CANFAR

Canadian Advanced Network for Astronomical Research

Cloud Scheduling System Design Document

Version: 0.1
Submission Date: 2010.01.31
Prepared by: Duncan Penfold-Brown, Patrick Armstrong

University of Victoria

UBC

CANARIE

NRC·CNRC
From Discovery to Innovation...
De la découverte à l'innovation...

1 Revision History

Name	Date	Comments	Version
Duncan Penfold-Brown	June 2009	Initial content discovery	0.0
Duncan Penfold-Brown	1 September 2009	Initial document structure	0.0
Duncan Penfold-Brown	13 January 2010	Body content for all sections	0.1
Patrick Armstrong	14 January 2010	Section 5 (Design Considerations)	0.1
Duncan Penfold-Brown	18 January 2010	Section 6 (Interfaces and Logging)	0.1
Duncan Penfold-Brown	19 January 2010	Section 7, 8, diagrams.	0.1

2 Introduction

2.1 Purpose

This document is meant to present the system architecture, components, design details, and testing environments of the Cloud Scheduling system and Cloud Scheduler software. It is intended for the current and future developers of the Cloud Scheduling system, and should serve as high-level guide for the system's functionality and future directions.

This document is further intended for the maintainers of the Cloud Scheduling system, who may use it as a rough outline of system function; and for the leaders of the CANFAR project, as a statement of the design and implementation of the system.

2.2 Scope

This document will describe, at a high level, the design of the entire Cloud Scheduling system. Implementation details of Cloud Scheduler functionality is left to other sources. The software and hardware dependencies of the Cloud Scheduler will also be described.

2.3 Terms and Definitions

Terms and Acronyms	Definitions
Cloud Scheduler	The software written for the CANFAR project to schedule job submissions onto the cloud.
Cloud Scheduling System	The entire system surrounding the Cloud Scheduler software – this includes the Cloud Scheduler itself, as well as the software on which it relies and the computing environments in which it runs.
VM	A virtual machine, roughly explained as a computer simulated inside (or “on top of”) a physical computer.
Job	A computational task described in an appropriate format. The Condor [a] job scheduler provides a format job submission.

2.4 References

This document references and is derived from the Cloud Scheduling System Requirements Specification document. The User Stories available on the Cloud Scheduler wiki has also influenced the material in this document.

[1] Cloud Scheduling System Requirements Specification. http://github.com/hep-gc/cloud-scheduler/raw/master/docs/cs-SRS-20090901.pdf

[2] Cloud Scheduler Github Wiki – User Stories. http://wiki.github.com/hep-gc/cloud-scheduler/user-stories

[a] XML-RPC main site. http://www.xmlrpc.com/

[b] Condor project homepage. http://www.cs.wisc.edu/condor/

[c] Python Distribution Utilities. http://docs.python.org/distutils/

[d] Python optparse. http://docs.python.org/library/optparse.html

[e] Python logging. http://docs.python.org/library/logging.html

[f] JSON. http://www.json.org/

[g] Python unittest. http://docs.python.org/library/unittest.html

2.5 Document Overview

The remainder of this document consists of a high-level description of the system architecture and an overview of the system's function (what the Cloud Scheduler does, and how the system works); a review of design assumptions and dependencies, as well as a discussion of related and integrated technologies; an explanation of the Cloud Scheduler's interfaces and logging practices; a description of the design details of the system, including components and their interactions; and finally an overview of the tools and environments built to test the Cloud Scheduler and Cloud Scheduling system.

3 System Architecture

The Cloud Scheduler software architecture is modular, with object-oriented functionality supported by classes organized into a number of categorical modules. These modules support extensions for specific tasks and functionality related to external softwares and technologies. The Cloud Scheduler also consists of a number of central processes that contain the main scheduling, initialization, and monitoring functionality.

The main modules of the Cloud Scheduler are as follows:

- **cluster_tools:** this module contains the functionality for representing and manipulating cluster resources. Cluster properties are stored. Clusters with specific cloud softwares are represented by subclasses of the cluster class. VMs are represented herein, as are all operations on them. All VM operations are methods of the cluster classes.
  - This module is extended by the **nimbus_xml** module, which contains technology-specific support for the required functionality within **cluster_tools**.
  - Other extension modules may be necessary in the future.
- **cloud_management:** this module contains functionality for managing groups of clusters as a distributed cloud. Clusters are gathered in a resource pool. Contains functionality for searching and evaluating cloud resources to find suitable spaces for starting VMs, and functionality for monitoring the Cloud Scheduler's status.
- **job_management:** this module contains all functionality for representing jobs in the Cloud Scheduling system, and for collecting those jobs into a job pool. All operations on Jobs are supported. Communication with the external job scheduler is supported.

