Runtime Virtual Machine Recontextualization for Clouds

Django Armstrong[1], Daniel Espling[2], Johan Tordsson[2], Karim Djemame[1], and Erik Elmroth[2]

[1] University of Leeds, United Kingdom
[2] Umeå University, Sweden

Abstract. We introduce and define the concept of recontextualization for cloud applications by extending contextualization, i.e. the dynamic configuration of virtual machines (VM) upon initialization, with autonomous updates during runtime. Recontextualization allows VM images and instances to be dynamically re-configured without restarts or downtime, and the concept is applicable to all aspects of configuring a VM from virtual hardware to multi-tier software stacks. Moreover, we propose a runtime cloud recontextualization mechanism based on virtual device management that enables recontextualization without the need to customize the guest VM. We illustrate our concept and validate our mechanism via a use case demonstration: the reconfiguration of a cross-cloud migratable monitoring service in a dynamic cloud environment. We discuss the details of the interoperable recontextualization mechanism, its architecture and demonstrate a proof of concept implementation. A performance evaluation illustrates the feasibility of the approach and shows that the recontextualization mechanism performs adequately with an overhead of 18% of the total migration time.

1 Introduction

Infrastructure as a Service (IaaS) clouds are commonly based on virtualized hardware platforms executing and orchestrating self-contained virtual machines (VMs), which are comprised of multiple virtual devices. A cloud application is typically subdivided into individual components, each component bundled into a specific type of VM. Several VM instances can be started using the same type of VM (using the same master disk image) and each new VM instance is uniquely configured, *contextualized*, with instance specific settings at the early stages of execution. The capacity of the cloud application can be adjusted by changing the amount of VM instances. For clarity, our definition of contextualization is as follows:

Definition. *Contextualization* is the autonomous configuration of individual components of an application and supporting software stack during deployment to a specific environment.

I. Caragiannis et al. (Eds.): Euro-Par 2012 Workshops, LNCS 7640, pp. 567–576, 2013.

In this work we introduce the concept of *recontextualization*. Recontextualization can be used to adapt to any system changes, including making newly migrated VMs operate properly in the (potentially different) system environment of a new host. We define recontextualization as follows:

Definition. *Recontextualization* is the autonomous updating of configuration for individual components of an application and supporting software stack during runtime for the purpose of adapting to a new environment.

The life-cycle of a cloud application is comprised of three individual phases as shown in Figure 1. The Construction phase refers to the development of a cloud application making use of platform services and dividing that application into a set of VM images. In the Deployment phase a constructed application is deployed on to suitable infrastructure and finally in the Operation phase the cloud application is executed. The application can be configured in the Construction phase and contextualized with specifics of a provider's environment in the Deployment phase. Recontextualization offers dynamic reconfiguration in the Operation phase.

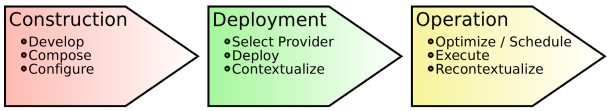

Fig. 1. The life-cycle of a cloud application

Recent work on IaaS systems have a lot in common with the vision of autonomic computing, as outlined by Kephart and Chess [9]. One of the major aspects of autonomic computing that has yet to be realized is self-configuration, the automated configuration and adjustment of systems and components. Our earlier work on contextualization [1] presents a mechanism for boot-time self-configuration of VMs. This work extends state of the art and our earlier efforts by introducing runtime recontextualization, enabling adaptation of VM behavior in response to internal changes in the application to which the VM belongs or to external changes affecting the execution environment of the VM. The concept can enable applications at the PaaS to adapt to different provider application middleware services through the dynamic binding of APIs, enabling the execution of site specific code. This, however, is out of scope in this paper.

