CRANFIELD UNIVERSITY

Léo Unbekandt

Investigation and implementation of resource allocation algorithms for containerized web applications in a cloud environment

School of Engineering

Computational and Software Techniques in Engineering

MSc

Academic Year: 2013 - 2014

Supervisor: Mark L. Stillwell

August 2014

CRANFIELD UNIVERSITY

School of Engineering

Computational and Software Techniques in Engineering

MSc

Academic Year: 2013 - 2014

Léo Unbekandt

(leo@unbekandt.eu)

Investigation and implementation of resource allocation algorithms for containerized web applications in a cloud environment

Supervisor: Mark L. Stillwell

August 2014

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science

Declaration of Authorship

I, Léo Unbekandt, declare that this thesis titled, 'Allocation and migration of web-services isolated in Linux Containers' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor…

Source code license

All the source code developped in the scope of the experiments done in this thesis are developed under the MIT License. The integrality of the examples are publicly available on GitHub https://github.com/Soulou

The MIT License (MIT)

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contents

List of Figures

List of Tables

Abbreviations

VM	**V**irtual **M**achine
SaaS	**S**oftware **as a S**ervice
PaaS	**P**latform **as a S**ervice
IaaS	**I**nfrastructure **as a S**ervice
DBMS	**D**ata**B**ase **M**anagement **S**ystem
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
HTML	**H**yper **T**ext **M**arkup **L**anguage
CSS	**C**ascading **S**tyle**S**heet
JS	**J**ava**S**cript
XML	e**X**tensible **M**arktup **L**anguage
JSON	**J**ava**S**cript **O**bject **N**otation
PID	**P**rocess **ID**entifier
CM	**C**onfiguration **M**anager

Introduction

An interesting definition of the Cloud Computing has been written by the National Institute of Standards and Technology [Mell and Grance, 2011]:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

Different kind of clouds are specified, if Amazon Web Services provides services which are part of a "public" cloud, this is not the only way to use a cloud infrastructure: private cloud or hybrid clouds mixing private and public cloud infrastructures are being developed more and more. Thanks to open-source projects like openstack, cloud environments can be installed on private infrastructures. This is sometimes necessary or requested for security, performance or data control purposes.

The evolution of the paradigm of cloud computing has been made possible thanks to different technologies. The virtualisation, as explained by Paul, Dragovic, Fraser, Hand, Harris, Ho, Neugebauer, Pratt, and Warfield [2003] allows servers to be splitted in different sub-components, isolated from each other, sharing the resources of the physical machine.

Technologies have been developed to give people much more flexibility in the way to manage their applications, their products. Virtual machines got live migration, a process which is detailed in the work of Christopher, Fraser, Hand, Hansen, Jul, Limpach, Pratt, and Warfield [2005]. The feature has been built to move instances from one physical host to another without interrupting the activity of anything running in the virtual machine. The memory is kept intact of course, but also the running connections. The instance may seem frozen for a few second when the migration is finalized, but nothing is disrupted.

Virtual machines have been used and studied for a decade now, this work will focus on another way to isolate resources on a server: containers. In some way, those are lightweight virtual machines. However, instead of having a global focus (virtualisation of hardware + operating system), they focus at the application level. Containers are designed to isolate applications from each other on a similar host. This host can be a virtual machine or more directly a physical machine. Already in 2007, Soltesz et al. [2007] have worked on the possibility to use container-based virtualization instead of hypervisors and virtual machines as a high-performance alternative.

This technology has been more and more used in the industry these last 3-5 years, more and more companies are adopting it. It may be to offer services for companies like OpenShift (Red Hat), Cloud Foundry, Heroku, MongoLab, etc. or to manage the hosting of their own projects: Google, Ebay, Spotify. What kind of applications are containerized? Any software is able to run in a container, the work done by **?** shows in the field of HPC, containers are mature enough to replace virtual machines and get better performance. Recently, more and more companies are building their products using the micro services architecture Arkency. In this model, a set of loosely coupled softwares are communicating together using a communication protocol. The most often, the web (HTTP) is used, and those services are sending and receiving requests through REST API. One of the main advantages of those applications is that they are stateless, as a result, it is much more easy to migrate them.

In this work, the focus will be on those web applications, isolated thanks to containers, hosted on virtual machines. How those services can be load balanced and how is it possible to keep the load balanced over a cluster a servers, with each of them running a different amount of containers.

Chapter 1

Literature Review

1.1 Motivation

The legitimate question is "Why do people migrate their infrastructure to a cloud infrastructure?". Whether it concers virtual machines, whether is is linked to containers, the answers are multiple, Valentina Salapura explains how a virtualized environment improves the resiliency of an infrastructure [Salapura, 2012]. More precisely, when a service requires to be scalable, highly available and fault tolerant, using cloud technologies is essential. In the case of disaster recovery scenarios, they are highly simplified and cheaper thanks to those environments.

As a result the infrastructures are composed of a certain amount of physical machines (PMs) which could be dispatched among different data centers, and each of these PMs, contains a variable number of virtual machines (VMs), then each of them hosts a set of containerized applicatios. The problematic which is now interesting concerns the assignment of these applications, what is the optimal distributions of the containers among the different servers? It depends of what characteristic has to be optimized.

At the scope of the physical server, Thomas Setzer and Alexander Stage base their study on the statement that energy represents up to 50% of operating costs of an infrastructure [Setzer and Stage, 2010]. That's why there is a need to optimize it. Using the virtual machine reassignment through live migrations, they are looking at consolidating the VMs on the physical servers. Consolidating an infrastructure consists in reducing the number of PMs which are hosting instances without disturbing the performance of these. After this operation, useless PMs can be suspended and electricity is saved, then when more computational power is required they are resumed dynamically.

In the publication *An adaptive Resource Provisioning for the Cloud Using Online Bin Packing* [Song, Xiao, Chen, and Luo, 2013a], the authors also introduce their subject by explaining that it has been estimated that Amazon manages more than half a million of physical servers around the world and that it must be a priority for them to reduce their expenses by consolidating their infrastructure.

For consumers of commercial *IaaS* offers, the main goal is to use the minimum number of virtual machines while having enough resources for all the applications running on their current infrastructure. They do not directly pay the electricity, it is included in the price paid to the provider, the focus is on the level of performance directly.

1.2 Algorithms

We have seen that cloud computing is a hot topic in the Internet industry which results in a lot of new problematics in computer science. The resource allocation problem is one of them. All over the world, universities have started studying different approaches of allocation optimisation. The different algorithms listed in this document gather publications around the virtual machine assignment and reassignment on a set of physical machines.

1.2.1 Linear Programming

Also known as Linear optimization. It is specialisation of mathematical programming, which is focused on linear functions. The main goal of linear programming is to find a maximum or a minimum to a linear function given a set of constraints, in other words: maximizing profits while minimizing costs. In scope of resource allocation, it is required to define the different variables, the function we want to optimize and the constraints linked to the variables.

In their work, Ruben Van den Bossche and Broeckhov [2010] are working with linear programming. The aim of their study is to define a way to optimize the number of allocated virtual machines splitted in different cloud infrastructures. Different constraints are defined to setup the scope of the function to minimize.

Equation 1 Example of linear optimization problem

$$Minimize \sum_{k=1}^{A}\sum_{l=1}^{T_k}\sum_{i=1}^{I}\sum_{j=1}^{C}(y_{klij} \cdot (ni_{kl} \cdot pi_j + no_{kl} \cdot po_j) + \sum_{s=1}^{S}(p_{ij} \cdot x_{klijs}))$$

Equation 1 is the problem they want to solve, in this case a cost minimization problem. How can we minimize for each task t of each application k in each virtual machine i of each cloud infrastructure j the price of the input and output bandwidth ($ni \cdot pi_j$ and $no_{kl} \cdot po_j$) and the price the requested virtual machines ($x_{klijs} \cdot p_{ij}$) at each unit of time (S)

Equation 2 Example of constraints in a linear program

$$\forall j \in [1, C], s \in [1, S] : \sum_{k=1}^{A}\sum_{l=1}^{T_k}\sum_{i=1}^{I} cpu_i \cdot x_{klijs} \leq maxcpu_j$$

The *Equation 2* defines a constraint from the linear problem, which explains that in each cloud, at each unit of time, the sum of all the tasks run on all the virtual machines instantiated should be less than the number of CPUs available. (There is note that in the case of public clouds, the amount of CPU is considered unlimited so this constraint becomes void).

The work of Stillwell, Vivien, and Casanova [2012], which focuses virtual machine resources allocation in heterogeneous environment also start by defining a formal model based on linear programming. However, as explained in this publication, resolving such a problem requires an exponential time, linked to the amount of allocations to achieve. As a result using directly this solution on an important workload is not feasible.

The work of Young [1995] about linear optimization relaxation has been used to simplify the original problem and transform it from an exponential complexity to a polynomial complexity. The "random rounding" is a probabilistic approach which modifies some of the constraints by a weaker one.

Equation 3 Application of random rounding

$$\text{constraint before: } 0 \leq x \leq 1$$

$$\text{constraint after: } x_r \in 0, 1$$

$$x_r = 1 \text{ with a probability of } x\text{, otherwise: } 0$$

However, the RRND approaches is quickly discarded as the results are not good enough in the case of resource allocations in heterogeneous environment.

1.2.2 Bin packing

Bin packing is one of the most common approach to resource allocation or re-allocation in a cloud environment. It consists in representing "bins" associated to a storage capacity and "items" which have to be packed into those bins.

