Scaling Virtual Organization Clusters over a Wide Area Network using the Kestrel Workload Management System

Lance Stout, Michael Fenn, Michael A. Murphy, and Sebastien Goasguen
School of Computing
Clemson University
Clemson, SC 29634-0974 USA
{lstout, mfenn, mamurph, sebgoa}@clemson.edu

ABSTRACT

This paper presents the results of using the Kestrel workload management system to test operating a Virtual Organization Cluster (VOC) across multiple sites. A Many-Task Computing (MTC) framework based on the Extensible Messaging and Presence Protocol (XMPP), Kestrel presents a special purpose scheduler that can offer better VOC scalability under certain workload assumptions, namely CPU bound processes and bag-of-tasks jobs.

Experimental results have shown that Kestrel is capable of operating a VOC of at least 1600 worker nodes with all nodes visible to the scheduler at once. When using multiple sites located in both North America and Europe, the latencies introduced to the round trip time of messages were on the order of 0.3 seconds. To offset the overhead of XMPP processing, a task execution time of 2 seconds is sufficient for a pool of 900 workers on a single site to operate at near 100% use. Requiring tasks that take on the order of 30 seconds to a minute to execute would compensate for increased latency during job dispatch across multiple sites.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Design, Experimentation

Keywords

MTC, XMPP, grid, distributed, cluster, scheduling, VOC

1. INTRODUCTION

When scaling a Virtual Organization Cluster (VOC) [1, 2, 3] from a single site to multiple sites across a Wide Area Network (WAN), several challenges arise, including Network Address Translation (NAT) traversal and increased latency. The VOC workload management system Kestrel has been designed to address these issues and to allow for easier scaling of VOCs. Thus, to show Kestrel's ability to scale a VOC across sites, two criteria must be met. The first is that Kestrel must be able to acknowledge and manage a large number of worker nodes across NAT boundaries, and the second is that the latencies involved in cross-site communication should not excessively impede job dispatch rates.

Based on the Extensible Messaging and Presence Protocol (XMPP) [4, 5], Kestrel uses a pull based model to initiate communications with an XMPP server using long lived TCP connections. While messages sent and received by Kestrel agents are pushed using the established TCP connections, the benefits of the initial pull request remain. Under this model, NAT traversal is not an issue since all requests originate behind the boundary. With the simplification in networking provided by XMPP, a special purpose scheduler can be created that does not have to implement NAT traversal on its own. By forming the equivalent architecture of an overlay network using XMPP, without having to use an actual overlay networking layer, Kestrel allows for scaling across physical sites a VOC that is expected to receive certain kinds of workloads, in particular, computationally intensive, bag-of-tasks jobs.

In high throughput computing, the focus is on long running tasks which overshadow the time spent during dispatch [6, 7]. However, for many-task computing where many short lived tasks are typically scheduled [8], the dispatch time can introduce significant overhead. For Kestrel to scale across sites, it should not be affected by latency during job dispatch, given tasks that take on the order of 30 seconds to a minute in duration to execute.

In the remainder of this paper, related work is discussed in Section 2. The VOC Model is described in Section 3. Section 4 discusses the Kestrel XMPP based workload management system. The procedures and results of the experiments are presented in Section 5. Some initial conclusions are presented Section 7.

2. RELATED WORK

The Condor High Throughput Computing (HTC) system is designed to both scavenge idle CPU cycles from machines and to use those cycles to execute computationally intensive jobs [6, 7], and is used globally to manage both grids and clusters. The architecture for Condor calls for direct connections between worker and job submission nodes in order to transfer job data and output. While sufficient when all machines are present in the same subnet behind a NAT

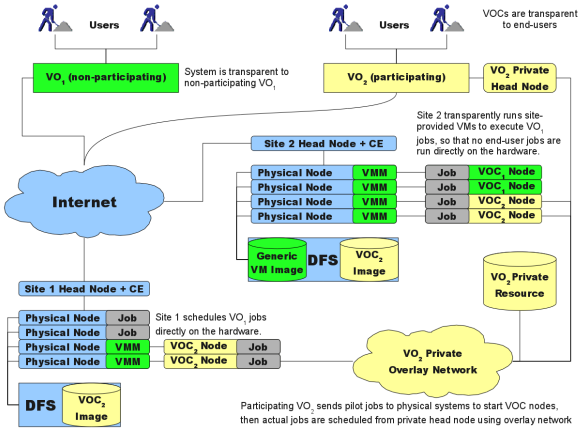

Figure 1: Virtual Organization Clusters can span multiple grid sites when an overlay mechanism is used

boundary, using Condor with machines on opposite sides of a NAT can introduce complications.

