Automated Instantiation of Heterogeneous FastFlow CPU/GPU Parallel Pattern Applications in Clouds

Boob Suresh

Submitted as part of the requirements for the degree
of MSc in Cloud Computing
at the School of Computing,
National College of Ireland
Dublin, Ireland.

September 2013

Supervisor
Horacio González-Vélez
Alina Mădălina Popescu

Abstract

Parallel scientific workloads typically entail highly-customised software environments, involving complex data structures, specialised systems software and even distinct hardware, where virtualisation is not necessarily supported by third-party providers. Considering the expansion of cloud computing in different domains and the development of different proprietary and open source cloud platforms, users should arguably be able to automatically and seamlessly migrate their parallel workloads across cloud platforms using standardised virtual machines based on elasticity rules. However, even if migrating the workload between the nodes is easier when the nodes have similar configuration on the same platform, the transition between different platforms raises different issues such as vendor lock-in, portability and interoperability.

Moreover, the static distribution of virtual appliances was not proved straightforward because of the time required for user to do all the migrations steps and because of the vendor lock-in issues.

The aim of this paper is to automate the portability of FastFlow—a C/C++ pattern-based programming framework for multi/many-core and distributed platforms—using virtual machines for both CPU and GPU-based environments between heterogeneous virtualised platforms. Our approach relies on the standard Open Virtualization Format (OVF) in order to achieve a universal description of virtual appliances. Such description is not only useful to migrate but also to determine the hardware/system software configuration needed for switching into any new (cloud) image format. We have successfully evaluated our work using virtual machines based on VirtualBox and AWS on local cluster and public cloud providers.

Keywords: Parallel Patterns, FastFlow, automation, portability, interoperability, elasticity, Cloud Computing, Vagrant, Chef.

Acknowledgements

This Thesis ‘Automated Instantiation of Heterogeneous FastFlow CPU/GPU Parallel Pattern Applications in Clouds.’ for Masters in Science degree, was carried out in the Cloud Competency Centre, National College of Ireland, Dublin.

I would like to express my sincere thanks to my supervisors Horacio González-Vélez and Alina Mădălina Popescu. I am grateful to them for there effort, valuable guidance, discussions and for raising up my spirits throughout the work. Every meeting with them broadened my perspective on the issue and added new aspects to the implementation.

I wish to thank with sincere gratitude to Pramod Pathak, Vikas Sahni, Michael Bradford, Robert Duncan and Keith Brittle for their help in successful completion of this course. I fail in my duty, if I dont acknowledge Danu Technologies Ltd., for giving me an oppurtunity to work with them on the real time problems which helped me to perform well in my thesis. I thank all the people, who helped me directly or indirectly in completing my thesis work.

I would like to thank my friends (SAT) for supporting me in through this difficult time. Their help and care helped me a lot to stay concentrated on my studies. I appreciate their help and I am grateful to them for helping me to adjust in a new country.

I express my deep and ever lasting love and gratitude to my parents and family for their continuous support and encouragement. This work is specially dedicated to mama, papa and dadi. To Ishi for cheering me up.

Declaration

I confirm that the work contained in this MSc project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.

Signed .... Date ....
Boob Suresh

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Structured parallel programming facilitates the design of parallel applications, through the composition of pre-defined parallel patterns (algorithmic skeletons) which are accepted as a mechanism for simplifying the code and making it more understandable [26]. In this work, we have used FastFlow, a C/C++ structured parallel programming framework designed to execute fine-grained parallel applications on multi-core CPU, GPU, and loosely-coupled heterogenous distributed systems [2].

Distributed FastFlow applications have been previously tested on virtual machines running on multi-core clusters and public cloud infrastructure [7]. However, although there is no need of additional programming efforts when running the FastFlow applications on different parallel and distributed virtualised platforms, preparing any host environment for FastFlow application execution currently requires manual intervention. That is to say, there exists the need to automate the instantiation/termination of FastFlow-based virtual machines based *on demand*, in order to optimally scale pattern-based applications by migrating the entire FastFlow framework and its underlying systems software from local clusters to cloud platforms.

Furthermore, complex scientific applications typically face the time-consuming manual staging and configuration of systems software in every new instance in new hardware infrastructures. Such configuration assumes substantial expertise of end-users and the application tuning and optimisation can produce minor system errors. In fact, the automatic instantiation of parallel scientific applications to comply with different cloud-specific APIs has been regarded as an open problem for public cloud infrastructures [11]. This is because as we know each CSP has its own Application Programming Interface (API) that allows customers to deploy and manage the cloud resources. However, users who are accessing multiple CSPs will face the big challenge of interacting in a multi-cloud environment due to the incompatibility between those APIs. In order, to use different cloud platforms the business logic of the propreitery APIs have to be reimplemented. In this context, open-source projects (i.e. Apache LibCloud, DeltaCloud) and proprietary APIs (i.e. RightScale, Enomaly) were implemented for managing the cloud services across multi-clouds [16].

Thus, Cloud Computing is not still mature, although numerous efforts were realized around this topic. Those efforts are reflected by the number of Cloud Service Providers (CSP) and open-source cloud platforms, by the appearance of several management interfaces for cloud services (i.e. open-source and proprietary), by the research work in terms of advantages and challenges raised by this technology and by the emerging of the cloud standardization bodies, non-profit groups and member-operated organizations.

In order to take the advantage of the characteristics of cloud computing, organizations have to seriously consider the various issues of this new computing environment for achieving highest financial profits. The major characteristic of cloud computing is shared resources, thus resource allocation plays a critical role for optimal utilization. Many challenges are being raised in the real time provisioning of resources to meet user's requirement at run time. For better performance of the cloud computing systems, the resources will be provisioned to the users efficiently and effectively as requested. Efficient resource provisioning is to allocate the resources to the users as they demand at anytime and anywhere without disruption.

The resources are provided to the users by means of a standard compute instance known as virtual machine. In order to provision the resources efficiently, different vendors have to follow the standards by solving the vendor lock-in problem, which it is the major issue halting the users from adopting the cloud technology [25].

In this sense and for providing on-demand infrastructure, Data Management Task Force (DMTF) supports the Open Virtualization Format (OVF), which was extended as well for cloud based systems in the new version (i.e. OVF 2.0). Nevertheless, the OVF seems that it had not gain enough adoption in the last years among cloud providers. Moreover, Amazon provides the AWS EC2's VM Import/Export Tool for importing and exporting instances from Amazon EC2 to VMware, but for the moment, the conversion into OVF format implies only the Windows VMs. Furthermore, OVF format standard is considered to be used in our solution with the purpose of achieving interoperability across clouds and to exploit the full potential of cloud computing [16].

Thence, in terms of the issues that cloud computing community embraces [24], this paper mainly focuses on the interoperability and portability challenges of complex scientific environments—in this case, a parallel pattern-based one—in public cloud infrastructures. By automating the installation and configuration of the FastFlow framework, it deals with the integration of OpenCL code into FastFlow [14], allowing the seamless porting and execution of FastFlow applications in distributed (cloud) CPU/GPU environments.

This thesis addresses the following contributions:

1. The proposal of the heterogeneous migration of FastFlow-based virtual machines across local resources and public cloud, considering the heterogeneity of the hypervisors, the heterogeneity of the instances (i.e. CPU-based and GPU-based) and the heterogeneity of the architectures (i.e. 32 bits and 64 bits) with different flavours of Linux.

2. The implementation and evaluation of the solution

1.1 Thesis Structure

The structure of the thesis will be as follows:

Chapter 1: Provides the goal and motivation of this dissertation and discusses about the issues that cloud computing is facing today

Chapter 2: Critically analyses the research work on the selected issues

Chapter 3: Gives a detailed explanation of the proposed automation tool

Chapter 4: Explains the environmental set up with different software installation and the tool development

Chapter 5: Discusses about the analysis of the test run of the developed tool. The test is carried with different applications.

Chapter 6: Draws the final conclusions about the thesis work.

Chapter 2

Literature Review

This chapter will focus on critically analysing the resource allocation methods which resolves the issues for the cloud service providers and runs the business smoothly by increasing the cloud adoption rate. This discussion is organized as follows, the issues related to interoperability and portability is described in the subsection 2.1. Different parameters to be considered in allocation of resources is described in the subsection 2.2. The methods followed to allocate the resources in different clouds and solutions are described in the subsection 2.3. The organizational bodies working for standardization of cloud computing are subjected in the subsection 2.4. The issues and solutions are analysed and the research gap is provided in the subsection 2.5.

2.1 Interoperability and Portability issues

Cloud computing is a paradigm in which compute, storage and network are provided to the users on demand. It enables hardware and software services to public and business markets as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS) [4]. Instead of purchasing actual physical resources, users lease these resources from a cloud provider as an outsourced service. As cloud computing is based on novel sharing infrastructure, maintaining a huge pool of resources plays a critical role in effective utilization of resources [6]. Cloud computing services have improved the efficiency of resource discovery, resource matching, task scheduling and execution. However, provisioning of resources to all the tasks in cloud computing is the key concern [43]. As the performance of the system depends on how efficiently the resources are being used. If the resources are provisioned and are in idle state for a long time it degrades the performance of the system. A detailed, well-written and rigorous account of research by Nathani et. al.[33] introduced the key things to be handled to optimize resource utilization. The key things are ”Where to place newly created virtual machine? When to dispatch newly created virtual machine to a particular physical machine?” [33] as resources are provisioned to the users in the form of creating virtual machines, the problem arises when the resources are not available to accept the request of the user that is resource scarcity. Ye Hu et. al.[19] argue that the point of assigning a newly created virtual machine to a physical machine completely depends on the scheduler used to provision resources . Several attempts have been made for effective provisioning of resources to meet the given quality of service and to provision the needs as for the peak demands. The framework of Vazquez et. al. [46] provides the areas where meeting a Service Level Agreement (SLA) and satisfying the quality of service (QoS) when peak demands arises, which is challenging at certain times. This explanation has a few weaknesses that other researcher have pointed out clearly. The applications demand resources as in need, which cannot be predicted. This raises the issue of having idle resources or lack of resources. In the case of lack of resources, the resources should be provided from other virtualized platforms with same environment.

Sotomayor et. al.[42] related their research on provisioning of resources from the local IT infrastructure that is a private cloud with the requests for more resources. This local provisioning has very limited resources and cannot handle the requests more than its capability; therefore the suggestion made by the author is to build a Hybrid cloud, which allows to provision resources from the public cloud. In this way the resources can be provided. The issue with hybrid cloud is the compatibility and interoperability of the resources between the infrastructure of private and public cloud. But, the resource contention that provides interoperability among infrastructures in the hybrid cloud failed. The cooperation strategy of dynamic resource allocation by Dai et. al.[10] in their research overcomes the resource contention problem. The beginning of this article provides an informative overview on effectively utilizing the resources by using parallel processing technique in cloud computing like MapReduce. However, this parallel processing technique has no consideration of the status information of the local resources, which leads to diminish the effectiveness of resource provisioning that is resource fragmentation. The users always raises a question, to adopt cloud computing ”Whether the cloud workload can be migrated to other service provider without modifying the system?” Dowell et. al. clearly presents the portability of the system by differentiating the cloud artifacts. Portability is to move the image in the power off state from one system to the other system and then to boot it in the destination machine. In order to advance the cloud interoperability to mature state, the virtual machine images and cloud application interfaces should be standardized [12].

