Experiences designing and building a PRAGMA Cloud Scheduler

[Short Paper]

Shava Smallen
San Diego Supercomputer Center
University of California San Diego
ssmallen@sdsc.edu

Nadya Williams
San Diego Supercomputer Center
University of California San Diego
nadya@sdsc.edu

Philip Papadopoulos
San Diego Supercomputer Center
University of California San Diego
phil@sdsc.edu

ABSTRACT

PRAGMA is a community of research scientists and institutions from around the Pacific Rim that work together to enable scientific expeditions in areas of biodiversity distribution and lake ecology. Over the past four years, the technology focus for PRAGMA partners has shifted to cloud and software defined networking as enabling technologies. During PRAGMA 27 meeting, the Resources Working Group discussed rebooting the persistent PRAGMA Testbed as a way for sites to contribute and use shared resources, leveraging technologies such as PRAGMA Boot, Personal Cloud Controller (PCC), and virtual network overlays (i.e. IPOP and ViNe). To make PRAGMA resource sharing easier, a lightweight scheduler was proposed and discussed as a way to enable access to the resources and to manage resources reservations. This short paper discusses the design process and initial prototype of a simple cloud scheduler for PRAGMA.

Keywords

Cloud, Scheduling, Resource Sharing, Virtualization

1. INTRODUCTION

The Pacific Rim Application and Grid Middleware Assembly (PRAGMA) researches actively collaborate to enable scientific expeditions in areas of computational chemistry, telescience, biodiversity, and lake ecology [11]. Founded in 2002, PRAGMA originally sought to advance the use of grid technologies in applications among a community of investigators working with leading institution sites around the Pacific Rim [12]. This included the deployment of a shared PRAGMA Grid testbed where participating sites could contribute and use resources as needed to develop and test new middleware and to conduct scientific experiments. Testbed sites needed to install at a minimum Globus [1] and a local HPC job scheduler as well as other optional software such as MPICH-G2 [4] and Ninf-G [17]. In November 2006, nineteen sites in thirteen countries were part of the testbed. By 2009, the testbed had grown to twenty-seven sites in fifteen countries with a total of 1008 CPUs, more than 1.3 terabytes of memory, and over 24.7 terabytes of online storage. However, as of 2011, the number of sites started to decline and maintaining the numerous and complex middleware and scientific applications at each local site required significant people effort and expertise. Since PRAGMA participants often have varying levels of of funding, staff, and expertise, PRAGMA started to shift away from grid towards cloud technologies to simplify the infrastructure and to lower the amount of effort any site would need to participate in the testbed.

The first phase of the PRAGMA cloud testbed started with three sites and focused around the creation of application-specific virtual machines and making them portable to different cloud hosting environments like Rocks [10], OpenNebula [7], and Eucalyptus [8]. One early example was an AutoDock [2] virtual cluster created for the Avian Flu Grid research team in 2011 [13]. By 2012, the cloud testbed had ten sites with a total of 367 CPUs, 2.5 terabytes of memory, and 657 terabytes of online storage. In 2013 PRAGMA shifted its focus to the creation and management of virtual clusters to make it easier for users to assemble a multi-node virtual environment for running their scientific experiments. The following technologies have been explored to facilitate the operation and use of virtual clusters on the PRAGMA cloud testbed.

Pragma_boot: rather than requiring a single cloud system at all sites, PRAGMA encourages the sites to deploy any virtualization technology that best fits their expertise. However, enabling a virtual cluster to be ported to different cloud hosting environment is a complex problem and requires tooling to retain the cluster relationship between head and worker nodes as well as software configurations. In 2013, three pilot sites (UCSD, AIST, NCHC) developed a script for an automated virtual cluster porting and demonstrated the same virtual cluster image being booted in three different cloud hosting environments, including Rocks/Xen, OpenNebula/KVM, and Amazon EC2. Based on those experiences, the porting script was redesigned and reimplemented as *pragma_boot* toolkit with "drivers" to support virtual cluster migration to virtual environments enabled

on Rocks and OpenNebula clusters. In 2014, a feature was added to pragma_boot to download and boot virtual cluster images from Amazon CloudFront [**?**].

