HEP Data-Intensive Distributed Cloud Computing System Requirements Specification Document

CANARIE NEP-101 Project
University of Victoria HEP Computing Group

December 18, 2013

Version 1.0

Revision History

Date	Reason for Changes	Version
30/10/2013	Skeleton.	0.1
18/12/2013	Content added.	1.0

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to present the system requirements specifications for the High Energy Physics Data-Intensive Distributed Cloud Computing **NEP-101** project. The document explains the purpose, features, and interfaces of the system as well as the constraints under which it operates. This document is intended for the stakeholders and the developers, and will be delivered to CANARIE as a work product of the NEP-101 project.

1.2 Scope

The NEP-101 project will extend the capabilities of the distributed cloud computing model to allow the processing of ATLAS production requiring large input data sets. Though focused on ATLAS production, the system extensions will make this computing model more applicable to a broader range of modern experiments. For example, the distributed compute cloud system is used by the Belle-II project at the KEK Laboratory in Japan.

Users of this system will have the ability to process workloads requiring large amounts of input data on clouds spread throughout the world, they will be able to manage their application and data resources, and they will be able to do this efficiently with the network resources.

The use of cloud computing and virtualization technology allows for a decoupling of site resources from the application environment. As such, a user will be able to take advantage of both public and private cloud resources that are made available on the globe and to take advantage of replicated copies of data stored around the world.

The ultimate outcome of the NEP-101 project is to provide an infrastructure where a user can submit jobs to a single work queue, have those jobs dispatched to various clouds, no matter where they are located, and have each job retrieve and process the required data from the nearest available repository.

1.3 Definitions, acronyms and abbreviations

APF	The ATLAS AutoPyFactory generates batch jobs in response to workloads within the ATLAS PanDA queue. These APF batch jobs are payloadless, that is they do not contain the application or data to perform ATLAS analysis or simulation, but they are used to secure computing resources in a distributed grid or cloud environment. Having secured the required computing resources, an APF job selects and pulls a unit of work from the PanDA queue for execution.
ATLAS	A HEP experiment at CERN laboratory in Geneva, Switzerland (http://atlas.ch)
Cloud	A collection of computing resources and software that provide on demand through Web Services, Infrastructure As A Service (IaaS) Virtual Machines (see below).
DCCM	Distributed Cloud Computing Model as developed by NEP-52 and enhanced by NEP-101.
HEP	High Energy Physics, sometimes referred to as Particle Physics or Nuclear Physics
Image Repository	A storage capability employed by the NEP-101 system to manage virtual machine images. Depending on its' type, an image repository may be dedicated to a single cloud or shared by multiple clouds.
NEP-52 [1]	HEP Legacy Data project (Oct 2009 - Mar 2012), created the DCCM for high throughput, modest data input/output serial processing.
NEP-101	HEP Data-Intensive Distributed Cloud Computing project (Oct 2013 - Dec 2014), will extend the DCCM for data intensive ATLAS applications.
Virtual Machine (VM)	A complete computer system represented in software

1.4 Overview

The rest of this document is organized as follows. Section 2 gives an overview of the functionality of the entire system. It describes the general structure of the system and its informal requirements. Section 3 takes a closer look at the constituent subsystems and, for each subsystem, provides an overview, a list of enhancements, and references to in depth technical information.

[1] https://wiki.heprc.uvic.ca/twiki/bin/view/Main/CanarieProjectNEP52

2 Overall description

The distributed cloud computing model established by NEP-52 provided researchers with both interactive and batch services for data analysis, VM image management, credential management, and low volume data access [1]. These services were particularly suited to large scale serial batch processing, consistent with image analysis and Monte Carlo simulation prevalent in the astronomy and physics disciplines. The NEP-101 project will extend this capability, by providing both new and enhanced services, to provide access to large volumes of data. By enabling-proximity aware data sources and by making optimal use of the networks on which this computing model is so dependent, these new features will target a whole new class of HEP processing and extend this system's applicability to a wider range of applications. In addition, modifications will exploit new features of the underlying open source components to improve usability and system performance.

The system, as modified by NEP-101, will have the following subsystems: Batch Services, Software Distribution, Storage Federation, VM Distribution, and VM Optimization.

Section 3 provides both a textual and graphical overview of these subsystems together with the specific requirements of each.

