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

25 juillet 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

25 juillet 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 Licence. The integrality of the examples are publicly available on GitHub https://github.com/Soulou

Table des matières

Table des figures

Liste des tableaux

Abbreviations

VM	**V**irtual **M**achine
IaaS	**I**nfrastructure **a**s **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

Introduction

An interesting definition of the Cloud Computing has been written by the National Institute of Standards and Technology [**?**] :

> "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 **?**, 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 **?** 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 **?**. 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, **?** 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 **?**. 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.

Chapitre 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 [**?**]. 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 [**?**]. 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* [**?**], 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, **?** 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 **?**, 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 **?** 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 **?** :

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 **?** 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 **?**. 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 **?**, 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**?**, 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 **?**, 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 **?**, **?** and **?**, the ant colony algorithms are studied. As we can see in the following graph :

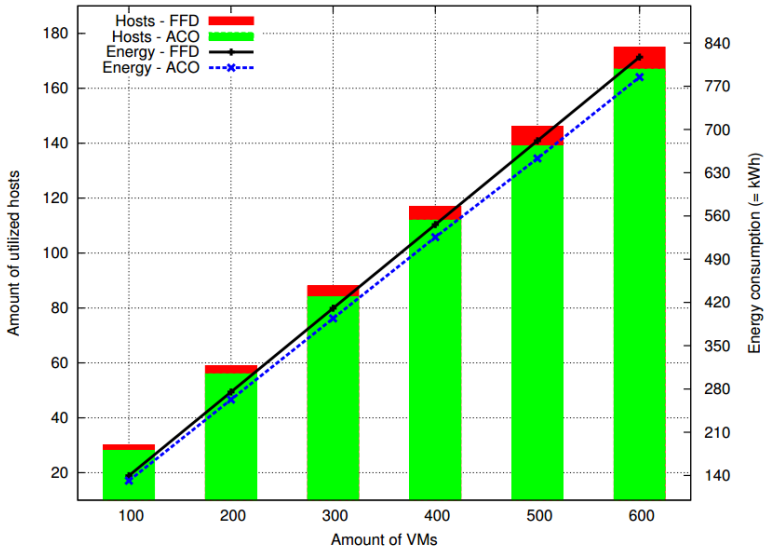

FIGURE 1.1 – Comparison between First Fit Decreased and Ant Colony algorithms in **?**

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 **?**

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. **?** and **?** focused on them to solve the virtual machines assignment problem. In the work of David Wilcox et al.**?**, simulations are comparing GA with *-Fit algorithms.

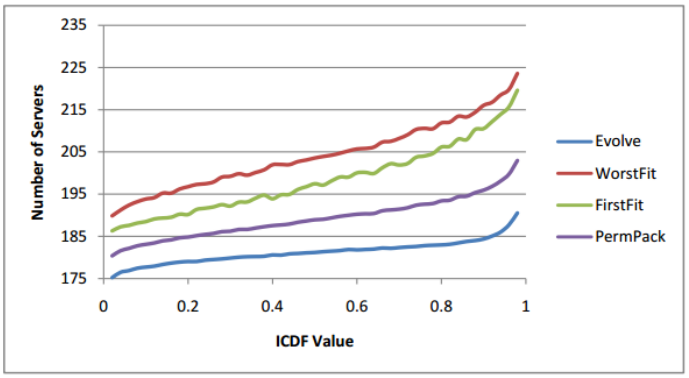

Fig. 7. A comparison of the the number of servers found.

FIGURE 1.4 – Results of simulations using a genetic algorithm**?**

On the following graphs, ICDF stands for "inverse cumulative distribution function" also known as "quantile function", the authors use it to represent the load :

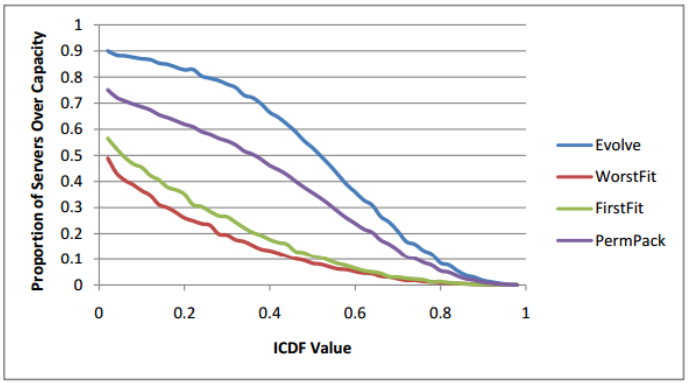

Fig. 6. A comparison of the proportion of servers over capacity.

