University of Victoria
Faculty of Engineering
May 7, 2010

Cloud Aggregator: Improving the Nimbus Monitoring System

Department of Physics
University of Victoria
Victoria, BC

Adam Bishop
V00699182
Work Term 2
Electrical Engineering
ahbishop@uvic.ca

May 7, 2010

In partial fulfillment of the requirements of the
Bachelor of Electrical 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

Cloud Aggregator: Improving the Nimbus Monitoring System

Adam Bishop - ahbishop@uvic.ca

May 7, 2010

Abstract

Cloud Aggregator is a new software utility to provide data aggregation and reporting functionality to the Globus Resource Monitoring & Discovery system. The deprecation of previous reporting software necessitated the need for Cloud Aggregator. Cloud Aggregator gathers information about various cloud locations and makes the aggregated data available to other cloud software, notably the Cloud Scheduler. Cloud Aggregator requires a specific XML data format, which required modifying existing components of the Nimbus Monitoring & Discovery system. This new data scheme is significantly more descriptive and conveys additional information not possible with the previous scheme.

1 Introduction

The High Energy Physics Research Computing (HEPRC) group at the University of Victoria is applying emerging computing technologies to physics research. Cloud computing and virtualization are emerging computing technologies and are utilized by the HEPRC group.

Cloud computing has numerous interpretations depending on one's application needs. The University of Victoria HEPRC group considers cloud computing synonymous with Infrastructure-as-a-Service or IaaS[1]. IaaS strives to provide computing resources to users in a system agnostic manner. This technology allows multiple users, with heterogeneous work loads and system requirements, to utilize shared physical resources. To facilitate this sharing, virtualization technology is used to abstract a user's workload from the physical cluster resources into a virtual machine (VM). The VM encapsulates a user's unique software environment, including data and tool-chain requirements, into a single file. The cloud is concerned only with the life cycle management of VMs, not the unique software dependencies of a job.

The Nimbus[2] project is an IaaS software suite that handles VM life-cycle management as well as security, data transfer services and other key features of cloud enabled middleware. A missing link in this chain is a cloud resource monitoring and discovery service. As clouds are multi-user entities, knowing the current state of a clouds resources, including what VMs are active, how much free memory exists on compute nodes and other statistics, is vital. The Nimbus Resource Monitoring & Discovery project was developed to provide this functionality. Cloud Scheduler, another cloud enabled project developed by the HEPRC group, will utilize the complete Cloud Aggregator system to monitor cloud resources.

A subsequent co-op student utilized the Nimbus Monitoring & Discovery system and provided valuable feedback on the system. It was discovered that the system's XML output data was structured in an inflexible fashion, making it difficult to work with. Also, the Globus MDS utility, the software used to publish the Monitoring & Discovery system's data, has been deprecated. These two issues must be addressed to make the Monitoring and Discovery system usable.

The lack of a utility to publish data to and query against necessitated the design and development of the Cloud Aggregator utility. Development of this utility was the focus of the work term's efforts. Tied to this was the re-factoring of the Discovery & Monitoring system's XML[3] output data.

2 Discussion

2.1 Cloud Aggregator

2.1.1 Overview

Cloud Aggregator is a data gathering and aggregation utility. Its purpose is to gather XML data from remote Nimbus cloud sites, aggregate this XML into a single document and finally store it into a shared, public database. Cloud software requiring access to this information can then query this shared database to retrieve the XML. Previously Globus provided this functionality, and other advanced features, through it's Monitoring and Discovery Service (MDS). However, this utility is now deprecated and has been removed from current Globus releases, necessitating the creation of Cloud Aggregator.

The 2009 XML format allowed plug-in XML data to simply be appended together, the new format requires the construction and manipulation of the documents DOM[4] tree, as well as XSLTs[5] to produce the desired output. Both DOM manipulation and XSLT are memory intensive, especially for larger XML documents. However, testing showed the memory and computing cost to have a neglible performance impact on the system. Discussion amongst the HEPRC group revealed that simple gathering, aggregation and storage functionality was all that would be required for Cloud Aggregator. Advanced features present in MDS, such as the ability to cascade MDS' from multiple locations and XPath[6] query support were deemed unnecessary for the Cloud Aggregator.