One method to allow Condor to work across NAT is IPOP, or Internet Protocol over Peer-to-Peer. IPOP implements a full networking stack on top of a P2P network that does not require any centralized control or routing configuration [9]. The result is a virtual network that can span across NAT boundaries. IPOP provides network transports that behave as Edge objects, enabling TCP, UDP, and TLS/SSL. In addition, its distributed tunneling system can compensate for some routing difficulties such as an un-traversable NAT or a firewall, without destroying the network topology.

As a method to parallelize the computation of optimized meander line radio frequency identification (RFID) antennas, an XMPP based scheduler was developed by Weis and Lewis [10]. The resource pool for the scheduler was a static list of available, physical machines. The managing XMPP client contacted each machine in the list that did not already have an XMPP client running, and start a new XMPP worker. While it uses XMPP for ad-hoc grid computing in a similar fashion to Kestrel, the system is not intended as a general purpose job distribution framework.

3. VIRTUAL ORGANIZATION CLUSTERS

Virtual Organization Clusters (VOCs) [1, 2, 3] enable the creation of virtual cluster environments that are compatible across sites, transparent to end users, implementable in a phased and non-disruptive manner, optionally customizable by Virtual Organizations, and designed according to an architectural specification. Since VOCs are constructed from virtual machines (VMs), and multiple VM instances can be spawned from a single image, VOC environments are nominally homogeneous (and therefore software compatible) across grid sites. Once operational, VOCs remain completely transparent to the end user, since the virtual environments are autonomically managed without explicit resource reservation requests. VOCs also remain transparent to Virtual Organizations and other entities that choose not to deploy them, allowing VOC implementations to be added to existing production grids without disrupting the operational infrastructure. Different technologies can be utilized to implement VOCs, since a VOC is simply an implementation of a system that conforms to the specifications presented in the VOC Model.

Jobs can be submitted directly to a VOC through one of several mechanisms. Dedicated grid interconnection middleware (such as Globus [11]) can be employed to permit the VOC to join the grid as a site dedicated to a single VO. Alternatively, VOs can provide direct submission mechanisms to permit members to submit workloads to the cluster without requiring the use of grid middleware. Regardless of the approach utilized to receive user jobs, VOCs autonomically provision themselves to adapt to changing user workload demands by means of pilot jobs that obtain physical resources from other grid sites. These pilot jobs start virtual machines, which are dynamically configured, or "contextualized" [12], at boot time. Once operational, the virtual worker nodes connect to a dedicated VO server and join the virtual cluster, enabling execution of user jobs according to policies set by the VO.

The mechanism by which VOCs are adapted to span multiple grid sites over a Wide Area Network varies by implementation. As illustrated in figure 1, the worker nodes of a VOC may be joined into a virtual private cluster by means of an overlay network such as IPOP [9], Violin [13], or ViNe [14]. Once joined together, a general purpose scheduling system such as Condor [6] may be used to distribute jobs to worker nodes. Alternatively, if the jobs to be executed by the virtual cluster do not require simultaneous interprocess communication (such as bag-of-tasks applications [15]), a scheduling system capable of direct operation over a Wide Area Network, such as Kestrel as described in Section 4, could be employed without the use of an overlay network.

4. KESTREL

Kestrel is a Many-Task Computing system based on the Extensible Messaging and Presence Protocol (XMPP) [4, 5] and is built using Python [16] and SleekXMPP [17]. It has been developed at Clemson University in order to explore methods of creating a fault-tolerant, scalable, and cross-platform job scheduling system [18]. The main problems addressed by Kestrel are intermittently connected machines and traversing of NAT boundaries.