FIGURE 1.3 – Results of simulations using a genetic algorithm**?**

"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".

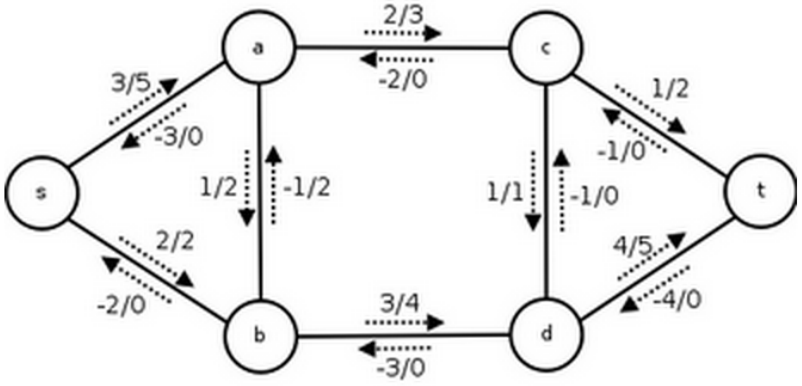

FIGURE 1.5 – Example of network flow directed graph

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 datacenter**?**, considering an heterogeneous environment as in **?**. 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 SimGrid**?** or CloudSim**?** 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)**?**. 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. Souza**?**. They based their work on linear programming. However in **?** and **?**, 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.

Chapitre 2

Container load balancing in cloud environment

2.1 Containers - Operating System-level Virtualization

2.1.1 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

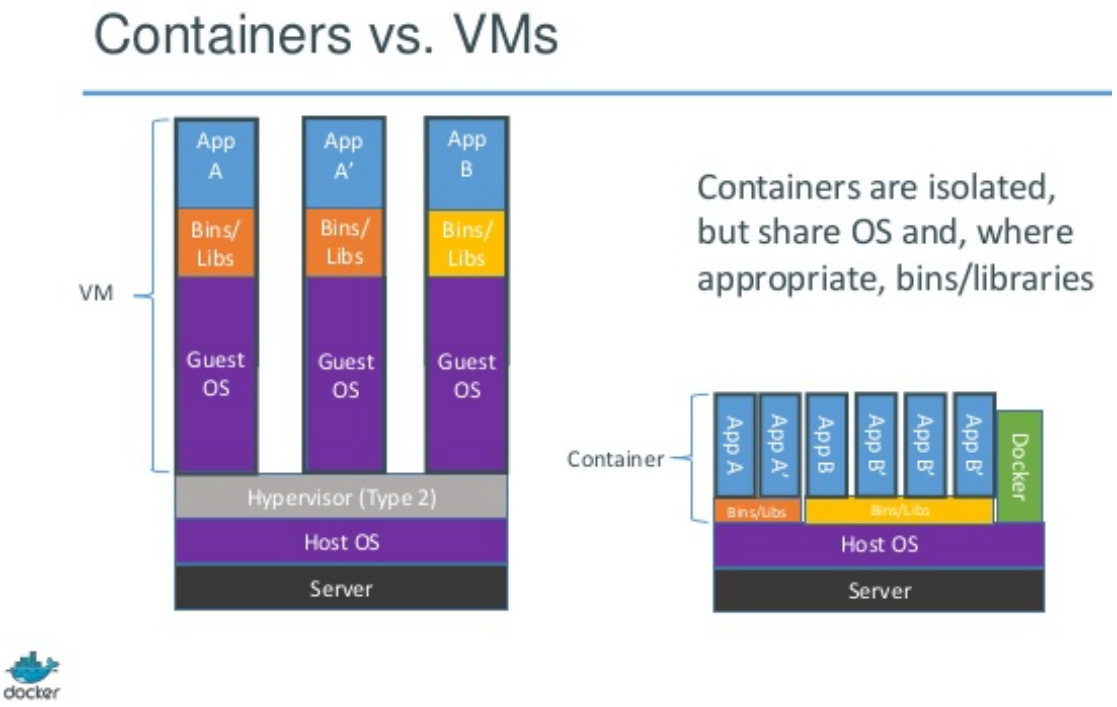

FIGURE 2.1 – Structural difference between containers and VMs

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.

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.

2.1.2 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).

2.1.3 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 **?** 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.

2.1.4 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 **?** 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.

2.1.5 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.

