FluidCloud: An Open Framework for Relocation of Cloud Services

Andy Edmonds
Zürcher Hochschule für Angewandte Wissenschaften

Thijs Metsch
Intel Ireland Limited

Dana Petcu
Institute e-Austria Timisoara

Erik Elmroth
Umeå University

Jamie Marshall
Prologue

Plamen Ganchosov
CloudSigma

Abstract

Cloud computing delivers new levels of being connected, instead of the once disconnected PC-type systems. The proposal in this paper extends that level of connectedness in the cloud such that cloud service instances, hosted by providers, can relocate between clouds. This is key in order to provide economical and regulatory benefits but more importantly liberation and positive market disruption.

While service providers want to lock in their customer's services, FluidCloud wants the liberation of those and thereby allow the service owner to freely choose the best matching provider at any time. In the cloud world of competing cloud standards and software solutions, each only partially complete, the central research question which this paper intends to answer: **How to intrinsically enable and fully automate relocation of service instances between clouds?**

1 Introduction

Today, cloud computing [1] service instances cannot easily move from one cloud service provider to another. Cloud standards are seen to be the panacea, yet have little adoption by the market, especially by the market's dominant players. When adopted, *de jure* standards are not as widely adopted as *de facto* standards (e.g. Amazon EC2). Software libraries and frameworks that abstract cloud computing services to common interfaces are more widely adopted (see "Related Work"). However, even the most relevant standards or software libraries have little or no service instance relocation functionality. Ultimately, those cloud service instances remain locked under the control of the service provider, unless significant manual and/or ad hoc efforts are spent by the service instance owners.

The proposed solution is the FluidCloud framework which aims to make relocating services instances easier any autonomous. From this work a number of research and engineering challenges arise including data optimisation, runtime architecture adaptation, and goal-oriented service instance relocation.

2 A Problem in the Cloud?

Cloud service instances remain locked under the control of the service provider. FluidCloud will liberate these instances. Having the ability for a cloud service instance to easily and seamlessly move from one provider to another will bring advantages to any cloud service owner. It would bring more freedom to cloud service owners and their instances. Essentially, it will bring service instance "movement rights" to the cloud. However there is no encompassing means to accomplish this.

FluidCloud fits within the soon future cloud. A reasonable view of this future cloud is the InterCloud. The InterCloud is described in [2], [3], [4] where the genesis and progression of it from singular and multi-cloud ecosystem is stated. The concept of the InterCloud is based on the proliferation and continued growth of public clouds ranging from IaaS, PaaS and up to SaaS. The ecosystem of these cloud service providers include the popular Amazon EC2, Rackspace, and CloudSigma.

2.1 The FluidCloud Concept

Within the highly populated ecosystem of cloud service providers combined with the InterCloud concept, it becomes crucial, both technically and financially, that service instances can be relocated and consequently adapted to their new service provider. In the vision of the FluidCloud framework, the hosting cloud provider of the service instance is not a concern anymore as the service can be fluidly (easily, on-demand and dynamically) relocated between providers. The FluidCloud concept addresses multiple benefits. FluidCloud will bring **liberation** to cloud service developers and operators who own services

and are responsible for end-user data. They should have the option of movement for those services instances and the efficient means to enable them. It will **enhance the InterCloud** by advancing the definition, architecture and implementations of cloud computing, yet for stakeholders makes the transition easy through the open source framework. It will make it more **economical** by suggesting new compatible service providers, based on economical differentiators. And should the service owner want it relocate their service instance to the suggested target provider. **Regulatory** control can be supplied so if a running cloud service instance is in an unrecognised or risky geographic region, the service provider can use FluidCloud to relocate that service instance at risk. Finally, with ease of relocation, **positive market disruption**, the market place is opened further, enabling greater competition based on service provider differentiation and not on technical lock-in or limitation is key.