Personal Cloud Controller: to provide users with an easy-to-use interface for managing the life cycle of their virtual clusters, a Personal Cloud Controller (PCC) was started in 2014. This lightweight tool was designed to manage startup, status monitoring, and shutdown of a virtual cluster and was built on top of pragma_boot and a well-known resource and workload management system HTCondor [5]. PCC also provided an option to create a multi-site virtual cluster leveraging an open-source virtual network overlay software IPOP [3] for creating its private network.

Software-Defined Networking: PRAGMA began investigating software-defined networking technologies such as OpenFlow [6] in 2013 as a way to create a private network between multi-site virtual clusters and to protect access to sensitive datasets [14]. PRAGMA then created the Experimental Network Testbed (PRAGMA-ENT) as a breakable, international testbed for use by PRAGMA researchers and collaborators. In 2014, The PRAGMA-ENT team worked to create an international reliable direct point-to-point data Layer-2 connection with network engineers and developed AutoVFlow [18] that allows to provision private virtual network slices for each application, user, and/or project [15].

In late 2014 during the PRAGMA27 workshop, a lightweight scheduler was proposed to coordinate virtual hosts and clusters running on the different cloud deployments via leverage the above described technologies. The next section discusses the general requirements of the cloud scheduler and Section 3 discusses the design options that were considered and why a simple calendar solution was selected. A summary of calendar systems that were examined is provided in Section 4. Section 5 discusses our early implementation with one of the calendar systems called Booked and Section 6 concludes with our planned future work.

2. SCHEDULER REQUIREMENTS

PRAGMA had the following three key requirements for a cloud testbed scheduler to enable multiple users access to the resources at a given time:

Low participation overhead: One key requirement for the cloud testbed scheduler was that it had to be lightweight and required minimal effort and minimal expertise for a site to add their cloud deployment to the list of resources available for scheduling. Similarly, the scheduler would need to work across multiple cloud deployments (e.g., Rocks, OpenNebula, and OpenStack [9]) so that sites do not have to learn and install specific cloud deployment tools in addition to their currently used.

Easy to use: Currently, to execute a virtual cluster on the PRAGMA cloud testbed, users needs to contact each site individually to gain access to their cloud deployment, upload their images, and launch their virtual cluster either using pragma_boot if it is available or manually via a sequence of defined steps. The design of the PRAGMA cloud scheduler should first simplify this process by ensuring that pragma_boot gets deployed more broadly and port it to more cloud environments as needed. The scheduler should also provide a simple Web interface for users to see the available resources and sites, construct their virtual cluster, and manage their images.

Scale to tens of users: While the PRAGMA community was composed of nineteen active institution sites in 2014, the initial number of expected users for the cloud testbed is in the tens of users, not hundreds nor thousands. Therefore, we can prioritize simplicity over scalability and give higher priority to the requirements of low participation overhead and ease of use.

3. SCHEDULER DESIGN

In our design discussions, we first looked at the existing solutions that could be adapted for PRAGMA's scheduling needs:

Existing HPC schedulers: Open source batch schedulers like Slurm [19], TORQUE [16], and HTCondor are commonly used in HPC to manage distributed shared resources. While these batch schedulers are widely used, they either do not work across multiple sites (TORQUE) or are too heavy weight for what PRAGMA needs, requiring custom configuration and multiple daemons at each site (Slurm and HTCondor). Furthermore, to be adopted by PRAGMA users who are scientists in many different domains, we wanted a low barrier to entry like a simple Web based interface for scheduling, rather than the command-line tools and queue abstractions that these batch schedulers provide.

Grid'5000: NEEDS TO BE WRITTEN. Add short description of Grid'5000. They use OAR which is a batch scheduler similar to above and the calendar gui called GridPrems is one of three options for interacting with Grid'5000. A key difference is that testbed resources are dedicated rather than like on PRAGMA where they could be shared; I believe also single admin model. Interesting set of tools. We can say we liked their calendar approach the best.

DHCP Leases: NEEDS TO BE WRITTEN.