Cloud Service Providers (CSPs) typically have their own Application Programming Interfaces (APIs) that allow customers to deploy and manage their cloud resources. However, users accessing multiple CSPs face the challenge of adapting their applications to a multiplicity of cloud environments with mostly incompatible APIs. Besides this drive, the multi-cloud migration is necessary for backup purposes when a CSP becomes unavailable at a certain point [36]. Moreover, the financial argument is another motivation for approaching the portability challenge as it enables seamless switching between CSPs when economic factors change [16, 17].

Specifically, APIs are built for specific cloud infrastructure, which leads to a problem of vendor lock-in. Furthermore, cloud-based templates and application deployment compound the challenge cloud platform. Ergo, true cloud interoperability and vendor lock-in are currently perceived as a drawback of adopting the cloud services [25].

It has become clear that there exists a emerging need to increase standardisation in the field to allow the elastic execution of applications [31]. Hence, in order to automatically increase or decrease the resources on different cloud platforms depending on the load of the system, several researchers have provided solutions based on distinct brokering systems using federated clouds. Such brokering systems mediate the services between the cloud consumer and cloud provider. The federated cloud for scalable provisioning of resources and services by Buyya et. al. [5], delivers an architecture for federated cloud environments. This architecture defines different components called cloud coordinator services, cloud exchange, and client brokering. The consumer states the cloud broker all the services needed and all the services of the cloud coordinator are published in the federated environment.

All the service providers and clients are contracted by means of cloud exchange. The different communications between the components follows the SLA message schema. Similarly, the Contrail project uses a Virtual Execution Platform (VEP) [18] to provides the virtual distribution of the resources and deploy the users applications independently. But, each CSP should have VEP running on their IaaS layer to federate with the CONTRAIL project and automate the deployment, requiring a significant level of installation and configuration to interact with different nodes.

The EU-funded SLA@SOI [44] provides a framework where different Cloud providers can form their SLAs with the consumers based on their needs with the support of service providers. In this approach, Marco et. al. [8] keep resource utilisation logs between the consumer, service provider and the cloud infrastructure provider to enable metering. Similarly, the EU RESERVOIR [35] project has worked on building a brokerage system for the cloud providers, focusing on the Small and Medium Enterprises for their migration into Cloud systems. Its CLAUDIA toolkit [36] provides a medium to deploy services in different cloud platforms through a plug-in mechanism for federated clouds. This approach maximises the benefits by optimising the services between the federated Clouds. CSPs use the RESERVOIR sites and make the contracts with the federated cloud providers. By combining the benefits of the SLA@SOI and RESERVOIR project, Metsch et. al. [27] have delivered a framework with OCCI. This framework provides the interface for interoperating between the clouds. The consumer needs the services, which can be deployed on the RESERVOIR [35] sites, which can be managed using SLA@SOI framework [44].

2.2 Parameterized Resource allocation

A study of the research of Sindelar et. al.[41] determines the initial mapping of resources to the virtual machines into two levels, that is at the cluster and the node. This research has structured the provisioning of the huge pool of resources to the requests of the users. This study can be used in division of the virtual machine types. Mishra et. al. [30] pointed out the problem with traditional bin-packing which many researchers used to determine the virtual machine types by summarizing different parameters of the virtual machines like CPU usage, memory usage and network usage. This division of virtual machine types can be much useful for mapping the virtual machines without over-provisioning and under-provisioning problem. The idea of dividing the virtual machine types fails when, there is a request from the user for a virtual machine which is not categorized in the virtual machine type. So, along with the virtual machine parameter summarization, the attributes of the users, providers are to be considered to categorize the virtual machine types.

The above mentioned virtual machine placement method work well only in the specific conditions. In order, to efficiently provision the resources and improve the performance of the cloud computing system the decision of selecting the most optimal method is most important. As discussed, each method requires different parameters to process and make the optimal utilization. So, the performance of the method depends on whether the parameters are specified properly or not. Nevertheless, to efficiently provision the resources some parameters has to be considered that are related to resource allocation in cloud computing. The table 2.1 shows the different parameters and their dependent variables examined during the allocation of resources with advantages and disadvantages.

Parameter	Dependent	Advantages and Disadvantages
Market based	Resource utilization	-No over provisioning of resources -Cost consideration is beneficial
Auction based	Pricing schema	-Provides efficient cost factor -Sudden changes in the cost may effect
SLA based	Contract between stakeholders	-Known services and downtime -Efficient services with promised resources
Scheduler based	Resource schedule algorithm	-Depends on resource availability
Rules based	Task and Constraints	-Based on the priority of the tasks -Few tasks may get into deadlock state
Scalable computing based	Reserved schema	-May get over provisioned -Guaranteed resources provisioned
Congestion control based	Compression technique	-Unpredictive change in the requirement may effect

Table 2.1: Resource allocation parameters

2.2.1 Market based resource allocation

Resource allocation based on market is proposed by Salehi, et. al. [37]. To advance the resource utilization from large pool of resources of the cloud providers with high degree of Quality of Service to cloud users different schedulers are used. According to different constraints of the user, the design and allocation strategy are reconstructed. The proposed method allocates resources to different users according to their resource necessities. This efficiently improves the provisioning as the consumption is advanced. This improves the profits of users and the provider.

2.2.2 Resource allocation based on auctions

For eficient resource provisioning in the cloud environment several auction based techniques have been proposed. Li, et. al. [23] proposed a second-level auction method with the assistance of pricing and truth telling methods. In the research, a negotiation-based approach for heterogeneous computing like cloud was proposed, it presented a protocol using market based system in between the resource manager and scheduler. A knowledge based continuous auction trade model was proposed in [39] which achieved higher efficiency and provided stability in pricing transaction. Though these auction based model has improved the efficiency these are based on single-item.

2.2.3 Resource allocation based on Service Level Agreements (SLA)

Service Level Agreements are used to deliver services to the consumers. As allocating resources in a scalable on demand approach is an indispensable characteristic. In order to avoid the penalties, SLA has to be maintained and the resources are reserved to the users according to the SLA [48]. Prodan, et. al. [34] has proposed 3 types of resource scheduling layers in cloud:

1. Infrastructure as a Service (Iaas)
2. Platform as a Service (PaaS)
3. Software as a Service (SaaS)

Efficiently provisioning and scheduling resources at these layers is significant at different constraints and necessities of the users. The resources are provisioned to the virtual machines based on the SLA phrases. With this method, the resource provisioning is optimized and the potential of SLA violation is condensed. As the SLA satisfies it clearly points that the request rejection rates for the resource provisioning is low.

2.2.4 Resource allocation based on schedulers

Different schedulers are used to provision resources to the users like immediate, best effort, advance reservation [33]. An optimized scheduling algorithm is selected based on the research on Infrastructure as a service cloud systems. Using these schedulers, the virtual machines (VMs) are incorporated with the resources as from their availability. A scheme of Dividend policy [38] is used to select a finest allocation strategy for the users demand. As the scheduler allocates the resources efficiently to the customers demand, optimal utilization of the resources increases.

2.2.5 Resource allocation based on Rules

Rules allow different components to safely access the resources. In cloud computing, services are provided to the users based on customer's constraints [33]. So, efficient allocation of the requested resources by the customers is a challenge. For this rule based resource allocation, resources are allocated based on the task priority. This system follows queuing policy, where the criticality of the requests is calculated and resources are allocated [22]. This system overcomes the problem of under utilization and over utilization of the resources which enables the efficient utilisation of provisioned resources.

2.2.6 Resource allocation by scalable computing

The cloud IaaS service, of providing the users with the resources by leasing over the internet. The user selects and utilises these resources based on their necessity. But, the real time tasks which has to be completed before the deadline approaches. At a particular point of time the users may require additional resources, which are provisioned as required and charged for the usage period that is scalable computing [34]. The users request for the additional resources and is fixed only for the rental period. Special schedulers are used to allocate resources for real time tasks and reserved schema of resources has to be maintained for such requests by the providers to decrease the rejection rates of the requests.

2.2.7 Resource allocation for congestion control

To efficiently provision the resources, congestion has to be controlled. As multiple resources are provisioned to the users as a service by the providers, congestion might occur. Jianfeng, et. al. [49] described a congestion control technique which decreases the size of the resources in the situation of congestion. This technique can be vastly used to optimize the utilization of resources as the size of resources are reduced the idle part of the resources can be used for request of other users. This congestion control methods can be much optimised to maintain the Quality of service [45]. Congestion control mainly deals with the efficiency of the resource provisioning.

2.3 Resource Allocation System (RAS)

In general, Cloud allocates resources to the users' requests according to the defined resource allocation policies. Most of the cloud providers like Amazon, Eucalyptus, etc rely on simple resource allocation policies like immediate and best effort. In immediate resource allocation policy provider allocates resources as for user's request and rejects the request if the resources are not available that leads to high request rejection rate. In best effort, the requests are placed in the FIFO queue when resources are not available. Nathani et. al.[33] proposed a detailed well-written and rigorous account of resource lease manager Haizea. It is used as a scheduler for OpenNebula. The Haizea provides advance reservation of capacity by using the methods like swapping and backfilling. It is used to address the issue of resource scarcity by providing the resources for requested duration at required start time. Though Haizea provides resources, it is noted that the users has to reserve the resources in advance for duration. This results in effective provisioning of resources to users when needed. Nevertheless, the resource interoperability carries out when resources of different infrastructure are used.

The interoperability of the resources from different infrastructure providers can be solved from the research of Vazquez et. al. [46] which provided different adapters to act as interface. This adapter enables the access to different infrastructure resources. The component monitors the resources availability and when the load of the system exceeds a defined threshold, it grows the resources by contracting with different cloud providers using specific adapters. The rejection rates of the system are decreased as the required resources are provisioned from the other cloud providers, which satisfies the Quality of Service (QoS), and to qualify the Service Level Agreement (SLA). However, the users might be over-provisioned. Keeping track of the resources provisioned accurately is the key concern. Therefore, the suggestion made by the author is difficult to support

Sotomayor et. al. [42] discusses this over-provisioning of the resources. The researchers have set up a monitoring manager called Virtual Infrastructure manager (VI). The VI management is an automation tool that manages the dynamic allocation of resources from the pool of available resources. The VI management uses Haizea [33] as a scheduling backend tool, which has different schedulers to provision the resources like Immediate, Best effort, Advance Reservation. If the users reserve the resources in advance and when a new user demands for the resource the status of the reserved resource should be monitored. By using the methods of Nathani et. al.[33] like backfilling and swapping the status of the reserved resources can be tracked. For effective and efficient process, Parallel processing technique is used. The dynamic resource allocation using parallel processing which distributes the tasks overall the computing nodes and balances the capabilities of the computing nodes. Sotomayor et. al. briefly described the advantage of parallel processing. In parallel processing, no resources are left idle as the tasks are distributed among all the computing nodes [42]. This processing technique efficiently utilizes the resources and releases all the resources as the tasks are completed which makes the request rejection rate minimum by provisioning the resources to the requests of the new user new user but the issue is all the resources are in processing state though there is a chance of running the process on few resources. Consolidation of resources is required for better performance with effective resource provisioning.