2.1 User Interfaces

Users will interact with various subsystems and components of the system with the following interfaces:

- **Web interfaces:** Where appropriate, DCCM components should offer web based dashboards accessible by standard web browsers for users to access and manipulate its' services.
- **Command line clients:** As with the case of all OpenStack [2] IaaS middle ware, each component should provide a command client to allow for scripting of the provided function.
- **Application Programming Interfaces (API):** DCCM components should expose the API on which both the web based interface and the command line client is based.
- **Linux commands:** Finally, some services of DCCM (e.g., batch job preparation, batch job submission, etc.), are provided through standard Linux [3] commands.

2.2 Constraints

All users must be authorized to use the system and be authenticated either by x509 certificates and certificate mapping, or by user ID and password.

2.3 Assumptions and dependencies

The system depends on the availability of virtualization software (e.g.. Xen or KVM), functioning IaaS software (e.g., OpenStack, Nimbus, Eucalyptus, Open Nebula), and job scheduling software which recognizes dynamically allocated resources.

[2] http://www.openstack.org/
[3] http://www.linux.org/

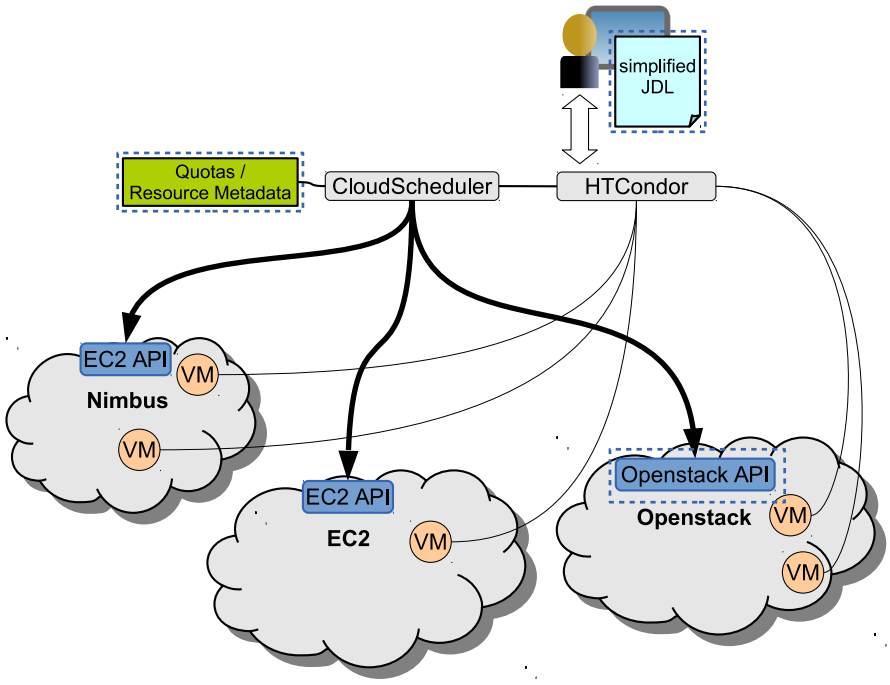

Figure 1: Batch Services Overview: The user prepares a text file containing Job Description Language (JDL) and submits it to the job scheduler (HTCondor). Cloud Scheduler monitors the job scheduler queue and starts and stops VMs as required by jobs in the queue. When a VM starts, it registers with the job scheduler and receives one or more jobs to execute. The area within the dotted rectangle represents new functionality to be provided by the NEP-101 project.

3 Specific requirements

This section is supplemental to the Use Case document. We can classify the specific requirements of the project into the following areas:

3.1 Batch Services

DCCM batch services provide the researcher with a traditional high throughput computing workflow. A researcher prepares a collection of batch jobs and submits them to a single job scheduler work queue for processing. The Cloud Scheduler [4] component of DCCM automates the creation of VMs to service the job scheduler work queue, distributing the work across multiple clouds, sharing the resources equally among users, and terminating VMs when they are no longer required. Cloud Scheduler will be extended in the following areas:

- **Native support for the OpenStack API:** Cloud Scheduler's support for a cloud type (OpenStack, EC2, Nimbus, etc.) is contingent on the availability of an Application Programming Interface (API). Currently, Cloud Scheduler uses the boto library bindings to the EC2 API which OpenStack supports. This API, though functional, does not provides access to all the services that OpenStack provides.

