

Continuous Adaptation for High Performance Throughput Computing across Distributed Clusters

Edward Walker

Texas Advanced Computing Center, The University of Texas at Austin

J. J. Pickle Research Campus, Austin, Texas, USA

ewalker@tacc.utexas.edu

Abstract—A job proxy is an abstraction for provisioning CPU resources. This paper proposes an adaptive algorithm for allocating job proxies to distributed host clusters with the objective of improving large-scale job ensemble throughput. Specifically, the paper proposes a decision metric for selecting appropriate pending job proxies for migration between host clusters, and a self-synchronizing Paxos-style distributed consensus algorithm for performing the migration of these selected job proxies. The algorithm is further described in the context of a concrete application, the MyCluster system, which implements a framework for submitting, managing and adapting job proxies across distributed High Performance Computing (HPC) host clusters. To date, the system has been used to provision many hundreds of thousands of CPUs for computational experiments requiring high throughput on HPC infrastructures like the NSF TeraGrid. Experimental evaluation of the proposed algorithm shows significant improvement in user job throughput: an average of 8% in simulation, and 15% in a real-world experiment.

I. INTRODUCTION

MyCluster [2][3][4] is a software system that builds personal clusters on-demand. It does this by provisioning CPU resources from distributed host clusters on a network into personal cluster constructed from commodity job management systems. To date, the system has been used to provision hundreds of thousands of CPU resources annually on HPC infrastructures like the NSF TeraGrid [1], enabling very large-scale computational experiments across a broad range of scientific disciplines [5][6].

MyCluster provisions CPU resources by submitting a job abstraction, called a job proxy, to a host cluster in-lieu of the actual user job. This job proxy when dispatched at the host cluster provision CPU resources for the personal cluster created by the system. Multiple job proxies submitted to multiple host HPC clusters in a distributed HPC infrastructure, such as the TeraGrid, can then aggregate CPU resources within personal clusters for conducting computational experiments.

These dynamically created personal clusters provide to scientists the freedom to configure the user job queues to optimally support the type of job submission required by their specific computational experiment. For example, many host clusters do not allow the simultaneous submission of thousands of short, serial, computation tasks. MyCluster allows a scientist to create a cluster with job queues defined to allow this, whilst the MyCluster infrastructure maintains job proxies

at the contributing host cluster sites that conform to the local host job submission limits.

The job proxies themselves are scheduled by the local host cluster schedulers. The scientists using MyCluster will therefore see their personal clusters expand and shrink over time, depending on when job proxies are dispatched and when they are terminated by these local host cluster schedulers. Thus it is intuitive that the more quickly the system can create a large personal cluster, and sustain this during its execution period, the better the achievable user job throughput.

This paper proposes an algorithm for mapping job proxies to host clusters with the objective of improving the overall user job throughput. The algorithm is adaptive, reacting to the changing load conditions at host cluster sites, because large-scale computational experiments can run for long periods of time, i.e. weeks. Specifically, the contribution of this paper is a new decision metric for selecting the appropriate pending job proxies for re-distribution, and a self-synchronising Paxos-style distributed consensus algorithm for performing the migration of these selected job proxies across host clusters over time. Collectively the algorithm is shown, in simulation and real implementation, to significantly improve the user job throughput.

The rest of the paper will be as follows. Section II summarizes the MyCluster process architecture and further discusses the concept of job proxies. Section III describes the proposed job proxy migration algorithm: the decision metric and the consensus algorithm for implementing the job proxy migration. Section IV describes the evaluation of our proposed algorithm through simulation and implementation on real host clusters on the NSF TeraGrid. Section V describes some related work, and finally section VI concludes this paper.