The contributions of this paper are: i) The concept and definition of recontextualization ii) The development of an architecture and mechanism for the purpose of recontextualization. iii) A demonstration and evaluation of a recontextualization system. The rest of this paper is organized as follows: Section 2 outlines the problem to be solved, a set of requirements for any approach to recontextualization and an illustrative scenario of recontexualization for application monitoring. Section 3 discusses different approaches considered for runtime

recontextualization. Section 4 presents our proposed solution for recontextualization of VMs including an evaluation of the approach. Finally, a conclusion and future work are presented in Section 5.

2 Problem Statement and Requirements

A motivational factor behind the need for runtime recontextualization stems from VM migration in clouds [3,15]. Using migration, a VM can be transferred from one physical host to another without explicitly shutting down and subsequently restarting the VM [4]. The entire state of the VM, including e.g., memory pages, are transferred to the new host and the VM can resume its execution from its state prior to migration. As a consequence of this, no contextualization is triggered again when the VM is resumed, as the level of abstraction provided by virtualization is insufficient for platform services. In this paper we consider migration from and to identical hypervisor technology, interoperable migration is out of scope but is considered in [12]. As presented in [6], there are several different cloud scenarios:

- Bursting - The partial or full migration of an application to a third party IaaS provider, this may occur when local resources are near exhaustion.
- Federation - The migration of an applications workload between a group of IaaS providers, e.g., when a single provider's resources are insufficient for maintaining the high availability of an application through redundancy.
- Brokering - The migration of an application's VMs, e.g., for the purpose of maintaining an agreed Quality of Service (QoS) in the case of an end-user utilizing a broker to select a provider given a set of selection criteria.

In all these cloud scenarios VM migration is a necessity, e.g., for the purpose of consolidating resources and maintaining levels of QoS. We have used these scenarios to guide us when the defining of requirements for any potential recontextualization mechanism. We consider the following requirements as imperative:

i. A triggering mechanism for recontextualization on VM migration.
ii. A secure process to gather and recreate contextualization data after migration.
iii. A hypervisor agnostic solution that maintains IaaS provider interoperability.
iv. An approach that is none pervasive and minimizes modifications at the IaaS level.

We make a case for each of these scenarios requiring recontextualization at runtime. In the Bursting scenario, if an IaaS provider is not obligated to divulge third party providers used for outsourcing of computational resources, an application may end up deployed on to a third party's infrastructure that requires use of their local infrastructure services. A dynamic federation of IaaS providers created during negotiation time that alters during the operation phase requires infrastructure services to be discovered dynamically. The same is applicable in

the case of a Broker, knowledge of a providers local infrastructure services is not available during deployment until after the Broker has selected a provider.

The lack of knowledge on the attributes of an IaaS provider's local infrastructure service available during deployment time further motivates our work. An example of such a service that exhibits configuration issues after resource migration is application-level monitoring.

In this example the monitoring service endpoint, to which application Key Performance Indicators (KPI) are reported, can be configured by contextualization during the deployment phase of an application's life cycle. However, the endpoint may change during the application lifetime, either as a result of changes in the local system environment or due to migration of the application to a new host. This example motivates the need for a mechanism to fetch configuration data during application operation and provide new context to application dependencies, thus *recontextualization*. In the following section, we illustrate recontextualization with service-level monitoring [7] as an example scenario.

2.1 Example Scenario

A typical cloud application must be continually monitored during runtime, an example of this is shown in Figure 2. Monitoring data can be used for several purposes, e.g., for automatic application scaling or to assess the likelihood and prevent the breaching of a Service Level Agreement (SLA). Application level metrics, know as KPIs, are sent from inside the VM to an external monitoring endpoint for processing.