In [47] a new reverse auction based allocation mechanism has been proposed named Reverse batch Matching Auction for effective resource allocation in cloud computing systems. This mechanism mainly follows batch matching for better performance and efficiency is improved by twice punishment method.

In order to provision the resources intelligent capacity planning system has to be implemented. This system should have the capability to estimate the computing instance, operating system and bandwidth with different pricing models according to the users utilization[40]. If the capacity planning system works well, most of the issues with the resource provisioning can be solved with an improvement in the resource utilization.

Though these systems provides the efficiency in allocating the resources many other issues arises with this. In order to overcome the issues, different organizational bodies are working to build a common standard to all the CSPs to make the work easier. The following section provides the information of different bodies that are working for standardization.

2.4 Cloud Computing Standards

As the growth in the Cloud computing technology increased, the issues related to cloud are getting much concerned as discussed. Many groups and organizations are working to solve these issues by standardizing the cloud industry. Some of the active groups working for cloud standardization are Data Management Task Force (DMTF), Open Grid Forum and Storage Networking Industry Association. The Open cloud standardizing working groups for the cloud computing in the market are presented in this section.

2.4.1 Data Management Task Force CIMI and OVF

Developed by the Distributed Management Task Force, the Cloud Infrastructure Management Interface (CIMI) provides full support to the Open Virtualization Format (OVF) with different RESTful APIs for JSON and XML formats. The objective of OVF is to provide the description for the virtual appliances in an XML format and vendor-free standard. It provides all the features of the virtual machines like hardware description, software description, network and start up order. The OVF 2.0 also provides the package encryption capabilities. Notwithstanding, the OVF has not gained enough adoption in the last years among the cloud providers [9].

2.4.2 Open Grid Forum WS-Agreement and OCCI

Web Services-Agreement provides a standard for Service Level agreement between the providers and the users. XML format is used for template and agreement specification.

WS-Agreement has major 3 parts, first describes the agreement, second the schema of the agreement description and third life cycle management operation description[3].

The Open Cloud Computing Interface (OCCI) [13] initiative endeavours to circumvent the vendor locks-in problem in the area of cloud computing by engaging with a wide range of industrial stakeholders such as mOSAIC [32], SLA@SOI [44], OpenNebula [29], and OpenStack [20]. The OCCI aims to standardise the RESTful APIs for task management. The standardisation was started by OCCI-working group at the beginning for the IaaS clouds and now the standards are extensible with the SaaS and PasS services. However, OCCI fails to exhibit common platform for cloud vendor APIs to define virtual machine and their operations [28].

2.4.3 Storage Network Industry Association CDMI

To perform the operations and to retrieve the cloud data,Cloud Data Management Interface(CDMI) has developed a RESTful interface for the users and applications. This standard interface enables the administrators to handle the metadata, and managing the credentials of cloud user accounts[21].

2.5 Research Gap

The aforementioned approaches do not necessarily take into account the structure of a given application when elastically expanding or contracting resources in a widely distributed environment such as public or hybrid clouds.

Pattern-based parallel programs are expressed by interweaving parameterised algorithmic skeletons analogously to the way in which sequential structured programs are constructed, fostering portability by focusing on the description of the algorithmic structure rather than on its detailed implementation [15]. This provides a clear and consistent behaviour across platforms, with the underlying structure depending on the particular implementation.

However, scant research has been devoted to take advantage of the computational nature of structured parallelism applications in cloud environments, and more importantly on the automatic deployment of pattern-based programming frameworks on clouds. From this perspective, the distributed version of FastFlow [1] can enable users to build a common API in order to enable pattern-based parallel applications to interact with different cloud platforms by using the standards such as OVF without depending on any brokering or third party system.

In this project, an automation tool for heterogeneous pattern-based applications with different hardware architecture requirements (CPU or GPU). Such tool has got no prerequisites (installation, configuration, or other) to burst into a public cloud with no federation rules. The feature of the proposed automation tool provides enhanced capability than the VEP [18], which configure the set up of the application deployment in the VEP tool interface. This requires changes in the VEP tool, when new dependency or configurations are added to the application during the new release. Though, VEP provides partial deployment of the application, while the proposed automation tool provides the full deployment of the application using the Chef recipes and the standardized data export format Open Virtualization Format (OVF), presented in Section. Hence, the technique of programming the VEP tool by remote login into the system for application deployment is not flexible for automation purpose. In this sense, there exist automation tool used for automating the deployment of application framework in the systems(i.e. Chef, Puppet Labs and CFEngine), where Chef is an open source automation tool and Puppet Labs and CFEngine are the commercial tools which also provide this automation techniques.

Chapter 3

Specification

This chapter provides the design goal of developing a tool to overcome the issue of vendor lock-in and migrate the virtual machines between the clouds from local resources to the other cloud platforms automatically. This automation tool follows the parameters like market based and pricing schema to migrate the virtual machines.

The proposed methodology is to automate the migration of virtual machines running FastFlow applications from the local resources to the public cloud platforms. The portability of FastFlow applications will be realized using the Automation Tool, which configures both the hardware and software requirements of the virtual machine on different IaaS platforms. While the Open Virtualization Format (OVF) standard is used to configure the hardware, Chef is used to automatically install and configure the FastFlow framework into the new virtual machine.

Thus, based on OVF, this work is proposing to create a specific deployment template based on the IaaS provider by deconstructing the complete description regarding the instance running in a standardized form. This deployment template is forwarded to instantiate the pre-configured FastFlow-based virtual machines in the specific IaaS platform. Moreover, the automated installation of FastFlow applications into the deployment template will be realized through the usage of Chef, where FastFlow cookbook will be created.

The proposed Automation Tool consists of two modules (Fig. 3.1): Automation Module and Monitoring Module.

Vagrant is used to automate the instance creation on the local virtualised environment and it is configured to use the Chef Server, which provides the cookbook for the FastFlow installation during the instance creation. Once, the automation tool is started

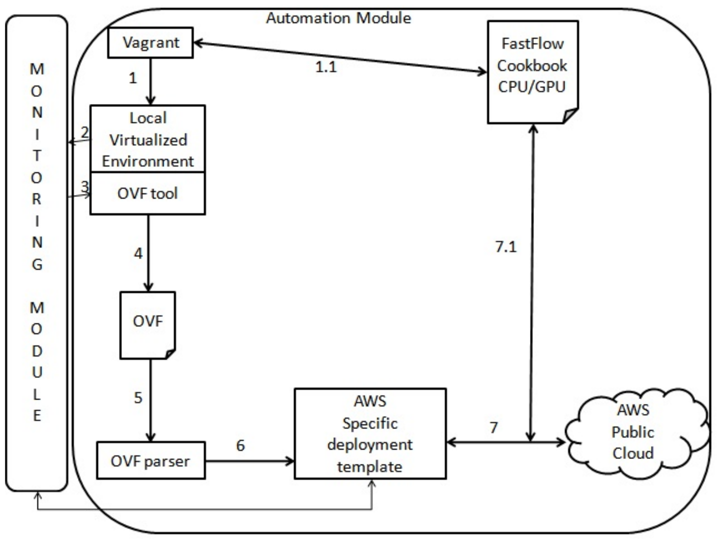


Figure 3.1: FastFlow Automation Tool.

with the number of virtual machines required with the FastFlow platform and applications deployed. It calls the vagrant and automatically creates the instance in the local virtualized environment and installs all the dependencies and the applications using the FastFlow cookbook deployed in the Chef Server.

The monitoring module (Fig. 3.2) continuously monitors the utilisation of the local resources and in case of more resources required, the automation tool automatically bursts into the public clouds to get more resources. The monitoring module is used to optimise the resource utilisation, the time for migration and the cost reduction. This module maintains the complete log of the instance related to the specific project and provides the efficiency in resource utilization.

These instances are notified to the monitoring module. The monitoring module decides whether to burst to public clouds or not. For the case when the decision is to burst the resources across public clouds, a cloud specific deployment template should be created. This is because the public cloud platforms has specific requirement to start instances. In order to get this specific requirement, a OVF file will be generated using the OVF tool. This OVF file is obtained from the running virtual machine in the local virtualized environment and it is deconstructed based on the nodes like operating systems, memory, hard disk, RAM size and product name. These deconstructed nodes are used to create a cloud specific deployment template. From the product section of the OVF, the name of the platform and the application running on it is gathered and those specific cookbooks from the chef server are called. Using the cookbooks, all the dependencies are installed into the newly created virtual machines of the public cloud platforms.

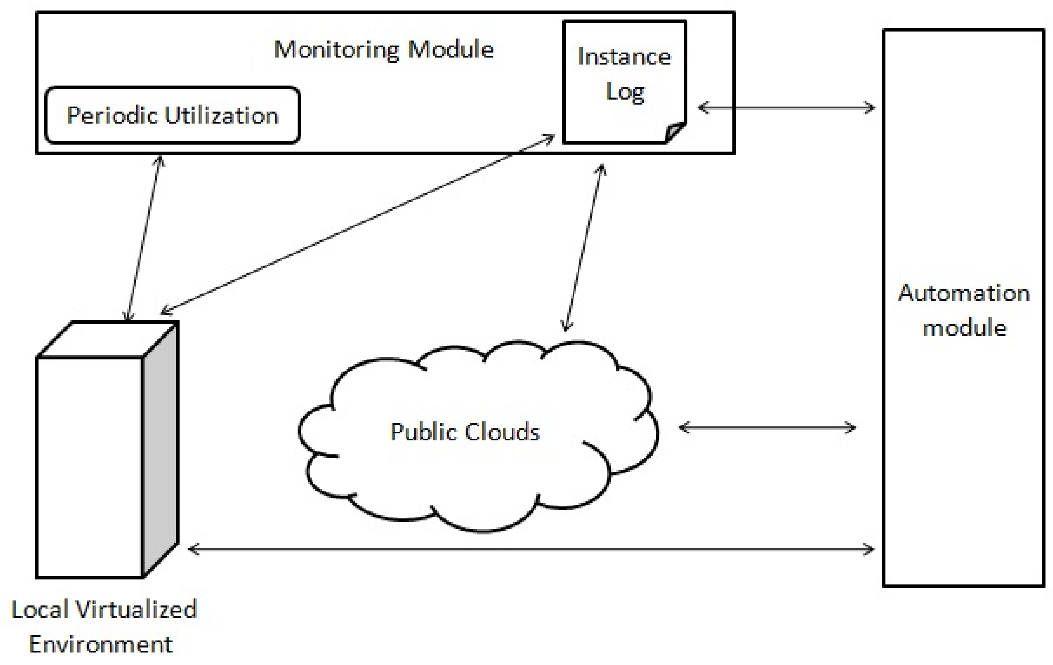


Figure 3.2: Monitoring Module.