TODO: I'm not sure I understand the "Regulatory" bit in the last paragraph of section 2.1. I think of Regulation as being about governments or overseeing bodies setting up rules, but that doesn't seem like what this means. This seems to be about recognizing with a service is at risk? Why is this regulatory?

2.2 FluidCloud Scenarios

To see how FluidCloud can be used in a practical sense, consider some examples which demonstrate the need for the FluidCloud framework.

A **Cloud Service Developer** (CSD, e.g. University startup). Take the example where a university startup implements a new service in the cloud. The type of service is one that is architected to handle bursty traffic as described in [5]. After a period of time, the selected provider does not satisfy from technical (e.g. provider outages), economical (e.g. provider increasing prices) or regulatory purposes, due to service offer changes. Now the benefits from a FluidCloud concept is that the startup can easily relocate their service to a new cloud service provider.

A **Cloud Service Provider** (CSP; e.g. CloudSigma, ProfitBricks). A cloud service provider would like to relocate (or "on board") new customers from other cloud service providers. What is needed is to relocate new end-user service instances to their services from other, potentially, competing, differentiated services. This would allow the CSP to let their customers seamlessly relocate to their service offering. The cloud service provider operates the FluidCloud framework and offers it as a service.

A **Cloud Broker** (e.g. CompatibleOne or Zimory). Developers and providers of cloud brokerage services and software should consider adding cloud service relocation functionality to their cloud brokerage offers. Typically, a cloud broker discovers and provisions a cloud service instance on the behalf of the service owner. Once that target service instance is provisioned the end-user interacts and uses the target service instance, either through interfaces provided by the cloud broker or directly using the interfaces of the provisioned target service instance. The cloud broker is aware of the end-users service instances and can continually watch for compatible service types that can be offered to the end-user as a replacement, based on economic/geo-location/ performance/feature/cost bases. If the service owner was interested the cloud broker can, using FluidCloud, relocate the service on the owner's behalf.

2.3 FluidCloud Contributions

For such scenarios, as described above, to be technically realised there is a set of missing technologies. FluidCloud fills these gaps by providing the following key contributions:

Service Instance Relocation – *Ensuring the overall orchestration and process of moving a cloud service from the source to the target cloud service provider*. There are two types of cloud services that FluidCloud will support and enable relocation for: IaaS and PaaS based services. The decision to relocate will be initiated by the owner of the service (e.g. through a user interface or API).

Service Instance Adaptation - *The conversion, transformation and movement of the service and its related data*. Related to relocating IaaS and PaaS services are the potential service adaptations that need to take place. Parts of the service might need to be adapted when relocated. For example virtual machines may need to be re-contextualised as its environment changes [6]. If we turn our attention to the PaaS area service instances need might need to be adopted (on source code level) for a certain target platform.

Data Relocation - *Relocation, migration, transformation and conversion of the data belonging to the service*. Relocation of data fundamentally means moving bits and bytes. Currently tools such as GlobusOnline provide a service for easily transferring data between Grid sites using the proven GridFTP protocol [7]. Certainly technologies like Software Defined Networking can help when data paths are needed on-demand to be established between two providers.

3 Architecture

Core to realising FluidCloud are the following components shown in the proposed logical architecture.

TODO: increase size of fig!

The key components are the following:

