Lightweight Scheduling for the PRAGMA Cloud Testbed

Shava Smallen[1]*, Nadya Williams[1], Philip Papadopoulos[1]

[1] *San Diego Supercomputer Center*
University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093

SUMMARY

The Pacific Rim Application and Grid Middleware Assembly (PRAGMA) is a community of research scientists and institutions from around the Pacific Rim that work together to enable scientific expeditions in areas such as biodiversity distribution and lake ecology. PRAGMA's members collaborate on a testbed infrastructure and over the past four years, the technology focus has shifted to cloud and software defined networking as enabling technologies. This paper describes the design of a web-based cloud scheduler reservation system that enables users to easily run and manage virtual clusters for their science. Based on a client-server architecture, the cloud scheduler is designed to require minimal installation and management effort for participating PRAGMA testbed sites. The PRAGMA cloud scheduler is built using a web-based room reservation tool called Booked and leverages several PRAGMA technologies such as pragma_boot, Personal Cloud Controller, and virtual network overlays (e.g., IPOP, ViNe). We discuss our implementation of the pilot cloud scheduler and then describe future work. Copyright © 2010 John Wiley & Sons, Ltd.

1. INTRODUCTION

The Pacific Rim Application and Grid Middleware Assembly (PRAGMA) researchers 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 for applications among a community of investigators working with leading institution sites around the Pacific Rim [27]. 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 the Globus toolkit [11], a local HPC job scheduler as well as other optional software such as MPICH-G2 [15] and Ninf-G [32]. 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 expend to participate in the testbed.

---

*Correspondence to: University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093. E-mail: ssmallen@sdsc.edu

The first phase of the PRAGMA cloud testbed started with three sites and focused on 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 in 2011 was an AutoDock virtual cluster created for the Avian Flu Grid research team [28]. 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. To make it easier for users to assemble a multi-node virtual environment for running their scientific experiments, PRAGMA shifted its focus in 2013 to the creation and management of virtual clusters. The following technologies have been explored to facilitate the operation and use of virtual clusters on the PRAGMA cloud testbed.

Pragma_boot: Rather than require a single cloud system at all sites, PRAGMA encourages sites to deploy any virtualization technology that best fits their expertise. However, enabling a virtual cluster to be ported to different cloud hosting environments 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 to automate 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 these experiences, the original prototype was redesigned and reimplemented as a *pragma_boot* toolkit with "drivers" to support Rocks and OpenNebula [26]. 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 researchers with an easy-to-use interface for managing the life cycle of their virtual clusters, a Personal Cloud Controller (PCC) was started in 2014 [17]. 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 [16]. PCC also provided an option to create a multi-site virtual cluster leveraging an open-source virtual network overlay software IPOP [14] for creating its private network.

Software-Defined Networking: PRAGMA community members began investigating software-defined networking technologies such as OpenFlow [18] in 2013 as a way to create a private network among multi-site virtual clusters and to protect access to sensitive datasets. 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 a reliable international Layer-2 connection with network engineers and developed AutoVFlow [36] that can 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 and leveraging the above described technologies. The work presented in this paper extends earlier work we presented at a PRAGMA workshop in October 2015 [30]. The next section discusses the general requirements of the cloud scheduler and Section 3 discusses the design options that were considered. The architecture of the cloud scheduler is provided in Section 4 and Section 5 discusses our pilot implementation. Section 6 concludes with our planned future work.

2. SCHEDULER REQUIREMENTS

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

Low participation overhead: One key requirement for the cloud testbed scheduler was that it be lightweight and require 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 systems (e.g., Rocks, OpenNebula, and OpenStack [23]) so that sites do not have to learn and install additional cloud deployment tools.