2.1.6 Operation on containers

2.1.6.1 Load balancing

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

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

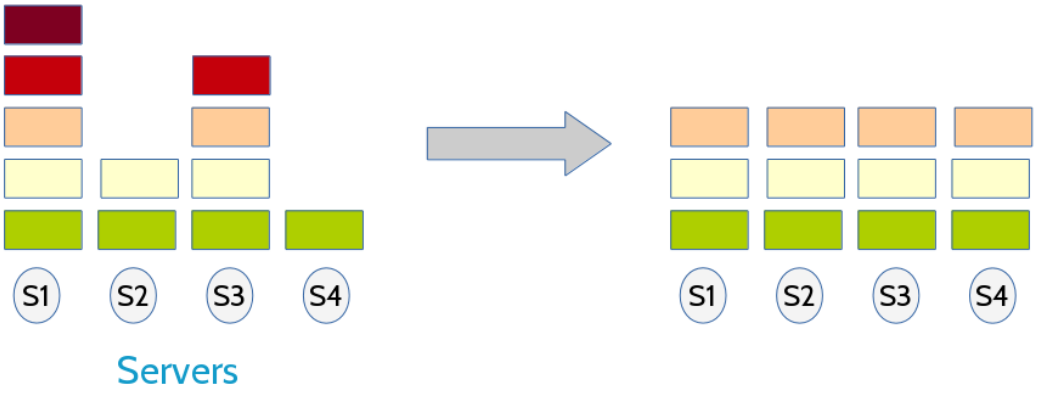

FIGURE 2.2 – Schema of a load balancing process

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.

2.1.6.2 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

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.

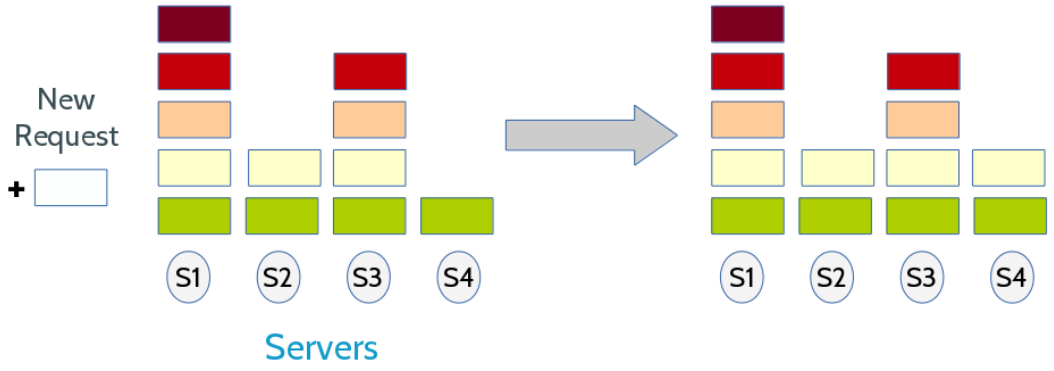

FIGURE 2.3 – Schema of a resource allocation process

Chapitre 3

The experiments

3.1 Study of the ability to isolate containers CPU usage using Linux control groups

3.1.1 Goal of the experiment

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 the CPU resources among the different containers.

3.1.2 Metrics

3.1.2.1 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%.

3.1.3 Setup

3.1.3.1 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

3.1.3.2 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**

3.1.4 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.

3.1.5 Results

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

3.1.6 On the laptop

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.