The central system function is represented in a multi-threaded process. This process contains the initialization and scheduling functionality of the system. Currently, scheduling takes place in a thread separate from the process's main thread. The scheduling functionality currently is all encompassing (that is, all scheduling functionality occurs in one thread), but may be expanded or separated into a number of separate threads to enhance efficiency.

External monitoring functionality is achieved through a separate process that can be executed remotely via XML-RPC [a].

Illustration 1 (below) shows the modules of the Cloud Scheduler software and the classes that make up their functionality.

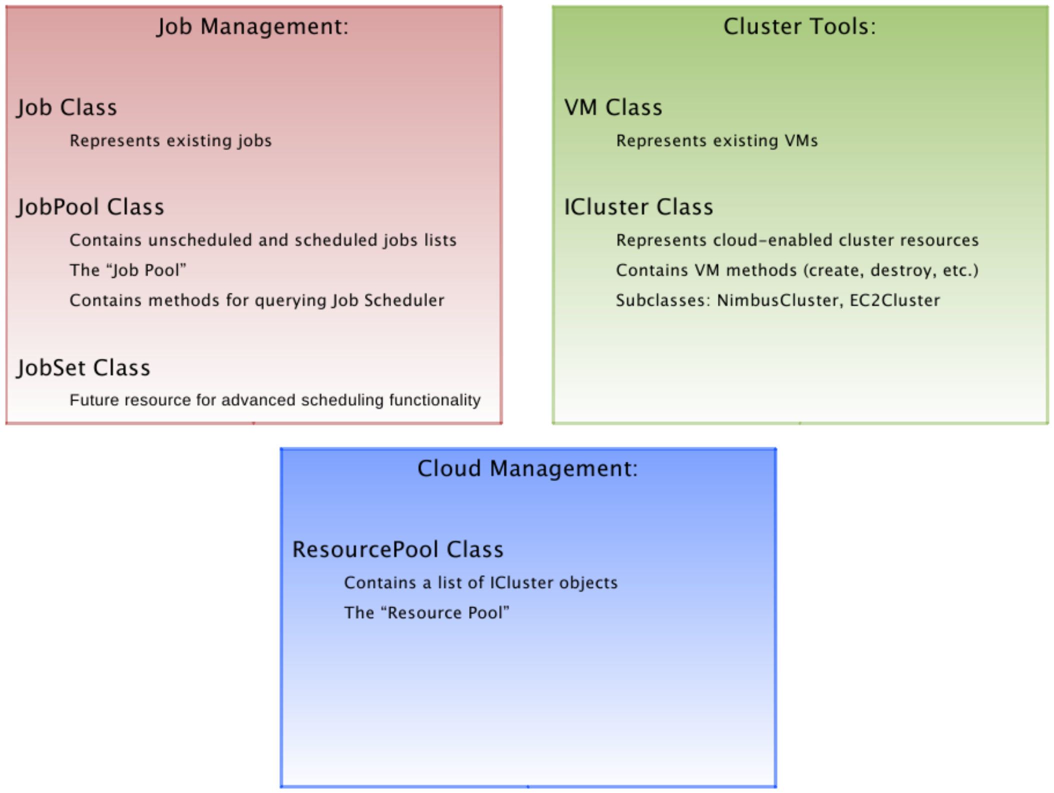

Job Management:
Job Class
Represents existing jobs
JobPool Class
Contains unscheduled and scheduled jobs lists
The "Job Pool"
Contains methods for querying Job Scheduler
JobSet Class
Future resource for advanced scheduling functionality
Cluster Tools:
VM Class
Represents existing VMs
ICluster Class
Represents cloud-enabled cluster resources
Contains VM methods (create, destroy, etc.)
Subclasses: NimbusCluster, EC2Cluster
Cloud Management:
ResourcePool Class
Contains a list of ICluster objects
The "Resource Pool"

Illustration 1: Modules of the Cloud Scheduling Software

4 System Function Overview

The Cloud Scheduling system is represented in Illustration 2.

