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's testbed has shifted to cloud and software defined networking as enabling technologies. This short paper describes the design of a simple cloud scheduler built on top of a Web reservation system that will enable users to easily run and manage virtual clusters for their science. It is also equally important that there be a low barrier to entry for PRAGMA institutions to become part of the PRAGMA cloud testbed and install the needed software. Therefore, the cloud scheduler has a simple client-server architecture and will be built upon existing technologies such as pragma_boot, Personal Cloud Controller, virtual network overlays (e.g.,. IPOP), and and also a Web reservation system. We discuss our experiences implementing our pilot cloud scheduler and then describe future work.

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 [25]. 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 [26]. 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 [11] and a local HPC job scheduler as well as other optional software such as MPICH-G2 [16] and Ninf-G [31]. 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 [24], OpenNebula [20], and Eucalyptus [22]. One early example was an AutoDock [12] virtual cluster created for the Avian Flu Grid research team in 2011 [27]. 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 ex-

periences, 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 [7].

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 [17]. PCC also provided an option to create a multi-site virtual cluster leveraging an open-source virtual network overlay software IPOP [15] for creating its private network.

Software-Defined Networking: PRAGMA began investigating software-defined networking technologies such as OpenFlow [18] in 2013 as a way to create a private network between multi-site virtual clusters and to protect access to sensitive datasets [28]. 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 [35] that allows to provision private virtual network slices for each application, user, and/or project [29].

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 **??**. 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 [23]) 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 some existing solutions that might be adapted for PRAGMA's scheduling needs. Open source batch schedulers like Slurm [36], TORQUE [30], or HTCondor, are used to manage space shared resources. There are also other testbeds like Grid'5000 [1], GENI [3], and PlanetLab [5] that have their own tools for using and managing their infrastructure. In general, we found these tools had different goals and were too heavy weight for PRAGMA. Some tools were better documented than others but all had a fairly high learning curve and would have been non-trivial to adapt for PRAGMA's cloud testbed, which had a strong application and virtual cluster focus.

We next looked more broadly for a simpler approach to scheduling and found a number of open source Web-based calendar reservation systems that seemed promising. These systems were designed to manage meeting room reservations, hotel room reservations, or equipment rentals. Their interfaces would be also be familiar to users in their everyday lives and thus well suited to the different PRAGMA users who have varying levels of distributed systems and networking expertise. Also, several of these tools were significantly less complex than the batch and testbed scheduling tools above. Later, we found a Web-based calendar reservation system called GridPrems [13] for Grid'5000 indicating a simpler interface was sometimes needed even for their users.

Some of the calendar reservation systems we looked at like Reservations [10], Merci [19], and Booking_calendar [34] were plugins for Drupal [9] or Wordpress [33]. However, the majority of tools were PHP based with a relational database backend like the Meeting Room Booking System [21], Booked [4], and Classroombookings [6]. For each system, we evaluated 1) how easy it would be to manage resources, reservations, and users as well as add new fields and features, 2) the GUI interface with respect to intuitive menus/navigation as well as a clean, modern, and uncluttered look, and 3) how easy it was to install and setup a demo instance. One deficiency that all systems had was that they only allowed one reservation per resource.

Out of the calendar reservation systems we evaluated, Booked [4] from Twinkle Toes software seemed the best suited. It was easy to setup, had a nice looking GUI interface, and was customizable. It also had some nice additional

features like usage reporting, a REST API interface, LDAP and Active Directory support, fine grained roles and permissions, and user/group quotas. After working with Booked during our pilot, we did find some weaknesses. First, the documentation was a bit sparse in areas and any needed PHP changes were time consuming due to its heavily object-oriented structure (i.e., we had to go thru many files to find the right place to change the code).