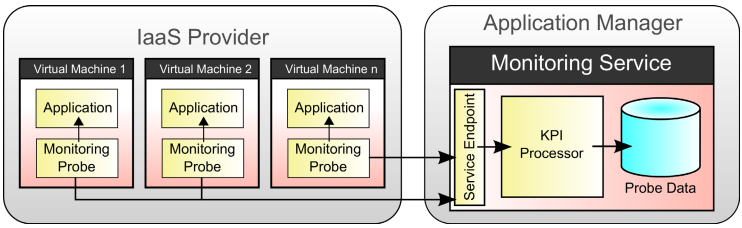

Fig. 2. Monitoring applications in a IaaS provider

Each monitoring probe that gathers KPI data must be configured with the endpoint or location of the monitoring service. The endpoint can be associated with a IaaS specific service or a service running at a remote location and depends on what entity within connected clouds has control over application management. When deploying to a IaaS provider, the endpoint for the monitoring service is configured using contextualization in the Deployment phase. However, in a multi-site scenario the VM maybe migrated to an unknown provider during runtime and must therefore be dynamically recontextualized with a new endpoint in the Operation phase.

3 Recontextualization Approaches

As far as we are aware no previous research has considered an approach for recontextualization. Keahey and Freeman [8] present fundamental work on contextualization in virtual clusters and recontextualization is mentioned but deemed out of scope for their work. In this section several different approaches for contextualization are considered for use in recontextualization. Any recontextualization approach has two major obstacles that must be dealt with; how is recontextualization triggered and where can the necessary information be found? Below are some approaches for recontextualization, listed and discussed, from the perspective of the above two challenges.

Contextualized direct addressing is based on a known endpoint address that is specified in the initial contextualization phase, as described by Armstrong et al. in [1]. A similar approach is used for Puppet [14], a mass-machine configuration tool for HPC-like environments. During operation, this endpoint address is queried for the updated context information. Furthermore, this approach is interoperable and requires no host and hypervisor modifications, but requires that the end point address is constant when a VM is migrated to other domains. This approach offers no procedure for triggering a new round of recontextualization, and has to rely on periodically polling the endpoint for updates.

Hypervisor network proxying also relies on periodically querying an external endpoint address for context information, but in this method a standard virtual network address is used and the hypervisor (and associated virtual network management) is responsible for routing this call to a host specific endpoint. This approach, used by Clayman et al. in [5], is transparent to the VM but requires modifications at the hypervisor level.

Hypervisor interaction by the guest can be used to offer contextualization data straight from the hypervisor itself, using a customized API both to react to changes in context information and to transfer new information. However, this solution requires modifications both to hypervisor and guest operating system software and would require considerable standardization to be widely available, with regards to the compatibility of virtual hardware APIs between hypervisor technologies.

Dynamic virtual device mounting is based on dynamically mounting virtual media containing newly generated content in a running VM via the reuse of existing hypervisor interfaces and procedures [1]. Interoperability is achieved by reusing existing drivers for removable media such as USB disks or CD-ROM drives. Recontextualization can be detected by the guest OS by reacting to events triggered when new USB or CD-ROM media is available.

We propose that the dynamic virtual device mounting approach is the most promising solution to recontextualization due to inherent interoperability and support in all major operating systems. The ability to manage virtual devices is also offered by the Libvirt API [11], inferring that there is fundamental support

for these operations in most major hypervisors. The following section describes our recontextualization solution in more detail.

4 A Recontextualization Solution

In this section, an implementation of a system for runtime recontextualization is described, followed by an evaluation to validate the suggested approach. The previously discussed virtual device mounting technique is used in response to migration events and thus enables automatic self-configuration of newly migrated VMs. The following subsections discuss the mechanism, architecture, and evaluation in more detail.

4.1 Mechanism

Figure 3 illustrates the recontextualization approach used in the implementation. Each VM is assigned a virtual CD-ROM device for contextualization on which the host-specific and thus provider contextualization data can be found. When a VM is migrated from one host to another events describing this action are triggered by the hypervisor, which can be registered to via the Libvirt API. In response to these events, the recontextualizer software triggers a detachment of the virtual device mounted with contextualization information, and once the migration is completed a new virtual device with context information relevant for the new host is automatically attached to the VM as it resumes operation after migration.