Kestrel's architecture is composed of four types of resources: manager, workers, users, and XMPP servers. With the exception of the XMPP server, each resource is an XMPP agent program that can use an opaque Jabber ID (JID) [4] to join the system. These JIDs are similar in appearance to email addresses and are of the form `username@server/resource` where the resource component identifies each connection when an agent has multiple sessions open with the server. While Kestrel agents can use any JID, experience has shown that each worker in particular should have a unique bare JID (a JID without the resource identifier) to reduce the number of presence notices issued. Early experiments with Kestrel used a single bare JID for a thousand worker agents; however, because each agent sharing a bare JID knew of all other agents using the same bare JID, when every worker changed its status to indicate it was executing a task, the result was a million presence notifications which crippled the XMPP server.

User agents are client applications capable of submitting

job requests and querying the status of previous jobs. These agents can be highly transient, only connecting to submit a job and to check its status. Since there are no direct connections between a user agent and a worker agent executing a task, there is no requirement that a user remain constantly tethered to the pool. Since the protocol Kestrel uses for messages between agents is based on JavaScript Object Notation (JSON) [19], a user could manually submit commands to Kestrel using a stock instant messaging client such as Pidgin [20], Adium [21], or Google Talk [22].

Manager agents serve as the analogue to the collector and negotiator daemons in a Condor pool. As such, they serve as the centers of communication in a Kestrel pool, either receiving or sending all messages. These agents must normally be implemented as an external component to an XMPP server instead of as a normal XMPP client due to the large number of entries (one for each worker and user agent) that must be included in its roster [23]. For small pools with less than a thousand workers, a normal client would suffice.

Worker agents are started as services preinstalled on VOC nodes. Once connected to the server, each worker agent notifies the manager that it is available and sends a copy of its profile. The worker profile is a list of tags specifying both the abilities of the VM and the job data stored in the VM, which allows the manager to perform matchmaking if needed. By switching its status to "busy," the worker alerts the manager that tasks should not be scheduled for that agent. Likewise, setting its status to "available" causes the manager to dispatch a task to the agent if at least one is available. In the event that the VM is terminated, or the worker agent disconnects for any reason, then the XMPP server will issue an "unavailable" presence on its behalf to the manager which can immediately restart any task that worker had been executing.

5. EXPERIMENTAL RESULTS

Several experiments were performed to test the scalability and usability of VOCs. The first, detailed in Section 5.1, compared the size of pools that could be generated from an overlay network. The second experiment measured the latency involved in running a VOC across multiple sites over a WAN. The final experiment measured throughput and dispatch rates using the Kestrel system to manage a VOC. While there were several XMPP server implementations available, including ejabberd [24], Openfire [25], and Tigase [26], an Openfire installation was used for all Kestrel related experiments to take advantage of its web based interface (shown in Figure 6).

5.1 Pool Sizes

To establish the need for a special-purpose WAN scheduler, it was necessary to test the scalability of disk subsystems and general-purpose overlay networks by starting large number of VMs, as described in Section 5.1.1. To compare the behavior of a general-purpose overlay network to a purpose-built WAN scheduler, an overlay combination of IPOP and Condor (section 5.1.2) was tested against Kestrel (section 5.1.3) in an environment with pervasive NAT boundaries. If the number of VMs (and thus IPOP nodes) that could be instantiated on a TOP100 cluster did not exceed the limits of the general-purpose overlay network, then a special-purpose scheduler is unnecessary.

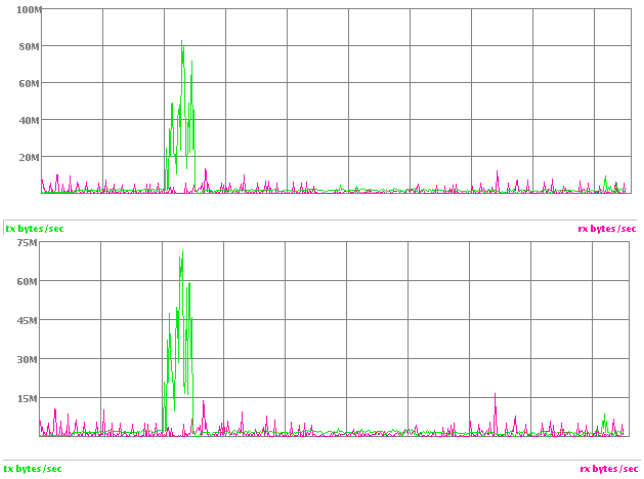

Figure 2: Aggregate Disk Throughput when starting 1000 VMs

5.1.1 VM Image Considerations

The simultaneous booting of thousands of VMs has the potential to place considerable strain on the test cluster's disk subsystem. This risk was mitigated by the use of the *snapshot* mode in QEMU/KVM. In snapshot mode, QEMU/-KVM did not write to the disk image from which the VM was instantiated. Instead, each instance of QEMU/KVM created a local copy-on-write file which overlaid the image of the original disk. Thus, many VMs could be instantiated from a single disk image. Furthermore, QEMU/KVM would only read blocks from this image as they were requested by the guest OS, greatly reducing the total amount of data that had to be transferred. The combination of these two factors allows the disk to be placed on a shared network store.