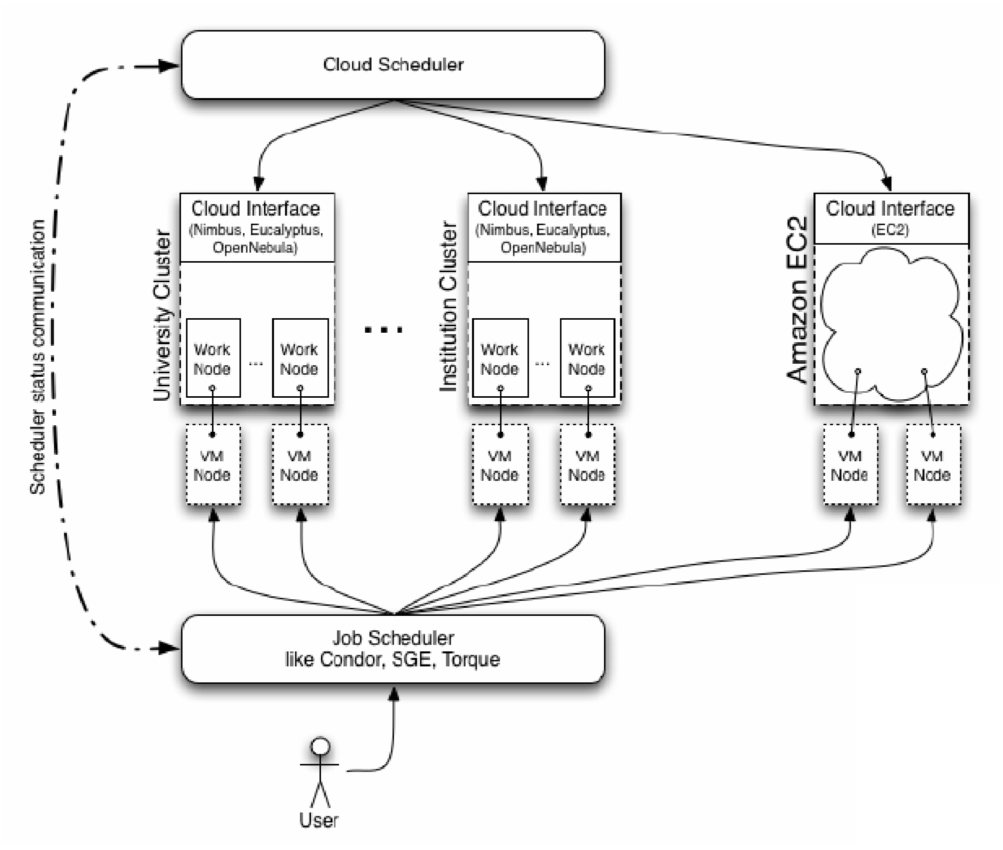


Illustration 2: The Cloud Scheduling System Overview

4.1 User Perspective

From the perspective of the user, the Cloud Scheduler should be completely invisible. The Cloud Scheduler is built in part to automate many tedious and difficult tasks that would normally be required of a user to run his or her job in a virtual, cloud-based environment. The user, then, submits a computational job the Condor [b] job scheduler. This job will have a number of extra fields specifying the environment in which the wants his or her job to run. The user may monitor his or her job submission with familiar job scheduler tools (for example, condor_q and condor_status). When the user's jobs have finished, output is staged back to the user. This workflow is familiar to scientists who have used batch scheduling systems for cluster computation.

4.2 The Cloud Scheduler

From the point of user job submission (as explained above), the Cloud Scheduler takes over as the cloud backend to create and manage a VM environment that meets the user's job requests. This section will briefly introduce the Cloud Scheduler's functionality in a roughly chronological sequence, from starting the Cloud Scheduler to its handling of user-submitted jobs. Note that some of these steps do no occur in strict chronological order, but may occur in parallel.

4.2.1 Initialization

When the Cloud Scheduler is first run, it reads in a number of configuration variables, initializes its logging functions, and proceeds to create an internal representation of the cluster resources available to it (specified in a resource configuration file). This involves storing the properties of a number of cloud-enabled clusters, including web address; available work nodes; free memory, CPU cores, and disk space; and available networking options. Once the Cloud Scheduler has created this internal representation of resources, it starts the scheduling thread and can proceed to detect and schedule resources for user jobs.

[Include diagrams from Ottawa presentation: user perspective and sample job file.]

4.2.2 Scheduling Jobs and VMs

The Cloud Scheduler's scheduling sequence proceeds as follows:

- The Cloud Scheduler first queries the cloud job scheduler to determine if there are any new jobs that have been submitted since its last check.
    - In this process, the Cloud Scheduler removes jobs that have finished or have been canceled from the system.
- The Cloud Scheduler then attempts to schedule all new jobs.
    - Scheduling involves parsing a job's requirements, finding fitting space for those requirements among the available cloud resources, and starting a VM of the specified type and build in that space.
    - User fairness, job priority, and anti-starvation policies play a part in which jobs are considered.
- When all new jobs are considered, the Cloud Scheduler clears all unneeded VMs from the system.
- The Cloud Scheduler then polls all its running VMs to check for status.