1.2.2.1 Different variants

Two main types of bin packing algorithms exist. On the one hand, those considered as "offline". They consider that we have access to all the items to find the optimal packing on the different bins. This problem is a NP-hard problem, there is no, to this day, a polynomial way to solve this problem. That is why to answer this problem in a reasonable duration, different heuristics have to be defined. The most common have been studied by Johnson [1967]:

Algorithm Name	Description
First Fit (FF)	Pack the item in the first bin with a large enough capacity
Best Fit	Pack the item in the bin which will have the less capacity after packing
Worst Fit	Opposite of Best Fit: Pack the item in the bin with the biggest capacity
Next Fit	Same as FF except that instead of reconsidering the first bin after packing, the current one then the next one is considered
*-Fit Decreasing	First, sort the items in a decreasing order, then apply any of the *-Fit algorithm

Those different algorithms reduce the complexity of the packing operation to $O(n \log n)$. But as Johnson [1967] title explains: they are "Near-Optimal". The issue is finally to find the best ratio optimality/complexity.

On the other hand, the "online" algorithms, which, on the contrary, are packing items at the time they are arriving. In this case bins are already partially filled with other items, and it is not always possible to move those. Thus, the main goal is to find the best assignment for the newly coming item. Previous *-Fit could be directly used. However, it is really limited to pack one item in a set of bin, this is why different algorithms have been developed

To answer more precisely to the cloud resource allocation problem, some people have defined some variants of those two main categories of bin packing algorithms. G. Gambosi and A. Postiglione and M. Talamo have developed a "relaxed online bin packing" algorithm Gambosi, Postiglione, and Talamo [2000]. It may be represent as a mix between online and offline bin packing. When a new item has to be packed, it allows an additional limited number of moves among the currently packed items.

Another interesting variant is the dynamic online bin packing defined by Joseph Wun-Tan. It differentiates itself from standard online bin packing by allowing items to be removed from bins. Static online bin packing does not allow these items changes, once an item has been placed it does not move anymore.

1.2.2.2 Their application in resource allocation

In the scope of containers assignment on a set of hosts, the bins are the differents servers and the items are the services we want to host. Some additional aspects have to be considered: applications need different resources like memory, CPU, persistent storage (disk), network input/output. So often, the items we want to pack are multidimensional items, and we speak of multidimensional vector bin packing. Another interesting point is that moving a container from one host to another has a cost which may be important, even if it is cheaper than migrating a virtual machine. As a result we can not execute numerous container migrations simultaneously.

In the work about online bin packing for virtual machines allocation of Song, Xiao, Chen, and Luo [2013b], the authors consider first, that a virtual machine only has one dimension, its CPU consumption. From that point they study which algorithm may fit this particular problem. They reject "strict" online bin packing, because in realistic situations it is uncommon to know exactly the future consumption of a virtual machine, so it is necessary to move it afterward, when we can measure it. Moreover, as VMs can be migrated easily, there is no reason not considering it if the resulting performance is better. "Relaxed online bin packing" allows movements when a new item is packed, but an item cannot be resized. "Dynamic online bin packing" is thought inadequate in this context too, but often, when an virtual machine has to move the best solution is not always to remove it then repack it, but to move others instances which are easier to move.

This is why in the publication of Song, Xiao, Chen, and Luo [2013b], they decided to build an online bin packing algorithm which suits the virtualisation environment: "Variable Item Size Bin Packing", its characteristics are the following.

- As relaxed online bin packing, it allows movements when a new item is packed
- Stronger limit of movements, to avoid executing too many migrations
- A `change` operation is defined to modify the size of an item in a bin

They extend their algorithm to multidimensional vectors by considering the biggest value among the different dimensions of a vector, so the problem returns to one-dimension. Using this way to simplify the problem is working in some cases. Commonly when a resource consumption increases the others are following. For example an application having a high network bandwidth requirement, would also have a high CPU consumption. Finally, they admit that this solution would work quite poorly in the case of instances with non-proportional requirements.

In Stillwell et al. [2012], we have seen that the first approach of the author was around linear programming, but the main part of their work is defining a way to apply multidimensional vector bin packing to heterogeneous environments. On a first side, they deal with the multidimensional aspect of this problem. It is necessary to specify how to sort the items because there is not natural way to sort these vector.

- Value of the maximal dimension
- Sum of all dimensions
- Ratio of the max/min
- Difference max-min

- Lexicographic order
- None

Most of the previous algorithms are not considering the way the bins are used. In this publication, as it is targeting heterogeneous infrastructure, the order the bins are sorted when executing any algorithm matters. All the previous way to sort the items can be applied to the set of bins.

All these previous possibilities of ordering among the virtual machines and physical hosts are combined and result in a "meta" algorithm (METAHVP) which takes the best result out of the different combinations of one items ordering and one bins ordering. After individual analysis, some sort types are removed from the meta algorithm to improve its runtime. (METAHVPLIGHT)

The simulation achieved to test these heuristics are comparing the results to those which have been found using the linear programming method and those obtained using greedy algorithms (*-Fit). The conclusion is that METAHVP has the best results over all the other, and METAHVPLIGHT achieves this result in on tenth of METAHVP's runtime.

Finally, according to what we want to study there are several possible solutions using bin packing. Semantically, it is really comfortable to compare bins with physical servers and items with virtual machines, it allows a very natural vision of this problem.

1.2.3 Others

To deal with mathematical optimization and approximative solution of NP-complete problems, Ants colony algorithms, genetic algorithms and some other famous methods, based on statistical analysis.

1.2.3.1 Ant colony algorithms

In Chimakurthi and SD [2011], Feller et al. [2011] and Zhu et al. [2012], the ant colony algorithms are studied. As we can see in the following graph:

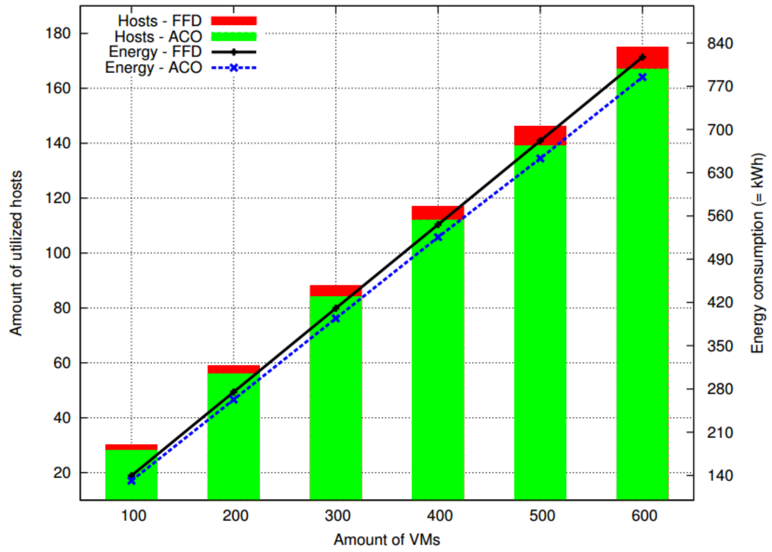

FIGURE 1.1: Comparison between First Fit Decreased and Ant Colony algorithms in Feller et al. [2011]

The simulation shows that the ant colony gets better performance than a simple greedy First Fit Decreasing, however this improvement is not free:

VMs	**Policy**	**Hosts**	**Execution time**	**Energy (= kWh)**	**Energy gain (= %)**
100	FFD	30	0.39 sec	139.62	
	ACO	28	37.46 sec	131.41	5.88
200	FFD	59	0.58 sec	275.13	
	ACO	56	4.51 min	262.83	4.47
300	FFD	88	0.77 sec	410.65	
	ACO	84	15.04 min	394.28	3.98
400	FFD	117	1.03 sec	546.16	
	ACO	112	34.23 min	525.75	3.73
500	FFD	146	1.39 sec	681.67	
	ACO	139	1.17 h	653.17	4.18
600	FFD	175	1.75 sec	817.19	
	ACO	167	2.01 h	784.75	3.96

FIGURE 1.2: Runtime of First Fit Decreased and Ant Colony algorithms in Feller et al. [2011]

When the number of nodes becomes bigger, the time spent to find the optimal allocation grows hugely, it is thousands times longer than a simple First Fit Decreasing for 3 to 5 percents of improvement. For analysis purpose it is something interesting to get better results, but in a realistic point of view, this operation can not take several hours as it should be repeated often.

1.2.3.2 Genetic algorithms

Genetic algorithms (GA) are heuristics based on natural selection. Generations of solutions are mutating, inheriting with and from each other to result in close to optimal results. Wilcox et al. [2010a] and Wilcox et al. [2010b] focused on them to solve the virtual machines assignment problem. In the work of David Wilcox et al.Wilcox et al. [2010a], simulations are comparing GA with *-Fit algorithms.

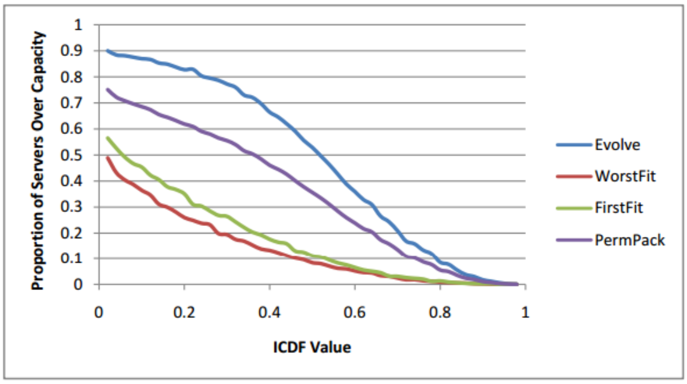

Fig. 6. A comparison of the proportion of servers over capacity.

FIGURE 1.3: Results of simulations using a genetic algorithmWilcox et al. [2010a]

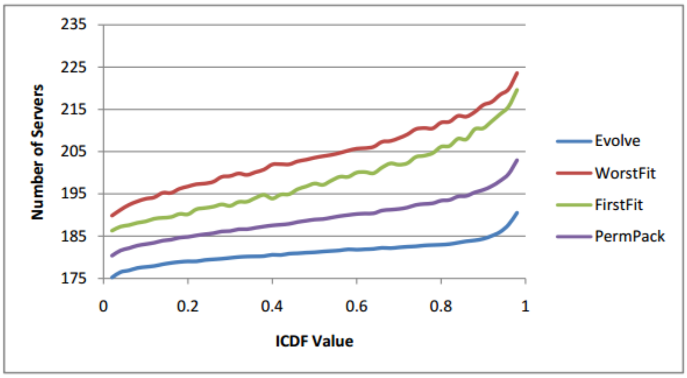

Fig. 7. A comparison of the the number of servers found.

