Proposed scheduling algorithm for deployment of fail-safe cloud services

Simon Kollberg
Abstract

The cloud is a fairly new concept in computer science, although almost everyone has been or is in contact with it on a regular basis. As more and more systems and applications are migrating from the desktop and into the cloud, keeping a high availability in cloud services is becoming increasingly important. Seeing that the cloud users are dependant on the cloud services being online and accessible via the Internet, creating fault tolerant cloud systems is key to keeping the cloud stable and trustworthy.

This thesis researches different aspects of the cloud infrastructure such as virtualization which detaches the services from the physical hardware, scheduling algorithms that decides the mapping of virtual machines onto physical machines and fault tolerance in the context of availability in the cloud. It then proposes a new scheduling algorithm with the purpose of deploying virtual machines in a way that active-passive replication can be sustained.
Acknowledgements

I want to thank Petter Svärd for all the valuable insights and a continued support, answering questions and helping me find good resources. I would also like to give thanks to Daniel Espling for helping me getting started on the project by setting up and introducing me to the OpenStack environment. I’m also very grateful for all the help from my friends and family for providing support and feedback during the whole process.
1 Introduction

What prompted the work behind this thesis is a project called ORBIT. Umeå University, through the ORBIT project, strives to improve a system called OpenStack. OpenStack is an open source cloud computing platform which involves many different collaborating partners [1]. One of the ways that the ORBIT project is improving OpenStack is to implementing fault tolerance (FT). The FT is brought about via replication of virtual machines (VMs), whereby a redundant secondary machine can be used to recover from hardware failures of a primary VM. This is achieved by mirroring I/O, computation and resource consolidation to get equal results on both VM pairs, also called active-passive replication. [2, 3]

Currently the scheduling in OpenStack does not take this kind of FT into consideration when deploying new VMs. When new VMs are deployed in OpenStack the backup VM can, with the current setup, theoretically be placed on the same physical machine (PM) which would render the mirroring useless. That is why a new scheduling algorithm, which is better suited for the described FT mechanism, is requested.

To be able to describe the work done this thesis is going to go through and explain, among others, the following concepts; cloud computing (CC), VM, scheduling and FT. Research into which aspects to consider when creating a scheduling algorithm for cloud services is also needed to get a good result when drafting the new proposed algorithm.

2 The cloud and virtual machines

When talking about the cloud in the context of computer science the term often refers to large distributed systems or clusters of computers collectively used for computation and storage, with the purpose of being utilized by non-local clients. CC is defined as "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." [4]. A cloud typically provides one of three service models, Infrastructure as a Service (IaaS) which offers an application environment through hardware and software, Platform as a Service (PaaS) that supplies an environment for application deployment, testing etc. and Software as a Service (SaaS) which is a software with a specific objective which is accessed and executed remotely. [5]

The term grid computing often comes up when talking about CC. Grid computing and CC are sometimes mistakenly used for one another though these terms are not completely synonymous. The grid is still a widely used distributed system which CC have evolved out of. Even though CC is built on top of the grid it is much more oriented towards economy-based services while the grid can be seen more as a raw storage and computing system. Computing in the grid compared to the cloud is also very different. The grid usually works with specific batches of jobs queued and executed for a certain amount of time. Resources in the grid are, at
execution, dedicated by the job-batch for the entire run-time. In the cloud, the usage of resources is dependant on the amount of users currently in the system since the cloud is mostly utilized by user-driven services.

Clouds are commonly set up through a VM environment where resources can be dynamically allocated as needed. A VM is a software that simulates a machine (i.e. a computer) that can execute programs either by emulating a whole operating system or emulating a process by running a single application [6]. The VMs becomes an abstraction on top of the hardware to create a unified pool of resources which can be deployed, suspended, migrated etc. [7]. This primarily correlates with the cloud when provided as an IaaS. With virtualization it is possible to overcome problems with availability, that are quite common when only utilizing physical machines (PMs) [8]. Availability failures caused by hardware crashes or heavy workloads can be avoided through migration of applications to different VMs since they are not bound to a specific computer.

In the context of an IaaS, a cloud service is a bundle of VMs running a single, or a collection of components forming an application [9]. A cloud service also includes a document describing the service and its constraints, called a service manifest. For example, a web application can consist of multiple components such as a database and HTTP server. In this model, just as threads does in a native computer program, virtualization creates the illusion of multiple applications running simulatiously for all its users in the cloud.