The monitoring module logs all the instances information like the type of the instance, operating systems, status of the instance and software installed. While bursting into the public cloud platform, this module checks whether any instance with same configuration is in stopped state. If it finds such instance, it starts that instance instead of creating the new virtual machine and installing all the dependencies.

As a proof of concept, the monitoring tool decides whether to use CPU based instance or GPU based instance based on the resource utilization of the local environment. Depending on this decision, the appropriate FastFlow CPU or GPU cookbook recipes from the Chef-server are used to deploy the application.

As the local resources are free, the monitoring module stops the instances in the public cloud and restarts the instances in the local virtualized environment. This saves the cost of running instances in the public cloud.

Chapter 4

Implementation

This chapter provides the implementation and the set up of the environment for migrating the virtual machines between different cloud providers. The implementation of the automation tool with the code snippets are presented in subsection 4.1 The set up of Chef-workstation is detailed in subsection 4.2 and the vagrant for automation in the local environment is explained in subsection 4.3.

4.1 Automation Tool

The goal of automatically bursting from local environment to cloud is achieved by developing the automation tool on Java platform; OVF tool version 1.0 is used to generate the OVF file; the virtualized environment is built using the Oracle VirtualBox version 2.2 and Chef. The migration of workload is based on the CPU utilization of the local system. The Cloud automation tool continuously monitors the CPU utilization and checks whether the average utilization of the CPU exceeds the maximum threshold value, like it is presented in the code below:

```
class CpuCalculation {
  static String cmdTop = "uptime";
    private StreamWrapper getAvgCpuUtilization(InputStream is, String type){
    return new StreamWrapper(is, type);
  }
  private cpuMonitor(cpuUtilizationValue){
  }
}
```

If the CPU utilization exceeds the average threshold, then the OVF is generated for one of the instance in the local system using OVF tool. The complete configuration of the instance including the application running is parsed from the OVF file. Then, the cloud specific template is created in order to instantiate an instance in the cloud with same configuration and all the dependencies required to start the application will be installled into the new instance using the Chef cookbook that was created for FastFlow applications (i.e. CPU or GPU based).

```
private File createOvf(Instance){
  return new runVbox(command);
  }
private getConfigOfInstance(File){
    }
```

The cloud specific template creates an instance in the public cloud with the same configuration and using the Chef cookbook recipes created for the FastFlow platform and applications. The FastFlow CPU or GPU based framework is installed and configured to run the applications in the newly created instance in the cloud.

```
private Reservation CreateInstance(configuration){
  return new runInstance();
  }
private installApplication(configuration){
  node aws = instance.getInstancePublicDnsName();
  runKnifeFastflow(aws);
    }
```

Finally, the instance automatically starts running the application that was recently running in the local virtualized system. The instances created in the public cloud are stopped as soon as the local resources get free and the utilization falls to minimum threshold value. This process runs in the background and continuously monitors the utilization when the instances in the local virtualized system or in the cloud are running.

To automatically deploy and configure the application on the virtual machines, Chef platform has been built on the Amazon public cloud.

4.2 Chef Platform

Chef is an automation tool, which provides a means to transform infrastructure as a code. The definitions and dependencies are defined in the cookbooks and recipes used in the chef environment. The workstation and nodes use the knife and chef-client API's to talk with the chef-server (Fig. 4.1).

The different parts in Chef architecture are:

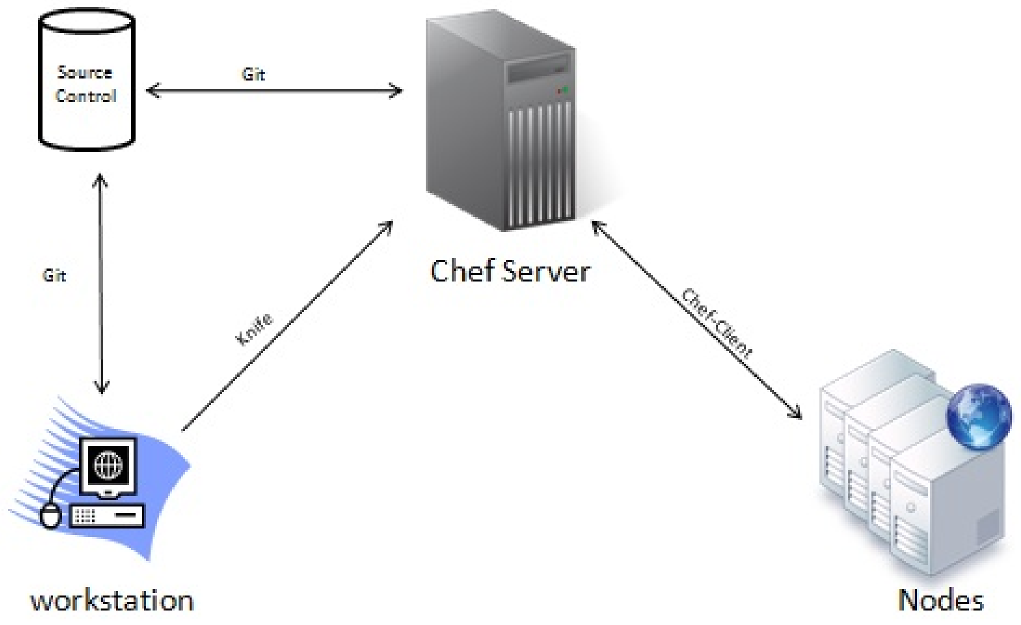


Figure 4.1: Chef Architecture.

- **Chef-Server:** The chef-server is the centralized store of infrastructure configuration. The FastFlow CPU or GPU based cookbooks, policies, data bags etc are placed in the chef-server. It handles all the nodes registered. The figures in the appendix A are the screen shots of the Chef-Server deployed in the Amazon Public Cloud.

- **Chef-Nodes:** The systems managed and configured by chef-server. Chef-client runs on the node retrieving configuration information from the chef-server. It installs all the dependencies and configures the system to run FastFlow applications.

- **Chef-Workstation:** The system with the local chef-repository and knife configured to talk with the chef-server.

A private chef-server is running in Amazon public cloud. A FastFlow cookbook for CPU and GPU based application was developed on this chef-server with all the dependencies and configuration recipes required to install on the Linux machines (Fig. A.5). The chef-client API talks to the chef-server for the specific recipe and runs the recipe on the node.

4.2.1 Setting Chef Environment

On Linux Machines:

- Installing Virtual Box and Vagrant

```
$ sudo apt-get install vagrant
```

  This will install VirtualBox and Vagrant on your system

- Installing omnibus installer, This will install Chef and Ruby on your system

```
$ curl -L https://www.opscode.com/chef/install.sh | sudo bash
```

On Windows System

- Download and install Virtual Box and Vagrant

```
VirtualBox: https://www.virtualbox.org/wiki/Downloads
Vagrant: http://downloads.vagrantup.com/
```

- Download and run the git installer with all the default options

```
Git Installer: http://git-scm.com/
```

- Download and run the Windows Chef installer with all the default options

```
Chef installer: http://opscode.com/chef/install.msi
```

  P.S: Run the Chef commands in the git bash only for windows system

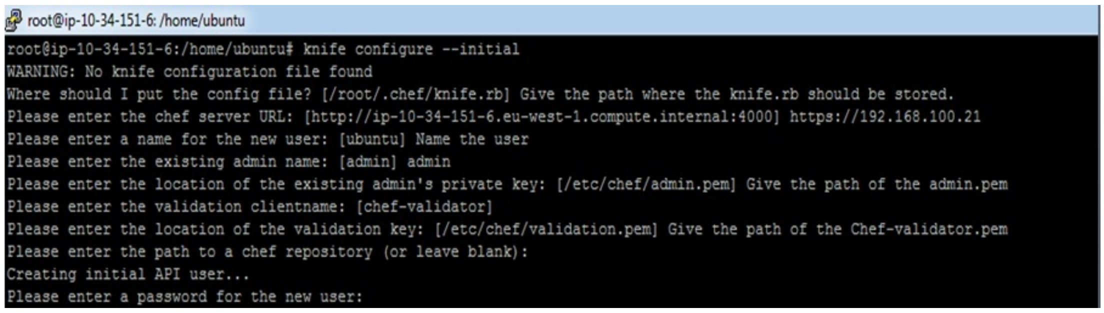
- Before, configuring the Chef-Workstation get the Chef-Validator.pem and client.pem keys from the Chef-Server. This keys are used to register the user with the chef-server and validate the authorized use of cookbooks

- Once the required keys are collected, place it in a particular directory or folder and run the command to configure the workstation.

```
$ knife configure --initial
```

  This command will prompt for the path to store the knife.rb, the address of the chef-server (https://54.217.223.76) and the paths of chef-validator.pem and client.pem.,place the appropriate path and set the password. The figure 4.2 below gives the description

- The above command will create a knife.rb file and the user.pem with all the configuration set



```
root@ip-10-34-151-6: /home/ubuntu
root@ip-10-34-151-6:/home/ubuntu# knife configure --initial
WARNING: No knife configuration file found
Where should I put the config file? [/root/.chef/knife.rb] Give the path where the knife.rb should be stored.
Please enter the chef server URL: [http://ip-10-34-151-6.eu-west-1.compute.internal:4000] https://192.168.100.21
Please enter a name for the new user: [ubuntu] Name the user
Please enter the existing admin name: [admin] admin
Please enter the location of the existing admin's private key: [/etc/chef/admin.pem] Give the path of the admin.pem
Please enter the validation clientname: [chef-validator]
Please enter the location of the validation key: [/etc/chef/validation.pem] Give the path of the Chef-validator.pem
Please enter the path to a chef repository (or leave blank):
Creating initial API user...
Please enter a password for the new user:
```

Figure 4.2: Initializing the knife configuration.

- Run the command to test the chef-workstation setup

```
knife client list
```

  This command will list out all the clients in the chef-server.