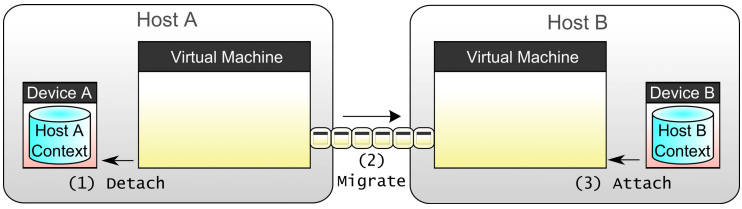

Fig. 3. Recontextualization approach overview

Event support including migrations is present in several hypervisors, including Xen [2] and KVM [10]. The Libvirt API enables a unified approach to VM management available and includes event support. An initial version of the recontextualization system was implemented using KVM with QEMU [13] specific event and control APIs and the second version was implemented using Libvirt to make the solution hypervisor independent. Libvirt provides a number of event types that can be monitored via a callback: i) Started, ii) Suspended, iii) Resumed, iv) Stopped, and v) Shutdown. Upon receiving an event callback details are returned on the specific cause of the event, for example the shutting down of a VM on a host machine triggered by migration terminating successfully.

4.2 Architecture

The architecture of the implemented system is shown in Figure 4. Up-to-date context data is dynamically bundled as ISO images on the host. The recontextualizer, implemented in Python, manages the attachment and detachment of virtual CD-ROM devices inside a VM that contain the data held within the ISO image media in response to events from the hypervisor. The Python Libvirt API bindings were used to access the Libvirtd daemon for the purpose of abstracting the specifics of the underlying hypervisor and to improve interoperability.

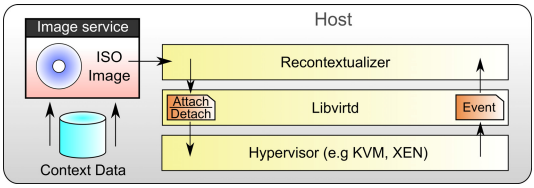

Fig. 4. Architecture overview

4.3 Evaluation

A series of tests to evaluate the feasibility of the approach have been performed. For all tests, Libvirt version 0.9.9 was used to monitor and manage the VMs. QEMU-KVM version 1.0.50 and Xen version 4.0.0 were used as hypervisors, both running on the same hardware using CentOS 5.5 (final) with kernel version 2.6.32.24. The hosts used in these tests are on the same subnet, have shared storage and are comprised of a quad core Intel Xeon X3430 CPU @ 2.40GHz, 4GB DDR3 @ 1333MHz, 1GBit NIC and a 250GB 7200RPM WD RE3 HDD.

The results of the evaluation are shown in Figure 5. The first set of bars illustrate the time to migrate a VM from one host to another with recontextualization running and context data attached, and the second set of columns illustrate the same migrations with recontextualization turned off and no virtual devices mounted. The third column illustrates the time spent within the recontextualizer software during the tests from the first column, measured from when the event for migration was received in the recontextualizer until the device had been removed and reattached. The values shown are the averages from ten runs, and all columns have error bars with the (marginal) standard deviations which are all in the 0.03 to 0.07s range.

Based on the evaluation we conclude that the recontextualization process adds about an 18% overhead using either hypervisor compared to doing normal migrations. For KVM, most of the extra time required for recontextualization is spent outside the bounds of our component, likely associated with processing events and extra overhead imposed by preparing migration with virtual devices attached. In the case of Xen the device management functionality in Libvirt proved unreliable and we therefore had to bypass the Libvirt API and rely on sub-process calls from the recontextualizer to Xen using the *xm* utility. This workaround increased the time needed for recontextualization in the Xen case.