4. SCHEDULER ARCHITECTURE

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

5. SCHEDULER PILOT

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

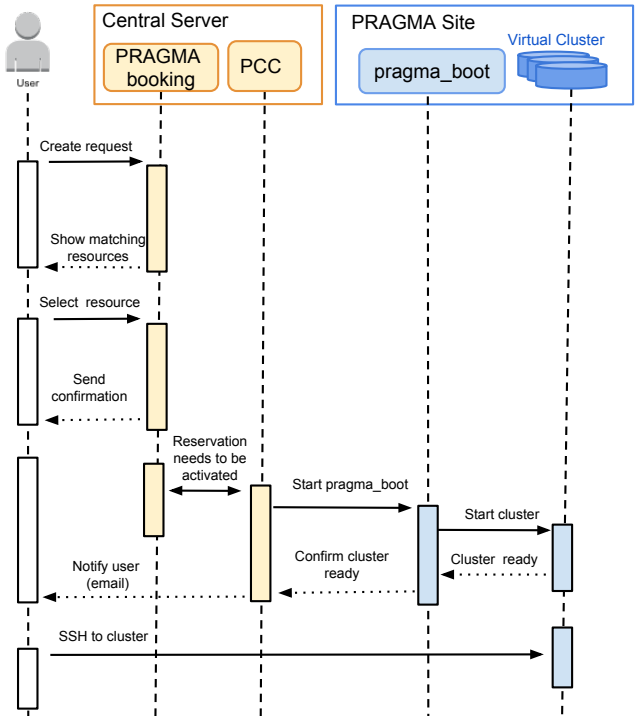

Figure 1: The flow of a user request to start a virtual cluster and the components that are involved in the cloud scheduler.

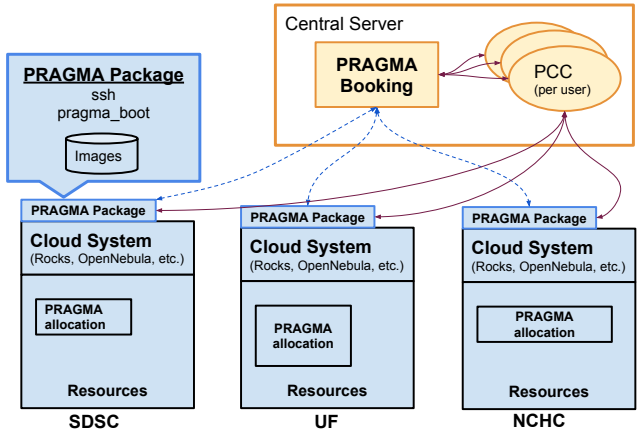

Figure 2: Architecture for the PRAGMA cloud scheduler.

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 [2] compute virtual cluster.

Developed and used a PCC stub: 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 launches 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 [14] that will automate the IPOP installation and install the configure scripts. Since IPOP is installed at the user-level, it can be used by any PRAGMA site but some sites may support better performing network overlays created with ViNE [32] or software-defined networking tools. Therefore, our long-term goal is to automatically select the most appropriate network overlay depending on the sites that a user wants to include in their virtual cluster where IPOP is the lowest common denominator.

Much of our work in this pilot focused on creating the PRAGMA Booking interface. This required the following customizations to Booked:

- Added a custom field for a public SSH key text to the user profile. When PCC launches a virtual cluster, it will pull the user's public SSH key from their profile and give it to pragma_boot so that the user can login as root once the virtual cluster is launched.
- Added custom fields for CPU count, amount of memory per host, and virtual cluster image name to reservations.
- Added custom fields for CPU count, amount of memory per host, site hostname, and ENT-enabled to resources.
- Added a numeric count as a custom field type. Booked only provided text matching on search so if a host had 16 CPUs available and the user requested 8 CPUs, the search would fail since "16" != "8". By adding count as a custom field type, the search would do a numeric comparison on the values and the search would succeed since 8 <= 16.
- Added custom reservation statuses: "Starting", "Running", and "Stopping". Each reservation status is shown as a different color in the calendar view.
- Added the ability to retrieve and set the reservation status from the Booked REST API.
- Added the PRAGMA logo to the header.

We also fixed the following bugs in Booked and plan to work with the developer to integrate them back into the distribution:

- Fixed bug that allowed users from making reservations for past time frames.
- Fixed a time conversion bug that was preventing updates to existing reservations.
- Fixed bug where the username was getting set to a blank value when a reservation was updated via the REST API.
- Fixed bug that would not recognize zero as a valid value (e.g., specifying a virtual cluster with just a frontend and 0 compute nodes).

Some sample screenshots of the PRAGMA Booking user interface is shown in Figure 3. Likewise, Figure 4 shows some screenshots of the administrations options and usage reports. Our customizations for Booked were packaged into a Rocks roll [8], which automates its installation and publishes some sample data.

6. CONCLUSIONS AND FUTURE WORK

This paper described the the design and pilot implementation of a scheduler for the PRAGMA cloud testbed. Since the scheduler does not need to scale beyond tens of users, we prioritized ease of use and low installation/maintenance overhead and opted for a simple client-server design. The server has an intuitive Web GUI frontend based on the Booked Web reservation system software.

