University of Victoria
Faculty of Engineering
Winter 2010 Work Term Report

# Dynamic Resource Distribution Across Clouds

Department of Physics
University of Victoria
Victoria, BC

Michael Paterson
V00214440
Work Term 4
Software Engineering
mhp@uvic.ca

April 21, 2010

In partial fulfillment of the requirements of the
Bachelor of Software Engineering Degree

**Supervisor's Approval: To be completed by Co-op Employer**

I approve the release of this report to the University of Victoria for evaluation purposes only.
The report is to be considered (**select one**): ☐ NOT CONFIDENTIAL ☐ CONFIDENTIAL

Signature: _____ Position: _____ Date: _____

Name (print): _____ E-Mail: _____ Fax #: _____

If a report is deemed CONFIDENTIAL, a non-disclosure form signed by an evaluator will be faxed to the employer. The report will be destroyed following evaluation. If the report is NOT CONFIDENTIAL, it will be returned to the student following evaluation.

# Contents



# List of Figures

# Dynamic Resource Distribution Across Clouds

Michael Paterson
mhp@uvic.ca

April 21, 2010

**Abstract**

The Cloud Scheduler is a cloud enabled distributed resource manager, being developed at the University of Victoria. Using the Cloud Scheduler allows processing jobs to be submitted to a job queue and have the appropriate virtual machine provisioned to run the jobs across multiple cloud installations. The early releases of the Cloud Scheduler only supported a First In First Out(FIFO) provisioning of virtual machines. In order to support multiple users and give each of them fair access to the available computing resources, it was necessary to provide dynamic provisioning of resources as new users submitted processing jobs, and existing jobs finished execution. The latest version of the Cloud Scheduler now supports a basic dynamic resource distribution implementation that treats all virtual machines the same with regard to resource usage. In future releases the Cloud Scheduler will take the actual resource consumption of the virtual machines memory, CPU cores, and other metrics into account when determining resource distribution.

# 1 Report Specification

## 1.1 Audience

This report is intended for members of the High Energy Physics Grid Computing Group at the University of Victoria, and future co-op students. In addition to those wishing to know more about the internal workings of Cloud Scheduler.

## 1.2 Prerequisites

A general understanding of virtualization, distributed computing, clusters, and batch job processing is assumed.

## 1.3 Purpose

The report provides an overview of Cloud Scheduler, Nimbus, and Condor. With a discussion of using the information provided by each of the aforementioned to manage virtual machines and create a fair distribution of resources for users. Possible enhancements to the scheduling algorithm and other metrics to consider in creating the distributions are also covered.

# 2 Introduction

The execution environment requirements of HEP applications are frequently dependent on the specific versions of the tools under which they were written. This makes maintaining dedicated execution environments unfeasible particularly for older applications which do not run on current operating systems. Using Virtual Machines(VMs) alleviates this problem by allowing the customized VM to be deployed on to any cluster with a Virtual Machine Manager(VMM) such as Xen[1] that handles the creation, execution, and shutdown