II. MYCLUSTER

A. Process Architecture

MyCluster builds Condor [7][8], SGE [10] or OpenPBS [9] clusters when a user creates a *virtual login session*. Within this virtual login session, users can submit, monitor and manage jobs through a single job management interface, emulating the experience of a traditional cluster login session. The MyCluster architecture has been described in detail in prior publications [2], but we reiterate a summary description of the system in this section for clarity in our subsequent discussions.

```

ewalker@blanco ~]$ vo-login -S
Enter GRID pass phrase:
Spawning on tg-login.tacc.teragrid.org
Spawning on tg-login3.ncsa.teragrid.org
Setting up VO participants .....Done

Welcome to your MyCluster/Sun Grid Engine environment
To shutdown environment, type "exit"
To detach from environment, type "detach"

blanco(grid)% agent_jobs
GATEWAY: lonestar.tacc.utexas.edu
753018 (RUNNING)
GATEWAY: tg-login3.ncsa.teragrid.org
526521.tg-master.ncsa.teragrid.org (RUNNING)
blanco(grid)% qhost
HOSTNAME      ARCH      NPROC  LOAD  MEMTOT  MEMUSE  SHAPTO  SHAPUS
-----
global        -         -      -      -        -        -        -
mc1-14        glinux    2  0.03  2.0G    79.1M   2.0G    0.0
mc1-9         glinux    2  0.00  2.0G    92.6M   2.0G    0.0
tg-c491       ia64linux 2  0.65  3.9G    383.3M  5.9G    23.5M
tg-c861       ia64linux 2  0.00  3.9G    404.7M  6.0G    16.5M

blanco(grid)% qsub -t 1000 sub.cmd
your job-array 1.1000-1000:1 ("sub.cmd") has been submitted
blanco(grid)% qdel 1
ewalker has registered the job-array task 1.1000 for deletion

```

(a)

```

ewalker@blanco ~]$ vo-login
Spawning on tg-login.tacc.teragrid.org
Spawning on tg-login3.ncsa.teragrid.org
Setting up VO participants .....Done

Welcome to your MyCluster/Condor environment
To shutdown environment, type "exit"
To detach from environment, type "detach"

blanco(grid)% agent_jobs
GATEWAY: lonestar.tacc.utexas.edu
753019 (RUNNING)
GATEWAY: tg-login3.ncsa.teragrid.org
526522.tg-master.ncsa.teragrid.org (RUNNING)
blanco(grid)% condor_status

Name      OpSys      Arch      State      Activity      LoadAv Mem      ActvtyTime
-----
9947@tg-c861. LINUX IA64 Unclaimed Idle      0.000 4009 0+00:00:04
9947@tg-c861. LINUX IA64 Unclaimed Idle      0.000 4009 0+00:00:04
...
1958@compute- LINUX INTEL Unclaimed Idle      0.160 2026 0+00:00:04
1959@compute- LINUX INTEL Unclaimed Idle      0.160 2026 0+00:00:04
...

Machines Owner Claimed Unclaimed Matched Preempting
-----
IA64/LINUX      4  0  0  4  0  0
INTEL/LINUX     4  0  0  4  0  0

Total           8  0  0  8  0  0

blanco(grid)% condor_submit sub.cmd
Submitting job(s).....
1000 job(s) submitted to cluster 1.
blanco(grid)% condor_rm 1

```

(b)

Fig 1. Formatted screenshots of (a) SGE virtual login session, and (b) Condor virtual login session.

Fig 1 shows an example of a SGE and Condor virtual login session. A number of features are highlighted in this figure. First, the `vo-login` command is invoked by the user. This command reads a configuration file, located in the users home directory, containing the list of host clusters participating in the virtual login session. Second, the command request from the user a GSI [11] or SSH password if required, to allow the system to spawn a semi-autonomous agent at each participating host cluster. Third, a shell prompt is returned to the user allowing standard SGE or Condor commands to be issued. Fourth, the user queries the status of all remote job proxies with the `agent_jobs` command. Fifth, (not shown in the figure) the user can detach from the virtual login shell and reattach to it at some future time from a different terminal using the `vo-attach` command. And finally, sixth, the system automatically cleans up the environment and shuts down all remote agents when the user exits the shell.

A high level overview of the MyCluster processes relevant to the discussion in our paper is shown in Fig 2. When a user first starts a virtual login session on the desktop, the system remotely spawns a semi-autonomous *proxy manager* agent at each host cluster contributing resources to the session. These proxy managers are responsible for submitting and managing *job proxies* to the local scheduler at each cluster site. When the job proxies run, they start the appropriate job starter daemons for the user-selected job management system of choice, i.e. Condor, SGE or OpenPBS. The job starter daemons then register back with the master job queue manager at the scientist desktop across the network. Jobs submitted by the scientist from the desktop to the personal cluster are then dispatched to the newly registered job starter daemons, with the scientist seeing an expanding and shrinking cluster as job starter daemons register and terminate over time.