Summarized, upon receiving a number of jobs submitted by a user, the Cloud Scheduler would create VM resources in accordance with the requirements of those jobs. When the jobs finish, the Cloud Scheduler will shut down the resources started to support the jobs if the resources can no longer be used by other jobs in the system.

5 Design Considerations

5.1 Assumptions

We assume that cloud resources will have some type of IaaS software installed, and will use Xen on the backend. We also assume that we will be able to access VM nodes booted on these resources from outside the cluster, either with a public IP address, or using a solution like OpenVPN.

5.2 Dependencies and Constraints

One of the most important constraints to take into consideration for Cloud Scheduler is that our users will have little control over the technology choices made on the resource provider side. At best, we can hope that the resource provider uses one of the Cloud Management technologies mentioned below, as they all have usable APIs that will allow us to securely boot VMs remotely.

5.3 Cloud Management Technologies

The three main competing Cloud Management (IaaS) technologies are OpenNebula, Nimbus, and Eucalyptus. We would ideally like to support all three of these technologies, as we would like our users to be able to use the largest number of resources possible.

5.3.1 OpenNebula

OpenNebula is is a product of a project by the Distributed Systems Architecture Research Group at Universidad Computense de Madrid. Originally used for organizing VMs within a single data center, the latest release has focused on supporting externally available APIs, like the de-facto standard Amazon EC2 Query API, and the new OGC OCCI API, a new REST based, standards-driven API.

5.3.2 Nimbus

Nimbus is virtual machine provisioning software created as a part of the Globus project, at the University of Chicago and Argonne National Laboratory. Nimbus is one of the most mature pieces of IaaS software available, its distinguishing feature being that it can coexist with existing batch-queue job systems at the backend, making it easier for a site to transition to a cloud provider model, or even a hybrid cloud/batch model. In addition, Nimbus supports the grid standard X509 proxy certificates (RFC 3820), which are well supported by existing grid infrastructure.

5.3.3 Eucalyptus

Eucalyptus is product of a UC Santa Barbara research project, and is now a product of Eucalyptus Systems. It is designed to emulate the Amazon Elastic Compute Cloud (EC2) API, and allow a user to use his own cluster in a way that is similar to how a user would use Amazon's EC2 service.

5.4 Job System Technologies

Some of the possible job system technologies considered for this project were Sun Grid Engine, Cluster Resources Torque (A PBS clone), as well as Condor. After experimenting with these three technologies, we decided to focus on Condor. The primary reason for this choice is that Grid Engine and Torque are primarily intended for static, homogeneous resources. For example, to add a node to Torque, an administrator must edit a text file, then restart the Torque server.

Condor, on the other hand, was designed from the outset to make use of heterogeneous resources, that may or may not be available at any one time. This is a good match for dynamic resources booted in an ad-hoc way. As a result, we decided to use Condor for our initial implementation, but to do so in a way where it wouldn't be overly difficult to transition to another jobs system if it became obvious that there was a significant reason to do so.

5.5 Additional Technologies

Other supporting technologies will most likely be from the standard open source software stack. For example, we are using OpenVPN to manage connections to nodes that aren't available via a public IP address and Apache for serving images via http. Neither of these technologies are the only way to accomplish these ends. In fact, we are not yet sure that these are the best or most sustainable methods of accomplishing these. As a result, this section is intentionally left fairly vague at this point, and will be revised as the project continues and matures.

6 Interfaces and Logging

6.1 Administrator Interface

The administrator of the Cloud Scheduling system is responsible for installing, configuring, and running the Cloud Scheduler software. The design of these tasks in the Cloud Scheduling system is as follows.

6.1.1 Installation

Installation of the Cloud Scheduler software is facilitated by the Python distribution utilities (Distutils) [c], which provide the programmer with an easy way to specify an installation scheme. The setup.py script containing the installation specifications for the Cloud Scheduler software resides in the Cloud Scheduler's base directory.

The installation process currently moves the two Cloud Scheduler configuration files (discussed below) into the /etc/cloudscheduler directory, and the two run-scripts, cloud_scheduler and cloud_status, into the default Distutils script location.

The install process also imports the cloudscheduler package into the target system's Python libraries.

6.1.2 Configuration

There are two configuration files that define the function of the Cloud Scheduler. These are the cloud scheduler software configuration file and the cloud resource configuration file.

