International Journal of Science Engineering and Advance Technology

Cloud Changing Paradigm With The Meta Cloud

G Ravindra kumar#1, P P S Naik#2
#1Student of M.Tech (CSE) and Department of Computer Since Engineering,
#2 Department of Computer ScienceDr.Samuel George Institute of Engineering & Technology,Markapur

Abstract— It is a techno-business disruptive model of using distributed large-scale data centers either private or public or hybrid offering customers a scalable virtualized infrastructure or an abstracted set of services qualified by service-level agreements (SLAs) and charged only by the abstracted IT resources consumed.The occurrence of yet more cloud contributions from a crowd of service providers calls for a Meta cloud to smoothen the edges of the pointed cloud background. This Meta cloud could solve the hawker lock-in problems that current public and hybrid cloud users face.

Index Terms— Cloud, Resource Patterns, Migration and Deployment Recipes, Hawker, Proxy.

1. Introduction:

The cloud computing paradigm has achieved prevalent implementation in recent years. Its success is due principally to customers' ability to use services on demand with a pay-as-you go pricing model, which has proved suitable in many compliments. Low costs and high flexibility make migrating to the cloud compelling. Although its understandable advantages, however, many companies waver to "move to the cloud," mainly because of worries related to service availability, data lock-in, and legal suspicions.

1 Lock- in is mainly complicated. For one thing, even though public cloud availability is generally high, outages still occur.

2 Businesses locked into such a cloud are fundamentally at a standstill until the cloud is back online. Moreover, public cloud providers generally don't guarantee particular service level contracts (SLAs)

3 — that is, businesses locked into a cloud have no promises that it will continue to provide the essential quality of service (QoS).

As a final point, most public cloud providers' terms of service let that provider separately change pricing at any time. Hence, a business locked into a cloud has no mid- or long- term control over its own IT costs. At the fundamental of all these difficulties, we can classify a need for businesses to eternally monitor the cloud they're using and be able to hastily "change horses" — that is, migrate to a different cloud if they discover problems or if their estimates predict future issues. However, migration is currently far from insignificant. Many cloud Providers are swamping the market with an uncertain body of services, including compute services such as the Amazon Elastic Compute Cloud (EC2) and VMware vCloud, or key-value stores, such as the Amazon Simple Storage Service (S3). Some of these services are conceptually comparable to each other, whereas others are infinitely different, but they're all, ultimately, technically incompatible and follow no standards but their own. To further complicate the situation, many companies not (only) build on public clouds for their cloud computing needs, but combine public offerings with their own private clouds, leading to so-called hybrid cloud setups.Here, we introduce the concept of a Meta cloud that incorporates design time and runtime mechanisms. This Meta cloud would abstract away from surviving offerings' technical in compatibilities, thus mitigating hawker lock-in. It helps users find the right set of cloud services for a particular use case and supports an application's initial deployment and runtime migration.

2. Current Weather in the (Meta) Cloud

First, harmonized programming APIs must enable developers to create cloud-neutral applications that aren't hardwired to any single provider or cloud service. Cloud provider abstract- ion libraries such as libcloud (http:// libcloud.apache.org), fog (http: //fog. io), and jclouds (www.jclouds.org) provide unified APIs for accessing different hawkers' cloud products. Using these libraries, developers are relieved of technological hawker lock- in because they can switch cloud providers for their applications with relatively low overhead. As a second ingredient, the Meta cloud uses resource templates to define concrete features that the application requires from the cloud. For instance, an application must be able to specify that it requires a given number of com- putting resources, Internet access, and database storage. Some current tools and initiatives — for example, Amazon's Cloud Formation (http:// aws.amazon.com/cloud formation/) or the upcoming TOSCA specification (www.oasis -open.org/committees/ Tosca) — are working toward similar goals and can be adapted to provide these required features for the Meta cloud. In addition to resource templates, the automated formation and pro- visioning of cloud applications also depends on sophisticated features to actually deploy and install applications automatically.