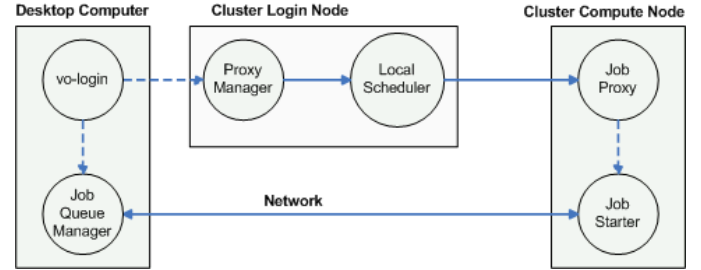


Fig 2. MyCluster process architecture overview

B. Job Proxies

In MyCluster, job proxies are resource provisioning containers. Job proxies are used because they have some beneficial properties. First, job proxies are spatially malleable. For example, multiple job proxies can be submitted with arbitrary size requirements. Thus to build a personal cluster of X CPUs, we can submit n job proxies of sizes S_i each, where $X = \sum_{i=0}^n S_i$.

Second, job proxies are also temporally malleable. For example, we can define job proxies to run for some time period K , within the allowable run-time limit of a host cluster. This allows the CPU resource at the host cluster to be provisioned for multiple user jobs runs during this K time period.

Note, that the above two properties allow job proxies to be migrated between participating host clusters in isolation from the requirements of the user jobs. For example, the user jobs may be serial (single-CPU) jobs that run for a short time, but we can adopt strategies to migrate job proxies between sites, adjusting spatial and temporal requirements of the job proxies to improve the throughput of the personal clusters created.

We define a proxy allocation at time t to be a mapping of $J_i(t)$ job proxies to host cluster i . Therefore, we are fundamentally interested in the problem of determining a good decision metric to decide what the job proxy allocation for N host clusters should be over some time period $t=0$ to ρ where ρ is the run-time of the entire computational science experiment.

III. JOB PROXY MIGRATION ALGORITHM

A. Liability-Adjusted Throughput

Now assume a cluster i is composed of CPUs of some normalized FLOPS rate F^l . This cluster has a job proxy allocation $J_i(t)$ with proxies in the running and pending states, represented by $R_i(t)$ and $P_i(t)$ respectively. Assuming that the job proxies are all submitted with the fixed size requirement S , we define the relative throughput of cluster i at time t as

$$C(t) = R(t) * S * F \quad (1)$$

We also define the pending job liability of cluster i at time t as

$$L(t) = P(t) * S * F \quad (2)$$

Intuitively, we wish to migrate job proxies to host clusters with the highest throughput (i.e. running the most job proxies on better CPUs), but we also do not wish to migrate job proxies to clusters which already have many job proxies in the pending state. This intuition is derived from the fact that the pending job proxies are expected to consume any additional throughput capacity of the cluster when it becomes available.

We therefore defined the liability-adjusted throughput rate as

$$\tau(t) = C(t) - L(t) \quad (3)$$

Equation (3) determines how much future “spare” capacity is available at a host cluster site. A high value for this metric indicates a high future capacity is available at a site, and hence more job proxies should be assigned to it if possible.

Finally, because we wish to filter out any noise component in the metric, we incorporate historical values into the adjusted throughput calculation. This prevents anomalous spikes in the adjusted throughput calculation from affecting our job proxy allocation decision. These anomalous spikes can occur for example if a large parallel job at a busy cluster completes, providing a lot of temporary spare CPU capacity for a short period of time.

To incorporate historical information of a host cluster adjusted throughput value, we compute the harmonic series weighted sum as follows

$$\begin{aligned} T(t) &= \tau(t) + \frac{1}{2} \tau(t-1) + \frac{1}{3} \tau(t-2) + \frac{1}{4} \tau(t-3) + \dots \\ \Rightarrow T(t) &= \sum_{n=1}^k \frac{1}{n} \tau(t-n+1) \\ \Rightarrow T(t) &= \sum_{n=1}^k \frac{SF}{n} (R(t-n+1) - P(t-n+1)) \end{aligned} \quad (4)$$

Equation (4) is the liability-adjusted throughput value we calculate for each host cluster in our algorithm.