An example of knife.rb file generated as part of this implementation is

```
log_level              :info
log_location           STDOUT
node_name              'admin'
client_key             '/root/.chef/admin.pem'
validation_client_name 'chef-validator'
validation_key         '/root/.chef/chef-validator.pem'
chef_server_url        'https://54.217.223.76'
syntax_check_cache_path '/root/.chef/syntax_check_cache'
```

The recipes of FastFlow applications for CPU and GPU environments gather all the dependencies like: C compiler, ZeroMQ version 2.2.0 and configures the paths to compile the applications. Where as for FastFlow application with OpenCl code, the NVIDIA drivers are required to be downloaded and installed. The GPU based recipe of FastFlow handles all the required dependencies and installs the applications to keep them running on GPU clusters. The decision of running a CPU or GPU based instance with pre-configured FastFlow framework is based on resource utilization. Thus, if the utilization is greater than 60%, then a GPU based instance is created in the AWS public cloud and the GPU recipes of FastFlow are used to deploy and configure the new instance. Otherwise, a CPU based instance will be created. In Fig. 4.3 the structure of Chef-Cookbook for the FastFlow platform is presented.

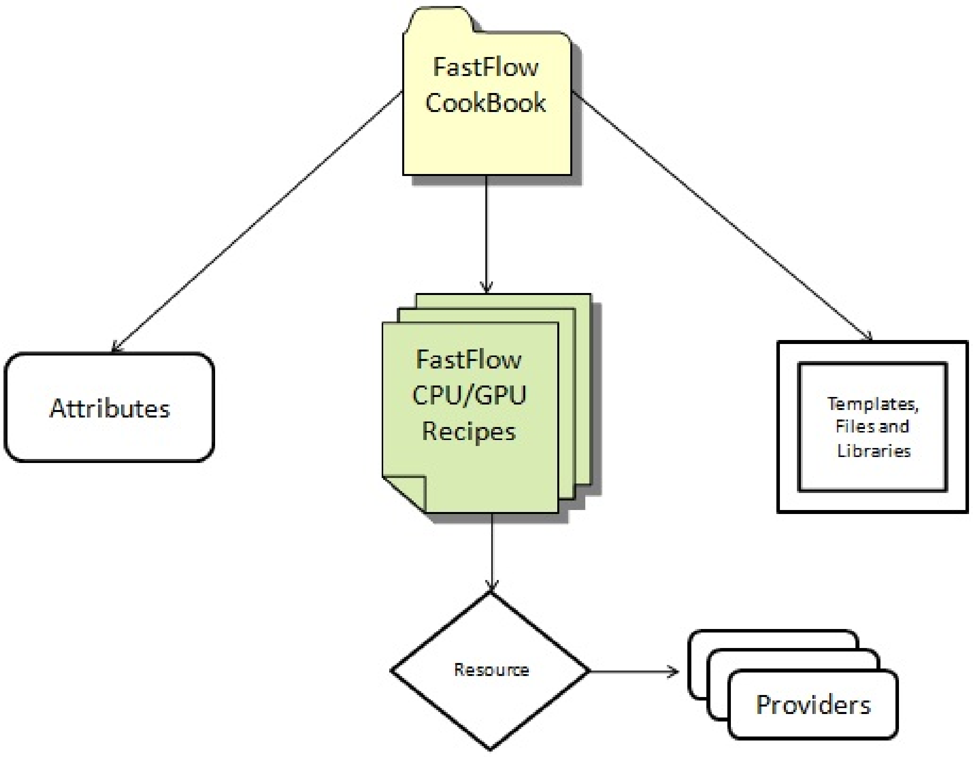


Figure 4.3: FastFlow CookBook Structure.

```
ark node['application']['fastflow'] do
  version "2.0.0"
  url node['fastflow']['git']
  checksum template['key']['valueff']
  action :put
end

ark node['zeromq']['fastflow'] do
  version "2.0.0"
  url node['zeromq']['git']
  checksum template['key']['valuezmq']
  action :put
end
%w[
    make && make install
  ].each do |pkg|
  package pkg
end
```

This automation tool is designed to work with different open source and commercial IaaS cloud platforms. In the initial effort, the support to AWS cloud is provided. Later on, the work for Microsoft Azure, Eucalyptus, Open Nebula etc., will be developed.

The pre-requisites to use this automation tool are Vagrant, VirtualBox version 2.2.0, AWS public cloud credentials and access to the Chef-server.

4.3 Vagrant

Vagrant is an open source automation tool used to build the development environment by maximizing the flexibility and productivity of the work environments. By using Vagrant, creation of virtual machines on top of VirtualBox can be automated. Configuring the Vagrant with Chef the required framework can be installed automatically in the local virtualized environment built with VirtualBox.

The Vagrant file is scripted to use the Chef cookbooks in order to create a virtual machine on top of the VirtualBox with the FastFlow framework.

Setting Vagrant

- Create a new directory and run

```
$  vagrant init
```

  This command will create a Vagrantfile in the newly created directory.

- Open the Vagrantfile and copy this setting in the Vagrant file in order to use the Chef Server and to install the FastFlow framework

```
config.vm.provision :chef_client do |chef|
  chef.chef_server_url = "IP address of the Chef Server"
  chef.validation_key_path = "Path of the Validation key"
  chef.node_name = "Name of the Node "
  chef.provisioning_path = "Path where to provision"
  chef.add_role("Role Name")
end
```

- Now, run the following command

```
$ vagrant up
```

  For the first time it will take few minutes in order to download the ubuntu box (Operating System).Now, the instance is ready with the application installed with all the dependencies configured.

- SSH into the instance

```
$ vagrant ssh
```

The vagrantfile in the automation tool for the FastFlow framework with Ubuntu vagrant box is

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
time = Time.new
name = time.min
Vagrant::Config.run do |config|
config.vm.define :name do |config|
config.vm.box = "opscode-ubuntu-12.04-chef11"
config.vm.box_url = "https://opscode-vm.s3.amazonaws.com/vagrant/opscode_ubuntu ↩
    -12.04_chef-11.2.0.box"
config.vm.network :bridged
config.vm.provision :chef_client do |chef|