FIGURE 1.4: Results of simulations using a genetic algorithmWilcox et al. [2010a]

On the following graphs, ICDF stands for "inverse cumulative distribution function" also known as "quantile function", the authors use it to represent the load: "Using the icdf, we can specify a percentile value and obtain a corresponding load which can be passed to the assignment algorithm".

The conclusion which is that GA tends to consume less physical hosts, at any load, the number of PMs is largely under the amount of servers used by the other bin packing algorithms. As a direct consequence, the PMs which are over-capacitated (where the amount of VMs exceed the resource capacity of the physical sever), is much more high. For this reason, this approach can hardly be used in environment where a SLA (Service Level Agreement) has to be respected, because if there are overloaded servers, some applications or tasks running of them will be slowed by this situation.

1.2.3.3 Network flows

Network flows are basically directed graphs where each edge has a capacity and a flow. The main property is that each node of this graph must have an equal sum of flows from the edges directed to it and leaving from it, except for two particular type of nodes: "the source node" and "the sink node".

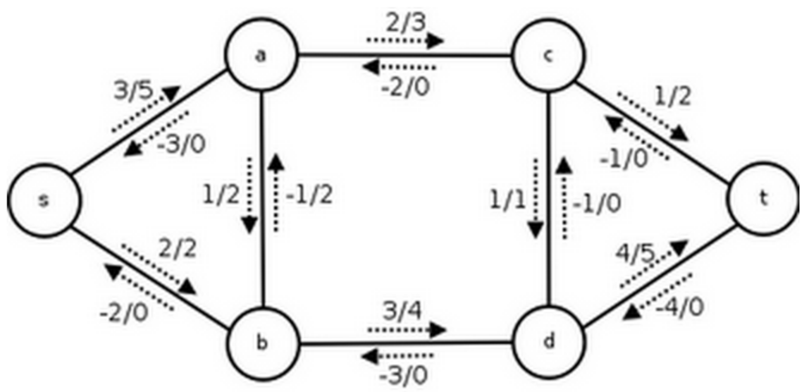

FIGURE 1.5: Example of network flow directed graph

Some people have used this concept to build a model to solve the resource allocation problem, to fine a close to optimal solution. Kimish Patel, Murali Annavaram and Massoud Pedram worked on resource assignment in datacenterPatel et al. [2013], considering an heterogeneous environment as in Stillwell et al. [2012]. Each set of similar servers, considered as a pool of servers is represented by a node, with a capacity different from each other according to the differences between two pools of servers.

Unfortunately, this technique does not seem to be used for virtual machines allocation, and the link between this method and the problem we are dealing with is not obvious at all.

1.3 Real data analysis

Most of the cited works in the literature review are basing their work on simulations. In the experiments, simulation tools like SimGridINRIA or CloudSimUniversity are used to simulate the behavior of one or multiple cloud infrastructures.

The data may be generated randomly or following some statistical rules, but often, workloads are based of extract of real workload. Typically, Google is releasing workloads of its own production infrastructure.

In 2012, Google sponsored the ROADEF contest (Operational research and decision support French society)members. The contest was focusing the machine reassignment problem based on Google workload. Each attendee had to find the best solution find solution. Some of them resulted in an official publication like "Heuristics and matheuristics for a real-life machine reassignment problem" from Ramon Lopes, Vinicius W.C. Morais, Thiago F. Noronha and Vitor A.A. SouzaLopes et al.. They based their work on linear programming. However in Masson et al. [2013] and Kell and van Hoeve [2013], the authors have used around the bin packing algorithms. Unfortunately, the work of the winner has not been published so we are not able to see which algorithm has been used to achieve the best reassignment.

Chapter 2

Container load balancing in cloud environment

2.1 Definition

The technology of the operating system-level virtualization is composed of different mecanisms to create isolated environments in the user-space. Each of those environment can gather one or several running applications and has access to different resources. Those environment are commonly called containers from the tool which popularized them: LXC (LinuX Containers). This technology is

Operating system-level virtualization has been existing for a long time, it appeared first in the BSD kernel (1998), where the technology is called **Jails**. Then, Sun developed Solaris (Sun UNIX operating system) **zones** in 2005, the same year as the **OpenVZ** implementation for the Linux kernel.

Containers are running over the same operating system as the host system, they are sharing the same drivers, but all the processes contained in them are limited by this same operating system. The memory consumption, the CPU usage, the network and disk IO are monitored and managed by these container engines sending the corresponding instructions to their respective kernel.

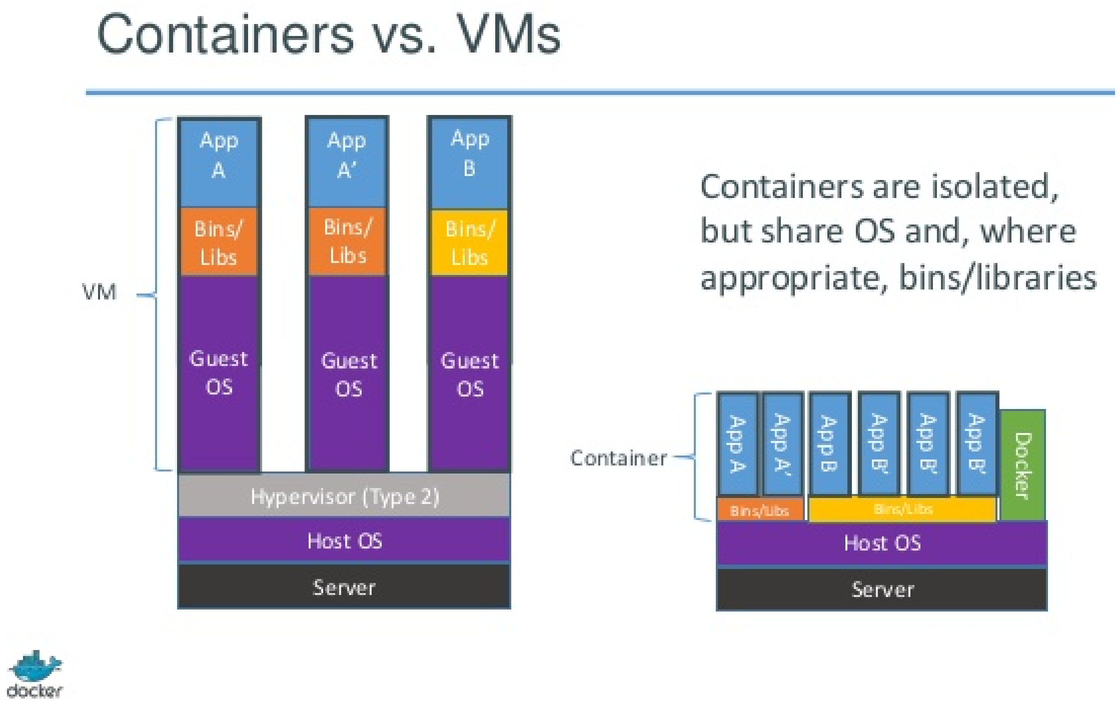

FIGURE 2.1: Structural difference between containers and VMs

This is a completely different approach to process isolation compare to classical virtual machines. Where hypervisors and VM have been following the paradigm where everything is virtualised, creating overhead and slower performance, then we look at optimising by accessing hardware in order to reduce binary translations and other slow operations. The main idea for containers is, based on the host operating system, only the required devices/features will be virtualised, and finally the level of performance is close to native efficiency.

2.2 Docker container engine

2.2.1 A bit of history

Docker is a recent project. Introduced in May 2013 by the Platform as a Service provider **Dotcloud** with the ambition to create a standard way to manage multi-platform containers, **docker** has rapidely been promoted as a mainstream project supported by all the main tech companies. Their catchword is "Build once, run everywhere", but this is only a half truth.

The project has been created as a REST-ish API server, using LXC tools to manage the containers themselves, however LXC is, as its name shows (LinuX Container) specific to Linux-based operating system, at that time it was even more restrected, **docker** was only working on Ubuntu Linux amd64. This is why the "run anywhere" was a bit biased at that time. One year later, the project has abandonned LXC to create their own library named *libcontainer*, which is able to run on mostly any Linux. In the future, the project leaders want docker to be able to run on any kind of containerization system : BSD Jails, Solaris Zones, and Linux Containers, to become the real interoperable standard for containerization.

2.2.2 Its contribution to operating system level virtualization

As stated previously, containers were existing for a long time before **docker**, but this project has succeeded to create a wave of motivation and keen interest around this technology. People were so enthusiastic that **Dotcloud** pivoted their activity, to focus on **docker** and changed its name to **Docker inc.**

What has been brought by this container engine is a simple way to manage and deploy containers one a large amount of server through a simple API. This simplification took over the LXC tools which were known to be difficult to handle.

Docker also use copy on write filesystems, it means that if different applications are using the same base files, they only need to be present once. The precise caracteristic let users create containers in milliseconds.

Currently, thanks to **Docker inc.** containers are really fashinoable, every tech companies is looking at them and their evolution and more and more people try to get rid of heavy virtual machines, because even if containers do not have all the features of virtual machines, they are good enough in a lot of cases.

2.3 Advantages

Studying containers is not a random choice. They have been more and more present in the industry these last years. Companies keep externalizing their infrastructure, and the hosting of their services. The phenomena happens for various reasons. A company infrastructure has to be robust, and available most of the time. Nowadays, unavailability means important losses of money. As hosting is a craft by itself, most of the companies do not have enough fund to invest in a dedicated IT department, so they have to externalize thoses processes.

The amount of resource providers, whether it is an application (Software as a Service - SaaS), a platform (Platform as a Service - PaaS), or an infrastructure (Infrastructure as a Service - IaaS), is increasing heavily, because for the final users, it is cheaper than doing it themselves, and it is easy to use, the internal mecanisms are abstracted.