GENI/PlanetLab: NEEDS TO BE WRITTEN. ORCA

Out of these approaches, we liked the Grid'5000 calendar reservation interface, GridPrems, for scheduling the best. The calendar reservation scheduling is an interface that all users are familiar with in their everyday lives and would be well suited to the different PRAGMA users who have varying levels of cloud expertise.

4. CALENDAR SYSTEMS

Since we were primarily interested in a scheduler with a Web based calendar reservation interface like GridPrems, we decided to look at other open source calendar reservation systems that were available. We looked at several systems used primarily for tracking room and/or equipment reservations and evaluated the following criteria:

Open Source and customizable: Open source software is generally preferable because it provides it is free to use and can be customized as needed. Since we did not know of any existing Web interfaces for managing virtual clusters, we

limited our search to customizable Open Source tools and evaluated how easy it would be to manage resources and users as well as add new fields and features.

Nice GUI interface: We evaluated the GUI interfaces of several calendar reservation tools and looked for those that had intuitive menus and navigation and a clean and uncluttered look. Some of the GUI interfaces we looked at just used basic HTML tags and looked very clunky so we limited our search to tools that leveraged CSS and Javascript and looked more professional.

Installation and maintenance: We downloaded a number of tools and reviewed their installation instructions to see how easy they were to setup a demo instance. Most of the tools we looked at were developed in PHP and had a relational database backend that was could be served from a basic Apache web serverc̃iteapache setup. A few of the tools were plugins for Drupal. However, we eliminated a number of tools because they had either too little documentation or complex installation instructions with several manual steps.

We searched for candidate reservation tools using Google search and also searching specifically on Github and SourceForge. The above criteria filtered out several tools.

One category of tools we looked at were plugins to existing frameworks like Drupal [**?**] and Wordpressc̃itewordpress. We looked at Booking_calendar for Wordpress as well as Merci and Reservations for Drupal One category of tools that we looked at were developed in PHP and had a relational backend database such as Postgres an MySQL. We looked at Meeting Room Booking System on SourceForge [**?**], Booked from Twinkle Toes Software, and Classroombookings on Github,

Discuss selection of Booked.

Open source Easy to setup Nice GUI interface Report features REST API Customizable-ish LDAP and Active Directory support. Fine tuned roles and permissions. User and group quotas.

Can only handle one reservation per resource at a time PHP changes can be painful (heavy OO makes it hard to find right files) Doc is sparse

http://mrbs.sourceforge.net PHP, Postgres/MySQL Demo was a bit slow and a bit clunky looking Booked - http://www.twinkletoessoftware.com/products Nice interface, lot of customization features PHP, MySQL RESTFUL API http://classroombookings.com/ Nice interface PHP, MySQL http://www.unitime.org/unitime_intro.php Geeky looking, might have some good algorithms Interface not as smooth http://journal.code4lib.org/articles/2941 Clunky looking http://www.drupalrooms.com More for hotel web sites ? e.g., prices, etc. https://groups.drupal.org/node/137544 Merci - https://www.drupal.org/project/merci For equipment, etc. https://www.drupal.org/project/reservations Also seems pretty flexible

tried a few tools here is the summary: 1. OTA hotel management http://rocks-86.sdsc.edu/hotelmis/setup/initialsetup.php appears to be just for the hotel, no easy login, no easy change. Not useful 2. Mybooking - in Malay and after changing the language and trying enenglishigsh i did not get far. – hotel 3. Booking_calendar - strictly room booking, very small in features, too much to rewrite. – wordpress plugin 4 Optaplanner appears to be a java app that runs outside of web. I tried examples and i am not sure of its usability. The examples are running producing sometimes some output sometimes noe and they are just java apps on some data. The examples are not impressive even in terms of giving an idea how things are run.

A few more links are on http://opensource.com/business/15/1/top-project-management-tools-2015 Started looking at one of them, Tuleap, may be too big for what we need.