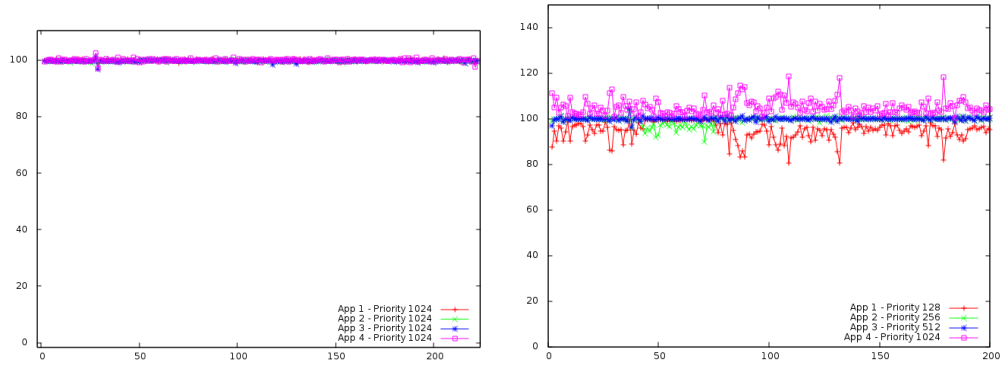

FIGURE 3.1 – 4 Processes with equal[1] and different[2] CPU shares

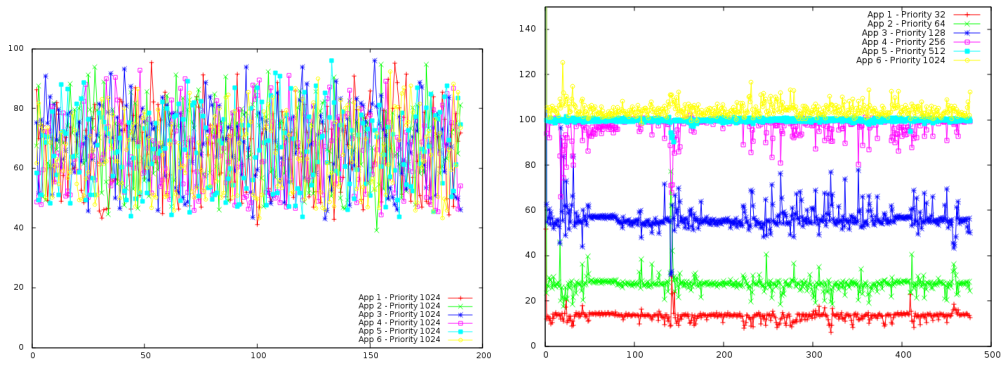

FIGURE 3.2 – 6 Processes with equal[1] and different[2] CPU shares

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 $\approx$ 15% of CPU, then the one with 64 shares has $\approx$ 30% of CPU consumption, and then, the third one has $\approx$ 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)

3.1.7 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.

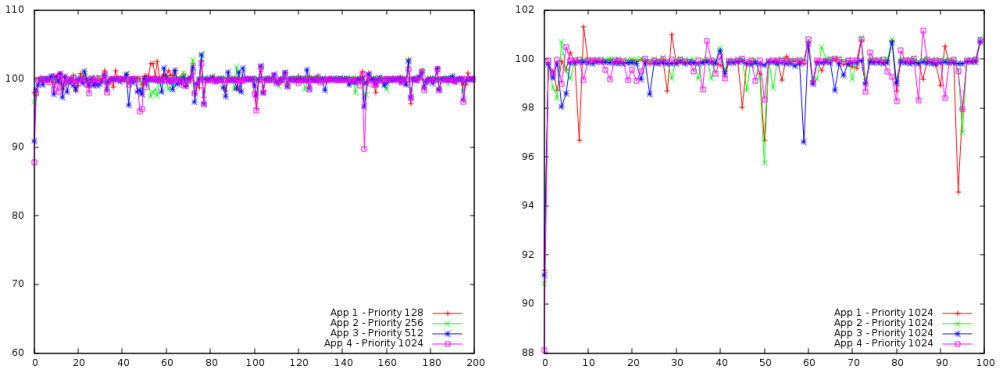

FIGURE 3.3 – 4 Processes with equal[1] and different[2] CPU shares

For

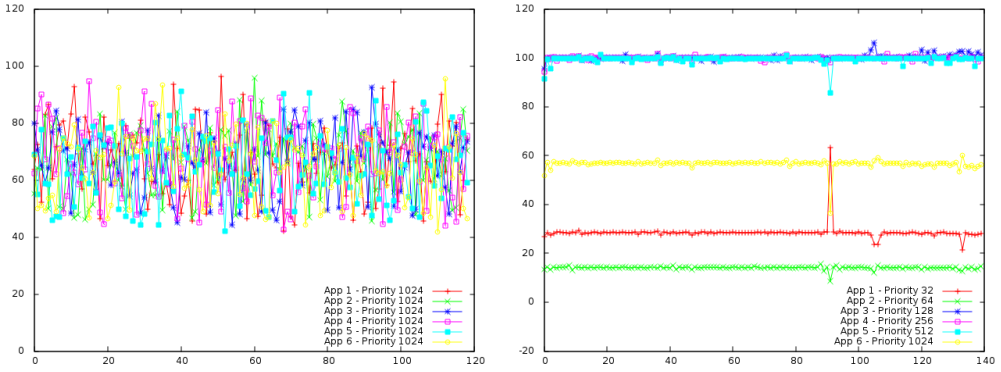

FIGURE 3.4 – 6 Processes with equal[1] and different[2] CPU shares

Ipsum lorem dolor sit amet

3.1.7.1 Comments on the results

3.1.8 Conclusion on the experiment

Annexe A

Appendix Title Here

Write your Appendix content here.