Results of a test that measured the aggregate disk throughput needed to boot 1000 VMs have been presented in Figure 2. A trace of the two 4Gb Fibre Channel ports connected to the Clemson University Palmetto Cluster's disk array was obtained from the storage management system. Packets were automatically load-balanced between the two ports, reaching a peak total transfer rate of approximately 154MB/s. This rate was well below the maximum capacity of the disk array. Thus it can be concluded that starting 1000 VMs with QEMU/KVM's snapshot mode did not place an onerous load on a cluster's disk subsystem.

5.1.2 IPOP/Condor Experiment

The scalability test for IPOP/Condor was conducted by starting a set of VMs using QEMU/KVM. Each VM utilized QEMU's user-mode networking stack for connectivity. This stack implemented a subset of the TCP/IP protocol and was used to provide network access to the VM without the need for administrative privileges on the host. Since the VM's network interface was not directly bridged to the the physical interface, the user-mode networking stack was required to implement NAT. With this setup, each VM was in a separate private network, maximizing the number of NAT traversals IPOP was required to provide. Furthermore, the Clemson Palmetto Cluster was located behind a NAT edge router to the Internet. Since the IPOP bootstraps were used for testing purposes, initial connections must be made through

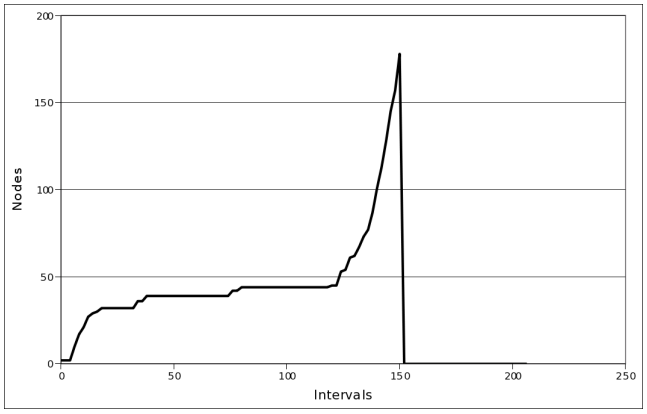

Figure 3: IPOP/Condor pool of 1600 nodes in one batch

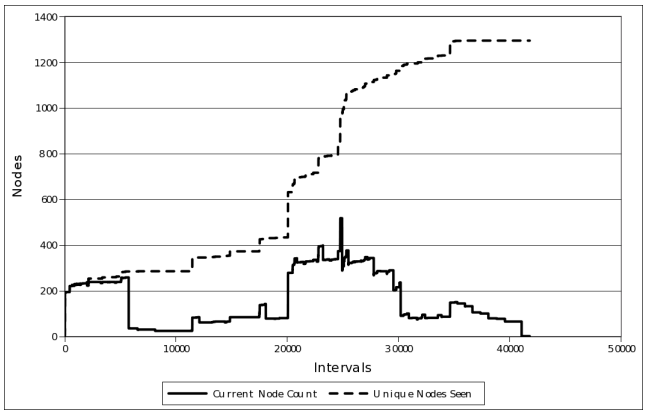

Figure 4: IPOP/Condor pool of 1600 nodes in 200 batches

two layers of NAT, further stressing the overlay network.

Two separate test scenarios differed in temporal characteristics. In the first test, a large number of IPOP nodes entered the overlay simultaneously. In the second test, nodes entered the overlay in a more incremental fashion. The total number of nodes was the same in both cases.

In order to measure the number of nodes which entered the overlay, an observation node was employed. This node monitored the status of the Condor pool. A worker node was considered to have entered the overlay whenever it appeared in the Condor pool.