5. CLOUD SCHEDULER PILOT

One key requirement for our cloud scheduler was to enable users to start and manage virtual clusters on the PRAGMA cloud testbed with minimal effort. The basic workflow of how a user would start a virtual cluster and the steps are summarized below:

1. A user does a general search for available resources based on the number of CPUs, memory size, and networking configuration (i.e., IPOP, ViNE, or PRAGMA-ENT) that is needed.
2. Available resources that fit the criteria are displayed to the user and their availability is shown in a calendar layout.
3. The user selects the subset of resources they want, the timeframe, virtual cluster image, and configuration details and submits it to the Cloud scheduler. Once the reservation is verified and confirmed, the user will get an email confirmation.
4. When the reservation is ready to be activated, the cloud scheduler will use PCC and pragma_boot to launch the virtual cluster, automatically configure the private and public network configuration on the virtual cluster nodes, and email the user when it's ready for them to login.

Figure 1 shows the above steps in a use case sequence diagram where *PRAGMA Booking* refers to our customized Booked instance. PCC will monitor the health of the virtual cluster and when the reservation is close to expiring, will notify the user in case they want to extend their reservation. Otherwise, when the time expires or the user chooses to end their reservation, the virtual cluster will be shut down.

The other key requirement for the cloud scheduler was that it should require minimal effort for a PRAGMA sites to participate in the PRAGMA testbed. Therefore, we opted for a simple client-server architecture, where the client is a small PRAGMA package that interfaces to a site's local cloud system. Currently, the PRAGMA Package is just a SSH service and the pragma_boot package, which is a small Python package that can be installed via RPM and can be configured in less than an hour. The server component is composed of the PRAGMA Booking interface and the PCC

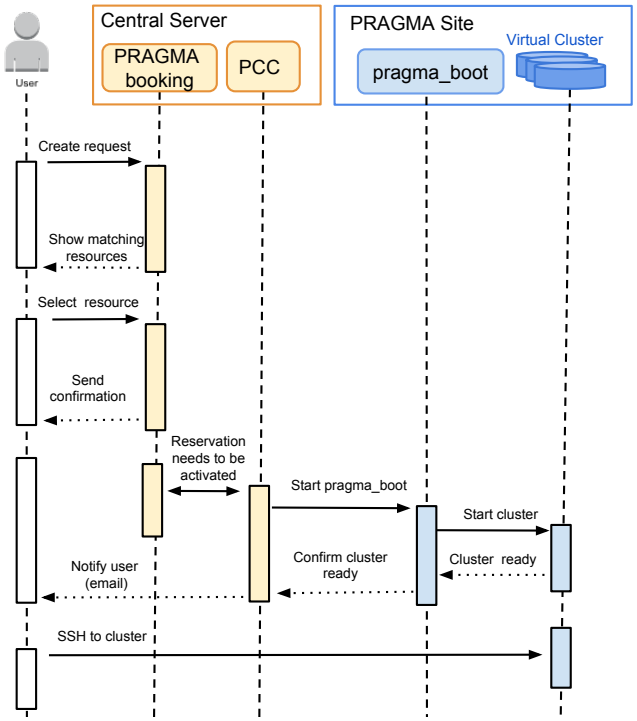

Figure 1: The flow of a user request to start a virtual cluster and the components that are involved in the cloud scheduler.

which gets launched for per user. The architecture for the PRAGMA cloud scheduler is shown in Figure 2.

In our pilot implementation of the cloud scheduler, we made a number of temporary assumptions to simplify the design.

- Only one reservation can be made per resource. While we made several changes in our pilot to customize the look of Booked, modifying it to allow for multiple reservations per resource would have required deeper changes that we wanted to defer to a later version.
- Virtual cluster images are already available at each site. For our pilot, we worked with a small number of virtual cluster images and pre-installed them to each cloud system so they were readily accessible by pragma_boot. We had the following three images: 1) a basic SGE Rocks virtual cluster, 2) an AutoDoc virtual cluster, and 3) a Lifemapper compute virtual cluster.
- Since PCC is also under active development and was not easy to install, we used a simple stub to substitute for its functionality. The stub is a small Python script called *pcc-check-reservations* that polls PRAGMA Booking via its REST interface and looked for reservations that are ready to be activated. Once it finds a reservation that need sto be activated, it launched pragma_boot via SSH. It does not monitor the virtual cluster nor email the user when their reservation is about to expire.
- Only single site virtual clusters can be launched. So far, we've only experimented with multi-site virtual clusters using IPOP and this requires the virtual cluster images to be pre-installed with IPOP and an automated configure script that will assign private IP addresses to each node on boot. For Rocks virtual clusters, there is a Rocks roll [**?**] that will automate the IPOP installation and install the configure scripts.