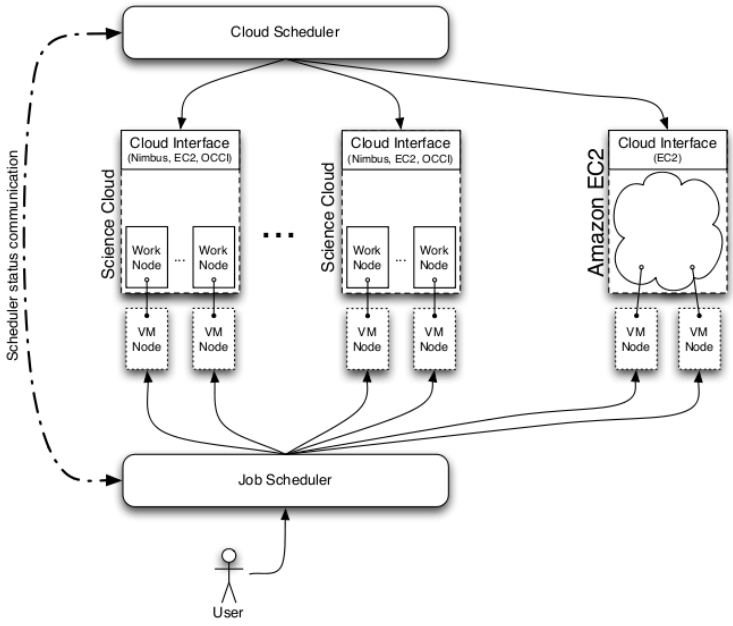


Figure 1: The Overall Architecture of Cloud Scheduler

of VMs. Infrastructure-as-a-Service(IaaS) clouds like Amazon EC2[2] allow remote deployment of virtual machines. Other IaaS platforms are available such as Nimbus[3], which provide services for creating and managing virtual machines through a VMM on a cluster. The ability to virtualize operating systems on a cluster can greatly increase the availability of computing resources to users.

In order to make use of computing resources available at different sites, a user would require valid credentials at each location to start VMs. However, users would still need convenient way to submit their processing jobs to these created VMs. Cloud Scheduler[4] alleviates the problem of multiple credentials by providing a single location to submit jobs. It does this by allowing users to submit their batch processing jobs to a central Condor job queue. Cloud Scheduler then provisions VMs that are required to run jobs in the queue. Figure 1 shows the overall architecture of Cloud Scheduler, which typically runs alongside the Condor Central Manager. The VM images include an installation of Condor so that they can register with the central Condor manager so Condor can begin submitting jobs to those VMs. To better support multiple users submitting jobs to Cloud Scheduler, the user fairness algorithms need to be improved to manage the resources so each user receives processing time.

# 3 Outline of user fairness issue

The first implementation of Cloud Scheduler's user fairness only took into consideration the case of all users submitting their jobs at the same time. Cloud Scheduler would check each user's job queue and start VMs in a round robin manner until no more resources were available, or no more jobs needed to be scheduled. In the case that a single user submitted enough jobs to fill all the available resources and still had jobs waiting in the queue, all subsequent users would get no computing resources until all jobs submitted by the first user had been drained from the queue and their VMs were shutdown.

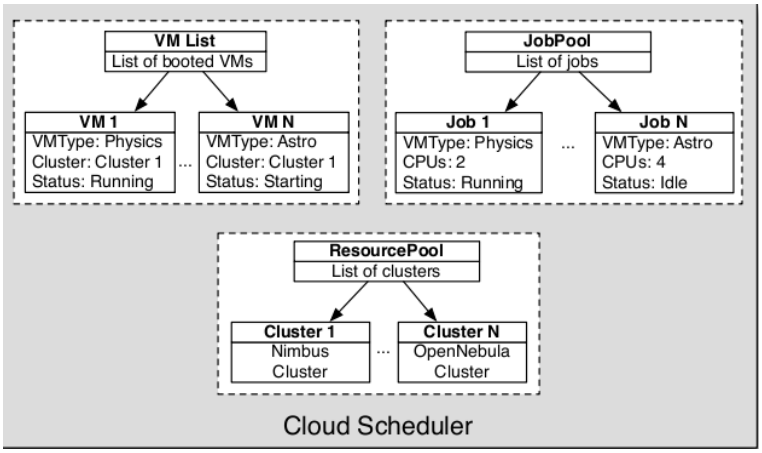


Figure 2: Cloud Scheduler Main Data Structures

# 4 Approach to improving user fairness

To get a fair distribution of VMs for users working it was decided to aim for reasonably fair as a starting point. In the first round of designing the solution to fair resource distribution all VMs were treated the same with regards to their actual resource usage. In reality VMs may be multiple or single CPU-core, and one or more gigabytes of memory, but the complexity to account for those differences can be added in later once the basic algorithm is working. To come up with a fair distribution the job requests can be examined for all users, this gives a desired distribution. The desired distribution is then compared with what is actually running and then adjustments are made by shutting down VMs that are in excess, and starting VMs that are lacking.