As illustrated in Figure 3, the catastrophic failure of the overlay when 1600 nodes attempted to enter within a short interval. Figure 4 shows the results of 200 batches of 8 nodes each entering the overlay. The overlay remained active, but approximately 300 of the requested nodes never entered the overlay. Additionally the maximum number of nodes in the overlay at any one time was approximately 500.

These tests represented a worst-case scenario for IPOP because of the rapid arrival of many nodes behind two layers of NAT. In this situation, the overlay became unbalanced, collapsing entirely.

5.1.3 Kestrel Experiment

As a special-purpose scheduler based on XMPP, Kestrel

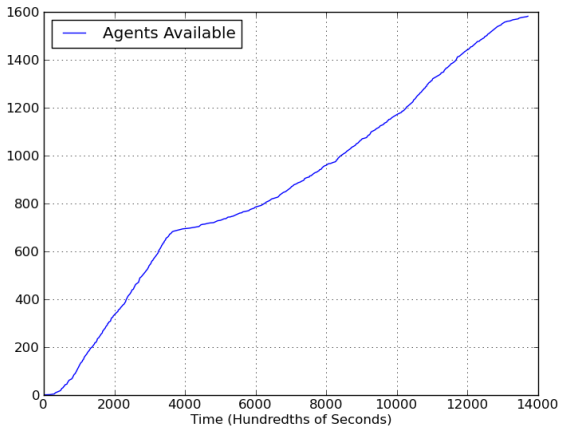

Figure 5: Kestrel Pool of 1600 Agents

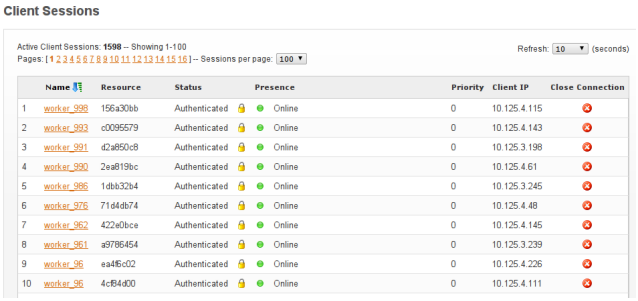

Figure 6: Openfire Showing Available Agents

did not have to implement a full, general-purpose networking stack like IPOP. Instead, Kestrel performed the specific function of task scheduling and distribution. However, applications distributed using Kestrel would not have access to a full network stack except for the one provided by the VM. To test how well Kestrel performed, 1600 VMs were instantiated on the Palmetto cluster using 200 physical nodes with 8 VMs per node. As a measure to reduce strain on the XMPP server, each worker waited for a random interval of up to 30 seconds before joining the pool. Since XMPP uses a pull-based model, the double NAT layer does not pose any issues.

Of the 1600 VMs submitted, 1598 Kestrel worker agents started properly and connected to the XMPP server. From the server's web interface all 1598 workers could be seen online at once.

A difficulty was initially encountered that prevented the Kestrel pool from growing beyond approximately 900 agents. The solution was to increase the number of open files allowed per process in `/etc/security/limits.conf` where the default value is 1024 files per process. In order to test the scalability of Kestrel, by allowing 16384 files per process and starting ten worker agents per VM, further experiments have shown that the XMPP server can accept at least 15,000 connections at once. However, Kestrel job dispatching has not yet been tested on a pool of that size.