The cloud scheduler software configuration file contains fields for defining Cloud Scheduler program functionality, including Condor job pool configuration information, logging information, and default cloud resource configuration options. The cloud scheduler configuration file can be manually specified on the command line when the Cloud Scheduler is run via the [-f FILE | --config-file=FILE] option, or can be stored in the following locations:

- ~/.cloudscheduler/cloud_scheduler.conf
- /etc/cloudscheduler/cloud_scheduler.conf

Run from the command line, the Cloud Scheduler will attempt first to get the software configuration file from the command-line, then from the ~/... directory, and finally from the /etc/... directory. If using the setup.py installation script, the Cloud Scheduler will be configured by default to search the /etc/cloudscheduler/ directory for its configuration files.

For future functionality implemented in the Cloud Scheduler, any parameters that would be useful for the administrator to be able to set and change should be present in the software configuration file.

The cloud resource configuration file contains information on the cloud-enabled clusters that the Cloud Scheduler will use as resources. The cloud resource configuration file can be specified on the command-line with the [-c | --cloud-config=FILE] option. If the cloud resource configuration file is not specified on the command line, it is taken from the location given in the cloud_resource_config field of the cloud scheduler software configuration file.

6.1.3 Running

There are two ways for the Cloud Scheduler software to be run: on the command line, or through an Unix/Linux-style init script.

For running the Cloud Scheduler on the command line, a number of option tags are supported. They are as follows:

- **-h | --help** – Obtain the Cloud Scheduler help message and usage notes.
- **-v | --version** – View the software version number.
- **-f | --config-file** – Specify the location of the software configuration file.
- **-c | --cloud-config** – Specify the location of the cloud resource configuration file.

- **-m | --MDS** – Specify the web address of an MDS server for obtaining up-to-date information on MDS-enabled clusters (currently not supported).

These command-line options are implemented via Python's optparse library [d], using the OptionParser utility. Command-line options are specified in the main Cloud Scheduler process file. More options can be easily added in the future.

After installation via the setup script, the Cloud Scheduler can be run through an init script in the /etc/init.d directory of the installation target machine. The init script options are as follows:

- **start** – Starts the Cloud Scheduler. Can not be run when the Cloud Scheduler is already running (will receive an error message).
- **stop** – Shut down the Cloud Scheduler.
- **restart** – Execute a stop, then a start.

The Cloud Scheduler init script should register itself as a system service on the machine on which the Scheduler is installed. The Cloud Scheduler can then be manipulated with standard Unix/Linux service syntax.

6.2 User and Monitoring Interface

The user interfaces and monitoring available in the Cloud Scheduling system come from two sources. The first is Condor, the tool through which users submit jobs to the system. The second is the cloud_status program, which can be run against the Cloud Scheduler software to obtain status information on the Scheduler's resources.

6.2.1 Condor Interfaces and Monitoring

From the user's perspective, the Condor job scheduler comes with a set of tools for submitting, managing, and monitoring jobs. These tools can be used to monitor jobs and resources in the Cloud Scheduling System. Condor's condor_q and condor_status commands are the two most common command-line tools for monitoring. They return the jobs in the system and the computational resources registered in the system, respectively. Status and user/owner information is included in the output of both tools.

6.2.2 Cloud Status Interfaces and Monitoring

The cloud_status process is a separate program that contacts a running Cloud Scheduler instance and queries it for information regarding the clusters and VMs that the Cloud Scheduler is managing. This information is available in user-readable format and also in a JSON [f] format that can be used programmatically to create graphical displays. The cloud_status program uses XMLRPC [a] calls to remotely query a Cloud Scheduler instance.

The Cloud Scheduler software implements a simple information server that fields requests from cloud_status (and other XMLRPC agents), returning requested information.

6.3 Scheduler Logging

The Cloud Scheduler software employs multi-level logging via the Python logging library [e]. A logging handler is created in the main Cloud Scheduler process and shared between all modules in the Cloud Scheduling system. The supported logging levels are:

- **DEBUG –** Status information for tracking programmatic progress and location.
- **INFO –** System function information useful to the user or administrator.
- **WARNING –** Warnings about system actions.
- **ERROR –** Reports of non-critical system failures.
- **CRITICAL –** Errors that cause the system to function improperly (illogically) or require a system restart.

The level of displayed log messages, log file and file size, and other logging options are specified in the Cloud Scheduler software configuration file.

7 Design Details

7.1 System Components

The main components of the Cloud Scheduler software correspond approximately to the modules described in section three, System Architecture. Additional components exist for procedural functionality within the Cloud Scheduler and external elements of the system.