Easy to use: Currently, to execute a virtual cluster on the PRAGMA cloud testbed, users must 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 systems 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 [37], TORQUE [31], 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 infrastructures. 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 searched 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 more familiar to users in their everyday lives and thus well suited to the different types of PRAGMA users. In addition, 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 [12] 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 [35] were plugins for Drupal [9] or Wordpress [34]. 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 to add new parameters and features; 2) how intuitive the GUI interface was with respect to menus and navigation and if it had a clean, modern, and uncluttered look; 3) how easy it was to install and setup a prototype instance. One deficiency that all systems had was that they only allowed one reservation per resource.

Booked [4] from Twinkle Toes software seemed the best suited to our needs compared to all the calendar reservation systems we evaluated. It was easy to setup, had a good look and feel, and was customizable. It also had some nice additional features like usage reporting, support for different time zones for each user, a REST API interface, LDAP and Active Directory support, fine grained roles and permissions management, and user and group quotas. After working with Booked during our pilot setup, we did find some weaknesses. First, the documentation was very sparse and any needed PHP changes were time consuming due to its heavily object-oriented structure. For example, we had to go through many source code files to find the right place to change or add functionality. Second, the reservation frequency of the event is currently limited to one hour. Based on today's PRAGMA needs this is sufficient because the time it takes to start up the virtual cluster is in the order of tens of minutes and the user reservation is expected to be in the order of days. In the future when the virtual cluster startup time shortens to minutes, the reservation frequency granularity can be optimized in order to avoid a potential of the underutilized resources.

4. SCHEDULER ARCHITECTURE

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

1. A user does a general search for available resources based on the number of CPUs, memory size, and networking configuration (e.g., IPOP, ViNE [33], 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 desired resources, the timeframe, virtual cluster image, and configuration details and submits them to the Cloud scheduler. Once the reservation is verified and confirmed, the user will get an email confirmation. In the event two users try to reserve the same resource at the same time, for example, one user has taken the resources while the second user was filling in the reservation fields, the second user will get a failure notification after hitting the submit button.
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 the reconfigured cluster is ready.

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 via email. The user then has an option to extend the reservation if resources are available. Otherwise, when the time expires or the user chooses to end the 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 with a site's local cloud system. Currently, the PRAGMA Package consists of an SSH service and pragma_boot, 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 PCC, which gets launched for each 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 simplifying assumptions to the design.

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 AutoDock virtual cluster; 3) a Lifemapper [2] compute virtual cluster.

Developed a stub for managing virtual clusters: To provide an interface for managing virtual clusters, we used a simple stub to emulate the functionality of PCC. The stub serves as a generic interface; it is a small Python script called *pcc-check-reservations* that polls PRAGMA Booking via its REST interface and monitors reservations. Once it finds a reservation that needs to be activated, it launches pragma_boot via SSH. It does not monitor the virtual cluster nor does it email the users when their reservation are about to expire.