Predictable and controlled application deployment

is a central issue for cost-effective and efficient deployments in the cloud, and even more so for the Meta cloud. Several application provisioning solutions exist, enabling developers and administrators to declaratively specify deployment artefacts and dependencies to allow for repeatable and managed resource provisioning. Notable examples include Opscode Chef (www.opscode.com/chef/), Puppet (http://puppetlabs.com), and juju (http://juju.ubuntu.com). At runtime, an important aspect of the Meta cloud is application monitoring, which enables the Meta cloud to decide whether it's necessary to provision new instances of the application or migrate parts of it. Various hawkers provide tools for cloud monitoring, ranging from system-level monitoring (such as CPU and bandwidth) to application-level monitoring (Amazon's Cloud Watch; http://aws.amazon.com/cloudwatch/) to SLA monitoring (as with moony- tis; http: //portal.monitis.com/index. php/cloud-monitoring). However, the Meta cloud requires more sophisticated monitoring techniques and, in particular, approaches for making automated provisioning decisions at runtime based on cur- rent application users' context and location.

3. Meta Cloud Proxy

The Meta cloud provides proxy objects, which are deployed with the application and run on the provisioned cloud resources. They serve as intermediaries between the application and the cloud provider. These proxies expose the Meta cloud API to the application, transform application requests into cloud-provider-specific requests, and forward them to the respective cloud services. Proxies provide a way to execute deployment

And migration recipes triggered by the Meta cloud's provisioning strategy. Moreover, proxy objects send QoS measurements to the resource observing component running within the Meta cloud. The Meta cloud obtains the data by intercepting the application's calls to the underlying cloud services and measuring their pro- cessing time, or by executing short benchmark programs. Applications can also define and monitor custom QoS metrics that the proxy objects send to the resource observing component to enable advanced, application-specific management schemes. To avoid high load and computational blockages, communication between proxies and the Meta cloud is kept at a minimum. Proxies don't run inside the Meta cloud, and regular service calls from the application to the proxy aren't routed through the Meta cloud, either.

3.1. Inside the Meta Cloud

To some extent, we can realize the Meta cloud based on a combination of existing tools and concepts, part of which we just examined. Figure 1 depicts the Meta cloud's main components. We can classify these components based on whether they're important mainly for cloud software engineers during development time or whether they perform tasks during runtime. We demonstrate their relationship using the sports betting portal example.

patterns allow developers to define constraints based on costs, component proximity, and geographical distribution.

4. Migration and Deployment Recipes

Deployment recipes are an important ingredient for automation in the Meta cloud infrastructure. Such recipes

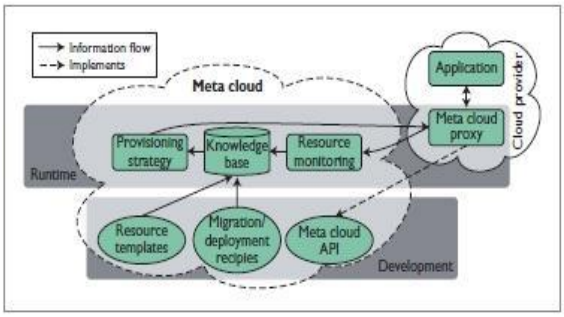

Figure 1. Theoretical Meta cloud summary. Developers create cloud applications using Meta cloud development mechanisms. The Meta cloud runtime extracts from provider particulars using proxy objects, and automates application life-cycle management.

3.2 Meta Cloud API

The Meta cloud API provides a unified programming interface to abstract from the differences among provider API operations. For customers, using this API prevents their application from being hard-wired to a specific cloud service offering**.** The Meta cloud API can build on available cloud provider abstraction APIs, as previously mentioned. While these deal mostly with key- value stores and compute services, in principle, all services can be covered that are abstract enough for more than one provider to offer and who's explicit APIs don't differ too much, conceptually.

3.3 Resource Patterns

Developers describe the cloud services necessary to run an application using resource patterns. They can specify service types with additional properties, and a graph model expresses the interrelation and functional dependencies between services. Developers create the Meta cloud resource patterns using a simple domain-specific language (DSL), letting them briefly specify required resources. Resource definitions are based on a hierarchical composition model; thus developers can create configurable and reusable pattern components, which enable them and their teams to share and reuse common resource patterns in different projects. Using the DSL, developers model their application components and their basic runtime requirements, such as (provider- independently normalized) CPU, memory, and I/O capacities, as well as dependencies and weighted communication relations between these components. The provisioning strategy uses the weighted

component relations to determine the application's optimal deployment configuration. Moreover, resource

Allow for controlled deployment of the application, including installing packages, starting required services, managing package and application parameters, and establishing links between related components. Automation tools such as Opscode Chef provide an extensive set of functionalities that are directly integrated into the Meta cloud environment. Migration recipes go one step further and describe how to migrate an application during runtime — for example, migrate storage functionality from one service provider to another. Recipes only describe initial deployment and migration; the provisioning strategy and the meta cloud proxy execute the actual pro-cess using the aforementioned automation tools.