7.1.1 Internal Components

Each internal component is a collection of classes, methods, and procedures, and encapsulates certain Cloud Scheduler functions. The internal components are as follows:

1. **Scheduling Component**
   - Coordinates and uses all other system components for main scheduling functionality – discovering jobs, matching jobs to resources, creating and managing VMs on the cloud, and clearing VMs and jobs from the system.
   - Constituents:
     - Cloud Scheduler main method: initialization of logging, threads, the info server, and resource and job pools.
     - Scheduling Thread: consists of a scheduling loop. Discovers jobs, creates resources for them, and clears old jobs and VMs from the system.
2. **Job Management Component**
   - Provides an interface for maintaining up-to-date job information and is responsible for all communication with the Job Scheduler.
   - Constituents:
     - Job class: represents a user-submitted job within the Cloud Scheduler.
     - JobPool class: represents the collection of jobs in the system. Contains scheduler and unscheduled job queues. Contains methods for querying and communication with Job Scheduler.
3. **Cloud Management Component**
   - Provides functionality for managing the cloud as a collection of heterogeneous cluster resources (based on the classes from the cluster component, below) and provides resource selection and searching.
   - Constituents:
     - ResourcePool class: represents a collection of cloud-enabled clusters. Contains methods for sorting, searching, and selecting resources based on job criteria. Supports balanced job-to-cluster distribution over the cloud at large.
4. **Cluster and VM Component**
   - Provides functionality for managing individual cluster resources with particular cloud softwares, and for managing VMs on cluster resources. Provides the Scheduling Component with a generic interface (not cloud-software specific) for managing VMs on resources.
   - Constituents:
     - VM class: represents a VM within the Cloud Scheduler.
     - ICluster class: provides general cluster parameters and an interface for subclasses to implement defining VM management on cluster resources.
     - NimbusCluster, EC2Cluster, etc: cloud-software specific subclasses of the ICluster class. Implement the required VM and cloud management functionality in accordance with the different cloud softwares. All cloud softwares described in section five-three (Cloud Management Technologies) will be supported.

Importantly, the ICluster and subclass structure is easily extensible to create subclasses for new cloud softwares.

7.1.2 External Components

The external components of the system are based on other softwares and technologies. They are not implemented or modified as part of this project, but are relied upon as dependencies of the Cloud Scheduler. These components are as follows:

1. Job Scheduler

- The Job Scheduler is an entity separate from the Cloud Scheduler software, and exists in the Cloud Scheduling system to accept user job submissions, and to manage the distribution and execution of these jobs on cloud resources managed by the Cloud Scheduler.
- The Job Scheduler is passive in the Cloud Scheduling system. The Cloud Scheduler is responsible for contacting and querying the Job Scheduler for information.
- Much effort has been dedicated to creating successful and robust Job Schedulers. Currently, the Cloud Scheduling system uses the Condor scheduler. Refer to section five for more details.

2. Cloud Interface (per cluster)

- The Cloud Interface component of the system refers to the particular cloud software (or cloud management technology) installed on any given cluster resource in the Cloud Scheduling system.
- The Cloud Interface component is manipulated by the Cloud Scheduler in order to manage VMs.
- The Cloud Interface component returns status information on VMs to the Cloud Scheduler.

7.2 System Diagrams

Illustration 3 depicts the Cloud Scheduling System at large. The entire current system is pictured in abstract form: the Job Scheduler and Cloud Interface external components are shown, as is the Cloud Scheduler. The internal components of the Cloud Scheduler itself are also shown in the form of the Scheduler, the Resource Pool, and the Job Pool.

Illustration 4 describes the main scheduling functionality of the Cloud Scheduler. This functionality forms the core of the Scheduling component described above.