cloud scheduler would leverage To create a lightweight scheduler that enabled users to start and manage virtual clusters on leveraged the pragma_boot, PCC, and software-defined networking tools that had been developed as part of PRAGMA As Our To schedule virtual clusters on the PRAGMA cloud testbed, Discuss creation of Cloud Scheduler Pilot.

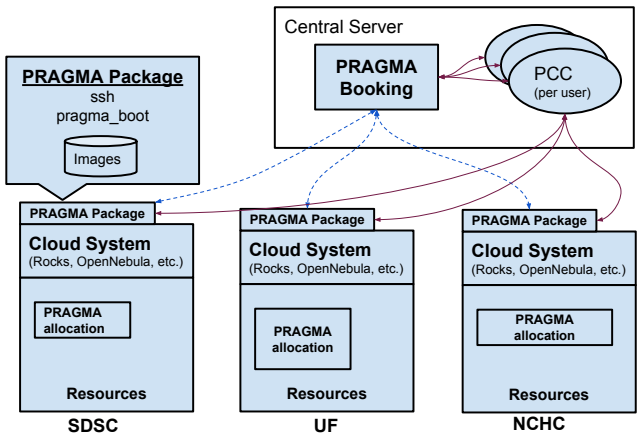

Figure 2: Architecture for the PRAGMA cloud scheduler.

6. CONCLUSIONS AND FUTURE WORK

Some interesting conclusion.

7. ACKNOWLEDGMENTS

The authors would like to thank Jose Fortes and Mauricio Tsugawa for their help during our initial design discussions and for providing resources at University of Florida during development.

8. REFERENCES

[1] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.

[2] D. S. Goodsell and A. J. Olson. Automated docking of substrates to proteins by simulated annealing. *Proteins: Structure, Function, and Bioinformatics*, 8(3):195–202, 1990.

[3] P. S. Juste, D. Wolinsky, P. Oscar Boykin, M. J. Covington, and R. J. Figueiredo. Socialvpn: Enabling wide-area collaboration with integrated social and overlay networks. *Comput. Netw.*, 54(12):1926–1938, Aug. 2010.

[4] N. T. Karonis, B. Toonen, and I. Foster. Mpich-g2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing*, 63(5):551–563, May 2003.

[5] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *ICDCS*, pages 104–111, 1988.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[7] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer*, 45(12):65–72, 2012.

[8] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.

[9] OpenStack Open Source Cloud Computing Software Website. `http://www.openstack.org`, 2015.

[10] P. M. Papadopoulos, M. J. Katz, and G. Bruno. Npaci rocks clusters: tools for easily deploying and maintaining manageable high-performance linux clusters. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 10–11. Springer, 2001.

[11] The Pacific Rim Application and Grid Middleware Assembly Website. `http://www.pragma-grid.net`, 2015.

[12] PRAGMA Collaboration Overview 2003-2004, November 2004.

[13] Collaboration Overview: Annual Report 2010-2011, November 2011.

[14] Collaboration Overview: Annual Report 2012-2013, November 2013.

[15] Collaboration Overview: Annual Report 2013-2014, November 2014.

[16] G. Staples. Torque resource manager. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 8. ACM, 2006.

[17] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka. Ninf-g: A reference implementation of rpc-based programming middleware for grid computing. *Journal of Grid computing*, 1(1):41–51, 2003.

[18] H. Yamanaka, E. Kawai, S. Ishii, and S. Shimojo. Autovflow: Autonomous virtualization for wide-area openflow networks. In *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pages 67–72. IEEE, 2014.

[19] A. B. Yoo, M. A. Jette, and M. Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.