B. Consensus Migration Algorithm

MyCluster uses a consensus algorithm to periodically select the host cluster with the lowest and highest liability-adjusted throughput value. It does this in order to move pending job proxies from the host cluster with the lowest value to the host cluster with the highest value. The algorithm is as follows:

1. Each host cluster calculates its adjusted throughput value at periodic (configurable) time intervals.
2. When a host cluster calculates its adjusted throughput value, it advertises this value to all other host clusters in the session in a proposal message. We call this host cluster the proposer, and this consensus round the *propose* phase.
3. When a host cluster receives a proposal value, it calculates its own adjusted throughput value if it has not done so already. If the host cluster adjusted throughput is lower than the proposal value, a message (called an ACK) with its own adjusted throughput value is sent to the proposer, or a message (called a NACK) is sent otherwise. We call this the *promise* phase.
4. If a proposer receives a NACK message, it terminates the consensus round it started and retires as a proposer.
5. The host cluster that sent the NACK message with the larger adjusted throughput value then initiates a consensus round from step 1 and begins life as a proposer.
6. If a proposer receives ACK messages from all the participating host clusters, it picks a host cluster with the lowest adjusted throughput value and sends it a request for a pending job proxy. We call this the *accept* phase.
7. If a host cluster receives a request for a pending proxy, it deletes a pending job proxy from its local scheduler queue if two or more pending proxies exist, and sends another ACK message back to the initiator. Otherwise, a NACK message is returned instead. We call this the *acknowledgement* phase.
8. If the proposer receives an ACK message for a pending job proxy request, it adds a new job proxy to its local job scheduler queue, otherwise no change occurs.

¹¹ The normalized FLOPS rate, F , is a comparative performance ratio to a baseline CPU. For example, using the SPEC CPU2006 benchmark results and the 1.6GHz Intel Xeon CPU as a baseline, the 2.4GHz Xeon will have an approximate value for F of $15/9=1.6$.

There are two things that should be noted from the above algorithm. First, our algorithm is a variation of the Paxos consensus algorithm [12], and second, it is self-synchronizing, requiring no global clock.

Fig 3 and Fig 4 illustrate the behaviour of our proposed consensus algorithm when no conflicts are encountered by the initial proposer, and when competing proposers are encountered respectively. In both illustrated scenarios four sites are involved: sites A, B, C and D.

In Fig 3 site A reaches its periodic interval for calculating its adjusted throughput value first. It then advertises its adjusted throughput value to sites B, C and D (propose phase). When all sites receive this proposal, they immediately calculate their adjusted throughput value, and compare their value with the proposed value. In this cases, all three sites have lower adjusted throughput values, so they respond to site A with an ACK message together with their own adjusted throughput value (promise phase). Site A then picks the site with the lowest adjusted throughput value (in this case it is site C), and sends a request to this site for a pending job proxy (accept phase). Finally site C responds with an ACK message after it successfully kills a pending job proxy (if any), allowing site A to submit an additional job proxy to its local job queue (acknowledge phase).