Only single site virtual clusters can be launched: So far, we have only experimented with multi-site virtual clusters using IPOP. When virtual cluster images have been IPOP-enabled, IPOP and an automated configure script will assign private IP addresses to each node on boot. For Rocks virtual clusters, there is a Rocks roll [13] that will automate the IPOP installation and install the configure scripts. IPOP can be used by any PRAGMA site but some sites may have support for better
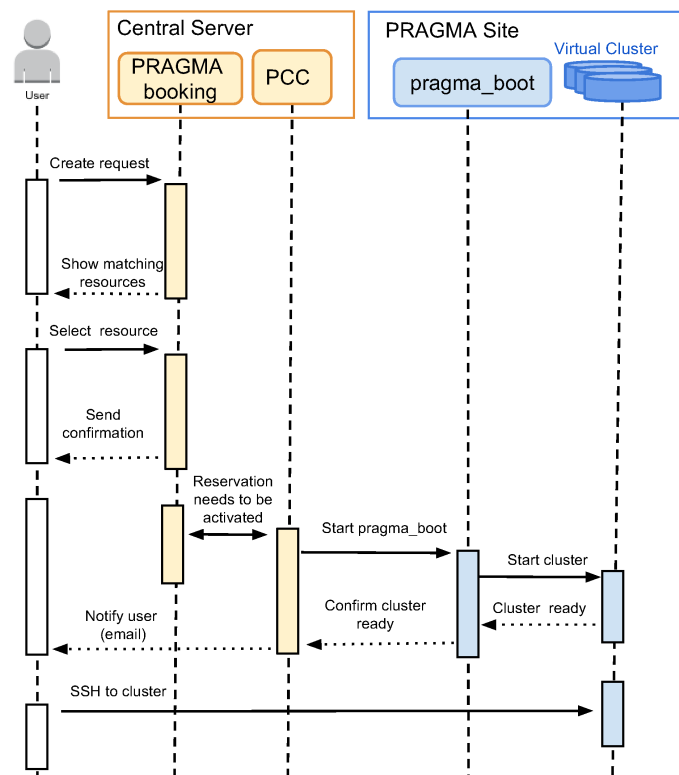
Figure 1. The flow of a user request to start a virtual cluster and the components that are involved in the cloud scheduler.
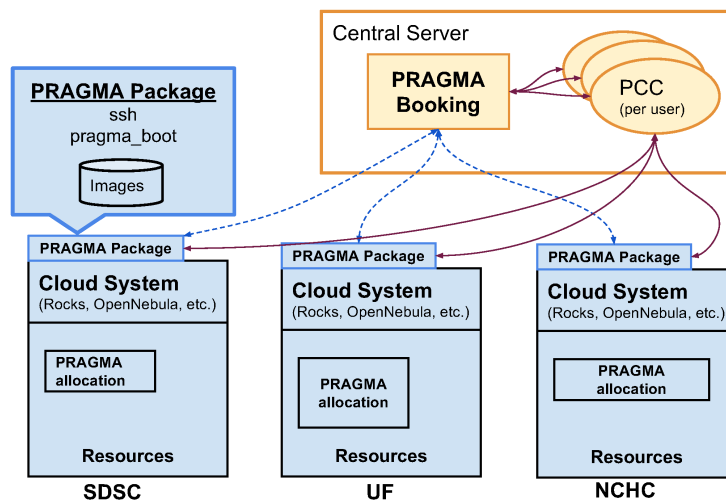

Figure 2. Architecture for the PRAGMA cloud scheduler.

performing network overlays created with ViNE or software-defined networking tools. Therefore, our long-term goal is to automatically select the most appropriate network overlay depending on the virtual cluster specification and leave IPOP as the lowest common denominator or default.

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

- Added the ability to make multiple reservations on a resource per time slot to allow more than one virtual cluster to run simultaneously on a site. This was the most extensive change we made to Booked. Fortunately, no changes were needed to the backend database but it required several changes to the GUI code to allow an additional reservation to be made on a resource and to display multiple reservations per time slot in the calendar views. In total, enhancements were made to seven PHP files and two template files.
- Added a custom field for a public SSH key to the user profile. When PCC launches a virtual cluster, it pulls the public SSH key from the user's profile and gives it to pragma_boot so that the user can login as root once the virtual cluster is launched. The ability to use a root login is not required for all but can be needed in some case if users want to do system modifications. A user with a root login has a full ownership of the virtual cluster.
- Added custom fields for CPU count, amount of memory per host, and virtual cluster image name to the reservation specification.
- Added custom fields for CPU count, amount of memory per host, site hostname, and ENT capability (enabled, disabled) to the resource specification.
- 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 per "16" != "8" inequality. By adding count as a custom field type, the search would do a numeric comparison on the values and the search would succeed for "8 <= 16".
- Added virtual cluster statuses: "Starting", "Running", and "Stopping". Each virtual cluster status is shown as a different color in the calendar view. "Starting" means the virtual cluster has been accepted by the scheduler and is being started up on the local cloud system, "Running" means that the virtual cluster is available for the user to log into, and "Stopping" means the virtual cluster is scheduled for shutdown and is no longer available for log in.
- 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 to make 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).