The pilot implementation of the cloud scheduler made a lot of temporary assumptions and so our future work addresses them. We will first enhance the Booked software to handle multiple reservations and then work on integrating IPOP and PRAGMA-ENT into the cloud scheduler as well. Next, we will rework the PCC software and integrate HTCondor so it's used in personal mode allowing us to keep a light footprint on each of the PRAGMA sites. Finally, we will investigate using the CloudFront option in pragma_boot to manage application virtual cluster images and staging to each of the sites. Documentation and packaging will also be done throughout so that new users and sites can get quickly onto the PRAGMA cloud testbed and use it as an experimental service to test new science and infrastructure tools..

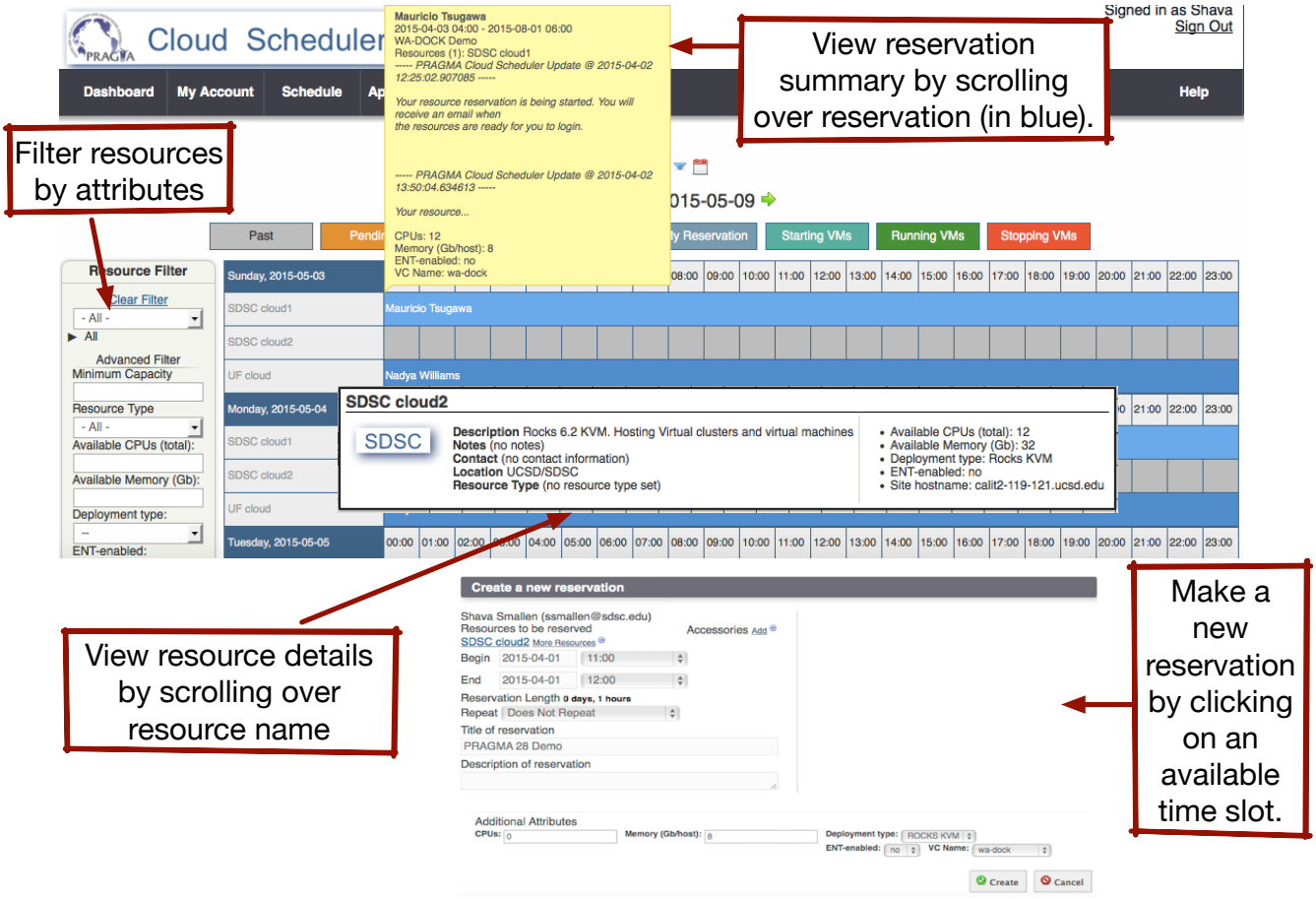

Figure 3: Screenshots of the PRAGMA Booking user interface showing how a user can view and filter resources, view reservation summaries, and create a new reservation.