Those providers have all the same problems: what is the best way to setup a multi-tenant architecture which is secure enough and fast enough. "Containers" is an answer to this issue. For example, the company **MongoLab** is hosting thousands of MongoDB databases. Data is something critical for any company, so **MongoLab** needs tot isolate each instance of MongoDB from each other. We can assume that most of the databases they are hosting don't have a really high traffic. Having a virtual machine for each of those instances is clearly something oversized and would result on high provisionning overhead (duration of virtual machine boot), storage overhead (1 full operating system per instance), etc. This company is using containers because, it allows them to isolate the databases, to provision them instantly, and the files required to run MongoDB are only present once on their servers. (physical or virtual).

2.4 Limits

Containers are not able to live-migrate from one host to another with a standard linux kernel yet. This feature is possible with a OpenVZ patched kernel because thoses patches implement the checkpoint/restore operations for the containers, but for a vanilla Linux kernel, it does not exist yet. Some developers/hackers are trying to clean the code of OpenVZ and push the features to the mainstream kernel with the CRIU project, but so far the results are mostly drafty and unstable.

This main limit results in the difficulty to host stateful applications like a database. It can be isolated in a container but we don't have the possibility to move it without any downtime, the container has to be stop first then restarted on another host. This is particularly blocking in the case of production environment where every downtime leads to money loses for instance.

2.5 Web Application

As containerized stateful applications can not be cleanly load balanced among a set of servers (a downtime is required), stateless web applications will be targeted, as stated in the introduction of this work.

A Web application is an applicative server which uses the web standards to communicate with clients. There are two main types of web services. The websites, which are rendering HTML/JS/CSS web pages to users, and web services defining an API and answering with standard data formats like XML or JSON. Both of them are using HTTP as transfer protocol.

By the nature of HTTP, web applications are mostly stateless. Each resource request is done using a new connection (except the case of reusing opened connections). When a web application is stateful it is linked to the application itself which is linking information to a local session or connection.

These last 5 years, more and more of the web services have been written based on some or all the principles of the REST method which declares as "best practice" to create complete stateless applications. Additionaly, another manifesto, the Factors has become a standard set of good practices for web development (website and web services)

The main advantage of stateless services is that they are able to scale horizontaly easily: the first step is to spawn new instances of the service, and then modify the routing table of a frontal reverse-proxy. As a result the requests will be distributed among all the instances.

2.6 Application balancing on the infrastructure

When a web application has to be moved from one host to another, there should be no unavailable time and the current requests have to stopped gracefuly. To solve the first issue, the following walkthrough has to be followed:

1. Create a new instance of the application - Instanciate a new container of a web application
2. Wait until the instance is available - TCP ping the application until a connection is established
3. Change reverse proxy routing to route requests to the new container and not the old one
4. Stop the old container to free its resources

To solve the second issue, it should be handle by the application itself. When the system is querying the old container to stop. It actually sends a signal to it. In most systems (Systemd, Upstart at the system level, or Heroku and Dotcloud at the PaaS level), SIGTERM is sent, then the application has some time to shutdown. In the case where the application is still running a while after receiving the signal, SIGKILL is sent to get rid of the process.

2.7 Operation on containers

2.7.1 Load balancing

The load balancing process consists in moving applications in order avoid having over-loaded and under-loaded hosts.

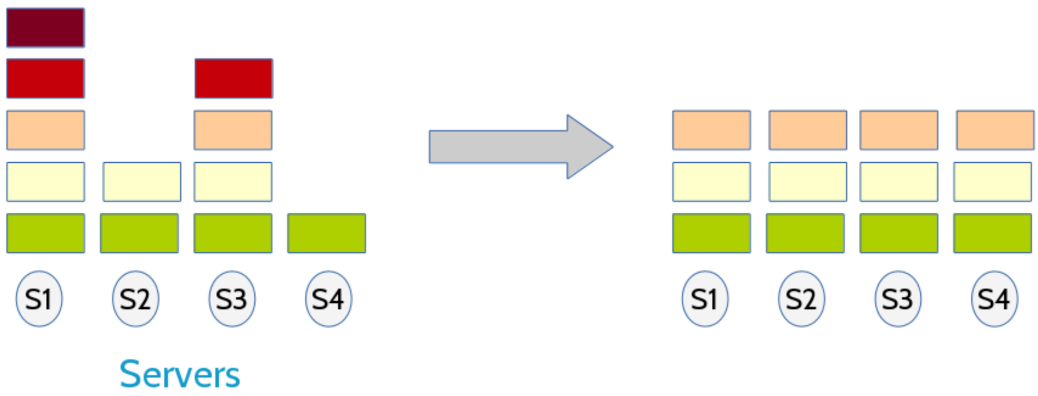

FIGURE 2.2: Schema of a load balancing process

In most of the case, we can't really predict the evolution of the resource usage of a service, this step has to be often to maintain a balance in an infrastructure. There are three potential outcome from this operation. The first is, that the current number of hosts is sufficient, so the containers are dispatched on them to get a balance of resource consumption. The second possibility is that all the hosts are completely busy. In this case some new servers should be provisionned. (Through a IaaS API, or more simply by sending an email to the infrastructure manager who will have to deal with the situation) The last case is when the hosts are not required anymore because there the applications can be packed on less nodes that before. There are different behavior which are possible in order to spare electricity and/or money. If the hosts are VMs, they can be shutdown, if they are physical servers, they could be suspended (If a mecanism like WakeOnLan is enabled to wake them when they are required again) for example, these operationnal pieces information are not in our scope.

2.7.2 Resource Allocation

Whan a new application has to start on the cluster, a container has to be created. At that step, it is required to find the best server to host this newly spawned application

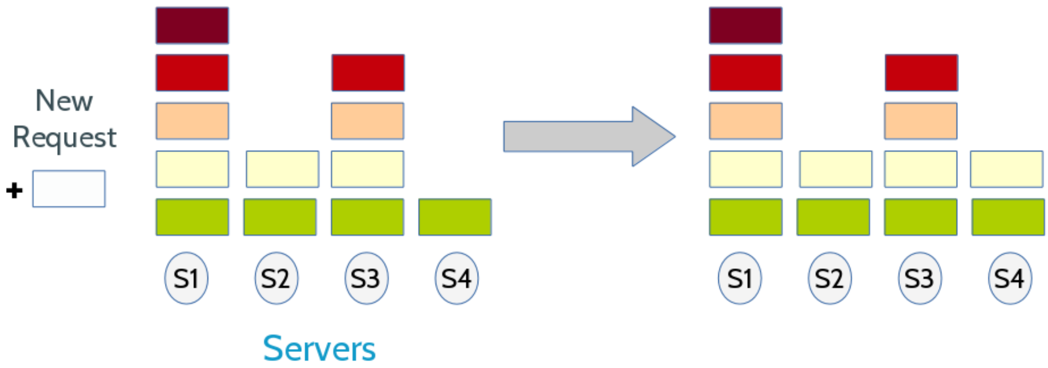

FIGURE 2.3: Schema of a resource allocation process

For that step, it is required to find the most available server, because deploying an application on a node which is already under an important load can have repercussions on all the different containers hosted on it.

Chapter 3

CPU allocation and scheduling for containerized processes

3.1 Goal of the experiment

This experiment has been defined to study of the ability to isolate containers CPU usage using Linux control groups Linux containers are sharing the same operating system, they are not fully isolated as we can see with complete virtual machines. To achieve this isolation, the control groups (cgroup) of the linux kernel are used to apply limits on the resource access right of each container.

This experiment aims at studying how these cgroups are working and how do they actually share and isolate the CPU resources among the different containers.

3.2 Metrics

3.2.1 Inputs

The number of CPUs that an application consumes has to be clearly defined. In each container, an application developped to consume a given number of CPU cores will be launched. The source code of the application can be found at https://github.com/Soulou/msc-thesis-cpu-burn.

```
# Parameter n: Number of core to consume
./msc-thesis-cpu-burn -nb-cpus=<n>
```

The second input corresponds to the number of shares a container can access on the CPUs of the running computer. This number is arbitrary as the shares are relative to each other.

> If a container does not have any cpu share number specified, the default value is: 1024

It is expected that if there are two containers, one with 1024 cpu shares and the other with 2048 CPU shares, the second container will have access to $2048/1024 = 200\%$ of the resources, for a single CPU: 33% and 66%.

3.3 Setup

3.3.1 Hosts

To test the capacity of the isolation by cpu shares, two different environments will be used. As the result are expected to be relative to the hardware their should not be any major differences between both, but as a sanity test, it is important execute it on two différents contexts

The first one my personal laptop, here are its caracteristics:

- CPU: Intel® Core™ i7-3537U CPU @ 2.00GHz (2 cores with hyperthreading)
- Memory: 8 GB RAM DDR3

- Disk: 256GB Solid State Drive

Then we'll study the results of the same experiment on a 4 cores virtual machine based on an OpenStack cluster:

- CPU: 4 KVM vCPUs
- Memory: 8 GB RAM
- Disk: Virtual HDD 80GB

3.3.2 Deployment

In order to simplify the reproduction of these experiments, the different applications have been packaged into container images. They can be found on the docker public repository:

- `soulou/msc-thesis-cpu-burn`
- `soulou/msc-thesis-docker-cpu-monitor`

In order to deploy them, simply install Docker on your host (http://docs.docker.com/installation/), then use the `docker pull` to get the container images locally.

```
docker run -d soulou/msc-thesis-cpu-burn -nb-cpus=<n>
...
# Run more instances according to what your want to test
...
docker run -i -t \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /sys/fs/cgroup/cpuacct/docker:/cgroup \
  soulou/msc-thesis-docker-cpu-monitor -cgroup-path=/cgroup
```

The cpu monitoring service will display in columns the cpu consumption of each container running on the host (including itself), the data are displayed to be quickly usable by a third-party data analysis tool like **R** or to draw graph with **Gnuplot**

3.4 Expected results

Four different experiments have been done:

- 4 Processes with equal CPU shares :
  The tested hosts have a total of 4 cores, normally 4 processes using 1 core each should be able to share it equaly, and each of the process should be able to get 100%
- 4 Processes with different CPU shares 128-256-512-1024 :
  For the same reason as the previous experiment, the shares should not change the results. Even if some processes have less priority over the CPU, as there is enough cores for all the processes, they should all be able to run their process at its maximum potential.
- 6 processes with equal CPU shares :
  This case is different, as there is a higher number of processes compared to the amount of available computation units. With an equal amount of CPU shares for each process, it is expected that each process will get 66% in average of CPU time. The results can't be stable as the mount of cores is not a divisor of the number of applications. In other words, there is no way the operating system can allocate an equal share of core per process, as the context of a process is linked to one core. An application can't be 33% on one core, and 33% on the other one at the same time.
- 6 processes with different CPU shares 32-64-128-256-512-1024 :
  According to the rule defined previously, a process with 64 shares should have twice more CPU time than a process with 32 shares but twice less than a 128-shares process.