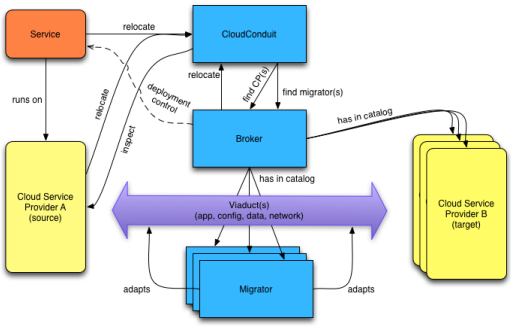

Figure 1: Conceptual Architecture

- **Service Instance** - A logical container that comprises the application and the data.
- **CloudConduit** - Orchestrates the process, introspects the service instances (incl. topology) to be relocated. It is also responsible for the lifecycle of Migrators and setup of viaducts. The relocation is triggered with this module.
- **Broker** - Discovers and provides both cloud provider services and Migrator facilities. Eventually it monitors and inspects the service instances. Also orchestrates the deployment of the Migrators.
- **Migrator** - These are the libraries and services for adaptation and carry out one specific task related to relocation (possibly partial) of service instances. Multiple Migrators might be needed to carry out the overall relocation.
- **Viaduct** - A logical path between two parties in which Migrators are organised. All together those accomplish the task of relocating a service instance from one cloud provider to another. Migrators as well as more network-oriented components (like proxies) are located on this path. All those components might be organised as workflows with multiple pipelines if needed.
- **Cloud Service Provider** - A provider of either IaaS or PaaS service types.

In the diagram above, the CloudConduit analyses the service to be relocated and based on that it uses the Broker to find suitable replacement providers. Based on the replacement provider the CloudConduit uses the Broker again to find suitable Migrators to aid the relocation process.

Cloud service implementations are varied because their implementation can use IaaS or PaaS, for example. Therefore FluidCloud addresses:

1. Relocation of IaaS-based Service Instances. FluidCloud will show the relocation of a service instance (and its data) running within virtual machines (on a local development machine, private end or public cloud). Triggers for the relocation can be scaling, costs, dependability or geo-location. The service instance will automatically be adapted to the new environment.
2. Relocation of PaaS-based Service Instances. The relocation of a PaaS based service instance and its data between providers. Key to this demonstration is the ability to adapt the service instance (on source code level) and convert it to the new environment. The motivation should be to bring PaaS services from closed environments (e.g. Google App Engine) into more open ones (e.g. CloudFoundry).
3. Service Instance Adaptation: IaaS to PaaS. A software developer has developed a service on his own virtual machine in a (Private) Cloud now he wants to roll-out this service on an available PaaS provider such as Google App Engine.

4 Implementation

The first proof of concept (PoC) of the logical architecture for IaaS-based relocation has been implemented using the Python programming language. Each of the components are standalone processes which communicate with each other using asynchronous messaging (See Figure 2). The prototype uses the Advanced Message Queuing Protocol (AMQP).

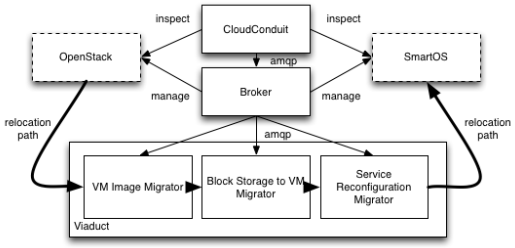

Figure 2: Implemented Architecture

TODO: increase size of fig!

The CloudConduit has capabilities to process requests for relocating service instances. When such a relocation is triggered it inspects the service instances for subcomponents (sub-services) and their dependencies. This is done through the RESTful Cloud APIs supported by both cloud providers, OpenStack[1] as source and Smar-

[1] http://www.openstack.org

tOS[2] as destination, in this case. Based on the inspection it creates a set of tasks which need to be executed. Currently, the tasks are executed in sequential order. Later on the scheduling of these tasks may become more complex.

The Broker now has the information to instantiate the appropriate Migrators that make up the Viaduct. The Migrators take care of the actual relocation and topology change of the service instance.

This implementation has been deployed within one data center across two different platforms. The trigger for the relocation is done based on a performance evaluation of both platforms. A virtual machine on the SmartOS platform has a significantly higher I/O throughput (60.2MB/s on KVM virtual machine under SmartOS and 43.2MB/s on KVM virtual image under OpenStack measured with *dd* on identical hardware).

In this evaluation, a simple node.js service which has been deployed with a virtual machine within OpenStack[3] is to be relocated to SmartOS. This virtual machine has block storage attached to it through an OpenStack cinder volume. The node.js also makes use of OpenStack Swift object storage.