[4] http://cloudscheduler.org/

It is therefore desirable to implement the native OpenStack API which will allow simplified job definition, quota and resource management (see below). For detailed technical information, refer to the OpenStack API documentation found at http://api.openstack.org/api-ref.html

- **Simplified JDL:** When a user wishes to run batch jobs, they must describe the jobs to be run in a simple text file using a Job Description Language (JDL). The JDL identifies the program to be executed along with any arguments to be passed to the program and the disposition of the input and output data. In addition, when jobs are to run on clouds, the characteristics of the VM need to be described in multiple JDL parameters, many peculiar to each target cloud. When the number of target clouds were small, this could easily be achieved by assigning an array of values to a single parameter, for example, the VM image ID might be defined as "+VMAMI='cloudA:ami000041,cloudB:ami000037'". However, this becomes unwieldy and error prone when the number of target clouds reaches double digits. The goal of this enhancement is to eliminate the need for cloud specific specification within the JDL by providing a uniform VM description applicable to all supported cloud types.

- **Simplified Quota & Resource Management:** When an administrator configures a cloud to participate in the batch processing environment, the location, type, and resources (eg. processors, memory, storage, etc.) are statically defined in a system wide configuration file. During operation, the system tracks resource allocation to ensure the availability of resources for new VMs. The goal of this enhancement is to use the OpenStack API to query the resource utilization and availability to allow more accurate tracking of capacity.

For specific technical information regarding these enhancements, please refer to section 3 within the companion Use Cases document.

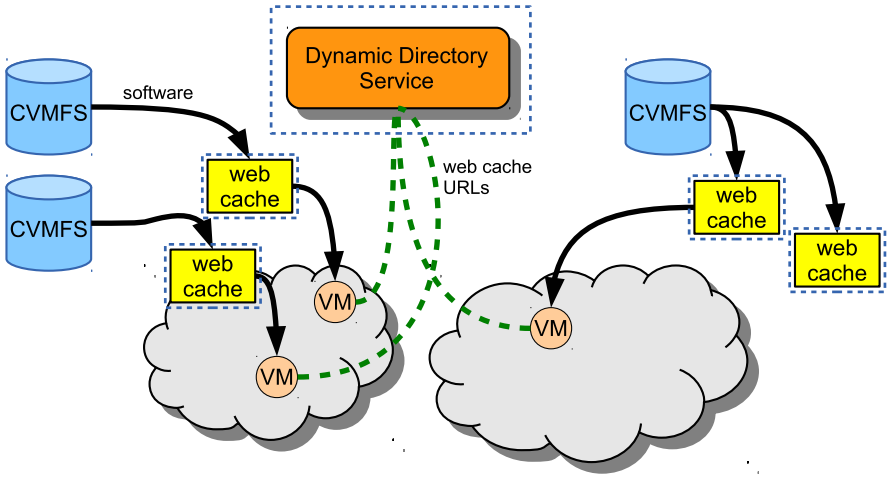

Figure 2: Software Distribution Overview: VMs requiring software query the Dynamic Directory Service for the location of the nearest Web Cache. Subsequently, the VMs retrieve software via the Web Cache. The area within the dotted rectangle represents new functionality to be provided by the NEP-101 project.

3.2 Software Distribution

The CVMFS server provides a way to simplify the distribution of software, to reduce the size of VM images, and increase the versatility of VM images by allowing a single image to run multiple applications. CVMFS application appliances are often created by application specialists and not often replicated to multiple locations. The remoteness of an application appliance somewhat negates the "transfer only what is needed" network efficiencies that is at the heart of the CVMFS philosophy. To counter this negation, web caches should be interposed to minimize the long haul network traffic and handle the many repetitive network requests locally. In addition, newly created VMs will need to connect to a single web cache to retrieve software from the CVMFS appliance. Since there will be many web caches, some kind of directory service is required to furnish newly created VMs with the nearest and least busy server available.

The following DCCM extensions are required:

- **Web caching:** Multiple web caching servers, capable of responding to VM web requests and contributing service information to a directory, need to be created and deployed across the DCCM.
- **Web cache location:** A dynamic directory service, one that tracks the availability, load and location of web cache servers needs to respond to requests from newly created VMs so that they can access software resources.