3.5 Results

All the following graphs represent the percentage of CPU time per process in function of the time in seconds.

3.6 On the laptop

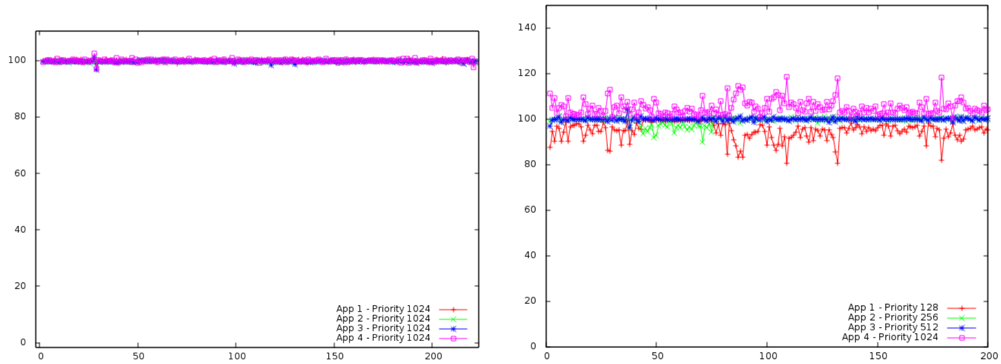

Figure 3.1: 4 Processes with equal[1] and different[2] CPU shares

Using 4 processes, the expectations are reached, even if there are some small differences between the excution with equal shares and the one without, it is clear that each service can use one complete core whatever are its CPU shares.

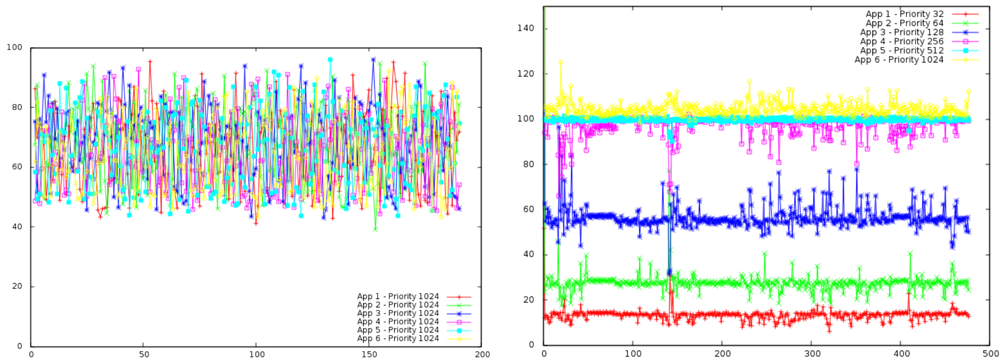

Figure 3.2: 6 Processes with equal[1] and different[2] CPU shares

When 6 processes are executing on the host the observed behavior is different. When shares are equal, the cpu consumption of each process is completely unstable. As explain in the expectation for this experiment, theoretically each process should have 66%, but as it's not possible because a process is only attached to one core at a precise time, the operating system is moving the processes during all the calculations. This is why the curves are so changing. But overall, if we measure the average and median of the CPU consumption of each application, the result is 66%, so the expectation is reached.

In the case where 6 processes are running with different CPU shares, the results are linked to what has been planned, but not only. The process with the minimum amount of shares (32) is using ≈ 15% of CPU, then the one with 64 shares has ≈ 30% of CPU consumption, and then, the third one has ≈ 60% of processor usage. These values are effectively each time twice higher as the previous one. However this rule is not respected afterwards. Three of the process are able to use one full core event if their shares are respectively really different (256, 512, 1024)

3.7 On the virtual machine

As previously said, the results of this experiment in another environment should not be fondamentally different. As the results are relative percentages, the same figures should be found.

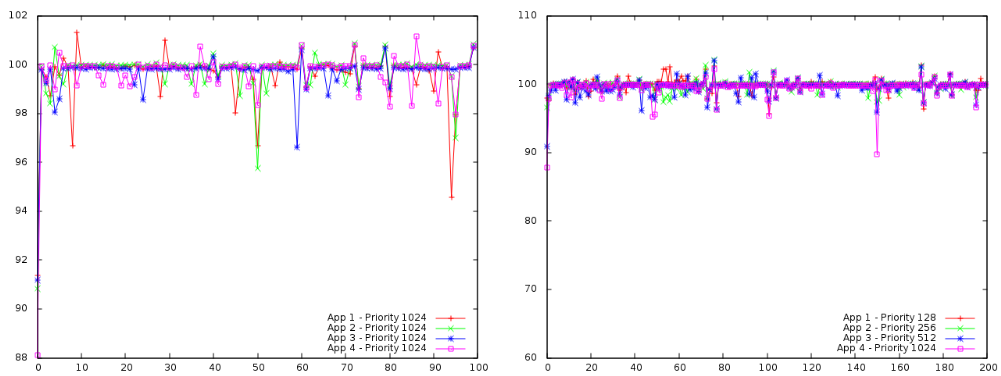

FIGURE 3.3: 4 Processes with equal[1] and different[2] CPU shares

When 4 processes are running, the results are not fondamentaly different with the privous ones. There are some instability which may be linked to the virtual machine environment, but concretely, each process got 100% of a core.

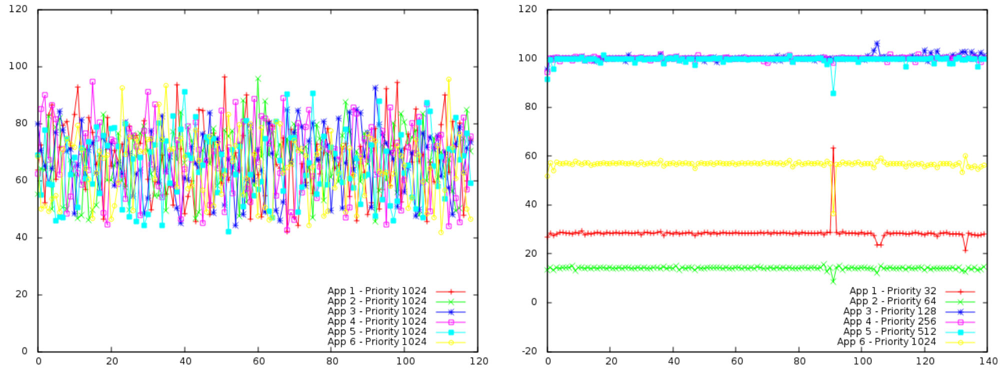

FIGURE 3.4: 6 Processes with equal[1] and different[2] CPU shares

Once more, the results are what we could expect. It is interesting that the consumption of 6 processes looks much more constant in this environment. It may be interesting to investigate that in order to know if it's because there are less applications running on the server than the laptop or if the vCPUs used by the virtual machine are rounding of the real CPU consumption on the physical host provided by the hypervisor. It doesn't change fundamentally the results, but there is a difference.

3.8 Additional analysis

We have seen that with 6 processes on 4 cores, 3 of them are able to get a full CPU, in this case the processes are limiting the experiment. That is why, the operation has been repeated with the following parameters.

```
docker run -d -c <shares> soulou/msc-thesis-cpu-burn -nb-cpus 2
```

In this case, each container will try to get 2 cores, so 200% of CPU.

	App6	App5	App4	App3	App2	App1
Median	144.0	90.11	63.29	38.58	23.59	16.04
Mean	145.6	90.22	64.48	40.20	25.07	17.52

TABLE 3.1: Median and mean of the different application CPU usage in %

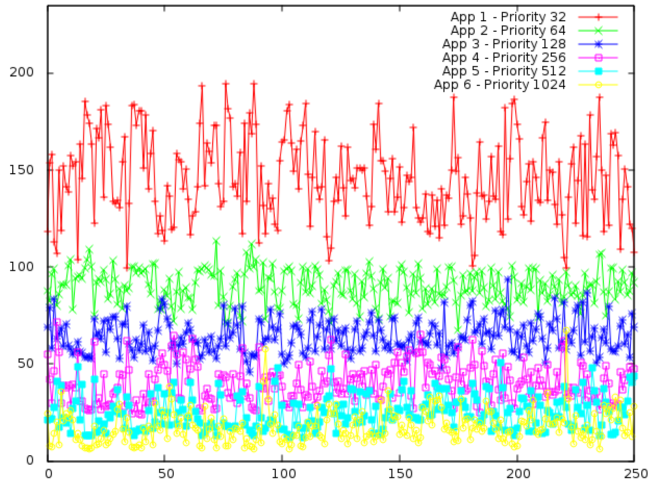

FIGURE 3.5: 6 Processes using 2 cores with different CPU shares

Compared to the previous experiment, we can first observe that the limit is not the process anymore, none of the applications have reached their maximal potential 200%. However, the observed relation between two tasks has been lost. The more shares it owns, the more CPU usage a process has, accordingly to the shares. The operating system is respecting the shares, in a best effort. The priority of one process over another can not be precisely defined.