After relocation the virtual machine will be running on the SmartOS platform. The data within the block storage will be relocated, whereas the data in the object storage will stay where it is, indeed the object storage could be hosted elsewhere e.g. Amazon S3. This will demonstrate that the service topology of the service instance can change after the relocation depending on the new destination service provider. This change in topology is done automatically. The service topology before and after relocation is shown in Figure 3:

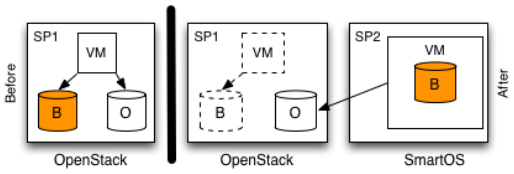

Figure 3: Service Instance Before and after Relocation

The decision for this service topology after relocation is made by the CloudConduit and should be guided by service owner policies. Overall, to relocate this simple node.js application the following Migrators were placed on the Viaduct:

- The virtual machine image Migrator will pause the virtual machine on the OpenStack side and create a new virtual machine on the SmartOS side. Both machines are then booted using a prepared ubuntu iso image[4]. After booting the data is copied over using *netcat* and the *dd* command. After copying the virtual machine on the SmartOS platform can be booted. This is one approach of many.

- The Relocator for the OpenStack block storage copies the data from the block storage device in OpenStack cinder onto the Filesystem of the virtual machine running on KVM in a SmartOS zone. This is done with the help of the *sftp* protocol.

- Reconfiguration of the node.js application's configuration file is based on regular expressions and a simple Python script. It changes the paths for node.js interpreter and the path to the data.

5 Evaluation

The first outcome of this initial PoC was to prove that the architecture satisfies the scenarios described in this paper. The separation between the CloudConduit and the Broker was found useful as several technologies for the Broker exist. Furthermore the Broker can be used to establish Viaduct(s) as this could be made up out of virtual machines, which each could host a Migrator[5]. The overall architecture therefore has been proven to satisfy the needs.

The second outcome was to have initial metrics about the runtime of a relocation process. In the end it will be crucial that the runtime is minimised and possibly reveal if actual online relocation can be achieved. For the relocation of the service instance in the PoC, a total downtime of $\sim$10 minutes was needed. The relocation was performed over a 1 Gb Ethernet network.

This combines the time for stopping the virtual machine and relocating it (seconds for stop, $\sim$5 minutes for relocation of the virtual machine image of 5.4GB), the time to move the data from the block storage to the virtual machine ($\sim$1 minute for 512MB test file within) and finally the reconfiguration ($\sim$10 seconds to copy the script and execute it). Times fluctuated over several runs of the relocation. In general the time-to-relocation using this PoC will depend on the data payload sizes of the virtual machine image and the data in the block storage.

Overall the PoC proved the concepts to be working. In particular the implemented architecture described in the last sections should demonstrate that the concepts of FluidCloud are technically feasible. More over, the important thing is that the concepts noted in this paper describe a way of enabling fluidity of services between clouds.

[2]http://www.smartos.org
[3]http://www.openstack.org

[4]In this case the virtual machine within OpenStack was initially booted from a volume instead of an image.
[5]The migrators were executed in sequential order.

6 Related Work

Standard organisations defined interfaces such as OCCI[6] [8], CIMI[7] or CDMI[8] which may realise interoperability but they do not necessarily solve the issue of relocation. The paper [9] reviews aspects related to portability and interoperability in clouds. It notes the lack of adoption of standards by vendors saying that "vendor[s] like[s] to put barriers to exit for their customers". Related thoughts are discussed in [10]. Here it is noted that cloud systems utilising different hypervisors will not interoperate, in part because they do not use the same data formats.