4.2 Cloud Computing Use Case

Let's consider a Web-based sports portal for an event such as the Olympic Games, which allows users to place bets. An event this large requires an enormously efficient and reliable infrastructure, and the cloud computing paradigm provides the necessary flexibility and elasticity for such a scenario. It lets service providers handle short-term usage spikes without needing respective dedicated resources available continuously. The problem, however, is that once an application has been developed based on one particular provider's cloud services and using its specific API, that application is bound to that provider; deploying it on another cloud would usually require completely redesigning and rewriting it. Such hawker lock-in leads to strong dependence on the cloud service operator. In the sports portal example, in addition to the ability to scale applications up and down by dynamically allocating and releasing resources, we must consider additional aspects, such as resource costs and regional communication bandwidth and latency. Let's assume the sports betting portal application is based on a load balancer that forwards HTTP requests to numerous computing nodes hosting a Web application that lets users submit a bet. Request handlers place bet records in a message queue and subsequently store them in a relational database. Let's further assume a service provider realizes this scenario using only Amazon Web Services (AWS), EC2 to host applications, Simple Queue Service (SQS) as its cloud message queue, and the Relational Database Service (RDS) as a database system. Instead of being bound to one cloud operator, however, the betting application should be hosted in an optimal cloud environment. To leverage a more diverse cloud landscape, support flexibility, and avoid hawker lock-in, the Meta cloud must achieve two main goals:

• find the optimal combination of cloud services for a certain application with regard to QoS for users and price for hosting; and

• develop a cloud-based application once, then run it anywhere, including support for runtime migration.

Lately, the Meta cloud idea has received some attention, and several approaches try to tackle at least parts of the problem.

4.3 Proposed Algorithm

4.3.1 Encryption:

```
byte
state[4,Nb]
state = in
AddRoundKey(state,keySchedule[0,
Nb-1]) for round = 1 step 1 to Nr–1
 {
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
AddRoundKey(state,keySchedule[round*Nb,
(round+1)*Nb-1])
}
SubBytes(state)
ShiftRows(state)
AddRoundKey(state,keySchedule[Nr*Nb,
(Nr+1)*Nb-1]) out = state
```

4.3.2 Decryption:

```
byte
state[4,Nb]
state = in
AddRoundKey(state,keySchedule[Nr*Nb,
(Nr+1)*Nb-1]) for round = Nr-1 step -1 downto 1 {
        InvShiftRows(state)
        InvSubBytes(state)
AddRoundKey(state,keySchedule[round*Nb,
(round+1)*Nb-1]) InvMixColumns(state)
}
InvShiftRows(state)
InvSubBytes(state)
AddRoundKey(state,keySchedule[0,
Nb-1]) out = state
```

4.3 Resource Monitoring

On an application's request, the resource monitoring component receives data collected by meta cloud proxies about the resources they're using. The component filters and pro- cesses these data and then stores them on the knowledge base for further processing. This helps generate compre- hensive QoS information about cloud service providers and the specific services they provide, including response time, availability, and more service-specific quality statements.

4.4 Knowledge Base

The knowledge base stores data about cloud provider services, their pricing and QoS, and information necessary to estimate migration costs. It also stores customer-provided resource templates and migration or deployment

recipes. 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 offer possibilities for creating (sub) accounts on the fly. Several information sources contribute to the knowledge base: Meta cloud proxies regularly send data about application behaviour and cloud service QoS. Users can add cloud service providers' pricing and capabilities manually or use crawling techniques that can get this information automatically.

5. A Meta Cloud Use Case

Let's come back to the sports application use case. A meta-cloud-compliant variant of this application accesses cloud services using the Meta cloud API and doesn't directly talk to the cloud-provider-specific service APIs. For our particular case, this means the application doesn't depend on Amazon EC2, SQS, or RDS service APIs, but rather on the Meta cloud's compute, message queue, and relational database service APIs.

For initial deployment, the developer submits the application's resource template to the Meta cloud. It specifies not only the three types of cloud services needed to run the sports application, but also their necessary properties and how they depend on each other. For compute resources, for instance, the developer can specify CPU, RAM, and disk space according to terminology defined by the Meta cloud resource template DSL. Each resource can be named in the template, which allows for referencing during deployment, runtime, and migration.