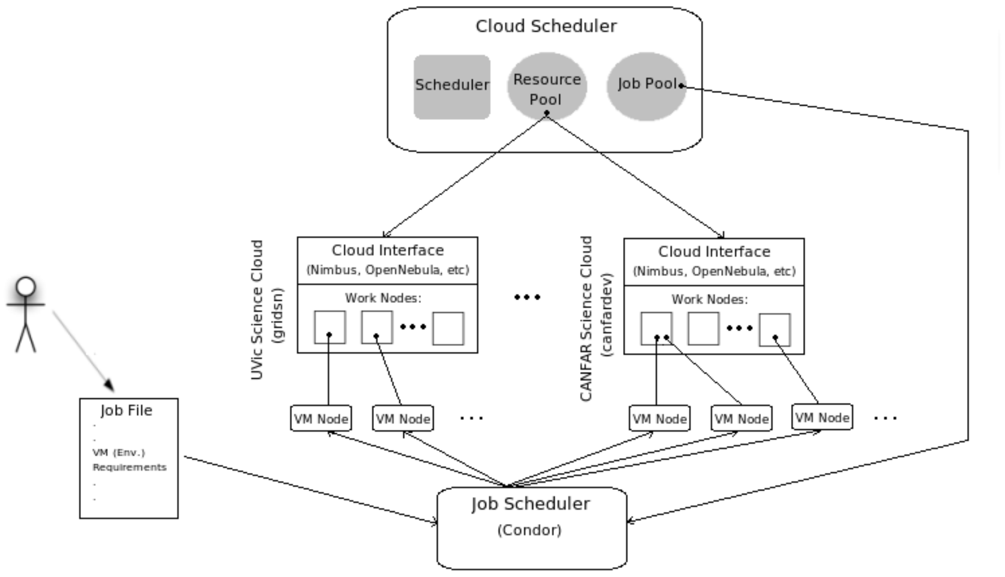
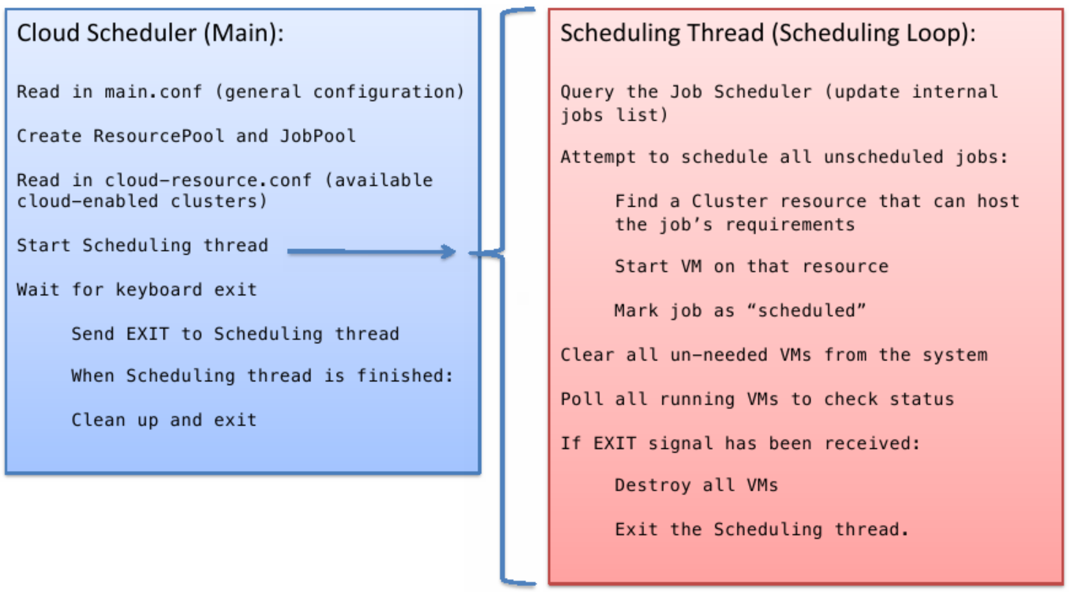


Illustration 3: Details of the Cloud Scheduling System



```
Cloud Scheduler (Main):

Read in main.conf (general configuration)
Create ResourcePool and JobPool
Read in cloud-resource.conf (available
cloud-enabled clusters)
Start Scheduling thread
Wait for keyboard exit
    Send EXIT to Scheduling thread
    When Scheduling thread is finished:
    Clean up and exit
```

```
Scheduling Thread (Scheduling Loop):

Query the Job Scheduler (update internal
jobs list)
Attempt to schedule all unscheduled jobs:
    Find a Cluster resource that can host
    the job's requirements
    Start VM on that resource
    Mark job as "scheduled"
Clear all un-needed VMs from the system
Poll all running VMs to check status
If EXIT signal has been received:
    Destroy all VMs
    Exit the Scheduling thread.
```

Illustration 4: Cloud Scheduler Main Scheduling Functionality

7.3 Class Diagrams

Illustration 5 depicts the main classes of the Cloud Scheduler software.

[CLASS DIAGRAM]

7.4 Additional Design Topics

The following topics are important in the design and continued evolution of the Cloud Scheduler and Cloud Scheduling system.

7.4.1 Modularity

The Cloud Scheduler software has adopted a modular architecture (described in section three) in order to avoid monolithic code structures and to support system organization. The modular structure currently employed facilitates a clear system concept, and makes extension, revision, and understanding of the code achievable. Future development of the Cloud Scheduling system should strive to be reasonably modular.