3 Scheduling in the cloud

Scheduling, as it pertains to computer science, is the method which distributes the workload of a system. However, within the context of this thesis, scheduling is referred to as the process of determining a mapping between VMs and PMs when resources are requested for cloud services. When allocating and configuring resources in the cloud, scheduling algorithms are used to get an optimal performance from the hardware. The purpose of the scheduling algorithm is to deploy the VMs at carefully chosen locations among the available PMs. This is done by considering different parameters such as CPU load, memory, storage, bandwidth usage, latency and availability. [10]

Scheduling algorithms are usually categorized into static or dynamic scheduling [11]. While a static algorithm can be more reliable and predictable a dynamic scheduling algorithm can adapt itself to create a more optimal placement. Dynamic algorithms are using the current state of the data center to decide the mapping while a static algorithm’s execution is not determined on any real-time information [12]. One needs to consider the trade-offs of choosing one type of algorithm over another when deciding a scheduling strategy.

Another thing to consider is the power consumption of data centers running clouds. The demand for large-scale computing infrastructure is increasing, therefore it is necessary to consider the social impacts such large data centers have. One major impact being the negative environmental effects. Proper allocation of the resources can reduce the environmental harm and at the same time reduce costs. [13]
While it is good not to overuse resources by, for example, deploying VMs on an excessive amount of PMs, distributing the load among the PMs is key to achieve a good result from the cloud. Without load balancing there is a chance that PMs get overloaded with too big of a workload on single machines. At the same time load balancing can be used to sustain big peaks in the workload as well. [14]

There are a lot of other possible optimizations including predicting future events [15] and rearranging of resources [7]. This thesis will mainly focus on usage of resources, load balancing and VM localization, however, future work of this kind would most likely improve the resulting scheduling.

4 Fault Tolerance

Making sure that cloud services are available and runs correctly is very important and a growing issue now that more applications are moved from being local and are instead running in the cloud.

When talking about Fault Tolerance (FT) in computer science it covers how to achieve robustness and dependability in a system. There are several techniques to create FT in the cloud. Among these are checkpointing, job migration and replication. [16] In this thesis, replication will be the main focus regarding FT.

One way of achieving FT in the cloud is by introducing redundant VMs that can recover from failures [17]. How well the resulting scheduling of these redundant VMs is can be affected by various aspects, one very impactful variable being network-awareness [18]. Minimizing the traffic of the communication between VMs by placing them wisely inside the network is important for scalability [19]. However, when introducing FT it is also significant to consider network failures. To achieve high availability and increase FT in cloud services with redundant VMs, the scheduler should consider placing these redundant VMs on different hardware in the network, such as different switches.

It is possible to achieve an even more robust FT through a broader geographically dispersed placement, where cloud services could withstand circumstances such as power-outages or natural disasters [20]. However, this thesis focuses on a single data-center.

5 Constraints

To ensure the correctness of a scheduling algorithm which is used in active-passive replication some constraints needs to be constructed.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_{k,i} )</td>
<td>Deployed primary VM ( k ) on PM ( i ).</td>
</tr>
<tr>
<td>( s_{k,j} )</td>
<td>Deployed secondary VM ( k ) on PM ( j ).</td>
</tr>
</tbody>
</table>
Given these symbols, a correct algorithm satisfying the FT replication mechanism described have the following constraints.

1. \( i \neq j \), two mirrored VMs must not be deployed on the same PM.

2. \( \forall p_k, \exists s_{k,j} \), both the primary VM and the secondary VM must be deployed else neither should be deployed.

### 6 Choice of algorithm

In this section different scheduling algorithms are described, analysed and compared with the background and constraints in mind. This is to find the best suited algorithm to use when implementing an active-passive replication. The algorithms currently do not support all of the constraints in section 5 and, therefore, will not fulfill them entirely. With this in mind, it is still possible to find a well suited algorithm to be modified later on.

The focused algorithms in this thesis will be based on algorithms used in different CC systems similar to OpenStack. In the CC system Eucalyptus two scheduling algorithms are used, the Round Robin and Greedy algorithm. OpenNebula is another CC system which uses the Match-Making algorithm in its scheduler. [21, 11]

Fittingly the Round Robin algorithm is a static algorithm while the Greedy and the Match-Making algorithm are dynamic algorithms and will therefore give the opportunity to weigh the advantages and disadvantages of the two scheduling categories.