  chef.chef_server_url = "https://54.217.223.76"
  chef.validation_key_path = "/root/.chef/chef-validator.pem"
  chef.node_name = "FastFlow-#{name}"
  chef.provisioning_path = "/etc/chef/"
  chef.add_role("FastFlow") //For CPU based virtual Machines
end
end
end
```

Chapter 5

Evaluation

The main aim of this automation tool is to reduce the time of migration from local virtualized resources to the cloud environment with minimal manual efforts. It also provides solution for vendor lock-in problem as this is a cloud-agnostic and hypervisor-agnostic solution. In general, to migrate an application from one platform to another cloud platform requires large number of manual steps to be followed. These steps may cause minor errors while installing the application. In case of installing the FastFlow framework within the local virtualized environment by a FastFlow experienced user, it takes 30 minutes (approx) for creating the virtual machine, booting up the operating system, installing and configuring the application.

In case of bursting into the public clouds, it will take longer because of the multitude of steps required: getting access to the public cloud, selecting the image, creating the security group, assigning the protocol rules, launching the instance, together with the installation steps of the FastFlow application within that specific instance. Furthermore, each public cloud has their own API for creating the instance and connecting to it, and a new cloud user has to learn how to use the cloud specific API to get access to a particular cloud. Hence, automating all this process of moving the FastFlow application from local virtualized environment into the cloud with the application ready to use, it is useful in the real-time research projects by reducing the time of setting up the systems.

Moreover, if a new version of FastFlow is released, it is a hectic process to reinstall the new version by installing all the dependencies required. The proposed solution provides the easy way of updating the application with all the required dependencies installed with no steps or process.

The below figures (i.e. Fig. 5.1 and 5.2) present: the timeline to build the environment with FastFlow platform installed manually by the experienced user on the local virtualized environment with Ubuntu 12.04 basic ISO image readily available and the timeline for migrating to the AWS public cloud for more resources.

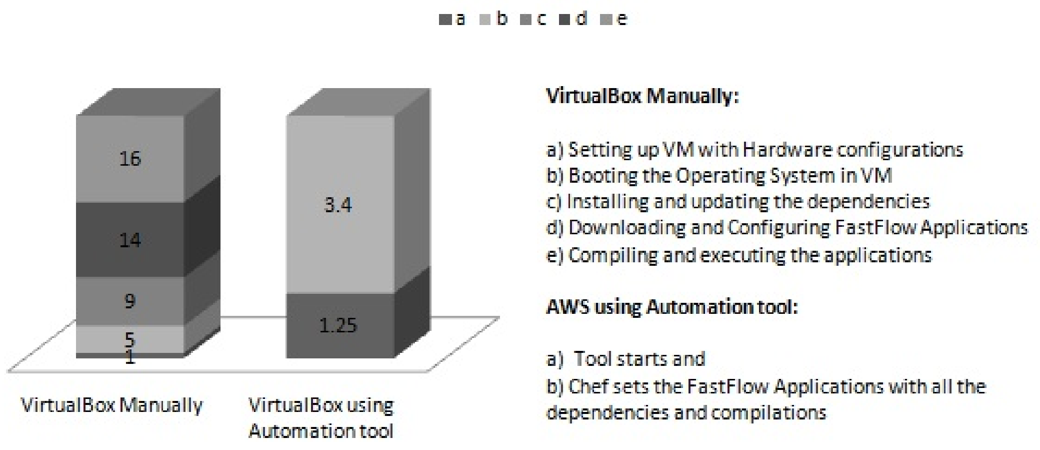


Figure 5.1: Time (min) for FastFlow CPU based virtual machine in VirtualBox.

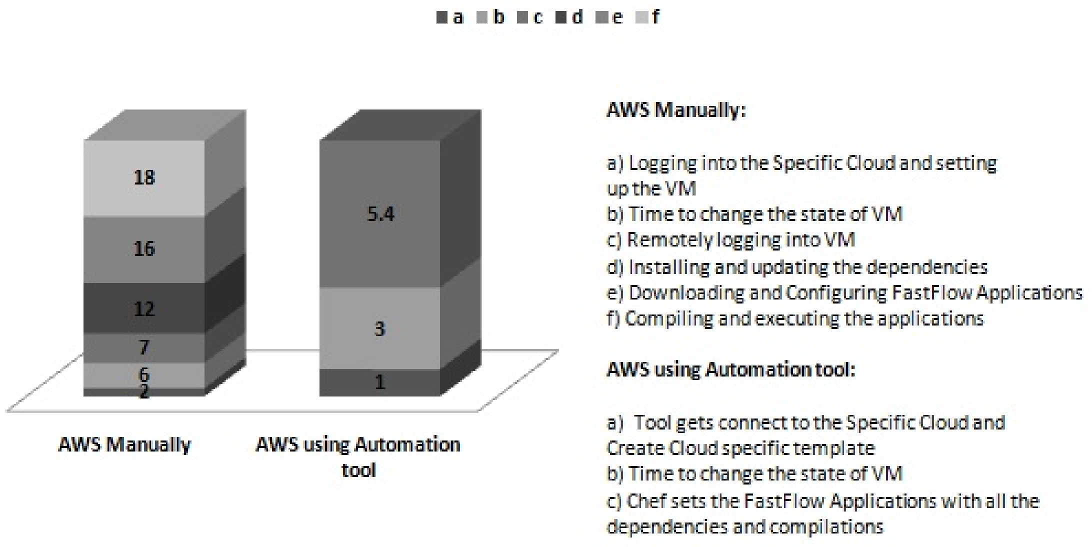


Figure 5.2: Time (min) for FastFlow CPU based virtual machine in AWS.

The timelines with the automation tool on the same platforms are discussed in Fig. 5.1 and 5.2.

From the above figures (i.e. Fig. 5.1, 5.2) it can be noticed that the average time for manually setting up the environment is much greater than using the proposed automation tool. This automation tool also monitors the local virtualized environment and self triggers to migrate to the cloud depending on the threshold values. It is also used to reduce the cost of using the instances in the public cloud when the local resources are freely available to use.

Furthermore, we used a simple benchmark application running on CPU-based virtual machines in VirtualBox and AWS using the proposed automation tool. The application that we selected for our tests is a 3-stage FastFlow distributed pipeline benchmark. The first stage called Unpacker is generating a stream of data. Further, the second stage is based on the task-farm pattern and it is internally parallel implemented. In this way, each worker thread is computing a numerical function (F) on the input data. The last stage called the Packer is collecting the results received from the middle stage(s). This application was executed with all stages mapped in a single virtual machine (i.e. first, the virtual machine created by the automation tool in VirtualBox and then the virtual machine created by the automation tool in AWS) and we consider 2 middle stages for the second stage of the application with 2 internal parallelism degree (i.e. 2 workers). The results of running the application are presented below in Fig. 5.3, where the length of the stream is varying from 128 to 1024.

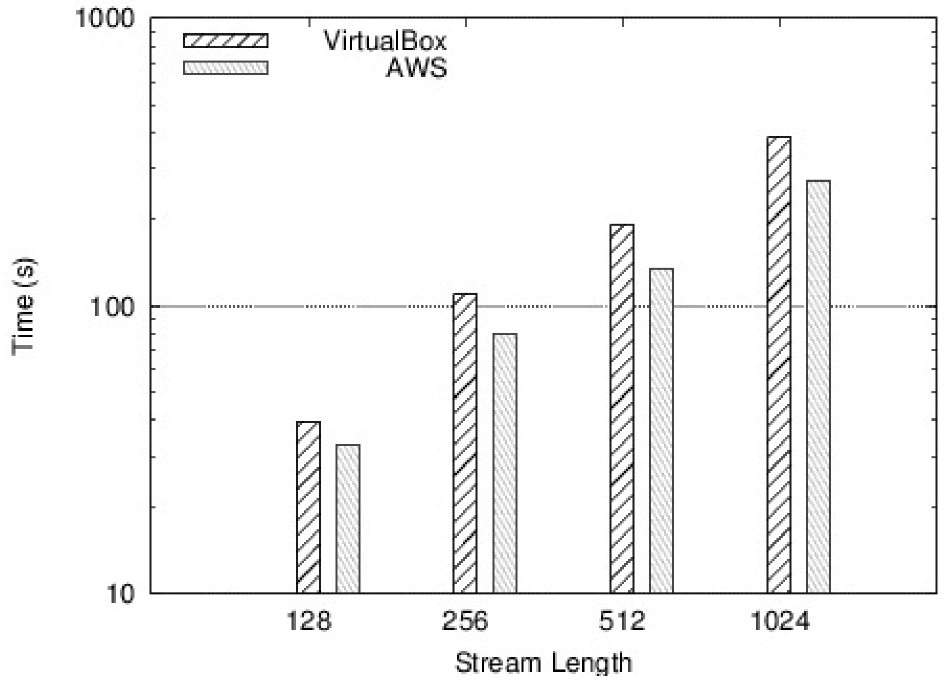
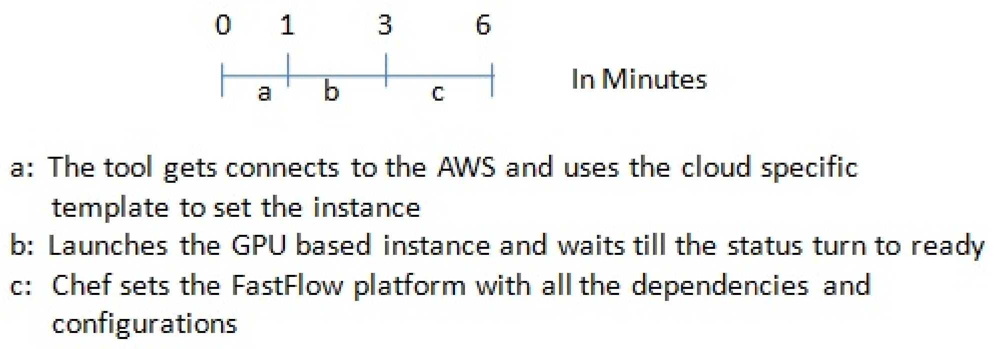


Figure 5.3: Time (s) for running the benchmark application in a CPU based virtual machine in VirtualBox and AWS with Automation Tool.

We have run for the GPU based virtual machine (Fig. 5.4 and 5.5) the same kind of experiments like the one we reported in Fig. 5.3 (i.e. the execution time for the benchmark application within the AWS CPU-based instance), and in this case we run the same benchmark application but in a mixed CPU/GPU system.



a: The tool gets connects to the AWS and uses the cloud specific template to set the instance
b: Launches the GPU based instance and waits till the status turn to ready
c: Chef sets the FastFlow platform with all the dependencies and configurations

Figure 5.4: Timeline for FastFlow GPU based virtual machine in AWS with Automation Tool.

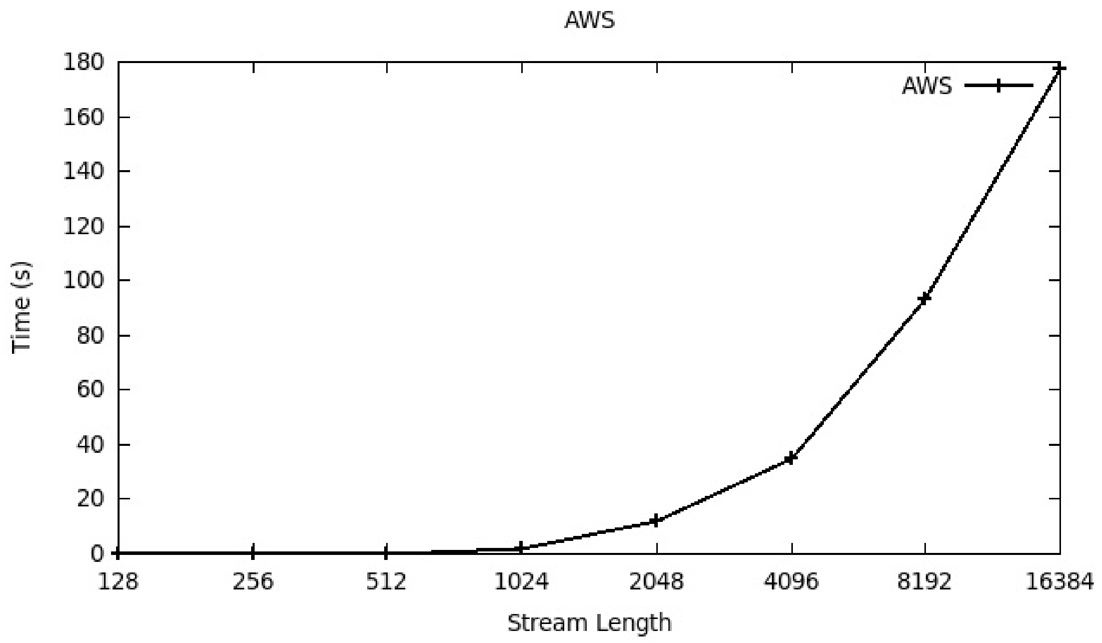


Figure 5.5: Time (s) for running the benchmark application in a GPU based virtual machine in AWS with Automation Tool.

Chapter 6

Conclusion

This thesis proposed a cloud automation tool for automatically deploying the FastFlow framework for both CPU or GPU based virtual machines, which are automatically migrated from local environment to cloud platform. This automation tool continuously monitors the resource utilization and based on the threshold it realizes the migration of the customized virtual machines between local and public cloud environment.

The evaluation chapter described different experiments, that have been tested using the proposed automation tool on both the CPU and GPU based instances. In this sense, different applications of FastFlow platform were successfully executed in the instances created using the Automation Tool.