The resource template specification should also contain interdependencies, such as the direct connection between the Web service compute instances and the message queue service. The rich information that resource templates provide helps the provisioning strategy component make profound decisions about cloud service ranking. We can explain the working principle for initial deployment with a Web search analogy, in which resource templates are queries and cloud service provider QoS and pricing information represent indexed documents. Algorithmic aspects of the actual ranking are beyond this article's scope. If some resources in the resource graph are only loosely coupled, then the Meta cloud will be more likely to select resources from different cloud providers for a single application. In our use case, however, we assume that the pro- visioning strategy ranks the respective Amazon cloud services first, and that the customer follows this recommendation. After the resources are deter-mined, the Meta cloud deploys the application, together with an instance of the Meta cloud proxy, according to customer-provided recipes.

During runtime, the Meta cloud proxy mediates between the application components and the Amazon cloud resources and sends monitoring data to the resource monitoring component running within the Meta cloud. Monitoring data helps refine the application's resource template and the provider's overall QoS values, both stored in the knowledge base. The provisioning strategy compo-nent regularly checks this updated information, which might trigger a migration. The Meta cloud could migrate front-end nodes to other providers to place them closer to the application's users, for example. Another reason for a migration might be updated pricing data.

After a price cut by Rack space, for example, services might migrate to its cloud offerings. To make these decisions, the provisioning strategy component must consider potential migration costs regarding time and money. The actual migration is per- formed based on customer-provided migration recipes. Working on the Meta cloud, we face the following technical challenges. Resource monitoring must collect and process data describing different cloud providers' services such that the provisioning strategy can compare and rank their QoS properties in a normalized, provider- independent fashion.

Although solutions for deployment in the cloud are relatively mature, application migration isn't as well supported. Finding the balance between migration facilities provided by the Meta cloud and the application is particularly important. Cloud-centric migration makes the Meta cloud infrastructure responsible for most migration aspects, leading to issues with application- specific intricacies, whereas in application-centric migration, the Meta cloud only triggers the migration process, leaving its execution mostly to the application. We argue that the Meta cloud should control the migration process but offer many interception points for applications to influence the process at all stages. The provisioning strategy — the most integrative component, which derives strategies mainly based on input from runtime monitoring and resource templates and effects them by executing migration and deployment recipes — requires further research into combining approaches from the information retrieval and autonomic computing fields.

6. Conclusions:

The Meta cloud can help mitigate hawker lock-in and promises apparent use of cloud computing services. Most of the basic technologies necessary to realize the Meta cloud already exist, yet require combination.

If we are facing any problems with the cloud service provider then we can migrate to another cloud service provider.

Thus, integrating these state-of-the-art tools promises a huge leap toward the Meta cloud. To avoid Meta cloud lock- in, the community must drive the ideas and create a truly open Meta cloud with added value for all customers and broad support for different providers and implementation technologies.

References:

1) M.D. Dikaiakos, A. Katsifodimos, and G. Pallis, "Minersoft: Software Retrieval in Grid and Cloud Computing Infrastruc- tures," ACM Trans. Internet Technology, vol. 12, no. 1, 2012, pp. 2:1–2:34.
2) B.P. Rimal, E. Choi, and I. Lumb, "A Tax-onomy and Survey of Cloud Computing Systems," Proc. Int'l Conf. Networked Com-puting and AdvancedInformation Man-agement, IEEE CS Press, 2009, pp. 44–51.
3) M. Armbrust et al., "A View of Cloud Computing," Comm. ACM, vol. 53, no. 4, 2010, pp. 50–58.
4) Q. Zhang, L. Cheng, and R. Boutaba, "Cloud Computing: State-of-the-Art and Research Challenges," J. Internet Services and Applications, vol. 1, no. 1, 2010, pp. 7–18.
5) J. Skene, D.D. Lamanna, and W. Emmerich, "Precise Service Level Agreements," Proc. 26th Int'l Conf. Software Eng. (ICSE 04), IEEE CS Press, 2004, pp. 179– 188.

Authors Profile

G Ravindra Kumar is a student of Computer Science Engineering from Dr.Samuel George Institute of Engineering & Technology, Presently pursuing M.Tech (CSE) from this college. He received B.Tech from JNTUK in the year Of 2012.

P P S Naik is a Associate Professor in the Department of CSE in Dr.Samuel George Institute of Engineering & Technology, JNTU Kakinada University