#### 6.1 Round Robin

The Round Robin algorithm works by distributing the workload equally over all the PMs in the data center by allocating resources on a new PM every time. The algorithm structures the PMs like a circular linked list. When a request is issued, the scheduler chooses a PM for allocation and remembers which one it chose. The scheduler will then place the next requested VM instance on the neighbouring PM from the previous remembered PM. This will result in all the PMs getting an equal load balance. [11, 22]

This algorithm would work well with constraint 1 in section 5 in that it would always work to place the primary and secondary VMs on different PMs. Although, there are some negative aspects of this algorithm. When used in the context of load balancing a data center, the Round Robin algorithm will create a high rate of power consumption. The algorithm would work to use the whole data center even though many tasks would not require all PMs available.

Since the Round Robin algorithm is static, it would have a very fast deployment time, this because it would never need to consider the current state of the data center except for when there is no more room on the PM. But seeing that the cloud is not supposed to run out of resources, all the PMs would rarely be found to be maxed out.
6.2 Greedy (First-Fit)

This algorithm iterates through the PMs of a data center until it finds an acceptable match. Unlike the Round Robin algorithm, the Greedy algorithm does not follow any particular order. The Greedy algorithm will choose the first PM that fits the requirements and has enough resources to allocate the VM. Even though there might be better solution for the deployment, the Greedy algorithm does not take this into consideration. From this, it is probable that the results from this algorithm would not be optimal; however, the execution time of the deployment would be very efficient. [10, 22]

Seeing that the Greedy algorithm would place all of the requested VM instances on the same PM, if it is possible, the latency between these VMs will be very quick. But it would create a bad load balance for the same reason.

6.3 Match-Making (Best-Fit)

What the Match-Making algorithm does is similar to the Greedy algorithm in that the Match-Making algorithm filters out the PMs that do not meet the requirements for the requested VM allocation. However, instead of choosing the first PM that fits the criteria the Match-Making algorithm does a more thorough search by weighing how well the PMs are suited for the placement to get the best possible mapping of VM to PM. This is done by considering the PMs’ different resources; such as CPU, memory and network. [11, 10, 22]

Given that the Match-Making algorithm is both dynamic and tries to find the best fit, it will not be as fast as other algorithms when it comes to deploying new VMs. However, compared to a task scheduler in an operating system, scheduling in the cloud is not going to be as frequent and therefore the cloud will not be affected much by a bit of extra latency at deployment. Getting a good placement is, on the other hand, much more important, and can greatly impact the efficiency of a cloud.

6.4 Summary

To summarize the analysis, the table below displays the advantages and disadvantages of utilizing the algorithms discussed as schedulers. The selected algorithm will be the one that best fits the needs, by looking at the advantages and disadvantages and suitability for the problem. Since the algorithm that is chosen is going to be modified to fit the FT constraints in section 5 the current FT capabilities of the algorithms will not be considered in the selection process.
### Table 2 Summary of advantages and disadvantages in scheduling algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round Robin</td>
<td>Deployment time, Load balancing</td>
<td>Cost, Power consumption</td>
</tr>
<tr>
<td>Greedy (First-Fit)</td>
<td>Deployment time, Cost, Power consumption, Latency</td>
<td>Load balancing, Utilization</td>
</tr>
<tr>
<td>Match-Making (Best-Fit)</td>
<td>Load balancing, Cost, Power consumption, Latency</td>
<td>Deployment time</td>
</tr>
</tbody>
</table>

It should be noted that the advantages of the Match-Making algorithm are dependent on a good ranking algorithm. If the ranking is not selecting PMs wisely, the mapping could be the worst of all three. A ranking algorithm which considers many important tasks necessary for good placement is therefore assumed. The Match-Making algorithm used in OpenNebula is also closely related to the filter scheduler in OpenStack and, therefore, is particularly suitable for this task [23].

With this being said, the Match-Making algorithm has been chosen as the scheduling algorithm best suited for this problem, and will therefore be modified and used to create the proposed algorithm in section 7.

### 7 The proposed algorithm

In this section the new scheduling algorithm is described, making note of which necessary features are currently not in the Match-Making algorithm and how it is modified to comply with the constraints in section 5.