For specific technical information regarding these enhancements, please refer to section 4 within the companion Use Cases document.

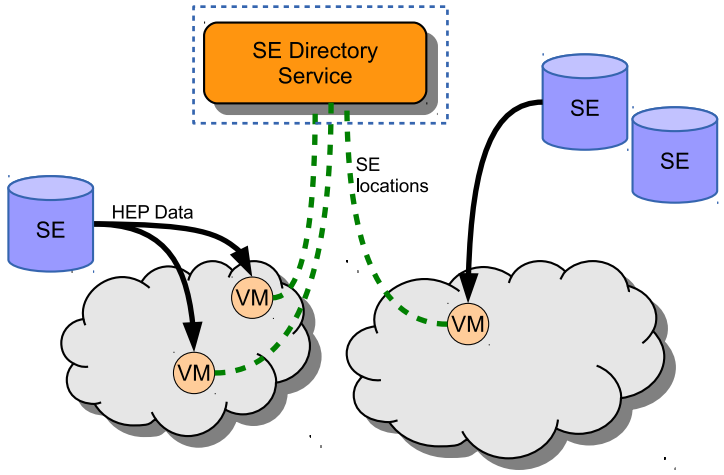

Figure 3: Storage Federation Overview: VMs requiring data ask the SE Directory Service for the locations of the nearest Storage Elements (SE, one or more depending on the location of the required data). Subsequently, the VMs retrieve data directly from the SEs. The area within the dotted rectangle represents new functionality to be provided by the NEP-101 project.

3.3 Storage Federation

To date, DCCM has run serial workloads requiring modest amounts of data. HEP processing has been confined to a particular class of work, Monte Carlo simulation, about thirty percent of the ATLAS project work load, with most of the modest data requirements being satisfied by one or two WebDAV servers. The goal of NEP-101 is to lift these barriers by incorporating other types of storage servers, or Storage Elements (SE) in ATLAS parlance, which are already distributed around the globe and contain vastly more data.

The following DCCM extensions are required:

- **Storage servers:** Data elements within the ATLAS storage servers are typically very large and may only be required by a few processes at a time. These attributes of the data make a caching solution non-viable as caches would be easily overwhelmed. However, these data elements are already replicated and disbarred around the world for data integrity reasons, and this fact should be leveraged to pull data from a "local" source.

- **Storage directory:** The advantages of having copies of data distributed to storage facilities around the world can only be fully realized with a comprehensive directory service. While the directory service does not have to deliver the actual data, it must be able to index new items as they are created and supply the requester with the properties and nearest location of all items cataloged. This service is required to provide jobs starting within VMs a single point of reference to access all the data they require from the nearest available storage facility, regardless of where any particular job is running.

For specific technical information regarding these enhancements, please refer to section 5 within the companion Use Cases document.