3.9 Conclusion on the experiment

The main goal of this experiment was to show that it is possible to give different priority levels to different applications running on a given host. This is something really important because in the scope of multiuser infrastructure, resources should be manageable. The **cgroups** already allow to set hard limit on memory consumption, but the CPU limits are different. The **cgroup** cpuset has not been mentionned in this experiment, it is another solution for CPU resource management but it is not integrated in **docker**. The **docker** developers are currently working on the addition of more resource control features but as the project is still quite recent, they are still not implemented.

The obtained results are in line with the expectations, as long as there is less processes than cores, the shares have no effect, but when the relation changes and the number of processes is increasing, the shares can have an important impact on how an application will be able to use the available cores. From a Quality of Service perspective, it allows to regulate the processes on a server base, to decrease the "Bad neighbour" effect, when one or different processes are slowing down all the othere processes of a host.

Chapter 4

Definition of the experimental setup

This chapter will be devoted to the study of the containerized web application allocation and load balancing in a cloud environment. Before speaking of the experiments themselves, it is important to define how those experiments have been setup. The target is the ability to test algorithms using a realistic infrastructure and not to create a simulation of it.

4.1 Hardware Infrastructure

The experiments have been done on a private cloud infrastructure, powered by Openstack (version: Grizzly). The amount and the capacities of the virtual machines have been different according to the experiment, but all the instances are always in the same private network.

This document won't cover how to install Openstack but there are plenty of tutorials on the web. Moreover this setup can be done in a public cloud infrastructure like Amazon Web Services EC2, there is no difference. We are going to use two distincts kind of nodes. The agents execute the different web applications, and the controller which is the interface to control the complete infrastructure.

4.2 Software Infrastructure

4.2.1 Operating System

All the virtual machines are running **Ubuntu Server 14.04 LTS**, this choice has been lead by the fact that this Linux distribution is probably the most standard worldwide and because *python 3.4* is required to run some libraries of the project to execute the experiments.

The cloud version of the distribution has been chosen[1] if order to be compatible with Openstack and correctly boot. To add the image to an openstack cluster, only one command is required:

```
glance add name="ubuntu-trusty" is_public=true \
  container_format=ovf disk_format=qcow2 < /path/to/file.img
```

To deploy, run, stop and migrate our applications, **Docker** will be used. More precisely its REST API. Actually all the requests to Docker are done through HTTP requests on a unix socket. (**Docker** is using a unix socket owned by root for security reasons, to avoid remote access to the host)

4.2.2 Service discovery

One of the common difficulties in cloud infrastrucure gathering numerous virtual machines is the service discovery. It is possible of course to use a configuration manager to generate static configuration on each node which will be used by the different services. However this system is static, doesn't scale well and is not fault resiliant. That is what it is a bad idea to write anything statically when deploying such infrastructure.

The project **Consul** has been used to achieve the feature. Consul is decentralized solution for service discovery based on two protocols. On the one hand, it is using

[1] Download page of Ubuntu 14.04 Cloud :http://cloud-images.ubuntu.com/releases/14.04/

a gossip protocol to manage the communication between nodes. This feature let **Consul** creates a decentralized cluster of servers. When a new node is available, it just needs to communicate with one node, whichever it is, to join the complete cluster and get access to the shared resources. On the second hand, the process is using a consensus algorithm to elect a leader node on the cluster, which has the responsability to keep the data consistent. Write operations have to be validated by the leader node, then spread to the rest of the servers. If the leader node crashes, another node is automatically elected by the others nodes.

Consul main usage is service discovery, so each node is regitering its running services to consul which will spread the information among the whole cluster. That is how services get to know each other.

The application is developped using the **Go** programming language, so the installation is trivial. To achieve the installation, whatever is the operating system, downloading the binary from the website http://www.consul.io/downloads.html and executing it enough. The configuration of each node service is done through a set of JSON files which have to be defined in **Consul** configuration directory.

4.2.3 Balancer agent

On each server which has to exectute the web applications, the installation of the balancer agent is required. It is a HTTP server written using python3. The source code can be found on GitHub [2]. The installation is straightforward.

```
git clone \
  https://github.com/Soulou/msc-thesis-container-balancer-agent
cd msc-thesis-container-balancer-agent
virtualenv -p /usr/bin/python3 .
source bin/activate
pip install -r requirements.txt
```

As the agent has to communicate with **Docker** is should be run as root:

[2] https://github.com/Soulou/msc-thesis-container-balancer-agent

```
sudo -E python agent.py
```

The server has two distinct roles. The first one is to execute instructions coming from a controller, the interface is an HTTP API. You can find its documentation in the Annexe A: HTTP API of the agent server

The other role of the agent is to achieve real time monitoring of the server itself and of each container running on it. Different threads starts in parallel with the HTTP Server. The reason why it is necessary to use separate threads is that at a given time it's not possible to get some relative data. This is how the different metrics are gathered by the agent.

For the entire server:

- CPU: The interface from the Linux Kernel to read the CPU usage is `/proc/stat`. When this virtual file is read, the kernel fills it with the current information about the CPU usage, the interruptions and the processes. However those data are cumulative. So each second the data are fetched, and compared to the CPU usage of the previous second. The data are un *User Hz*, this unit represent a tick in the user space, 100 of them are generated per second, so the value shown in this file are close to hundreths of second.

- Memory: This value is easier to access, the kernel provides `/proc/meminfo` which contains the real time data usage. There is no extra work to do in order to the clean pieces of information.

- Network I/O: `/proc/net/dev` contains all the information related to all the network interfaces of the server, in this file is displayed the amount of bytes and packets sent and received by each of them. The values are also cumulative, that's why the agent has to keep track of them. As a result when a request is done to get the system usage, the right data can be sent directly and it's not required to wait 1 second.

For the containers:

- CPU: The CPU usage of each container is accounted separately thanks to the *cpuacct* cgroup feature of the Linux Kernel. The communication from the userspace is done through a virtual file system located at `/sys/fs/cgroup`. As a result, for docker we can find the correct data at that path: `/sys/fs/cgroup/cpuacct/docker/:container_id/cpu.usage`. As precedently, the value is in *User Hz*, so the process to calculate the actual CPU usage is similar as the `/proc/stat` analysis.

- Memory: The cgroup *memory* manages the memory usage and limits per container, it is enough to read */sys/fs/cgroup/memory/docker/:container_id/memory.usag* to get the interesting piece of information.

- Network I/O: It is a bit more difficult to monitor the network usage of a container, as the resource management mecanisms is not part of the cgroup, it is another feature of the Linux Kernel called "network namespace". In order to get access to it, different steps are required.

  1. Find the PID of a process in the monitored container by looking in `/sys/fs/cgroup/:cgroup/docker/:container_id/tasks`
  2. Access the network namespace file located in `/proc/:pid/net/ns`
  3. Create a link of this namespace to `/var/run/netns/:container_id`
  4. Use IP command to get stats fron the desired namespace `ip netns exec :container_id netstat -i`

To sum up, three distinct threads are running in the Balancer agent, the HTTP server, the host system monitoring, and the containers monitoring. Those threads are used to be able to get accurate data instantly, otherwise 1 second should be waited to get interesting data.

3

[3] https://github.com/Soulou/msc-thesis-container-balancer-agent

4.2.4 Balancer controller

The second main brick of this infrastructure is the controller. The role of this software is to control all the different agents, to give them the instructions about which container to start and which container to stop. There is no need to configure this service particularly, the knowledge about the composition of the infrastructure is acquired through requests to the local **Consul** agent.

The running applications on the cluster are gathered by *service*. Each of them can contain a variable amount of containers hosted on the different agent.

Moreover, it updates dynamically the routing tables of the different services running on the infrastructure. If a service called "service-test" owns two containers, the incoming requests will be routed to both of thoses containers by following a round-robin algorithm. (C1 - C2 - C1 - C2 - ...).

Hipache[4] is used as a front HTTP working as reverse proxy, it is in charge of routing incoming requests to the different containers. Why **Hipache** has been used instead of a more classical server like **Apache**. The main reason is that **Hipache** is dynamically configurable thanks to different backends. The most common is the **redis** backend. **Redis** is a key-value store with really high performance as all the dataset stay in memory, and is asynchronously written to the disk. So when the controller sends request to start or stop a container, it also connects to a **redis** instance to update hipache configuration.

The controller also exposes a HTTP API which is described in the Annexe B HTTP API of the controller server Part of this API is only a proxy to the agents endpoint, in order to gather all the data to only one location, and on the other side, there are the active actions, linked to container scheduling on the infrastructure.

This component of the infrastructure is critical for the experiment detailed later in this work. The controller is charged to choose the nodes where the containers have to start. To select a strategy which will be used when a new controller has to start, it has to be specified as options:

[4] https://github.com/dotcloud/hipache

```
python3 controller.py --strategy random
```

In the previous example, the new containers will be placed randomly on the different vailable nodes. Others strategy are available:

- random: Choose randomly a node
- round-robin: Choose one node after the other, following a loop:

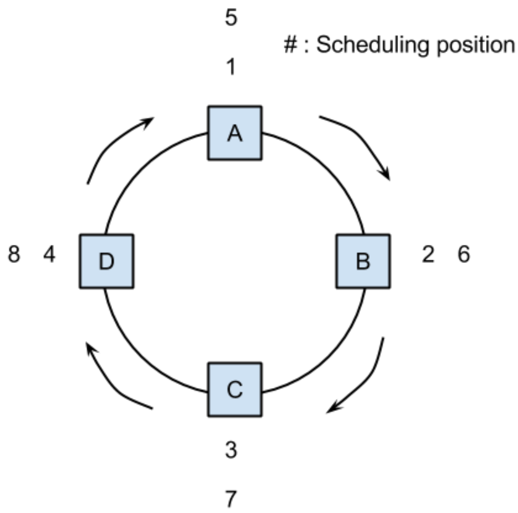

Figure 4.1: Scheme of a round-robin scheduling

5

4.2.5 Balancer client

The balancer client, is a command line tool to command the controller, it shows in a human-understandable way, the JSON results the controller and allows a user to execute requests easily. Its documentation can be found in the Annexe C: " Client command line tool documentation".