Thus, the proposed Automation Tool has the following advantages: it solves the vendor lock-in issues and the problems faced by the federation clouds, it does not need pre-requisites to use it and it is a very straightforward and quick tool, which registered less time than the usual manual migration and customization process of virtual machines.

Though, this Automation Tool was tested for FastFlow applications, different applications other than FastFlow could be automatically installed and configured, if the required Chef recipes for them will be developed. This is possible because the Automation Tool follows all the standardized tools and formats.

The initial development of this automation tool focused on migration from the local virtualized environment to the Amazon public cloud. The further work with other different public clouds like Microsoft Azure will be supported with the cost dependent parameter providing a Graphical User Interface. Another future task will be to automate the tasks allocation of FastFlow applications across heterogeneous cloud environments. Furthermore, different parameters as discussed in this paper will also be considered in the future work to allocate the resources for the virtual machines automatically.

Bibliography

[1] Marco Aldinucci, Sonia Campa, Marco Danelutto, Peter Kilpatrick, and Massimo Torquati. Targeting distributed systems in FastFlow. In *Euro-Par 2012 Workshops*, volume 7640 of *LNCS*, pages 47–56, Rhodes Islands, August 2012. Springer.

[2] Marco Aldinucci, Marco Danelutto, Peter Kilpatrick, Massimiliano Meneghin, and Massimo Torquati. Accelerating code on multi-cores with FastFlow. In *Euro-Par*, volume 6853 of *LNCS*, pages 170–181, Bordeaux, August 2011. Springer.

[3] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web services agreement specification (ws-agreement). In *Global Grid Forum*, volume 2, 2004.

[4] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[5] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Algorithms and architectures for parallel processing*, pages 13–31. Springer, 2010.

[6] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[7] Sonia Campa, Marco Danelutto, Horacio González-Vélez, Alina Mădălina Popescu, and Massimo Torquati. Towards the deployment of fastflow on distributed virtual architectures. In *Proc. of ECMS 2013*, pages 518–524, 2013.

[8] Marco Comuzzi, Constantinos Kotsokalis, Christoph Rathfelder, Wolfgang Theilmann, Ulrich Winkler, and Gabriele Zacco. A framework for multi-level sla management. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pages 187–196. Springer, 2010.

[9] Simon Crosby, Ron Doyle, Mike Gering, Michael Gionfriddo, Steffen Grarup, Steve Hand, Mark Hapner, Daniel Hiltgen, Michael Johanssen, Lawrence J Lamers, et al. Open virtualization format specification (ovf). Technical report, Technical Report DSP0243, Distributed Management Task Force, Inc, 2009.

[10] Jiangpeng Dai, Bin Hu, Lipeng Zhu, Haiyun Han, and Jianbo Liu. Research on dynamic resource allocation with cooperation strategy in cloud computing. In *System Science, Engineering Design and Manufacturing Informatization (ICSEM), 2012 3rd International Conference on*, volume 1, pages 193–196. IEEE, 2012.

[11] M.D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali. Cloud computing: Distributed internet computing for IT and scientific research. *IEEE Internet Computing*, 13(5):10–13, 2009.

[12] Scott Dowell, Albert Barreto, James Bret Michael, and Man-Tak Shing. Cloud to cloud interoperability. In *System of Systems Engineering (SoSE), 2011 6th International Conference on*, pages 258–263. IEEE, 2011.

[13] Andy Edmonds, Thijs Metsch, Alexander Papaspyrou, and Alexis Richardson. Toward an open cloud standard. *IEEE Internet Computing*, 16(4):15–25, 2012.

[14] Mehdi Goli and Horacio González-Vélez. Heterogeneous algorithmic skeletons for FastFlow with seamless coordination over hybrid architectures. In *PDP 2013*, pages 148–156, Belfast, February 2013. IEEE Computer Society.

[15] Horacio González-Vélez and Mario Leyton. A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers. *Software: Practice and Experience*, 40(12):1135–1160, 2010.

[16] Greek Interoperability Center. Deliverable d6.1 interoperability guide. Technical Report 6, 2012.

[17] Nikolay Grozev and Rajkumar Buyya. Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, pages 1–22, 2012.

[18] Piyush Harsh, Yvon Jegou, Roberto G Cascella, and Christine Morin. Contrail virtual execution platform challenges in being part of a cloud federation. In *Towards a Service-Based Internet*, pages 50–61. Springer, 2011.

[19] Ye Hu, Johnny Wong, Gabriel Iszlai, and Marin Litoiu. Resource provisioning for cloud computing. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, pages 101–111. ACM, 2009.

[20] Kevin Jackson. *OpenStack Cloud Computing Cookbook.* Packt Publishing, 2012.

[21] V Kowalski, H Shah, J Crandall, M Waschke, N Joy, S Neely, J Wheeler, S Pardikar, et al. Open virtualization format specification. *vol. DSP0243*, 2.

[22] Karthik Kumar, Jing Feng, Yamini Nimmagadda, and Yung-Hsiang Lu. Resource allocation for real-time tasks using cloud computing. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, pages 1–7. IEEE, 2011.

[23] Wei-Yu Lin, Guan-Yu Lin, and Hung-Yu Wei. Dynamic auction mechanism for cloud resource allocation. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 591–592. IEEE, 2010.

[24] Alina Mădălina Lonea, Daniela Elena Popescu, and Octavian Prostean. The overall process taken by enterprises to manage the iaas cloud services. In *Proc. of ECIME 2012*, pages 168–177. University College Cork, 2012.

[25] N. Loutas, E. Kamateri, F. Bosi, and K. Tarabanis. Cloud computing interoperability: The state of play. In *2011 IEEE CloudCom*, pages 752–757, Athens, November 2011. IEEE Computer Society.