There are four major phases associated with the recontextualization process. First, information about the VM corresponding to the event is resolved using Libvirt when the migration event is received. In the second phase, any current virtual contextualization device is identified and detached. Third, new contextualization information is prepared and bundled into a virtual device (ISO9660) image. Finally, the new virtual device is attached to the VM. A detailed breakdown of the time spent in different phases of recontextualization is presented in Figure 6. The above mentioned workaround for Xen interactions affects the second and fourth phase (detaching and attaching of devices), most likely increasing the time required for processing. In the first and third phases Xen requires significantly longer time than KVM despite the VMs being managed using the same calls in the Libvirt API, indicating performance flaws either in the link between Libvirt and Xen or in the core of Xen itself.

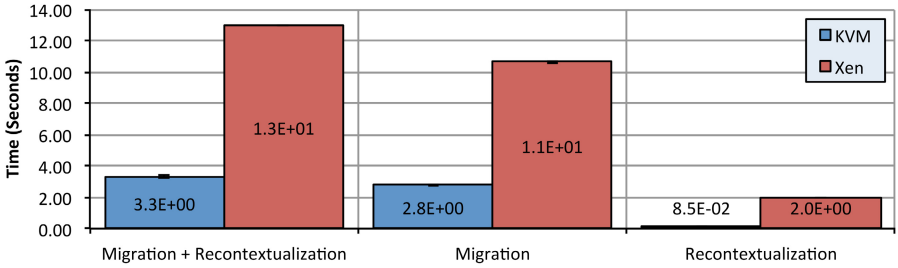

Fig. 5. Time measurements of recontextualization.

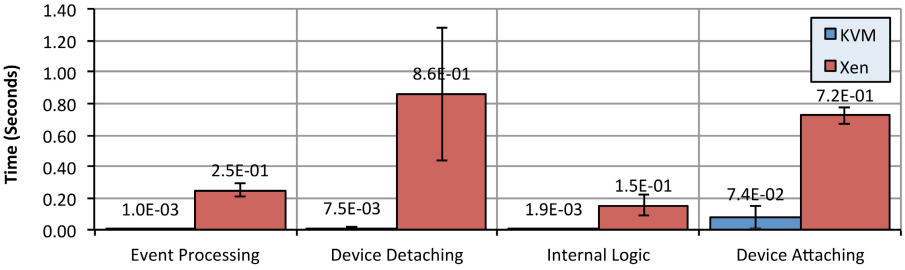

Fig. 6. Breakdown of time spent during recontextualization.

4.4 Practical Experiences

When creating the system a vast number of bugs and shortcomings both with Libvirt and the underlying hypervisors were experienced. It turned out that migrating VMs using KVM with USB devices attached periodically caused the migration to fail without any indicative error of the root cause. It was discovered after looking through the source code of qemu-kvm (necessary due to a lack of

documentation) that support for migration with USB devices is still to be fully implemented. To overcome this issue the use of a virtual CD-ROM device as a replacement was explored. Unfortunately this approach has the drawback of needing the guest OS to be configured to automatically re-mount the ISO image. We used *autofs* within our Debian guest VM for the purpose of testing. For other operating systems such as Windows that natively support the automatic mounting of CD-ROMs this would not be a problem. Using this device type in our system worked with KVM but Xen would not reliably release media mounted within a VM, causing the recontextualizer to fail in its attempt to provide new context data. To combat this issue we forced the removal of the entire CD-ROM device, reattaching another with a different ISO image.

Considering Libvirt's support of Xen events and the detaching of devices we initially tried to use a Hardware-assisted Virtualization (HVM) guest but found that Libvirt would not propagate any VM events from the hypervisor through its API. After discovering this issue we tried using a Paravirtualised (PV) Xen guest but found that only start and stop events were available. This has had the negative effect of altering the logic of the recontextualizer, where by detaching and attaching devices incurred an additional unnecessary overhead when a virtual machine starts, while for KVM this overhead only occurs after migration.