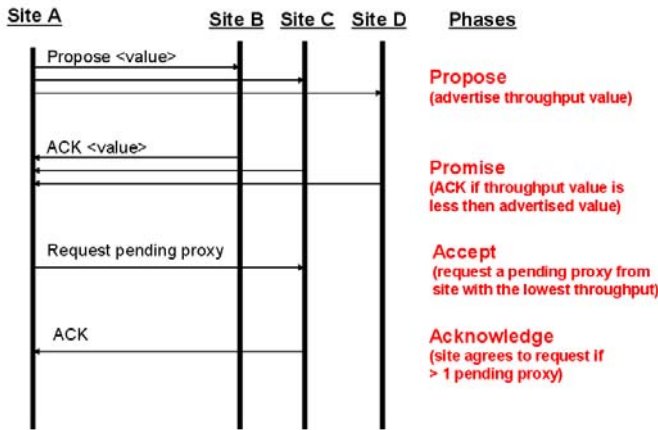


Fig 3. Consensus algorithm with no conflicts

In Fig 4 site A calculates its adjusted throughput value and advertises this to all the other sites (first propose phase). However, site C determines it has a higher adjusted throughput value, and responds with a NACK message (first promise phase). This causes site A to retire as a proposer. Site C then assumes the proposer role and advertises its own adjusted throughput value to all the other sites (second propose phase). However site B determines it has a higher adjusted throughput value, and responds with a NACK (second promise phase). Site B then assumes the proposer role and advertises its own adjusted throughput value to all the other sites (third propose phase). If all the other sites determine they have adjusted throughput values lower then this advertised value, the consensus algorithm ensues as illustrated in Fig 3.

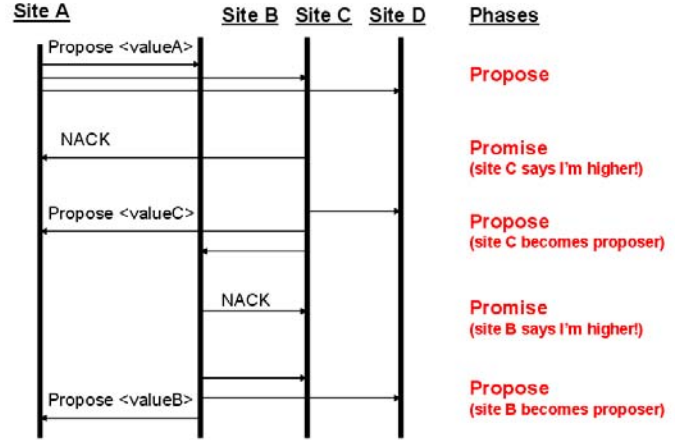


Fig 4. Consensus algorithm with competing proposers

IV. EXPERIMENTAL EVALUATION

A. Software Simulator

To evaluate the proposed job proxy migration technique we implemented a software simulator. The software simulator is used to measure the user job throughput rate achievable with and without the use of the proposed job proxy migration technique. The main purpose of the software simulator is to calculate the likelihood of a job proxy being dispatched on a set of distributed host clusters during a series of time ticks.

A job proxy of size (S) when submitted to a simulated host cluster is assumed to take three possible states: PEND (P), RUN (R), and KILL (K). The software simulator then simulates the following job proxy life cycle on a host cluster:

1. A job proxy is in the PEND state when it is first submitted to the job queue in a host cluster.
2. After some time ticks in this PEND state, T_p , the job proxy may be dispatched by the host cluster into the RUN state.
3. The job proxy in the RUN state will then run for some time ticks, T_R , after which it will be killed by the local host scheduler, where it takes on the KILL state.
4. The job proxy is then expected to be re-submitted into the job queue, where it re-assumes the WAIT state with T_p reset to 0.

Each simulated host cluster has a random average job queue wait time, T_W , assigned to it during the simulation. T_W is initially assigned a value randomly selected in some range, and re-assigned after some number of time ticks to simulate the time varying behaviour of the real job queue wait times in host clusters.

The simulator then calculates the probability of a job proxy J, which has been waiting for T_p time ticks in the job queue, of being dispatched at a host cluster where N_R job proxies from the user is already in the RUN state. The simulator uses the following probability distribution function to perform this calculation:

$$E(J) = \begin{cases} \frac{1}{e^{T_w - T_p}} & \text{if } T_p < T_w \\ \frac{1}{e} + \frac{1}{N_R + 2} & \text{if } T_p \geq T_w \end{cases} \quad (5)$$

Fig 5 and Fig 6 visually illustrates the job proxy dispatch probability distribution used by our software simulator for a simulated average wait time (T_w) of 7 time ticks. In Fig 5 we see that as T_p increases towards T_w , the probability of a job proxy dispatch increases exponentially up to a probability of 0.35. This ensures that the longer a job proxy is in the job queue, the more likely it will be dispatched.

Note that in our simulation, the T_p of a pending job proxy is always reset to 0 when it is migrated to a different cluster. This therefore results in the required penalty for moving an already pending job proxy and re-queuing it in a new host cluster.