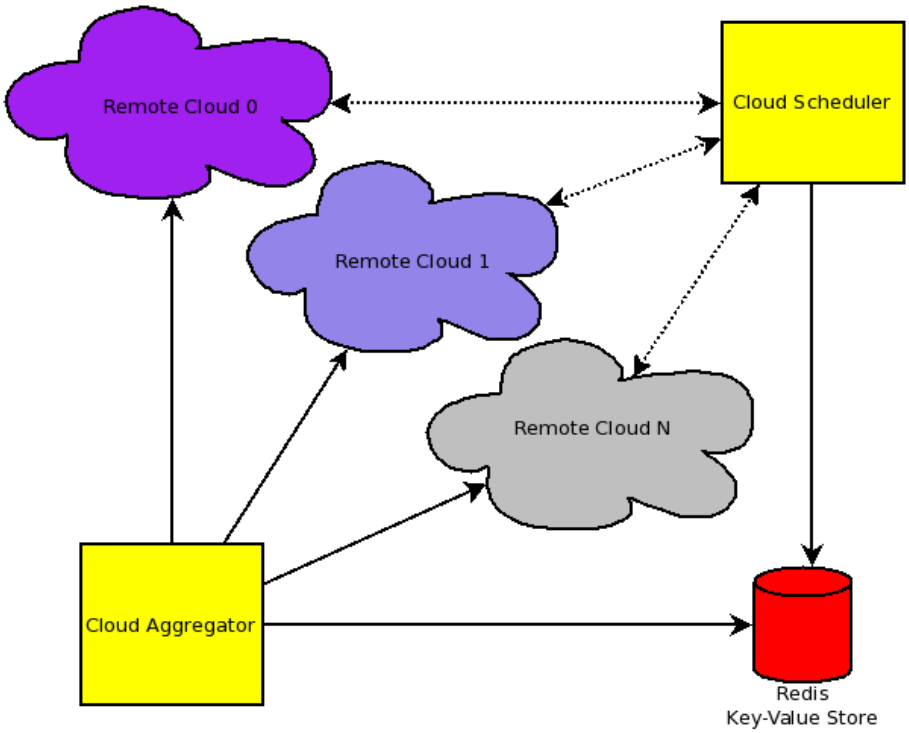

Figure 1: A high level overview of Cloud Aggregator and Cloud Scheduler interacting

2.1.2 Details

Cloud Aggregator is written in Python[7] and relies on the Redis[8] key-value database. Cloud Aggregator iterates over its configured list of remote cloud sites and issues an HTTP query for resource data. Cloud sites are obligated to provide a publicly accessible HTTP address to their resource XML data. How this XML data is published, be it by web server or some other method, is irrelevant to Cloud Aggregator, only that it is available. As a web server is required by Nagios[9], it is extremely likely that a cloud site will have an active web server. This simplifies installation and configuration requirements.

Data aggregation consists of combining all the individual cloud site's XML data into a single XML document. No additional processing or XSL transformations are required.

To persist the aggregated XML data, the Redis key-value store is utilized. Redis is a key-value store style database which does not utilize SQL. Redis exposes a simple, network based interface and text-based protocol. Interacting with the database is done through 'get' and 'set' operations that either retrieve the data stored with a given key, or set a unique key to point to some data. Redis supports more advanced features, such a list processing operations, set operations and indexing, but these features are not required by Cloud Aggregator's work-flow. Redis differs from traditional relational databases in that Redis is an in memory, "eventually consistent" database. This means that operations are not immediately serialized to disk and instead remain in memory, being written out to disk periodically. This is not a problem for Cloud Aggregator as any data loss from a server issue

will be rectified when Cloud Aggregator re-queries its remote sites. As this querying is done every few seconds to few minutes, any lost data will be quickly replaced.

Cloud Aggregator's data is gathered from two different sources at each cloud site. Normal data consists of resources gathered from the Nagios program that runs on each cloud site as part of the Nimbus Resource Monitoring project. This data consists primarily of physical hardware resources and status of worker nodes. The data is typically updated every few minutes to tens of minutes. Real time data represents data that needs to be reported every few seconds. Due to the high frequency of updates, this data must be gathered outside of Nagios. Nagios is not designed to handle high frequency updates and performs very poorly with this configuration.[10]

2.2 Realizing Cloud Aggregator

2.2.1 XML Schema

The 2009 version of the Nimbus Monitoring Project defined a simple XML format for resource data. This format allowed for easy software development, but was very difficult for other projects to interact with. An example of this schema is shown in Figure 3:

```
<ROOT>
  <RES LOC="IP Address" TYPE="Plug-in Type 1">
    <ENTRY ID="Identifier">
      "value"
    </ENTRY>
  </RES>
  <RES LOC="IP Address" TYPE="Plug-in Type 2">
    <ENTRY ID="Identifier 2">
      "value 2"
    </ENTRY>
    <ENTRY ID="Identifier 3">
      "value 3"
    </ENTRY>
  </RES>
</ROOT>
```

Figure 2: 2009 XML Scheme

This simple XML structure consists of one-to-many RES tags under the ROOT tag, with each RES tag having one-to-may entries. A RES tag represents a particular monitored resource, such as memory allocated to VMs on the host. Each VM would be represented by an ENTRY tag under the RES tag.

