowledge base serves as store for data about cloud provider services, their pricing and QoS, and information necessary to estimate migration costs. Customer provided resource templates and migration/deployment recipes are stored in the knowledge base as well. Also, the knowledge base indicates which cloud providers are eligible for a certain customer. These usually comprise all providers the customer has an account with and providers that over possibilities to create (sub) accounts on the fly. A number of different information sources contribute to the knowledge base: meta cloud proxies regularly send data about application behavior and cloud service QoS. Pricing and capabilities of cloud

International Journal of Electrical Electronics & Computer Science Engineering
Special Issue - EEECOS 2014, ISSN : 2348 2273
Available Online at www.ijeecse.com

service providers may be either added manually or by crawling techniques able to get this information automatically, like [2].

To a great extent, the meta cloud can be constructed and deployed based on the combination of existing tools, concepts and libraries, part of which are are discussed previously. The main components of the meta cloud, depicted in the following Figure 1, are described in the following and their interplay is explained using the previously introduced streaming portal example. The components of the meta cloud can be distinguished whether they are static or dynamic.

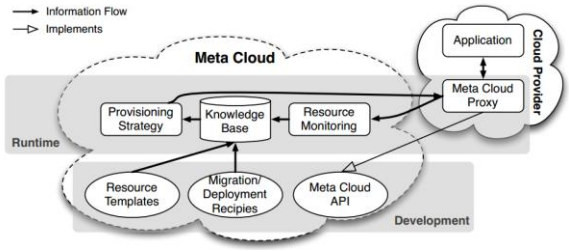

Fig. 1 : Architecture

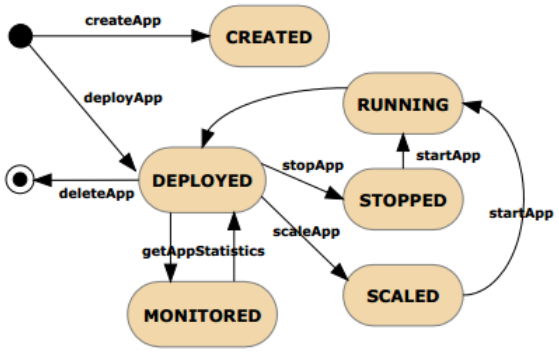

Fig. 2 : Application Lifecycle

The implemented algorithm besides avoiding interruptions during playback, maximizing video visible quality, reducing the number of video quality phase shifts, it importantly minimizes the delay between user's request and the start of the playback which the prior attempts failed to handle.

Although the meta cloud API provides a unified programming interface that prevents a client to their application from being hard-wired to a specific cloud service offering by joining a streaming cloud based on the estimations of a load balancer. Cloud Migration during real time is not supported yet. So we propose to implement the migration algorithm in cloud streaming servers that can help to achieve interactivity between them by aiding the transfer of client between different clouds based on activity. Helps to achieve better migration of client between different servers even after load balancing.

ADAPTATION ALGORITHM

Input: $(\sigma_i)_{i=1,\dots,n(t)}$
Output: $r_{n(t)+1}$, B_{delay}

static runningFastStart := true;
$B_{\text{delay}} := 0$;
$r_{n(t)+1} := r_{n(t)}$;
if runningFastStart ...
  $\wedge\ r_{n(t)} \neq r_{\max}$ ...
  $\wedge\ \beta_{\min}(t_1) \leq \beta_{\min}(t_2)\ \forall t_1 < t_2 \leq t$ ...
  $\wedge\ r_{n(t)} \leq \alpha_1 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
  if $\beta(t) < B_{\min}$ **then**
    if $r^{\uparrow}_{n(t)} \leq \alpha_2 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
      $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;
  else if $\beta(t) < B_{\text{low}}$ **then**
    if $r^{\uparrow}_{n(t)} \leq \alpha_3 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
      $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;
  else
    if $r^{\uparrow}_{n(t)} \leq \alpha_4 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
      $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;
    if $\beta(t) > B_{\text{high}}$ **then**
      $B_{\text{delay}} := B_{\text{high}} - \tau$;
else
  runningFastStart := false;
  if $\beta(t) < B_{\min}$ **then**
    $r_{n(t)+1} := r_{\min}$;
  else if $\beta(t) < B_{\text{low}}$ **then**
    if $r_{n(t)} \neq r_{\min} \wedge r_{n(t)} \geq \tilde{\rho}_{n(t)}$ **then**
      $r_{n(t)+1} := r^{\downarrow}_{n(t)}$;
  else if $\beta(t) < B_{\text{high}}$ **then**
    if $r_{n(t)} = r_{\max} \vee r^{\uparrow}_{n(t)} \geq \alpha_5 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
      $B_{\text{delay}} := \max(\beta(t) - \tau, B_{\text{opt}})$;
  else
    if $r_{n(t)} = r_{\max} \vee r^{\uparrow}_{n(t)} \geq \alpha_5 \cdot \tilde{\rho}(t - \Delta_t, t)$ **then**
      $B_{\text{delay}} := \max(\beta(t) - \tau, B_{\text{opt}})$;
    else
      $r_{n(t)+1} := r^{\uparrow}_{n(t)}$;

The Migration Algorithm.