# 5 Implementation details

User job requests are gathered from the Condor Scheduler daemon web-service and parsed into an internal representation, sorted by user and priority, for the Cloud Scheduler. Each user has a priority sorted list of jobs. Using the job lists, a simple distribution of requested VM types is calculated. The Cloud Scheduler then calculates the current distribution of VMs that it has started, currently this is a simple distribution where each type is counted and divided by the total number of VMs. The difference between the distribution of the currently running VMs, and the user request is taken. If the difference is negative more VMs of that type need to be created. If the difference is positive, that type of VM becomes a candidate type to shutdown. Figure 2 shows the primary data structures of Cloud Scheduler that have to be iterated through to produce the distributions. When the Cloud Scheduler tries to start new VMs for users it checks against the distribution difference to make sure a VM of the requested type is allowed to be created. Once scheduling decisions have been made the Cloud Scheduler attempts to remove unused machines or find machines that can be shutdown to re-balance resources. The Cloud Scheduler keeps track of the machines it has booted, but has no knowledge of what those machines are currently doing. To find out more information about a particular VM, the Condor Collector web-service is queried, Figure 3 shows the locations of Condor's daemons. The Condor Collector service provides a list of all the machines registered with Condor, and includes information about what job that machine is running. Condor returns machine information as a Condor classad data structure. The classad structure needs to be parsed into a more usable data structure before being used. The Cloud Scheduler compares the latest information with the previous output and tries to find VMs that have just finished executing their job and have started executing a different job id. If the job id has changed between scheduling cycles they are tagged and then checked against the distribution differences and if a VM

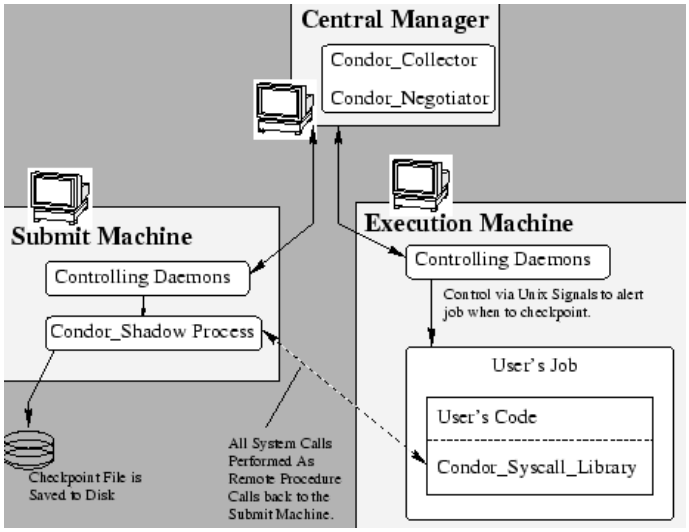


Figure 3: Condor's Pool Architecture[5]

of a type with a positive difference is found it flagged for shutdown. Using the parsed information from Condor the Cloud Scheduler can check for idle and unused machines to shutdown, and also shutdown any machines that had been previously flagged. On the next scheduling cycle the Cloud Scheduler should have the resources available to create a VM of a type that is under-represented in the system. As running jobs finish, their VMs will be shutdown as needed until the system reaches a state of reasonable fairness across the users' requested VM types.

# 6 Results

The initial test results are positive. Using a small cluster with limited space to create VMs a single user submitted enough jobs to occupy all available resources. Once the resources were filled a second test user submitted jobs to the system. The Cloud Scheduler was able to correctly identify finishing jobs and shutdown those machines then create VMs for the second users jobs to run. A subsequent test using 3 users submitting jobs identified an issue where a VM would be shutdown then recreated due to the nature of the distribution differences. This causes some inefficiency in the system and later versions of the Cloud Scheduler will address this issue.