5.2 Latency

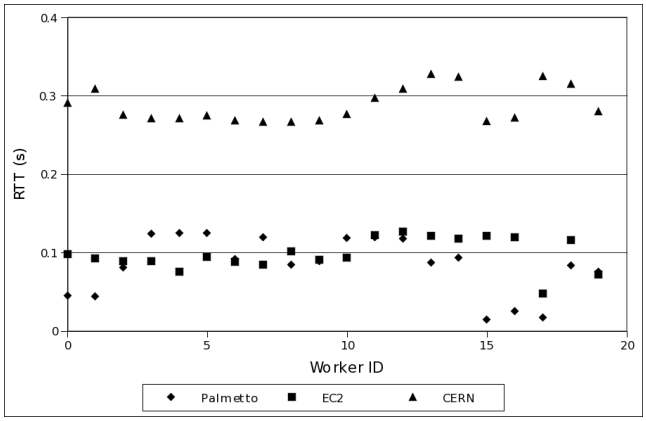

Figure 7: Ping Response Times

To determine the latency introduced by XMPP, a multi-site pool was started using Kestrel. The sites included Clemson University's Palmetto cluster, Amazon EC2, and CERN. The procedure for the experiment was to have a single node send a "ping" message to all the worker agents in the pool; to make processing easier, each "ping" message includes the time at which it was sent. These workers in turn replied to the sender with a "pong" message that included the sent time in the "ping" message. The current time was compared to the time stored in the "pong" message, and the round trip time (RTT) was calculated. Prior to running the experiment, the hypothesis was that it should be clear from a RTT graph that multiple sites were used in the pool since the data points should form clusters. A concern, however, was that the differences between Clemson's Palmetto cluster and Amazon's EC2 would not be distinct enough and would overlap.

The results of the experiment (illustrated in Figure 7) showed that a worker pool formed using XMPP can operate across multiple sites. The virtual machines at CERN responded in approximately 0.3 seconds whereas the the virtual machines on Palmetto and EC2 responded in roughly 0.1 seconds. The response times for Palmetto and EC2 overlapped considerably. The RTT times as measured through Kestrel were considerably longer than equivalent measurements through the system ping command, owing to the significant CPU overhead required to parse XML messages received via XMPP.

5.3 Kestrel Task Throughput and Dispatch Rates

In order to test the dispatch rate for Kestrel, four experiments were performed. Each experiment scheduled 50,000 sleep commands to about 900 worker agents. The first experiment used "sleep 0" to test Kestrel's performance under the most demanding workload of constantly scheduling. The second experiment used sleep with a half second delay, and the third used a two second delay. However, since a constant time for every task is not always accurate, a fourth trial used random length delays where each worker would sleep between 0 and 10 seconds.

For each run, the number of agents in three states (online, available, and busy) were tracked. The desired result would show the number of available workers mirror the number of online workers until the job was submitted. After the

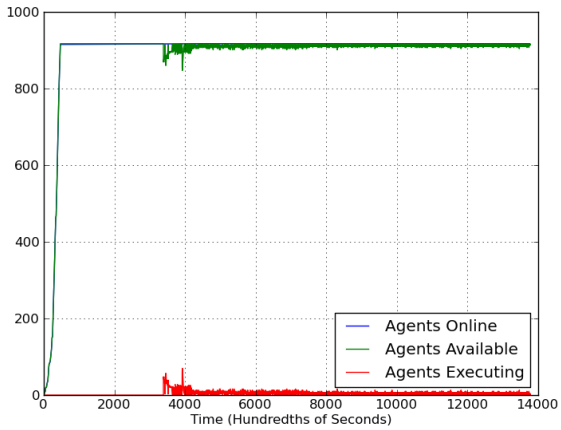

Figure 8: Trial #1 executing "sleep 0"

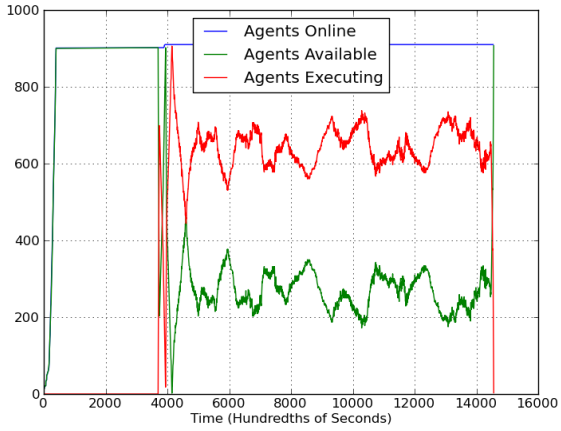

Figure 9: Trial #2, executing "sleep .5"

submission, the number of available workers would go to zero and the number of executing agents would rise to the number of online agents, representing a 100% usage of the pool. Such an arrangement would then last until all the tasks have been executed.

The results of the first experiment, shown in Figure 8, were disappointing in that only a small percentage of the pool was used at any given time. Since the scheduling strategy for Kestrel was to dispatch on demand when either a worker is available or a job was submitted, the scheduler could not keep up with the demand. Before the first round of tasks could be submitted to all available workers, the first workers given tasks finished and requested more tasks. The total time for completing the job was 357.38 seconds, giving a throughput rate of almost 140 tasks per second.