1: Each player i computes its anticipation to each commodity k in the market, *i.e.*, A_i^k. Then, it derives the relative distance d_i^k, and the corresponding utility-distance product ϕ_i^k, given by $\phi_i^k = d_i^k \cdot A_i^k$.
2: Each player i computes the cumulative utility-distance product p_i^k for all commodities, according to their relative distances d_i^k.
3: Each player i determines his pivot point μ_i
4: Each player i finds a subset of commodities $\mathcal{C}_i$, whose cumulative utility-distance products are not larger than μ_i, *i.e.*, $\mathcal{C}_i = \{k : p_i^k \leq \mu_i, \forall k \in \mathcal{M}\}$.
5: **if** Commodity k belongs to more than one $\mathcal{C}_i, \forall i \in \mathcal{N}$ **then**
6: The player i' $(k \in \mathcal{C}_{i'})$ who has the smallest d_i^k to commodity k will obtain this commodity.
7: **else**
8: The player i' $(k \in \mathcal{C}_{i'})$ will obtain this commodity.
9: **end if**
10: All players migrate their commodities according to the obtained assignment results.

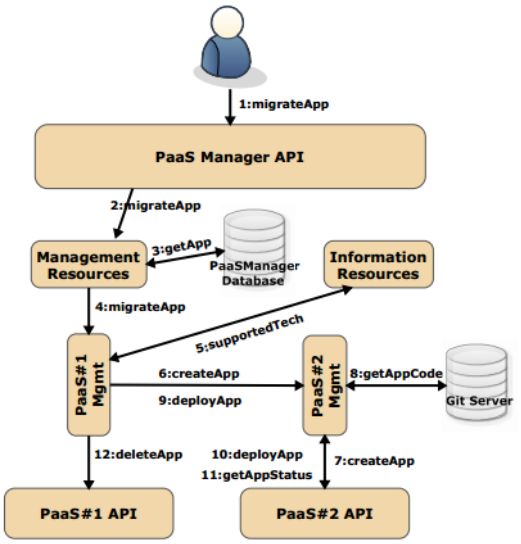

Fig. 3 : Migration

IV. CONCLUSION AND FUTUREWORK

With the increasing number of cloud service providers, the exploration for the best platform to deploy and manage application instances is always a critical factor and may not always lead to optimum results. This paper addresses the aforementioned inconvenient relating load balancing; cloud migration and importantly vendor lock in. It does so by proposing a high-level comprehensive cross platform inter-operable architecture intended to ease various application stages, such as, deployment, management, and monitoring of cloud applications over the surviving cloud service models. The proposed architecture is a key part of the framework by constantly delivering and evaluating, the most appropriate platform specifics from a catalog of cloud offerings, based on the application's resource usage profile or pre-defined cost thresholds.

The meta cloud is designed to mitigate vendor lock-in and attains transparent use of cloud computing services. Some basic technologies required to realize the meta cloud are already developed, yet they lack integration. The integration of these efficient and remarkable tools promises a huge leap in the direction of meta cloud. For avoiding vendor lock-in it is important that the cloud community drives these ideas, to create a truly open meta cloud interface with added utility for all customers with broad support from different providers and implementation technologies. Handling of Vendor lock in problem is PaaS and IaaS architectures are much more complex which involves Virtual machine migration algorithms. This can be attributed to a future research.

V. REFERENCES

[1] Michael Armbrust, Armando Fox, Rean Grith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. Commun. ACM, 53(4):50{58, April 2010.

[2] Marios D. Dikaiakos, Asterios Katsifodimos, and George Pallis. Minersoft: Software retrieval in grid and cloud computing infrastructures. ACM Transactions on Internet Technology (ACM TOIT), 12(1), July 2012.

[3] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A Taxonomy and Survey of Cloud Computing Systems. In International Conference on Networked Computing and Advanced InformationManagement, pages 44{51, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[4] James Skene, D. Davide Lamanna, and Wolfgang Emmerich. Precise Service Level Agreements. In Proceedings of the 26th International Conference on Software Engineering (ICSE'04), pages 179{188, Washington, DC, USA, 2004. IEEE Computer Society.

[5] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. J. Internet Services and Applications, 1(1):7{18, 2010.

[6] D. Collison, "Distributed Design and Architecture ofCloud Foundry,"http://www.slideshare.net/derekcollison/design-of-cloud-foundry, 2011,[Online; accessed 9 May 2012].

[7] F. Paraiso, N. Haderer, P. Merle, R. Rouvoy, and L. Seinturier, "A Federated Multi-Cloud PaaS Infrastructure," in 5th IEEE International Conference on Cloud Computing. hawaii,´ Etats-Unis: IEEE Xplore Digital Library, Jun. 2012. [Online]. Available: http://hal.inria.fr/hal-00694700.

[8] Z. ur Rehman, F. Hussain, and O. Hussain, "Towards multi-criteria cloud service selection," in Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on, 30 2011-july 2 2011, pp. 44 –48