This structure needed simple construction rules and was easy to implement for the developed Nagios plug-ins. Also, with no specification nor schema to conform to, this seemed to be the easiest development path. Parsing this XML back into a data structure was straightforward as only two unique tags were used.

Unfortunately, this structure isn't very 'human readable', which is a stated goal of XML. Another disadvantage is the reuse of only two tags. Many automatic XML parsing and visualization utilities rely on unique data being stored in tag names. The current scheme instead encodes unique information in attributes, which hides relevant information. These problems, combined with the inflexibility of the two tag format prompted a redesign.

The new XML scheme was decided on after consulting with members of the HEPRC group. Specific extensibility and readable goals were focused on and the following scheme was decided. The XML shown in Figure 4 was gathered from an active test system and is an accurate example of the system's XML scheme. Attempting to detail the scheme without example data, as was done above for the old scheme, would be difficult due to the verbosity and structure of the XML tags.

```
<Sky>
  <Cloud>
    <NetworkPools>
      <Pool>
        <Name> Public </Name>
        <TotalSlots> 10 </TotalSlots>
        <AvailableSlots> 8 </AvailableSlots>
      </Pool>
    </NetworkPools>
    <CloudName> TestCluster </CloudName>
    <Service>
      <Hostname> 142.104.63.29 </Hostname>
      <Port> 8443 </Port>
      <Type> Nimbus </Type>
      <Path> http://theGardenPath.com </Path>
    </Service>
    <VMMPools>
      <Pool>
        <Name> testpool-ANY </Name>
        <MemorySizeMB> 32000 </MemorySizeMB>
      </Pool>
    </VMMPools>
    <IaasService>
      <IaasInternalRepresentation> Consistent </IaasInternalRepresentation>
    </IaasService>
    <WorkerNodes>
      <Node>
        <CPUCores> 8 </CPUCores>
        <CPUArch>
          <MicroArchitecture> x86_64 </MicroArchitecture>
        </CPUArch>
        <CPUID>
          <Description> Intel(R) Xeon(R) CPU E5430 @ 2.66GHz </Description>
        </CPUID>
        <Memory>
          <TotalMB> 16378 </TotalMB>
          <FreeMB> 16378 </FreeMB>
        </Memory>
      </Node>
    </WorkerNodes>
  </Cloud>
</Sky>
```

Figure 3: 2010 XML Scheme

Obviously this structure is radically different and far more verbose than the original. While the previous structure was generic and allowed any type of resource to be adapted to the two tag model, the new layout dramatically improves readability. The new structure also encodes context. For example, individual XML snippets from a single worker node are merged together resource information from that node into a cohesive structure. Also interesting to note is the lack of attributes. Attributes are now used to transmit private and non-displayable information through the system. Thus, a XSLT is used to remove them from the final XML document structure before publishing.

2.2.2 Nimbus Monitoring Updates

As previously alluded to, the 2010 XML scheme required several updates to the original Nimbus Monitoring and Discovery project. Only minor updates to the Nagios plug-ins were required to conform to the new XML scheme. However, the Nagios data processing script used to generate the output XML required a complete rewrite to conform with the new XML scheme.

The drawback of this new format is increased processing complexity on the cloud head node. Whereas the old XML format allowed plug-in XML data to simply be appended together, the new format requires the construction and manipulation of the documents DOM tree, as well as XSLTs to produce the desired output.

As previously mentioned, Nagios can not be run at high frequencies without incurring major performance problems. Unfortunately, this makes gathering real time data, such as available IP addresses, impractical. IP addresses are considered VM slots when referring to Nimbus. As each VM configures networking, every VM needs an IP address from the pre-configured Nimbus pool. This means that a limited number of 'VM Slots' are available based on the number of free and used IP addresses. Knowing the available VM slots at a cloud site in real-time is critical for scheduling decisions. Thus, it is crucial that Cloud Aggregator be able to gather this data in real-time.

This feature was originally developed, in a previous work term, as a Nagios plug-in. However to avoid the Nagios performance issue, it had to be setup and executed outside of Nagios. This was a simple task, with the only modifications being the transfer of code from a Nagios plug-in to its own file and the addition of a configurable timing loop. To avoid conflicts with the existing Nagios plug-ins, the new real-time script saves its output XML to a separate file representing real-time information. Cloud Aggregator is setup to look for this additional data and integrate it into monitoring data gathered by Nagios.