We still need to address the race condition scenario where two users might try to reserve the same resources at the same time. Booked currently does not have any mechanisms in place that can handle two simultaneous user requests for the same resources gracefully. We also do not handle hardware failures at resource sites. When a virtual cluster goes down, the failure will be recorded but it will not be relocated to a different site. We do plan to address the fault tolerance of the cloud scheduler by deploying a backup instance at a different site so that we can failover when needed.

Some sample screenshots of the PRAGMA Booking user interface are shown in Figure 3. Likewise, Figure 4 shows some screenshots of the administrations options and usage reports. We created a Booked Rocks roll [8] which packages the software with our customizations and automates its installation and publishing of sample data.
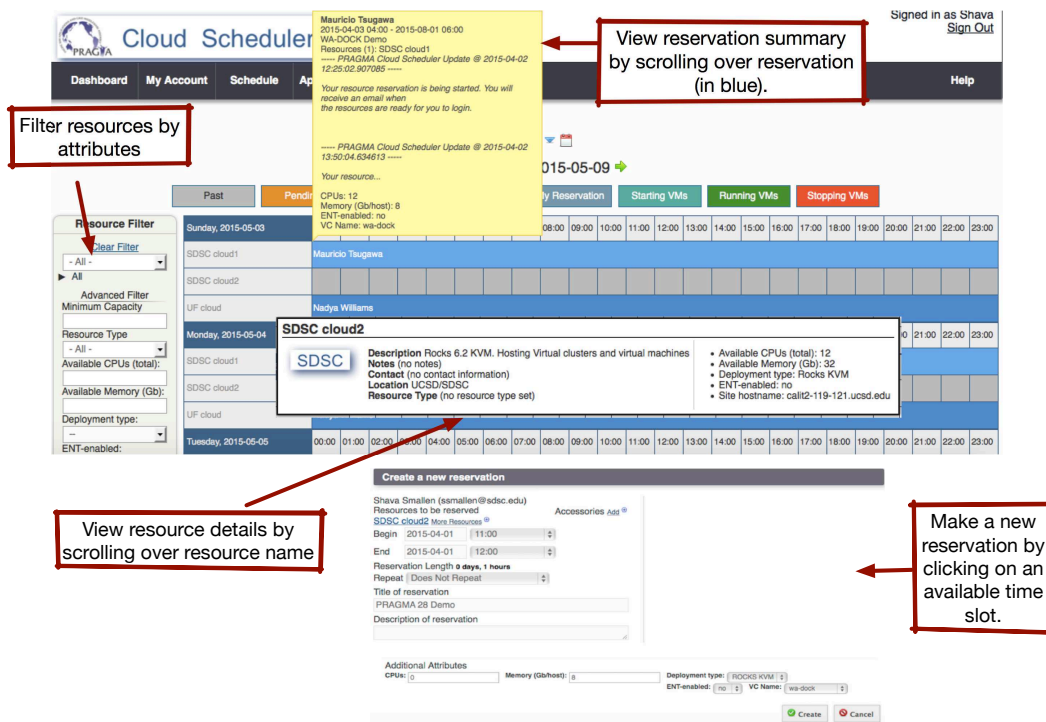
Figure 3. Screenshots of the PRAGMA Booking user interface showing how a user can view and filter resources, view reservation summaries, and create a new reservation.
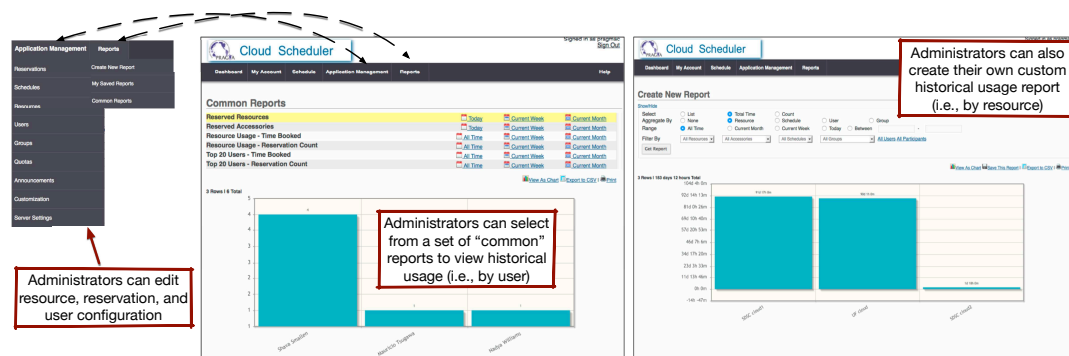