[26] Michael McCool, Arch D. Robison, and James Reinders. *Structured Parallel Programming: Patterns for Efficient Computation.* Morgan Kaufmann, Boston, 2012.

[27] Thijs Metsch, Andy Edmonds, and Victor Bayon. Using cloud standards for interoperability of cloud frameworks. *A technical RESERVOIR report*, 2010.

[28] Thijs Metsch, Andy Edmonds, R Nyrén, and A Papaspyrou. Open cloud computing interface–core. In *Open Grid Forum, OCCI-WG, Specification Document. Available at: http://www.ogf.org/documents/GFD.183.pdf*, 2010.

[29] Dejan S. Milojicic, Ignacio Martín Llorente, and Rubén S. Montero. Opennebula: A cloud management tool. *IEEE Internet Computing*, 15(2):11–14, 2011.

[30] Mayank Mishra and Anirudha Sahoo. On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 275–282. IEEE, 2011.

[31] R. Moreno-Vozmediano, R.S. Montero, and I.M. Llorente. Key challenges in cloud computing: Enabling the future internet of services. *Internet Computing, IEEE*, 17(4):18–25, 2013.

[32] mOSAIC FP7 project. Mosaic open source api and platform for multiple clouds. http://www.mosaic-cloud.eu/.

[33] Amit Nathani, Sanjay Chaudhary, and Gaurav Somani. Policy based resource allocation in iaas cloud. *Future Generation Computer Systems*, 28(1):94–103, 2012.

[34] Radu Prodan and Simon Ostermann. A survey and taxonomy of infrastructure as a service and web hosting cloud providers. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 17–25. IEEE, 2009.

[35] RESERVOIR FP7 project. Resources and services virtualization without barriers. http://www.reservoir-fp7.eu/.

[36] Luis Rodero-Merino, Luis M. Vaquero, Victor Gil, Fermín Galán, Javier Fontán, Rubén S. Montero, and Ignacio M. Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226 – 1240, 2010.

[37] Mohsen Amini Salehi and Rajkumar Buyya. Adapting market-oriented scheduling policies for cloud computing. In *Algorithms and Architectures for Parallel Processing*, pages 351–362. Springer, 2010.

[38] Tino Schlegel, Ryszard Kowalczyk, and Quoc Bao Vo. Decentralized co-allocation of interrelated resources in dynamic environments. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 2, pages 104–108. IEEE, 2008.

[39] Shifeng Shang, Jinlei Jiang, Yongwei Wu, Zhenchun Huang, Guangwen Yang, and Weimin Zheng. Dabgpm: A double auction bayesian game-based pricing model in cloud market. In *Network and Parallel Computing*, pages 155–164. Springer, 2010.

[40] Shifeng Shang, Yongwei Wu, Jinlei Jiang, and W Zheng. An intelligent capacity planning model for cloud market. *Journal of Internet Services and Information Security*, 1(1):37–45, 2011.

[41] Michael Sindelar, Ramesh K Sitaraman, and Prashant Shenoy. Sharing-aware algorithms for virtual machine colocation. In *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures*, pages 367–378. ACM, 2011.

[42] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *Internet Computing, IEEE*, 13(5):14–22, 2009.

[43] Jianzhe Tai, Juemin Zhang, Jun Li, Waleed Meleis, and Ningfang Mi. Ara: Adaptive resource allocation for cloud computing environments under bursty workloads. In *Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International*, pages 1–8. IEEE, 2011.

[44] Andy Edmonds Thijs Metsch and Victor Bayon. Using cloud standards for interoperability of cloud frameworks. Technical report, 2010.

[45] Takuro Tomita and Shin-ichi Kuribayashi. Congestion control method with fair resource allocation for cloud computing environments. In *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on*, pages 1–6. IEEE, 2011.

[46] Constantino Vázquez, Eduardo Huedo, Rubén S Montero, and Ignacio M Llorente. On the use of clouds for grid resource provisioning. *Future Generation Computer Systems*, 27(5):600–605, 2011.

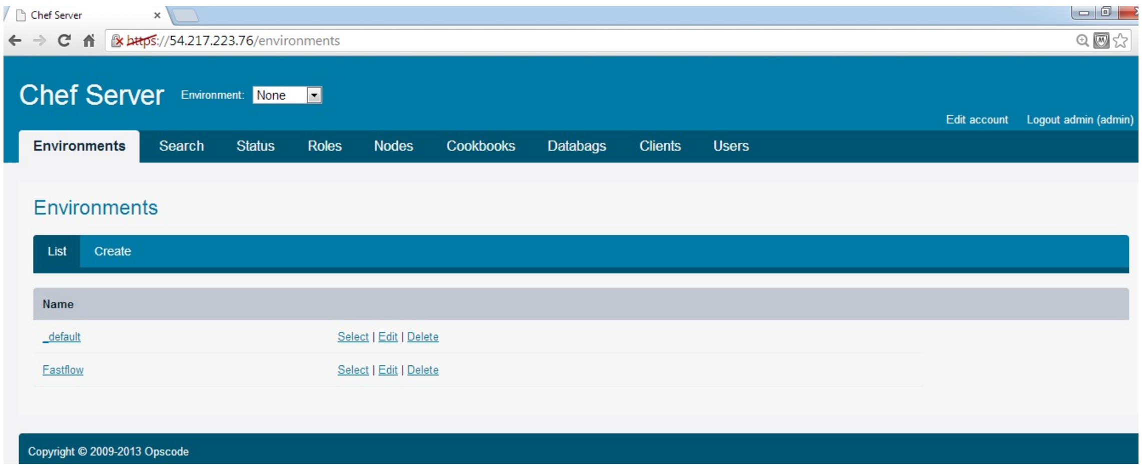
[47] Xingwei Wang, Jiajia Sun, Hongxing Li, Chuan Wu, and Min Huang. A reverse auction based allocation mechanism in the cloud computing environment. *Appl. Math*, 7(1L):75–84, 2013.

[48] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya. Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 195–204. IEEE, 2011.

[49] Jianfeng Yan and Wen-Syan Li. Calibrating resource allocation for parallel processing of analytic tasks. In *e-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*, pages 327–332. IEEE, 2009.

Appendix A

Chef Server

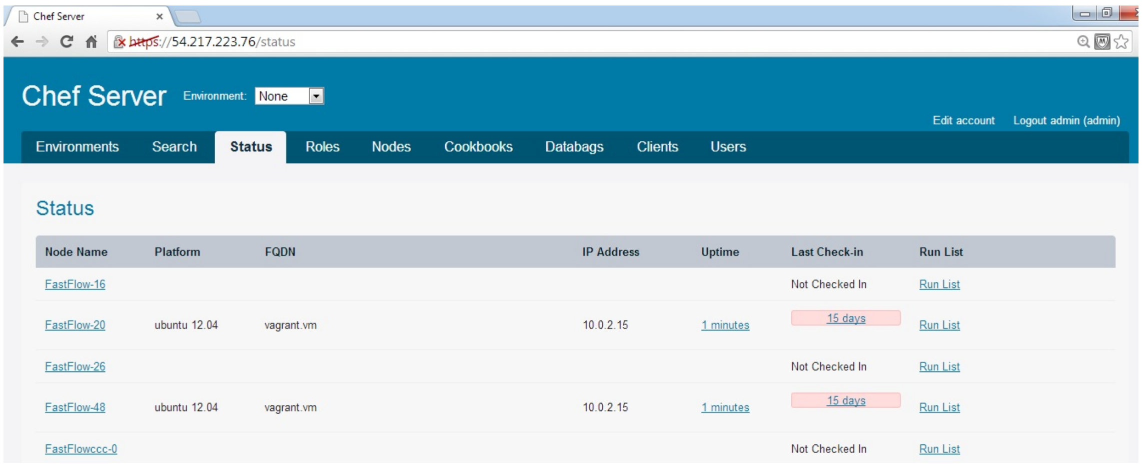


Environments

Name	
_default	Select | Edit | Delete
Fastflow	Select | Edit | Delete

Copyright © 2009-2013 Opscode

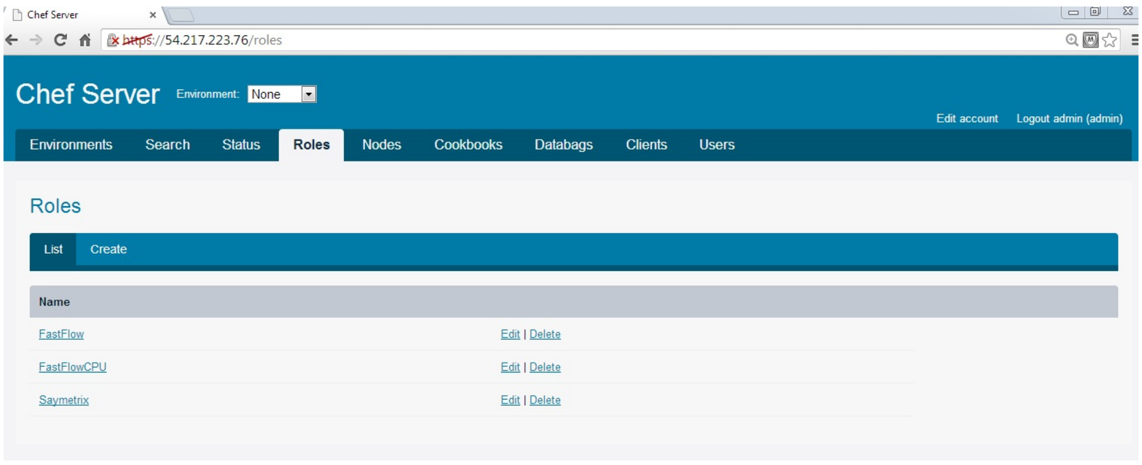
Figure A.1: Chef Environments



Status

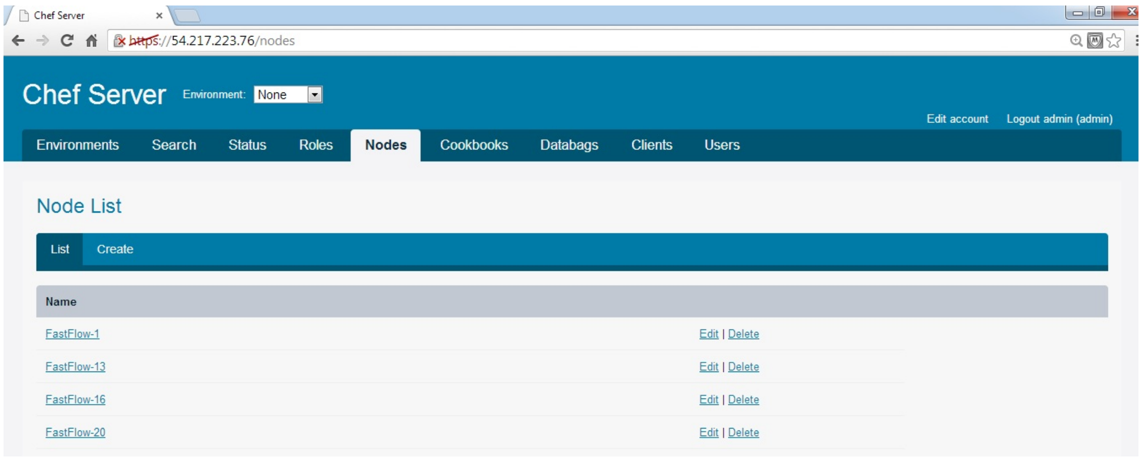
Node Name	Platform	FQDN	IP Address	Uptime	Last Check-in	Run List
FastFlow-16					Not Checked In	Run List
FastFlow-20	ubuntu 12.04	vagrant.vm	10.0.2.15	1 minutes	15 days	Run List
FastFlow-26					Not Checked In	Run List
FastFlow-48	ubuntu 12.04	vagrant.vm	10.0.2.15	1 minutes	15 days	Run List
FastFlowccc-0					Not Checked In	Run List

Figure A.2: Chef Roles



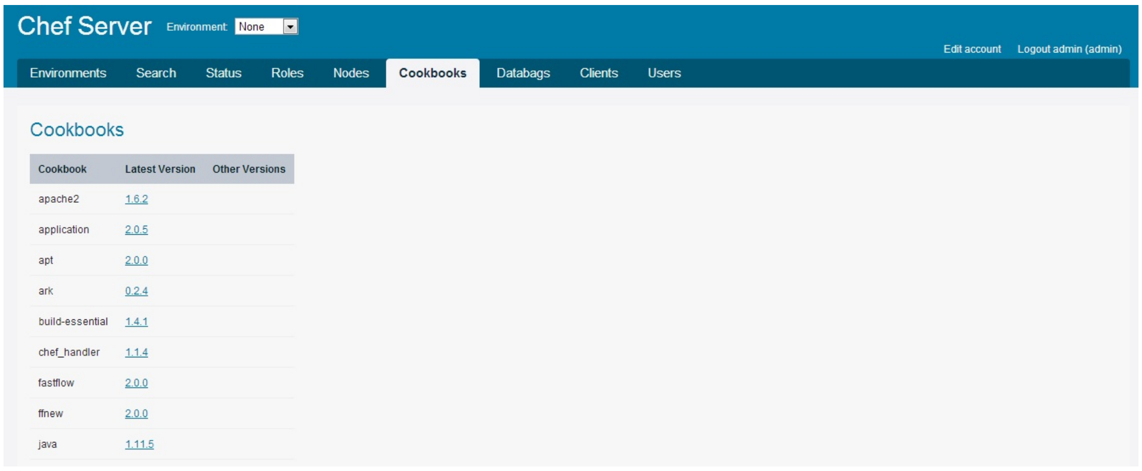
Name	
FastFlow	Edit | Delete
FastFlowCPU	Edit | Delete
Saymetrix	Edit | Delete

Figure A.3: Chef Status



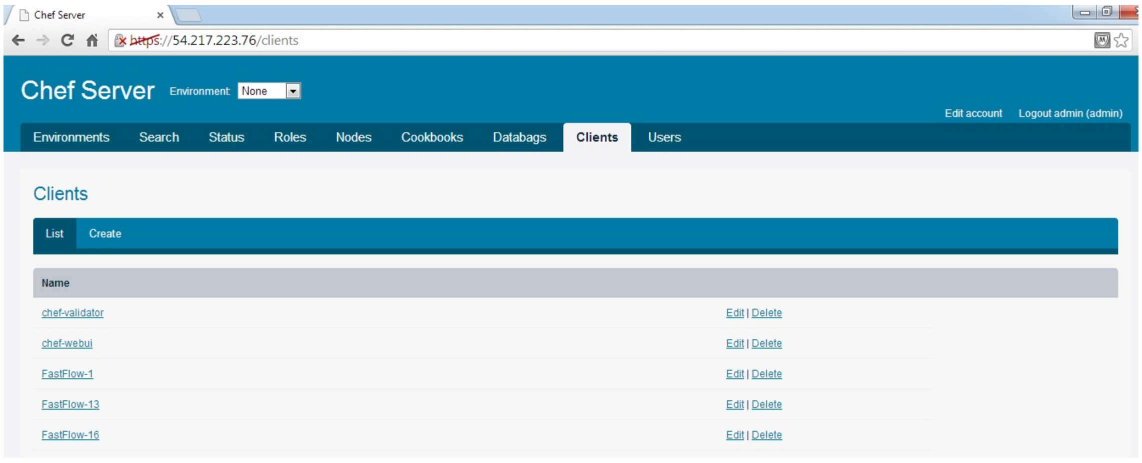
Name	
FastFlow-1	Edit | Delete
FastFlow-13	Edit | Delete
FastFlow-16	Edit | Delete
FastFlow-20	Edit | Delete

Figure A.4: Chef-Nodes



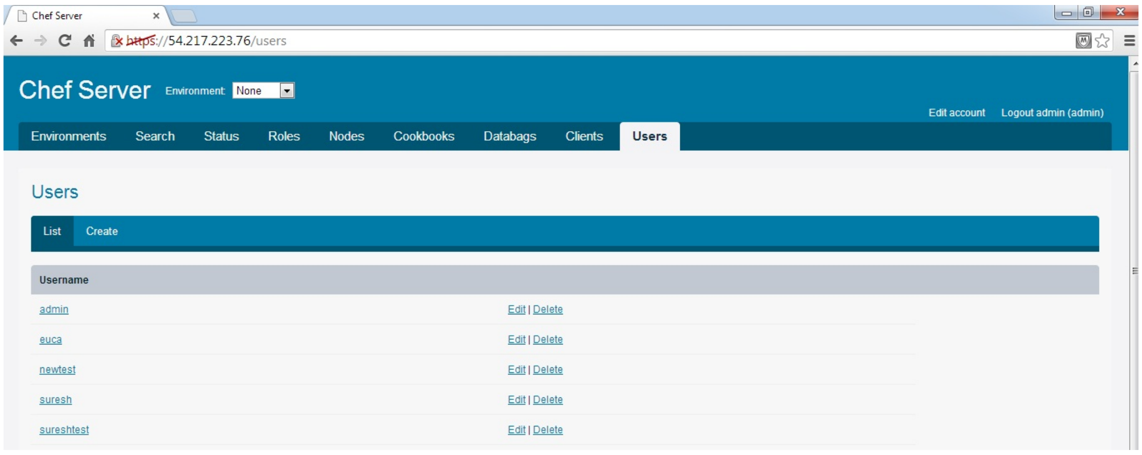
Cookbook	Latest Version	Other Versions
apache2	1.6.2	
application	2.0.5	
apt	2.0.0	
ark	0.2.4	
build-essential	1.4.1	
chef_handler	1.1.4	
fastflow	2.0.0	
ffnew	2.0.0	
java	1.11.5	

Figure A.5: Chef CookBooks



Name	
chef-validator	Edit | Delete
chef-webui	Edit | Delete
FastFlow-1	Edit | Delete
FastFlow-13	Edit | Delete
FastFlow-16	Edit | Delete

Figure A.6: Chef Clients



Username	
admin	Edit | Delete
euca	Edit | Delete
newtest	Edit | Delete
suresh	Edit | Delete
sureshtest	Edit | Delete

Figure A.7: Chef Users