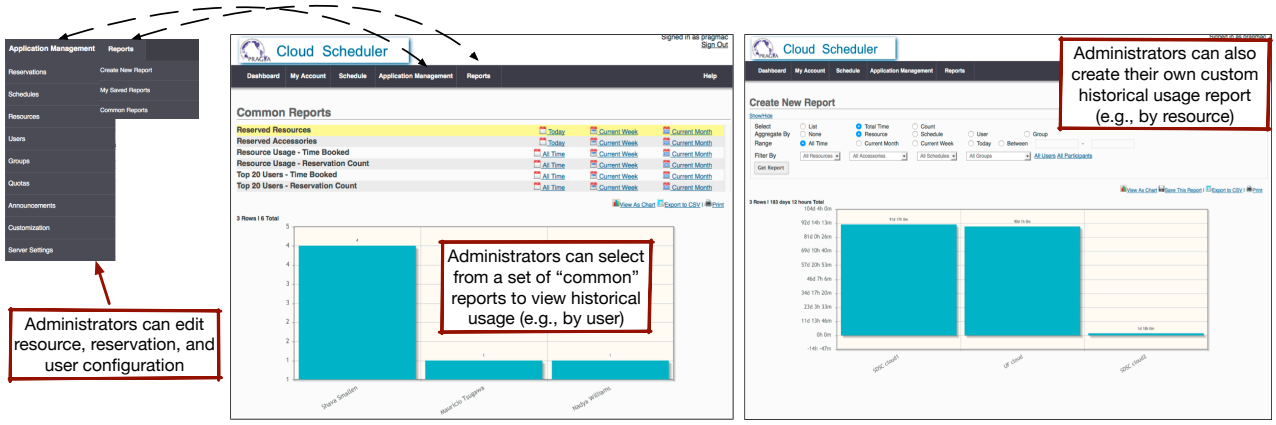

Figure 4: Screenshots of the PRAGMA Booking admin and reporting interface showing where an administrator can edit configuration details and then view usage reports.

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] D. Balouek, A. C. Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, et al. Adding virtualization capabilities to the gridâĂŹ5000 testbed. In *Cloud Computing and Services Science*, pages 3–20. Springer, 2013.

[2] J. Beach, A. M. Stewart, Y. Voronstov, R. Scachetti Pereira, D. R. Stockwell, D. A. Vieglais, and S. R. Downie. The future of biodiversity: Mapping and predicting the distribution of life with distributed computation. In *Proceedings of the ESRI User Conference*, 2002.

[3] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61(0):5 – 23, 2014. Special issue on Future Internet Testbeds âĂŞ Part I.

[4] Booked Website. `http://www.bookedscheduler.com/`, 2015.

[5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: An overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, July 2003.

[6] Classroombookings: the open source hassle-free room booking system for schools. `http://classroombookings.com/`, 2015.

[7] Amazon CloudFront CDN - Content Delivery Network & Streaming Website. `http://aws.amazon.com/cloudfront/`, 2015.

[8] PRAGMA Cloud Scheduler Rocks Roll Github Repository. `https://github.com/pragmagrid/cloud-scheduler`, 2015.

[9] Drupal Website. `https://www.drupal.org`, 2015.

[10] Drupal Reservations Website. `https://www.drupal.org/project/reservations`, 2015.

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

[12] 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.

[13] InriaForge: GridPrems Website. `https://gforge.inria.fr/projects/gridprems/`, 2015.

[14] IPOP Rocks Roll Github Repository. `https://github.com/pragmagrid/ipop`, 2015.

[15] 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.

[16] 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.

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

[18] 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.

[19] Drupal MERCI (Manage Equipment Reservations, Checkout and Inventory) Website. `https://www.drupal.org/project/merci`, 2015.

[20] 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.

[21] Meeting Room Booking System Website. `http://mrbs.sourceforge.net`, 2015.

[22] 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.

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

[24] 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.

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

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

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

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

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

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

[31] 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.

[32] M. Tsugawa and J. A. Fortes. A virtual network (vine) architecture for grid computing. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10–pp. IEEE, 2006.

[33] Wordpress Website. `https://wordpress.org/`, 2015.

[34] WordPress Booking Calendar Plugin Webiste. `https://wordpress.org/plugins/booking/`, 2015.

[35] 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.

[36] 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.