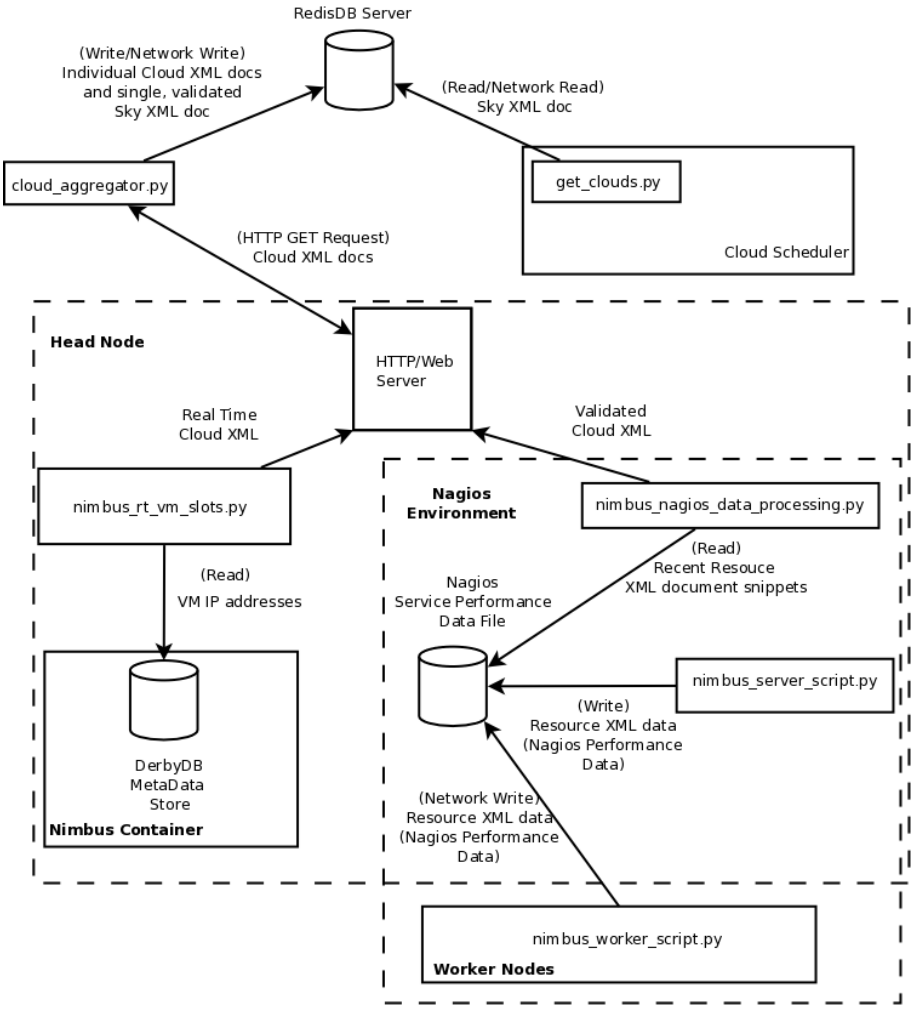

Figure 4: A detailed depiction of the complete Cloud Aggregator system

3 Future Directions

3.1 Production Deployment

Based on the currently understood requirements, Cloud Aggregator is feature complete. However, it has only been deployed and tested in the development environment provided by the HEPRC group. Real world deployment and testing is needed to ensure that Cloud Aggregator adequately supports its various use cases. Undoubtedly, third party interaction with Cloud Aggregator will expose the need for feature refinement and additions.

Integration of Cloud Aggregator into the Cloud Scheduler has yet to commence. Currently Cloud Scheduler is dependent on Cloud Aggregator to provide cloud resource data for intelligent VM scheduling decisions. Cloud Scheduler has been relying on alternative sources of cloud resource data while Cloud Aggregator is under development.

3.2 Project Integration

To ease future integration, a Cloud Aggregator 'client' has been developed. This small piece of code handles the duties of transforming XML data back into a Python data structure. This process is often referred to as XML 'binding'. The data structure's composition mimics that of the public XML scheme. This method should minimize difficulties in understanding where and how data is populated in the data structure.

4 Conclusion

The primary focus of this work term was the creation of the Cloud Aggregator utility. This task also dictated other modifications to the existing Nimbus Monitoring & Discovery software to support the Cloud Aggregator. Cloud Aggregator is a key building block to other cloud software, specifically the Cloud Scheduler. Additional Cloud Aggregator goals, features, and requirements will take shape as the Cloud Scheduler matures and begins relying on Cloud Aggregator.