Two ways to solve constraint 1 in section 5 would be either keeping which PM the primary VM is mapped to in memory, or check which VMs are placed on the PM the secondary VM is mapped to. However, here another problem is introduced. If the secondary VM fails to deploy on the selected PM due to the primary VM already being placed there and no more resources are available, the primary VM will be missing its mirror. This would violate constraint 2 in section 5.

The Match-Making algorithm proved to be already very fitting for the task. Since it works by filtering out multiple mapping-candidates and not just one, the Match-Making algorithm ensures that enough resources exist for deployment as long as more than one possible deployment candidate exists. From this, both the primary and the secondary VMs can be deployed on separate PMs. The secondary VM is also deployed right after the primary VM before any other VMs are deployed, and by that makes sure that no other VM can occupy any resources needed for the mirrored pair. This works correctly because scheduling requests are done sequentially in OpenStack.
and are therefore all queued and deployed one after the other.

The resulting PMs could here be selected very primitively and still meet the constraints in section 5. For example a valid mapping would be the primary VM mapped to the most fitting PM, and the secondary PM mapped to the second best match for the primary VM. However, a more sophisticated modification to the ranking mechanism will be implemented where the secondary PM is chosen. When finding the best suited PM for the secondary VM, the network topologies of the data center will be considered. A trade-off between latency and FT will be introduced in the ranking where communication between the VM pairs will be considered, but also having the mirrored VM pair not being dependant on the same network infrastructure as much as possible. For example, by deploying the primary and secondary VMs on PMs connected to different network switches.

Algorithm 1

**Input:** requirements, rank, hostsList

**Output:** primaryHost, secondaryHost

for each host in hostsList do

if host meets requirements then

candidates[] ← host

end if

end for

if length(candidates) < 2 then

return fail

else

pairs ← combine(candidates)

sortedPairs ← sortByRank(pairs, rank)

primaryHost, secondaryHost ← sortedPairs[0]

return primaryHost, secondaryHost

end if

Algorithm 1 shows the pseudocode of the new modified Match-Making scheduling algorithm. Original Match-Making algorithm pseudocode taken from [22]. One big difference from the original algorithm is that all the PM candidates are combined into pairs so that the ranking mechanism can consider both the primary and the secondary candidate together instead of them being ranked separately. This way it’s possible to rank the pair on how well they work with each other as well as how good of a fit they are for deployment of individual VMs.

It is worth mentioning that resources like CPU, RAM and network bandwidth can vary in different clouds. The resource capacity of the PMs is therefore dynamic and will be determined at the moment of deployment. OpenStack also allows overcommitting which means that deployments can be successful even though the PMs resources are maxed out. The allowed number of VMs will therefore also be determined by how much overcommitting is allowed [24].
8 Conclusion

In this thesis the mechanisms for deploying fail-safe cloud services by VM scheduling in a data center have been described and explained. The aspects that need to be considered when mapping VMs against PMs in a data center include load-balancing, power-consumption and availability of resources. This thesis explains how to achieve FT using redundant VMs, which can work as a backup replacement for failures. It also explains how to place these redundant VMs wisely. Finally, fulfilling the purpose of the thesis described in the introduction, a new scheduling algorithm has been proposed to be implemented in cloud computing platforms. The new algorithm is suited for an active-passive VM replication technique which fulfills the constraints in section 5.

9 Future work

The cloud has many different aspects that can be researched in more depth. The resulting scheduling algorithm of this thesis could be developed further in various ways. One way to improve the scheduling is through VM migration; the mechanism that reorganizes deployed VMs by mapping them onto different PMs, creating a more efficient load balancing while decreasing power consumption and cost. Another way to improve the scheduling of VMs is by implementing history awareness that keeps deployments in memory and use it to estimate a better mapping for future deployments. Genetic algorithms are also something that could be researched more to improve the results. Genetic algorithms adapt themselves by evaluating the efficiency from every deployment and gathers information about which were more beneficial than others, in a fashion similar to that of natural selection.

Further research into the OpenStack system will be done. The implementation of the scheduling algorithm will depend on what data can currently be gathered in the cloud computing system which it will be used in. New modifications to the system might be necessary to get an good result. A crucial element in the future work will be researching how effectively one can map the network topology in a datacenter to create a more fail-safe solution.
References


*Overcommitting in OpenStack*. OpenStack. URL: http://docs.openstack.org/trunk/openstack-ops/content/compute_nodes.html#overcommit.