A modular architecture also has the benefit of facilitating easy testing. For example, if a new cloud management technology becomes popular, writing a module to support that technology and importing it into the main Cloud Scheduler functionality is easier and cleaner than trying to modify the structure of the main Cloud Scheduler itself.

In the Cloud Scheduling system, modules should reflect a categorically consistent group of functionalities. For example, if real-time querying of cluster resources via MDS or other technologies is required, the implementation of such functionality could suitably be put into its own module.

7.4.2 Extensibility

Extensibility is in part achieved by modularity, as discussed in the previous subsection. Another important contributing factor to the extensibility of the Cloud Scheduling system is inheritance supported by classes and subclasses. The class/subclass structure suits supporting the specifics of new technologies and softwares in the case that the Cloud Scheduler requires a general interface to those technologies.

The most important example of this object-oriented inheritance-based extensibility is the ICluster class / NimbusCluster, EC2Cluster subclass structure in the cluster_management module. Extensibility is achieved here by the provision of a general cluster management interface in the ICluster class. Technology-independent details are specified in the subclasses of the ICluster class.

As the cloud computing environment, and the technology industry in general, are rapidly evolving, it is important to provide measures to achieve extensibility and to remain very flexible in the face of technology changes. The three main cloud management technologies mentioned in section five may eventually (or quickly) lose focus and popularity, in which case the Cloud Scheduler's flexible structure can be employed to adapt to new technologies. As such, flexibility and extensibility are major design goals of the Cloud Scheduling system.

7.4.3 Multi-threading

Currently, the main scheduling functionality of the Cloud Scheduler is designed monolithically. That is, all scheduling functionality is represented and executed in the scheduling thread of the main Cloud Scheduler process. This functionality includes querying the Job Scheduler of the system, parsing and managing jobs, managing VMs, and cleaning up the system.

Splitting the scheduling functionality of the Cloud Scheduler into multiple threads, to execute in parallel and asynchronously (where possible and reasonable), would benefit the Cloud Scheduler in terms of efficiency and scalability. Both efficiency and scalability are design goals of the Cloud Scheduling System.

8 System Testing

The Cloud Scheduling System and the Cloud Scheduler software require extensive testing to ensure successful functioning, stability and robustness, and scalability. To this end, the Cloud Scheduling system is tested with the following tools.

8.1 Test Scripts

A number of test scripts have been developed to automate and speed up testing of the Cloud Scheduler functionality. These scripts primarily test for Cloud Scheduler robustness and scalability.

Two general-function scripts have been useful in testing and analyzing the Cloud Scheduler. They are:

- **submit-multiples** – a shell script that submits a specified number of a given job file. This script helps to test for Cloud Scheduler scalability and robustness. Stress tests (huge numbers of jobs) can also be executed.
- **Job-distribution –** a shell script that, given the output files from a number of jobs, maps the jobs to the resources that ran them. Outputs numbers of jobs per cluster. This script tests for the fair distribution of jobs between resources.

Beyond these two general scripts, there are scripts to automatically acceptance-test specific releases of the Cloud Scheduler in accordance against the goals of those releases. These are located in the test-sets/ directory.

8.1.1 Test Job Script

Another important script in the testing of the Cloud Scheduling system is the script that a test job executes when scheduled to a cloud resource. This script executes a number of commands to perform reconnaissance on the resource on which the job is running.

Currently, the script obtains the resource's hostname and IP address; username running the job; memory available on the resource; CPU information on the resource, including number of cores, speed, and model; disk space available on the resource; VMType as specified in the VMs Condor configuration file; Condor job-pool the resource is registered with; and time of job arrival and execution. This information is obtained using Unix/Linux shell commands.

8.2 Sample Job Sets

To test the Cloud Scheduler's ability to support varied jobs, many test-job sets have been developed. Generally, these sets consist of a number of jobs with widely divergent requirements. Requirements are chosen specifically to test the limits of Cloud Scheduler functionality.

Each release has a test job set associated with it that specifically tests the features targeted for that release. Test jobs sets are usually cumulative in that they require functionality and features from previous releases to be received properly by the system. The test job sets associated with the Cloud Scheduler are located in the test-sets/ directory.

8.3 Unit Testing

Unit testing is also supported for the Cloud Scheduler software, by way of Python's unittest framework [g]. This framework provides simple assertion tests of Cloud Scheduler functions, whereby expected output can be compared against actual function output.

Unit testing functionality is located in the test.py file in the Cloud Scheduler directory.