5 Glossary

Cloud Computing - "The movement of services, computation, and/or data, for cost and business advantage, off-site to an internal or external, location transparent, centralized facility or contractor." Cloud computing involves the addition of a virtualization abstraction layer to traditional cluster configurations. Users operate within a virtualized environment instead of on the physical resources directly. This decoupling allows multiple clusters to make up a cloud and allows physical resources to change without affecting functionality.

DOM Document Object Model - "The Document Object Model is a platform and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page."

Globus The *de facto* grid middle-ware suite.

IaaS Infrastructure-as-a-Service - See Cloud Computing

MDS Monitoring and Discovery System - Information service component of the Globus Toolkit. It provides information about the available resources on registered grid resources and their status. This utility has been deprecated in current Globus and Nimbus releases.

Nagios - An open source system and network monitoring tool. "Nagios is a powerful monitoring system that enables organizations to identify and resolve IT infrastructure problems before they affect critical business processes." Nagios collects information from plug-ins that reside on various hosts, aggregates the information, and then reports the results through e-mail, web browser or any other communications mechanism.

Nimbus - An open source middle-ware stack responsible for VM life-cycle management. "Nimbus is a set of open source tools that together provide an 'Infrastructure-as-a-Service" cloud computing solution. [sic] Nimbus allows a client to lease remote resources by deploying virtual machines (VMs) on those resources and configuring them to represent an environment desired by the user."

Python - A freely available, interpreted, object oriented programming language.

SSH Secure SHell - "A secure shell provides an encrypted connection to a remote computer system, most often running Linux or some Unix variant. This allows a remote user or program to interact with a server over the Internet or a local network. The encryption for the communication is most often handled by SSL - Secure Sockets Layer. SSL is a standard and widely used protocol to handle encrypted communications across a network connection."[11]

SQL Structured Query Language - "SQL is a standard language for accessing and manipulating databases."[12]

VM Virtual Machine - "The abstraction of a computer system's hardware and software components into one or more isolated 'slices'. These 'slices' provide an encapsulated interface to the underlying hardware and are independent from other virtual machines running on the same system. That is, a virtual machine is unaware of other virtual machines sharing the physical hardware and provides the same functionality and services even if the underlying hardware is modified."[13]

XSD XML Schema Definition - "An XML schema describes the structure of an XML document." Schemas are used for defining exactly what XML structures a document should contain, including tag cardinality, attributes and structure.[14]

XML eXtensible Markup Language - "XML is a markup language for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents."

XSLT eXtensible Style-sheet Language Transform - "XSLT is designed for use as part of XSL, which is a style-sheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary."

6 Bibliography

[1] M. Creeger, "Cloud Computing: An Overview", *ACM Queue*, Vol 7 No.5, June 2009

[2] Nimbus homepage, [Online]. Availble: http://workspace.globus.org [Accessed April 17, 2010]

[3] O'Reilly XML homepage [Online]. Available: http://www.xml.com/pub/a/98/10/guide0.html?page=2#AEN58 [Accessed April 25, 2010]

[4] W3C homepage, [Online]. Available: http://www.w3.org/DOM/ [Accessed April 17, 2010]

[5] W3C homepage, [Online]. Available: http://www.w3.org/TR/xslt [Accessed April 19, 2010]

[6] W3C homepage, [Online]. Available: http://www.w3.org/TR/xpath/ [Accessed April 28, 2010]

[7] Python project homepage, [Online]. Available: http://www.python.org [Accessed April 17, 2010]

[8] Redis project homepage, [Online]. Available: http://code.google.com/p/redis/ [Accessed April 29,2010]

[9] Nagios homepage, [Online]. Available: http://www.nagios.org [Accessed April 17, 2010]

[10] Evaluation of Nagios for Real-time Cloud Virtual Machine Monitoring, [Online]. Available: http://heprc.phys.uvic.ca/sites/heprc.phys.uvic.ca/files/reports/paterson-wtr-nagios.pdf [Accessed April 30, 2010]

[11] OpenSSH homepage, [Online]. Available: http://www.openssh.org [Accessed April 25, 2010]

[12] W3C Schools homepage, [Online]. Availble: http://www.w3schools.com/sql/sql_intro.asp [Accessed April 27, 2010]

[13] M. Rosenblum, "The Reincarnation of Virtual Machines", *ACM Queue*, Vol 2 No.5, July/August 2004

[14] W3C School homepage, [Online]. Available: http://www.w3schools.com/Schema/schema_intro.asp [Accessed April 17, 2010]