Figure 4. Screenshots of the PRAGMA Booking admin and reporting interface showing where an administrator can edit configuration details and then view usage reports.

6. CONCLUSIONS AND FUTURE WORK

This paper describes 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, low installation and maintenance overhead, and thus opted for a simple client-server design. The server has an intuitive web GUI front-end based on the Booked web reservation system software.

In the pilot implementation of the cloud scheduler we made a lot of simplifying assumptions, and our future work will address these. Currently, the virtual cluster startup latency from the point in time of its reservation to its availability to the user has not been analyzed in detail but we observed it to be on the order of tens of minutes. Multiple factors influence this latency, such as the underlying

virtualization manager and load on the physical system where the virtual cluster is instantiated. There is additional latency if the cluster image is not available on the site as it will need to be downloaded from a remote repository and processed to make it available on the site. While this applies only to the first virtual cluster instantiation, it can add a considerable time latency depending on the cluster image size and the network speed from the site to the remote repository.

Once the cloud scheduler is deployed to more sites, we can gather statistics on its usability and user experiences with virtual clusters startup latencies in addition to its applicability to different scenarios. We will also work on integrating IPOP and PRAGMA-ENT into the cloud scheduler. Next, we will rework the PCC software and integrate HTCondor so it is used in personal mode allowing us to keep a light footprint at each of the PRAGMA sites. Finally, we will investigate using the remote repository option in pragma_boot to manage application-specific virtual cluster images and staging them to the sites. We will package and document the resulting software so that new users can quickly start using the PRAGMA cloud testbed as an experimental service to test new science and infrastructure tools.

7. ACKNOWLEDGMENTS

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 grid5000 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 Part I.
4. Booked Website. http://www.bookedscheduler.com/, 2016.
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/, 2016.
7. Amazon CloudFront CDN - Content Delivery Network & Streaming Website. http://aws.amazon.com/cloudfront/, 2016.
8. PRAGMA Cloud Scheduler Rocks Roll Github Repository. https://github.com/pragmagrid/cloud-scheduler, 2016.
9. Drupal Website. https://www.drupal.org, 2016.
10. Drupal Reservations Website. https://www.drupal.org/project/reservations, 2016.
11. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.
12. InriaForge: GridPrems Website. https://gforge.inria.fr/projects/gridprems/, 2016.
13. IPOP Rocks Roll Github Repository. https://github.com/pragmagrid/ipop, 2016.
14. 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.
15. 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.
16. M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *ICDCS*, pages 104–111, 1988.
17. Y. Luo. *Virtual cluster management for analysis of geographically distributed and immovable data*. PhD thesis, Indiana University, 2015.
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, 2016.
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, 2016.

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, 2016.

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, 2016.

26. PRAGMA Boot Github Repository. https://github.com/pragmagrid/pragma_boot, 2016.

27. PRAGMA Collaboration Overview 2003-2004, November 2004.

28. Collaboration Overview: Annual Report 2010-2011, November 2011.

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

30. S. Smallen, N. Williams, and P. Papadopoulos. Lightweight Scheduling for the PRAGMA Cloud Testbed. In *PRAGMA Workshop on International Clouds for Data Science*, Depok, Indonesia, October 2015.

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

32. 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.

33. 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.

34. Wordpress Website. https://wordpress.org/, 2016.

35. WordPress Booking Calendar Plugin Webiste. https://wordpress.org/plugins/booking/, 2016.

36. 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.

37. 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.