## 6.1 Areas where improvement is needed

### 6.1.1 Resource Distribution Calculations

Currently all VMs are treated as using the same amount of resources. If one user has VMs that require multiple CPU cores and several gigabytes of RAM, and another only requires a single CPU core with one or two gigabytes of RAM, both users will get roughly the same number of VMs but the first user will be consuming a larger share of the resources.

### 6.1.2 The shutdown of Virtual Machines

When selecting virtual machines to shutdown for distribution balancing, the Cloud Scheduler will find machines where a job has finished and a new job has just started. This causes a job to be preempted and

rescheduled. There is a potential for this behavior to cause problems if the job is able to run long enough and make a change to some external entity, such as a database.

## 6.2 Proposed enhancements

### 6.2.1 Improvement of distribution decisions based on resource metrics

Presently resource distribution works as outlined in Section 6.1.1. To ensure users get a fair distribution of the actual resources available, the RAM, CPU-cores, and disk space will be converted into a 3-dimensional volume representing the resource usage.

### 6.2.2 Improvements to selecting machines to shutdown

To avoid any problems caused by jobs being preempted and rescheduled when their machine is shutdown, it's desirable to have machines stop accepting new jobs. Methods of preventing job submission are being explored. It is not known whether any solution found will work on all cloud systems, as there are minor differences between various IaaS software. A partial feature has been implemented to target machines that are still in a Starting state to shutdown, since these machines are not doing any work. There is currently a problem in shutting down a machine in a Starting state, in that a job may have it's machine terminated and not be rescheduled, this is caused by how Cloud Scheduler moving the job into a scheduled queue once a VM has been created. Once the starvation issue with shutting down Starting machines this feature will be implemented. A tolerance range should be added to try to prevent VM thrashing, where a VM repeatedly boots and shuts down. There are cases where the creation of an additional machine of a certain type immediately flags that type to be shutdown, only to be restarted again since the removal of the VM makes it a candidate for creation.

# 7 Conclusion

The current implementation of resource distribution does a reasonable job of giving each user a share of the available computing resources, as long as the requirements of the virtual machines are similar. The current algorithm fixes the serious issue of having a single user occupying all the resources with only a single type of virtual machine. This will allow multiple users to start using the Cloud Scheduler for work and start providing feedback into features they would like to see and assisting in finding bugs that were not found during testing.

# 8 Future Work

The Cloud Scheduler is in on-going development. There are still several major features that need to be added. Integration of the Cloud Aggregator[6] to provide better reporting between Nimbus and the Cloud Scheduler. The state of the Cloud Scheduler needs to be persisted to recover from crashes or restarting the scheduler. Enhancements to the Cloud Status tool to provide more information on the state of the Cloud Scheduler, particularly in regard to job information. The resource distribution algorithms will be extended to use a more sophisticated calculation to account for memory and CPU core usage to give a more accurate picture of how the computing resources have been allocated.

# 9 Acknowledgments

I would like to thank Dr. Randall Sobie for this work term opportunity. Thanks to Patrick Armstrong for guidance on this work term. Additional thanks Ron Desmarais, Ian Gable, and Duncan Penfold-Brown for technical help and advice.

# 10 Glossary

**GT4** Globus Toolkit 4. The *de facto* grid middle ware.

**LRMS** Local Resource Management System. Manages jobs on local clusters.

**VM** Virtual Machine, an instance of a machine(computer) running in software.

**VMM** Virtual Machine Monitor, used for managing virtual machines.

**Xen** Open-source VMM used by Nimbus.

## References


[1] Xen hypervisor - http://www.xen.org/

[2] Amazon Elastic Compute Cloud - http://aws.amazon.com/ec2/

[3] Nimbus http://workspace.globus.org/index.html

[4] Cloud Scheduler http://github.com/hep-gc/cloud-scheduler/

[5] Condor Pool Architecture - http://www.cs.wisc.edu/condor/manual/v6.2/3_1Introduction.html

[6] Cloud Aggregator http://github.com/hep-gc/cloud-aggregator