The second experiment showed a large improvement over the first in Figure 9. The half second delay of "sleep .5" helped reduce the strain on the scheduler, allowing for a larger pool usage. However, roughly only 700 of the 900 workers were able to receive their first task during the half second delay. The result was severe thrashing immediately after the job was submitted as the first batch of tasks were

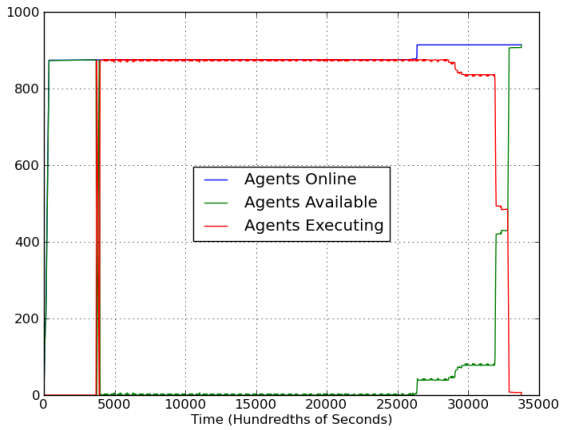

Figure 10: Trial #3, executing "sleep 2"

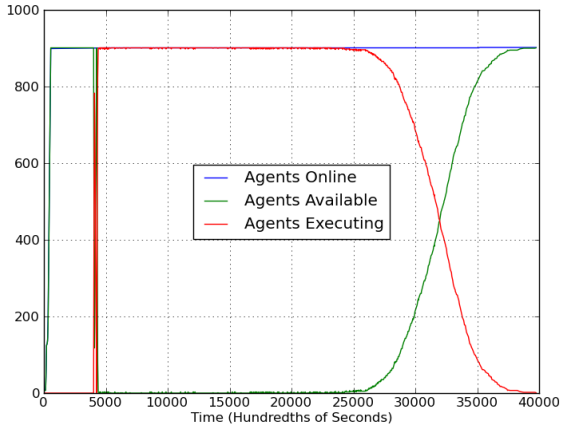

Figure 11: Trial #4, executing "sleep random(0,10)"

completed before the last group was fully dispatched. The system stabilized at close to a 66% usage for the remainder of the job. The total time for the job was 108.61 seconds, giving a task throughput rate of 460 tasks per second. The reduction in scheduler load compared to the first experiment allowed for the increased throughput despite the increased time per task.

The third trial in Figure 10 followed the expected, ideal pattern. The two second delay was sufficient to dispatch the first round of tasks to the 875 available workers. After the initial round, the scheduler was able to meet the demand for tasks, creating small, periodic blips of available workers, but otherwise remaining at close to 100% usage of the pool. Further experiments were later effected using task times of 5 and 10 seconds, and the results were similar. The time for completing the job was 300.56 seconds, giving a throughput rate of 166 tasks per second. The reduced throughput was due to the longer time per task. The time it took to distribute the 875 initial tasks was .51 seconds, giving a dispatch rate of 1715 tasks per second.

The fourth trial (Figure 11) used a random task length between 0 and 10 seconds. The pool usage remained near 100% for most of the life of the job, but gradually declined at the end as the remaining, long tasks complete and were not replaced. The scheduler was able to distribute 61 tasks in 0.04 seconds before a worker returned for another task, yielding a dispatch rate close to 1500 tasks per second.

6. ACKNOWLEDGMENT

The authors thank David Wolinsky of ACIS Laboratory at the University of Florida for help with IPOP configuration and testing.

7. CONCLUSIONS AND FUTURE WORK

As demonstrated through testing, it is possible to construct a VOC with over 1,500 VMs on a single physical cluster using a single VM image without straining the network by using QEMU/KVM in snapshot mode. However, for the case where many different VM images are needed, or one VM image per worker is required, pre-staging VMs could still be more efficient.

Establishing a VOC over a WAN is possible using XMPP. While a 0.3 second latency would cause network I/O intensive programs, such as those based on MPI, to perform poorly, computationally bound applications such as those assumed by the VOC Model should not be affected after the initial dispatch. Due to the possible overlap in RTTs between multiple physical sites, as demonstrated by the results from Palmetto and Amazon EC2, it is not always possible to determine the number of physical sites in a VOC based on RTT data. However, future work on scaling Kestrel could use RTT data to pick new manager nodes to better distribute load and improve dispatch rates.