Adapter libraries enabled the means to manage multiple cloud offerings. The most prominent of these are Apache libcloud, fog.io, RightScale, Enstratus and jClouds.

There are quite a number of Platform as a Service (PaaS) offerings available today. Including Heroku, Red Hat OpenShift, CloudFoundry and Google App Engine. The majority of the PaaS offerings leverage the existing interoperability work that each language (and its standard libraries - e.g. WSGI for Python) and supporting services (e.g. MySQL, RabbitMQ) already have. However, this is not uniform across all PaaS offerings.

The Open Data Center Alliance released a report [11] on long distance service instance relocation. Noted that relocation of workload possible but [...] migrations occur between disparate data centers of the same cloud provider [...] most of the time. So here the most obvious issues becomes clear: relocation is possible but mostly within a service providers domain, and that interdomain (InterCloud) relocation on IaaS and PaaS level needs more research.

Currently available software solutions for data management exist such as: Cloudant, Xeround, MongoLab or Amazon S3. But currently they lack the ability to convert data between the service instances, or relocate data.

The paper [12] looks at InterCloud more from the federation aspect and the authors describe their architectural vision of that InterCloud. One important aspect that the authors do note is the importance of cloud brokers in their architecture [13] as supported by [1].

7 Conclusions and Further Work

The need for service relocation will become ever needed the more cloud services are used and the more service owners move their services to the public cloud. FluidCloud will present a means for this to be supported a prototype framework is available. This framework will be released and supported under an Open Source license. There are further research and engineering challenges to be investigated including live service instance relocation between data centers, data payload minimisation, service decomposition over multiple target service providers and the leveraging of software-defined networking technologies.

TODO: There are a bunch of typos in the references. For instance, "Jr, S. O." should be "Ortiz Jr., S." or just Sixto Ortiz Jr. as some of the other references are spell out that way.

[6] http://www.occi-wg.org

[7] http://dmtf.org/standards/cmwg

[8] http://cdmi.sniacloud.com

References

[1] Grance, P. M. (2011). The NIST Definition of Cloud Computing. NIST special publication.

[2] Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., & Morrow, M. (2009). Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability. Internet and Web Applications and Services, 2009.

[3] D. Bernstein, D. V. (2010). Intercloud Exchanges and Roots Topology and Trust Blueprint.

[4] Y. Demchenko, M. X. (2012). Intercloud Architecture for Interoperability and Integration.

[5] Elson, J. (2008). Handling flash crowds from your garage. USENIX 2008 Annual Technical Conference.

[6] D. Armstrong, D. Espling, J. Tordsson, K. Djemame, and E. Elmroth. Runtime Virtual Machine Recontextualization for Clouds. Euro-Par 2012 Workshops, Lecture Notes of Compouting Science, Vol. 7640, Springer-Verlag, pp. 567 - 576, 2012.

[7] John Bresnahan, Michael Link, Gaurav Khanna, Zulfikar Imani, Rajkumar Kettimuthu and Ian Foster. Globus GridFTP. Proceedings of the First International Conference on Networks for Grid Applications (GridNets 2007), Oct, 2007

[8] A. Edmonds, T. Metsch, A. Papaspyrou, and A. Richardson, Toward an Open Cloud Standard, IEEE Internet Computing, vol. 16, no. 4, Jul. 2012.

[9] Petcu, D. (2011). Portability and interoperability between clouds: challenges and case study. Towards a Service-Based Internet , 62–74.

[10] Jr, S. O. (2011). The problem with cloud-computing standardization. Computer magazine , 13–16.

[11] Alliance, O. D. (2012). Long Distance Workload Migration. ODCA.

[12] C. Ward, N. A. (2010). Workload Migration into Clouds Challenges, Experiences, Opportunities. 2010 IEEE 3rd International Conference on Cloud Computing. IEEE.

[13] Daryl C. Plummer, B. J. (2011). Cloud Services Brokerage Is Dominated by Three Primary Roles. Gartner.