6

[5] https://github.com/Soulou/msc-thesis-container-balancer-controller
[6] https://github.com/Soulou/msc-thesis-container-balancer-client

4.2.6 Deployment

All those services and tools have to be deployed on the instances of the infrastructure. More and more configuration managers (CM) are released those days to answer the increasing needs to deploy applications in volatiles infrastructure, composed of temporary virtual machines. To setup all the different pieces of this infrastructure, **Ansible** has been used. There are two main models of CMs, thos centralized around a server, like **Chef Server** or **Puppet**, where the nodes synchronize themselves with this server. The devops do not connect directly to the nodes but give instructions to the central server. The second category are those who are run from a devops workstation, like **Chef Solo**, **Ansible**, **SaltStack**. The choice of **Ansible** has been done for it's ability to deploy nodes in parallel and by it's easiness of prototyping, its syntax is pretty simple.

All the mecanisms of deployment can be found on following *GitHub* page: `https://github.com/Soulou/msc-thesis-cloud-builder`. The script requires that **Openstack** credentials are stored in environment variables. (`OS_USERNAME, OS_TENANT_NAME, OS_PASSWORD, OS_AUTH_URL, OS_REGION_NAME`). Then it uses the API to instanciates VMs, and then deploy all the services on them using ansible [7].

[7] Ansible configuration: `https://github.com/Soulou/msc-thesis-cloud-builder/tree/master/ansible`

Chapter 5

Study of algorithms for Containers allocation and load balancing

5.1 Experimental applications

The experimental process has to measure the performance of a set of containers before and/or after using any resource allocation algorithm. First, we have to define which applications will be containerized for those operations. Those applications have to be deployed on all the agent servers and have to behave as standard web services. Whatever is the language or the framework used (except PHP), all the dependencies of the applications are loaded on application startup, so afterwards, there is no more file to read from the disk. As a result the disk input/output are really low for most of the applications and it this metric will not be measured in the following experiments.

The first service which has been used during the experiments is an application which is calculating N elements of the Fibonacci suite[1]. This weh application has only one endpoint: `GET /:n`, which returns the N^{th} items, of the Fibonacci

[1] **Docker** image: `soulou/msc-thesis-fibo-http-service`

sequence. This application only consumes CPU, the memory footprint is really negligible.

The second application which is used, has been designed to consume a precise amount of resource during a specified duration[2]. Its endpoint is `GET /:memory/:time` with the memory in megabytes and the time in milliseconds. For instance, `GET /10/100` will consume precisely 10MB for a duration of 0.1s. Thanks to this application, we can measure precisely how much should consume the application with a load of N requests per second of a certain amount of memory during a precise timelapse. The memory is allocated and initilized. As a ressult, even if we are requesting amount of memory, it also generates an important CPU load in the loops for memory application.

The source code of bother services can be found on GitHub:

- https://github.com/Soulou/msc-thesis-fibo-http-service
- https://github.com/Soulou/msc-thesis-memory-http-service

5.2 Load generation

An important question when measuring the performance against an infrastructure, in this case: web services, is the way to generation the requests. Do they have to reflect a realistic user load, is it possible to do it, is it pertinent? In this case, we are going to measure raw performance over the cluster, to gather data about the efficiency of a specific algorithm. When user load simulation algorithm is used, the benchmark is irregular, and finally getting information about the real impact of a given algorithm may be more difficult to do.

[2]**Docker** image: `soulou/msc-thesis/memory-http-service`

5.2.1 Tools

To generate web trafic, HTTP requests generators are required. Historically, the most used tool is part of the **Apache** web server tools, and is called Apache Bench *Command line name: ab*, released under the open-source *Apache Licence*. This utils is not really used anymore because it is single-threaded which is a very limitating parameter. Only one CPU core is used, it may not be enough to saturate a target and get a correct measure of the performance, if the measuring tool is limitating.

The tool which has been mostly used in the scope of this work is named **wrk**[3]. This is a benchmark tool able to send requests using a given number of connections, used in different parallel threads. (The opposite of `ab`, which sends all the request concurrently using one thread)

Usage example:

```
wrk -c 10 -t 2 -d 1m http://service1.thesis.dev
```

This example will send 10 requests concurrently during 1 minute using two threads. So we are sure that the targeted URL will receive a maximum of 10 request at a given time.

5.2.2 Different kinds of load

Combining **wrk** and the memory allocation HTTP service, we can simulate different kinds of load and different kinds of application.

In some cases, web applications are micro-services and their job is to do a small particular task. Commonly, requests are really quick and the treatment of each of them has a low memory footprint. However as those HTTP requests may be numerous, the overall processor and memory usage can be important.

[3] https://github.com/wg/wrk

5.3 Online bin packing algorithms

An online bin packing algorithm as defined in the first chapter of this work: Literature Review, is an algorithm which has to pack new items into bins without having the knowledge of what is already present in the bins. So bins have a remaining capacity, and each new items is known or partially known. In some cases, the new items are completely unknown.

In this infrastructure there are two different cases which can be distinguished:

- The new container is part of a new service: there is no information about the number of incoming requests, and what is the memory footprint of the new process
- The new container is from an existing service: in this case, the CPU consumption and memory usage can be estimated by looking the other containers of a similar service. For insntace, if two containers of `service1` are running using both 20% of CPU, it is probably true that the consumption of the new container will be capped at 20%

```
Round-Robin

Service1 Requests/sec:     16.71 16.69 16.91 16.74 16.67 16.48
Service2 Requests/sec:     22.92 23.76 23.16 23.87 23.31 23.20
Service3 Requests/sec:     27.68 28.30 28.13 28.69 27.18 28.83

Random

Service1 Requests/sec:     15.21 13.17 17.05 15.05 15.33
Service2 Requests/sec:     21.78 12.83 23.14 19.73 19.85
Service3 Requests/sec:     30.33 19.79 28.68 28.72 24.85
```

5.4 Offline bin packing algorithms

5.5 Results

Appendix A

HTTP API of the agent server

```
/*
 * Get current system info of the node itself (CPU, Memory, Net)
 */

// GET /status

// Code: 200 - OK
// Content-Type: application/json
{
  "cpus": {
    "cpu0": 44,
    "cpu1": 22,
  },
  "memory": 27440000,
  "network": {
    "eth0": {
      "tx": 98765,
      "rx": 12345
    }
  }
}

==================================================
```

```
/*
 * Get list of all running containers on the node
 */

// GET /containers

// Code: 200 - OK
// Content-Type: application/json
[
  {
    "Id": "0123456789abcdef",
    "Image": "soulou/msc-thesis-memory-http-service",
    "Ports": [
      {
        "PublicPort": 49127,
        "PrivatePort": 3000
      }
    ],
    "Names": [ "service1-1-837" ],
    "Created": 1723454345,
    "Status": "Up"
  },
  {
    ...
  }
]

==================================================

/*
 * Get information about a specific container
 */

// GET /container/:container_id

// Code: 404 - Container not found

// Code: 200 - OK
```

```
// Content-Type: application/json

// Docker JSON representation of a container:
// See: http://goo.gl/JrR6f6

==================================================

/*
 * Get information about the resource status of a container
 * (CPU, Memory, Net)
 */

// GET /container/:container_id/status

// Code: 404
//    - Container not found
//    - Data not ready (container launched for less than a second)

// Code 200 - OK
// Content-Type: application/json
{
  "cpu": 44,
  "free_memory": 123456,
  "memory": 2340354,
  "net": {
    "rx": 123456,
    "tx": 123564
  }
}

==================================================

/*
 * Create a new container
 * Params:
 *    image: Docker image to use
 *    service: Name of the service
 *    port (optional): Choose the given port instead
```

```
 *     of allocating one randomly
 */

// POST /containers

// Code 201 - Container created and started
// Content-Type: application/json

// Docker JSON representation of a container:
// See: http://goo.gl/JrR6f6

==================================================

/*
 * Stop and delete the container with the given ID.
 */

// DELETE /container/:container_id

// Code 204 - Container deleted, no content in response
```