Finally, Kestrel is able to manage a VOC effectively and distribute tasks with a dispatch rate of more than 1000 tasks per second. Due to the current scheduling algorithm, tasks that are too short will result in reduced performance from increased load on the manager. The minimum desired time was at most 2 seconds for a pool of 900 workers. Given the latency results found in Section 5.2, using tasks that are at least 30 seconds in duration should be sufficient to mask any dispatch delays caused by latency between sites. Future work on Kestrel will investigate the minimum desired time for task execution for pool sizes into the tens of thousands.

8. REFERENCES

[1] M. A. Murphy, M. Fenn, and S. Goasguen, "Virtual Organization Clusters," in *17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009)*, Weimar, Germany, February 2009.

[2] M. A. Murphy, B. Kagey, M. Fenn, and S. Goasguen, "Dynamic provisioning of Virtual Organization Clusters," in *9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '09)*, Shanghai, China, May 2009.

[3] M. A. Murphy, L. Abraham, M. Fenn, and S. Goasguen, "Autonomic clouds on the grid," *Journal of Grid Computing*, vol. 8, no. 1, pp. 1–18, March 2010.

[4] P. Saint-Andre. (2004, October) Extensible messaging and presence protocol (xmpp): Core. IETF. [Online]. Available: http://www.ietf.org/rfc/rfc3920.txt

[5] ——. (2004, October) Extensible messaging and presence protocol (xmpp): Instant messaging and presence. IETF. [Online]. Available: http://www.ietf.org/rfc/rfc3921.txt

[6] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor – a distributed job scheduler," in *Beowulf Cluster Computing with Linux*, T. Sterling, Ed. MIT Press, October 2001.

[7] M. L. Douglas Thain, Todd Tannenbaum, "How to measure a large open source distributed system," *Concurrency and Computation: Practice and Experience*, vol. 8, no. 15, December 2006.

[8] I. Raicu, I. Foster, and Y. Zhao, "Many-task computing for grids and supercomputers," in *Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop on*, Nov. 2008, pp. 1–11.

[9] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "IP over P2P: Enabling self-configuring virtual IP networks for grid computing," in *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.

[10] G. Weis and A. Lewis, "Using xmpp for ad-hoc grid computing - an application example using parallel ant colony optimisation," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, May 2009, pp. 1–4.

[11] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputing Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[12] K. Keahey and T. Freeman, "Contextualization: Providing one-click virtual clusters," in *4th IEEE International Conference on e-Science*, Indianapolis, IN, December 2008.

[13] P. Ruth, X. Jiang, D. Xu, and S. Goasguen, "Virtual distributed environments in a shared infrastructure," *Computer*, vol. 38, no. 5, pp. 63–69, 2005.

[14] M. Tsugawa and J. A. B. Fortes, "A virtual network (ViNe) architecture for grid computing," in *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.

[15] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert, "Centralized versus distributed schedulers for bag-of-tasks applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 5, pp. 698–709, May 2008.

[16] Python programming language. Python Software Foundation. [Online]. Available: http://www.python.org

[17] N. Fritz. [Online]. Available: http://code.google.com/p/sleekxmpp/

[18] L. Stout, M. A. Murphy, and S. Goasguen, "Kestrel: an xmpp-based framework for many task computing applications," in *MTAGS '09: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*. New York, NY, USA: ACM, 2009, pp. 1–6.

[19] D. Crockford. Introducing json. [Online]. Available: json.org

[20] Pidgin, the universal chat client. [Online]. Available: http://www.pidgin.im

[21] Adium. [Online]. Available: http://www.adium.im

[22] About google talk. Google. [Online]. Available: http://www.google.com/talk/about.html

[23] J. Moffitt. (2008, August) Thoughts on scalable xmpp bots. [Online]. Available: http://metajack.im/2008/08/04/thoughts-on-scalable-xmpp-bots/

[24] Ejabberd. Process One. [Online]. Available: http://www.process-one.net/en/ejabberd/

[25] Ignite realtime: Openfire server. Jive Software. [Online]. Available: http://www.igniterealtime.org/projects/openfire/

[26] tigase.org | open source and free (gplv3) jabber/xmpp server. [Online]. Available: http://www.tigase.org/