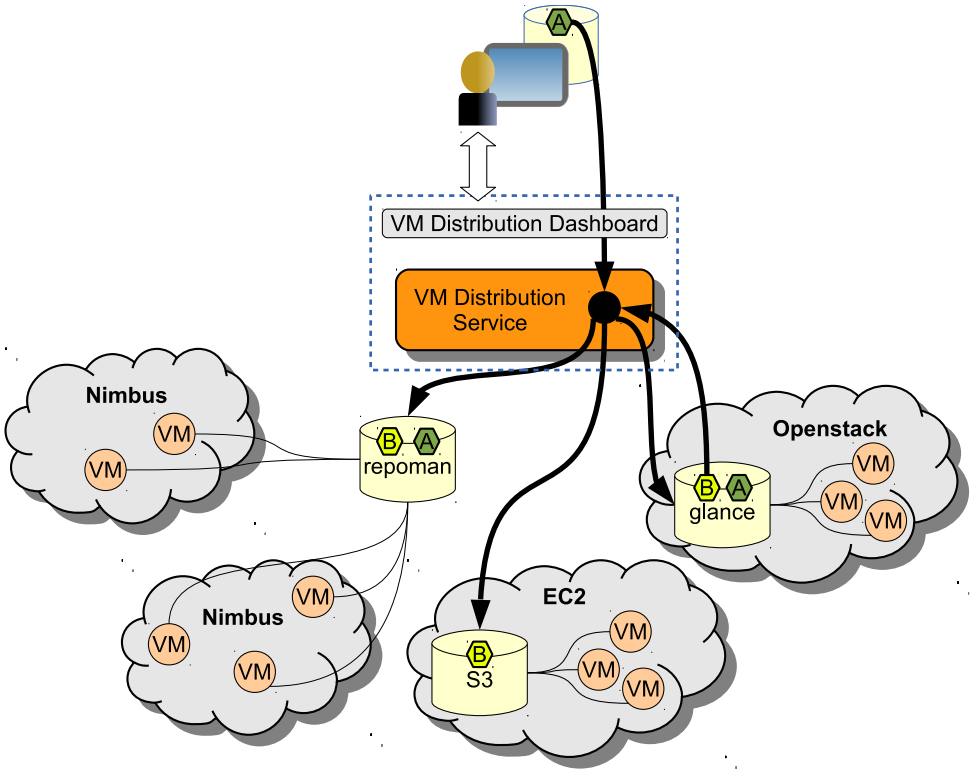

Figure 4: VM Image Distribution Overview: A user may create an image "A" on their workstation using standard Linux services, and then interact with the VM Distribution server to propagate that image to specific repositories. The user may also want to propagate an existing image "B" from one repository to several others. The area within the dotted rectangle represents new functionality to be provided by the NEP-101 project.

3.4 VM Distribution

Developed by NEP-52 and during that project's lifetime, DCCM was served, almost exclusively, by a single image repository. This was possible because the preferred Nimbus IaaS cloud middleware used to build many of the clouds exercised by the project allows for image propagation and instantiation through the HTTP protocol. The image repository manager Repoman was created as a gatekeeper to control access to the globally accessible images that it maintained and to manage any associated metadata. In more recent years, other IaaS cloud solutions have come to prominence, each of which require images be stored locally in a domain specific image repository. Since a fundamental concept of DCCM is to exploit many clouds regardless of location, and since many of these clouds now require a local copy of an image, users are required to propagate their images across the infrastructure before they can exploit DCCM.

Clearly, if the user has to manage a large number of images across many clouds, they need help and the following DCCM extensions are required (see Figure 4):

- **VM distribution service:** The VM distribution service is responsible for copying images from a source to all required destinations as identified by the user. It will be necessary for the service to identify the requester in order to restrict access to authorized users and to assign the user the correct

roles and privileges. In addition, the service will need to manage the users credentials for the variety of endpoints the user has access to. The service should provide a "window" on the resources consumed by the user, providing cloud-centric and image-centric reports. The service should also facilitate automation, allowing for the propagation of images without manual intervention and triggered by an event such as image modification.

- **VM distribution dashboard:** The primary user interface to the VM distribution service is a web based dashboard accessible through a web browser. The dashboard should provide dialogs for resource usage and credential management as well as propagation and image management services.
- **VM distribution client:** A secondary user interface to the VM distribution service is a command line client with access to same set of services provided through the dashboard. This interface is provided to allow scripting of the VM distribution services.
- **VM distribution API:** The VM distribution API is the foundation for the two client interfaces and is responsible for all interactions between a client and the VM distribution service. The API should employ well defined standards, such as JSON and XML, for communication between the application and the server.

For specific technical information regarding these enhancements, please refer to section 6 within the companion Use Cases document.

3.5 VM Optimization

The NEP-101 project will look for opportunities to optimize VM image performance in terms of image propagation, boot time, image re-usability, and image execution performance. The employment of CVMFS was an example of optimization. This one change reduced image size significantly by removing the application code from the saved image resulting in storage savings and improved image propagation time.

For specific technical information regarding these enhancements, please refer to section 7 within the companion Use Cases document.

References

[1] Randall Sobie, Ashok Agarwal, Ian Gable, Colin Leavett-Brown, Michael Paterson, Ryan Taylor, Andre Charbonneau, Roger Impey, and Wayne Podiama. 2013. HTC scientific computing in a distributed cloud environment. In Proceedings of the 4th ACM workshop on Scientific cloud computing (Science Cloud '13). ACM, New York, NY, USA, 45-52. DOI=10.1145/2465848.2465850 http://doi.acm.org/10.1145/2465848.2465850