Appendix B

HTTP API of the controller server

```
/*
 * Get the status of a given node, like the different
 * resources consumption and total
 */
// GET /node/:host/status

// Code 404 - Agent not found

// Code 200 - OK
// Content-Type: application/json
{
  "cpus": {
    "cpu0": 44,
    "cpu1": 22,
  },
  "free_memory": 12300000,
  "memory": 27440000,
  "network": {
    "eth0": {
      "tx": 98765,
```

```
      "rx": 12345
    }
  }
}

==================================================

/*
 * Return a list of containers running on a given host
 */
// GET /node/:host/containers

// Code 404 - Agent not found

// Code 200 - OK
// Content-Type: application/json
[
  {
    "Id": "0123456789abcdef",
    "Image": "soulou/msc-thesis-memory-http-service",
    "Ports": [
      {
        "PublicPort": 49127,
        "PrivatePort": 3000
      }
    ],
    "Names": [ "service1-1-837" ],
    "Created": 1723454345,
    "Status": "Up"
  },
  {
    ...
  }
]

==================================================

/*
```

```
 * Return the status of all the nodes of the cluster.
 */
// GET /nodes/status

// Code 200 - OK
// Content-Type: application/json
{
  "192.169.0.1" : {
    "cpus": {
      "cpu0": 44,
      "cpu1": 22,
    },
    "memory": 27440000,
    "network": {
      "eth0": {
        "tx": 98765,
        "rx": 12345
      }
    }
  },
  "192.168.0.2" : ...
}

==================================================

/*
 * This endpoint aims at executing an algorithm of load balancing.
 * The name of the algorithm has to be given in the strategy parameter
 * The containers and the agents nodes resource usage is gathered then normal
 * then the algorithm is executed.
 * As a result of the problem solving, a set of migration is applied to move
 * the containers according to the algorithm solution.
 */
// POST /balance
// Params:
//   strategy: "stillwell_current"

// Code 200 - OK
```

```
{
  "items": [[0.2, 0.3, 0], [...]],
  "bins": [[1.0, 2.0, 1.0], [...]],
  "problem": {
    "algo": "stillwell_current",
    "mapping": [0,0,0,1,2,3,4],
  },
  "migrations": [
    {
      "Service": "service-1",
      "Started": {
        "host": "182.168.0.2",
        "id": "0123456789abcdef"
      },
      "Stopped": {
        "host": "182.168.0.2",
        "id": "0123456789abcdef"
      }
    },
    {
    ...
    }
  ]
}

==================================================

/*
 * Return a list of all the containers running on the cluster
 * hosted by each alive node.
 */
// GET /containers

// Code: 200 - OK
// Content-Type: application/json
[
  {
    "Host": "192.168.1.2",
```

```
    "Id": "0123456789abcdef",
    "Image": "soulou/msc-thesis-memory-http-service",
    "Ports": [
      {
        "PublicPort": 49127,
        "PrivatePort": 3000
      }
    ],
    "Names": [ "service1-1-837" ],
    "Created": 1723454345,
    "Status": "Up"
  },
  {
    ...
  }
]

==================================================

/*
 * Create a new container
 */

// POST /containers

// Code: 201 - Container started

// Docker JSON representation of a container:
// With an additional field: "Host"
// See: http://goo.gl/JrR6f6

==================================================

/*
 * Stop and delete a container on a given host
 */

// DELETE /container/:host/:container_id
```

```
// Code: 404 - Host or Container not found
// Code: 204 - Container stopped

==================================================

/*
 * Migrate a container, to the most available node
 */

// POST /container/:host/:container_id/migrate

// Code 200 - Container migrated
{
  "Service": "service-1",
  "Started": {
    "host": "182.168.0.2",
    "id": "0123456789abcdef"
  },
  "Stopped": {
    "host": "182.168.0.2",
    "id": "0123456789abcdef"
  }
}
```

Appendix C

Client command line tool documentation

- Start a container

```
./client.py start [--image <image name>] <service name>

##
# The Image parameter correspond to the docker image which should be
# run on the agent, as at that time, the identify of this agent is not
# known, take care that all the agents have the requested docker image.
# By default a fibonacci web service is executed.
#
# The service name is the name of the group of containers that the
# newly created process will be appended, if the group doesn't exist,
# it is created.
#
# Example
#

./client.py start --image soulou/memory-http-service service1

-> Host: 192.168.114.13
-> ID: e6bbb8853678b67882ea6919c559d80a20
-> Service: service1
```

```
-> Image: soulou/msc-thesis-memory-http-service
-> Port: 49342
```

- Stop a container

```
./client.py stop --service <service_name> | --all <host> | <host> <ID>

##
# Three different options are available to stop a container.
# * All the container of a service ban be stopped
# * All the container of a precise server may be stopped
# * A precise container can be shutdown by giving its host and its ID
#
# Example
#

./client.py stop --service <service_name>

Container f6fe748fbf from 192.168.114.14 has been stopped
Container e7dc0f68f3 from 192.168.114.17 has been stopped
Container 6a28e4b2ab from 192.168.114.15 has been stopped
```

- Get status of the infrastructure

```
./client.py status

##
# Give the information about all the running containers
#
# Output example
#

+----------------+-------+-----------+
| Node           |  Port |  Service  |
+----------------+-------+-----------+
| 192.168.114.14 | 49196 |  service3 |
| 192.168.114.14 | 49195 |  service3 |
| 192.168.114.13 | 49342 |  service1 |
+----------------+-------+-----------+
```

```
+--------------------------------------+
|                Image                 |
+--------------------------------------+
| soulou/msc-thesis-memory-http-service |
| soulou/msc-thesis-memory-http-service |
| soulou/msc-thesis-memory-http-service |
+--------------------------------------+

+----------------+---------------------+
|       ID       |     Created At      |
+----------------+---------------------+
| 3c9345c19e00d21 | 2014-07-28 10:51:34 |
| 9a2e2b22448f766 | 2014-07-28 10:51:33 |
| e6bbb8853678b67 | 2014-07-29 15:52:15 |
+----------------+---------------------+
```

- Migrate an application to another node

```
./client.py migrate <host> <id>

##
# This operation asks the controller to migrate a container
# to another node
#
# Example
#

./client.py migrate 192.168.1.3

Container 192.168.1.3 - 9a2e2b2244 -> 192.168.1.18 - 406d5ce46d
```

- Get node or nodes status

```
./client.py node_status <host>
./client.py nodes_status

##
# Get the system metrics for on particular node or all the nodes
```

```
#

./client.py nodes_status
[ '192.168.114.19': {'cpus': {'cpu0': 0, 'cpu1': 0},
                     'free_memory': 3736440,
                     'memory': 4048292,
                     'nb_containers': 0,
                     'net': {'docker0': {'rx': 0, 'tx': 0},
                             'eth0': {'rx': 3728, 'tx': 1858},
                             'eth1': {'rx': 0, 'tx': 0},
                             'lo': {'rx': 0, 'tx': 0},
                             'veth2074': {'rx': 0, 'tx': 0},
                             'vetha4f2': {'rx': 913, 'tx': 1796},
                             'vethc390': {'rx': 0, 'tx': 0},
                             'vethd466': {'rx': 633, 'tx': 964},
                             'vethec34': {'rx': 0, 'tx': 0}}}}
]
```

Bibliography

Eugen Feller, Louis Rilling, and Christine Morin. Energy-aware ant colony based workload placement in clouds. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, 2011.

David Wilcox, Andrew McNabb, Kevin Seppi, and Kelly Flanagan. Probabilistic virtual machine assignment. In *The First International Conference on Cloud Computing, GRIDs, and Virtualization*, 2010a.

Peter Mell and Timothy Grance. The NIST definition of cloud computing. *Recommandation of National Institute of Standards and Technology*, Spetember 2011.

openstack. official website for the openstack project. http://www.openstack.org/.

Barham Paul, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualisation. In *19th ACM Symposium on Operating Systems Principles*, 2003.

Clark Christopher, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*, 2005.

Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *EuroSys European Conference on Computer Systems*, 2007.

Pawel Pacana Arkency. Microservices - 72 resources. http://blog.arkency.com/2014/07/microservices-72-resources/.

Valentina Salapura. Cloud computing: Virtualization and resiliency for data center computing. In *30th International Conference of Computer Design*, 2012.

Thomas Setzer and Alexander Stage. Decision support for virtual machine reassignments in enterprise data centers. In *Network Operations and Management Symposium*, 2010.

Weijia Song, Zhen Xiao, Qi Chen, and Haipeng Luo. An adaptive resource provisioning for the cloud using online bin packing. *Transactions on Computers*, 2013a.

Kurt Vanmechelen Ruben Van den Bossche and Jan Broeckhov. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *3rd International Conference on Cloud Computing*, 2010.

Mark Stillwell, Frédéric Vivien, and Henri Casanova. Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. In *Parallel & Distributed Processing Symposium*, 2012.

Neal E. Young. Randomized rounding without solving the linear program. In *6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.

David Stifler Johnson. *Near-Optimal Bin Packing Algorithms*. PhD thesis, Massachusetts Institute of Technology, 1967.

G. Gambosi, A. Postiglione, and M. Talamo. Algorithms for the relaxed online bin-packing model. *SIAM Journal on Computing*, 30(5), 2000.

Weijia Song, Zhen Xiao, Qi Chen, and Haipeng Luo. Adaptive resource provisioning for the cloud using online bin packing. In *Transactions on Computers*, 2013b.

Lskrao Chimakurthi and Madhu Kumar SD. Power efficient resource allocation for clouds using ant colony framework. *CoRR*, 2011.

Linan Zhu, Qingshui Li, and Lingna He. Study on cloud computing resource scheduling strategy based on the ant colony optimization algorithm. *IJCSI International Journal of Computer Science*, 9(5), 2012.

David Wilcox, Andrew McNabb, and Kevin Seppi. Solving virtual machine packing with a reordering grouping genetic algorithm. In *IEEE Congress on Evolutionary Computation (CEC)*, 2010b.

Kimish Patel, Murali Annavaram, and Massoud Pedram. Nfra: Generalized network flow-based resource allocation for hosting centers. *IEEE Transactions on Computers*, 62(9), 2013.

INRIA. Simgrid official website. http://simgrid.gforge.inria.fr/.

Melbourne University. Project page of cloudsim. https://code.google.com/p/cloudsim/.

ROADEF members. Roadef contest homepage. http://challenge.roadef.org/.

Ramon Lopes, Vinicius W.C. Morais, Thiago F. Noronha, and Vitor A.A. Souza. Heuristics and matheuristics for a real-life machine reassignment problem. *International Transactions in Operational Research.*

Renaud Masson, Thibaut Vidal, Julien Michallet, Puca Huachi Vaz Penna and-Vinicius Petrucci, Anand Subramanian, and Hugues Dubedout. An iterated local search heuristic for multi-capacity bin packing andmachine reassignment problem. *Expert Syst. Appl.*, 40(30), 2013.

Brian Kell and Willem-Jan van Hoeve. An mdd approach to multidimensional bin packing. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems Lecture Notes in Computer Science*, 7874, 2013.

CRIU. Checkpoint/restore in userspace.

12 Factors. The twelve factors.

Louis Colombus. Roundup of cloud computing forecasts update, 2013. http://www.forbes.com/sites/louiscolumbus/2013/11/16/roundup-of-cloud-computing-forecasts-update-2013/, November 2013.

Joseph Wun-Tat, Chan Tak-Wah Lam, and Prudence W.H. Wong. Dynamic bin packing of unit fractions items. *Theoretical Computer Science*, 409(8), 2008.