5 Conclusion and Future Work

We have described and defined recontextualization: the autonomous updating of configuration during runtime. Moreover, we have shown that recontextualization is a key enabler to using multiple cloud sites concurrently. We have evaluated different alternatives for recontextualization based on a set of requirements. Our approach, based on automatic mounting of dynamically generated images as virtual devices, is highly interoperable supporting a variety of hypervisors and virtually all operating systems. Apart from CD-ROM mounting routines, which are standard in most operating systems, no custom software is required inside the guest VM to make the contextualization data available.

Future work includes creating a unified mechanism for contextualization and recontextualization and integrating the solution with major software projects. In addition, recontextualization mechanisms for the dynamic binding of PaaS APIs will be explored. Finally, further studies and improvements on the implemented approach will be evaluated to reduce the overhead imposed by recontextualization.

Acknowledgments. The research that led to these results is partially supported by the European Commission's Seventh Framework Programme (FP7/2001-2013) under grant agreement no. 257115 (OPTIMIS). We would also like to thank Tomas Forsman for technical assistance and expertize.

References

1. Armstrong, D., Djemame, K., Nair, S., Tordsson, J., Ziegler, W.: Towards a contextualization solution for cloud platform services. In: 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), pp. 328–331. IEEE (2011)
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. SIGOPS Oper. Syst. Rev. 37(5), 164–177 (2003)
3. Bradford, R., Kotsovinos, E., Feldmann, A., Schiöberg, H.: Live wide-area migration of virtual machines including local persistent state. In: Proceedings of the 3rd International Conference on Virtual Execution Environments, pp. 169–179. ACM (June 2007)
4. Clark, C., Fraser, K., Hand, S., Hansen, J., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, vol. 2, pp. 273–286. USENIX Association (May 2005)
5. Clayman, S., Galis, A., Chapman, C., Toffetti, G., Rodero Merino, L., Vaquero, L., Nagin, K., Rochwerger, B.: Monitoring Service Clouds in the Future Internet. In: Towards the Future Internet - Emerging Trends from European Research, pp. 115–126. IOS Press, Amsterdam (2010)
6. Ferrer, A., Hernández, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R., Djemame, K., Ziegler, W., Dimitrakos, T., Nair, S., Kousiouris, G., Konstanteli, K., Varvarigou, T., Hudzia, B., Kipp, A., Wesner, S., Corrales, M., Forgó, N., Sharif, T., Sheridan, C.: OPTIMIS: a holistic approach to cloud service provisioning. Future Generation Computer Systems (2011)
7. Katsaros, G., Gallizo, G., Kübert, R., Wang, T., Oriol Fito, J., Henriksson, D.: A Multi-level Architecture for Collecting and Managing Monitoring Information in Cloud Environments. In: CLOSER 2011: International Conference on Cloud Computing and Services Science (CLOSER), Noordwijkerhout, The Netherlands (May 2011)
8. Keahey, K., Freeman, T.: Contextualization: Providing One-Click Virtual Clusters. In: Proceedings of the 4th IEEE International Conference on eScience (ESCIENCE 2008), pp. 301–308. IEEE, Washington, DC (2008)
9. Kephart, J., Chess, D.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
10. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux virtual machine monitor. In: Proceedings of the Linux Symposium, vol. 1, pp. 225–230 (2007)
11. Libvirt development team. Libvirt: The virtualization API (February 2012), `http://libvirt.org/`
12. Liu, P., Yang, Z., Song, X., Zhou, Y., Chen, H., Zang, B.: Heterogeneous live migration of virtual machines. In: International Workshop on Virtualization Technology, IWVT 2008 (2008)
13. QEMU development team. QEMU - An open source machine emulator and virtualizer (February 2012), `http://www.qemu.org`
14. Turnbull, J.: Pulling strings with puppet: configuration management made easy. Springer (2008)
15. Wood, T., Ramakrishnan, K.K., Shenoy, P., van der Merwe, J.: CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines. In: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 121–132. ACM (2011)