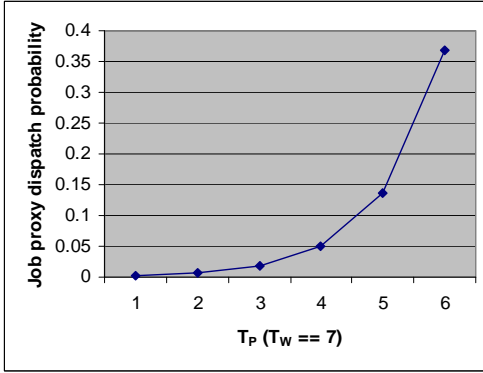


Fig 5. Probability of job proxy dispatch as T_p approaches T_w .

Fig 6 shows the job proxy dispatch probability distribution modified by the number of job proxies already in the RUN state in a host cluster. In the figure we see that the probability of job proxy dispatch is highest when there are no running job proxies that belong to the user and decreases as more job proxies are run. This distribution is used to simulate the fair share algorithm [23] used in many modern host cluster schedulers.

In fair share, a scheduler attempts to only dispatch jobs from a user to utilize some target share of the resource, representing some proportionally fair distribution of the users of the cluster. Typically, the more jobs a user has already running in a host cluster, the less priority is given to a new job from the user by the local scheduler. The fair share algorithm is intended to ensure that one user is not able to consume all the resources in a cluster at any one time; hence it is a feature that is implemented by many modern job schedulers and enabled in many real multi-user shared HPC clusters.

In our software simulator, the probability of a job proxy being dispatched after its wait time exceeds the simulated average job queue wait time (after $T_p \geq T_w$) is approximately 0.85 when no other job proxies from the

user is running. However, the probability of a job proxy dispatch decreases as more job proxies belonging to the user are run.

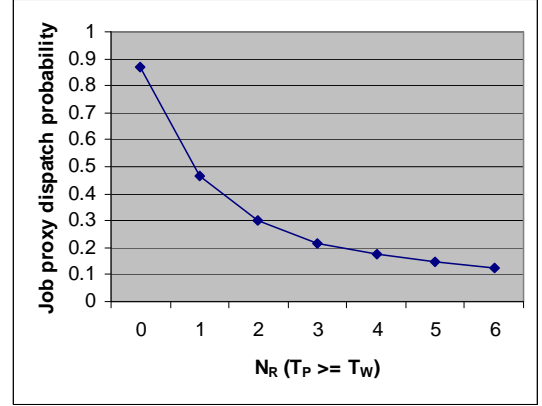


Fig 6. Probability of job proxy dispatch ($T_p \geq T_w$) modified by the number of already running job proxies.

Finally, as mentioned earlier, we assume the run time of a job proxy of size S on a host cluster is T_R . Thus if we also assume a user job of size S' has a run time of T_U ticks, a running job proxy can execute $(T_R * S) / (T_U * S')$ user jobs when it is dispatched.

B. Simulation Results

We used the software simulator to measure the expected throughput from the expanding and shrinking personal clusters created from the job proxies. We also implemented the job proxy migration algorithm in the simulator, and compared the throughput obtained when the proposed technique was enabled and disabled.

In our simulations, we assumed that the system is aggregating CPUs from 16 participating host clusters, with an initial condition of 6 job proxies, of size 32 CPUs each, assigned to each cluster site. The other pertinent properties of our simulation are summarized in TABLE I.

TABLE I. SUMMARY OF SIMULATION RUN PROPERTIES

Symbol	Description	Value
N	Number of clusters	16
S	Job proxy size	32
T_R	Job proxy runtime	4 ticks
S	Job proxy size	32
T_U	Average user job runtime	1 tick
S'	User job size	4
T_w	Average queue wait time	[2,20] ticks
F	Cluster normalized FLOPS	[1,3]
ρ	Total simulation	600 ticks

Fig 7 and Fig 8 show the user job throughput from 1000 simulated runs of the system with job proxy migration disabled and enabled respectively. The charts show the histogram of observed user job throughput in a range of increasing bin size. In Fig 7 the bin sizes are between 300,000 and 470,000 user jobs completed in each simulated run. In Fig 8 the bin sizes are between 330,000 and 520,000 user jobs completed in each simulated run.

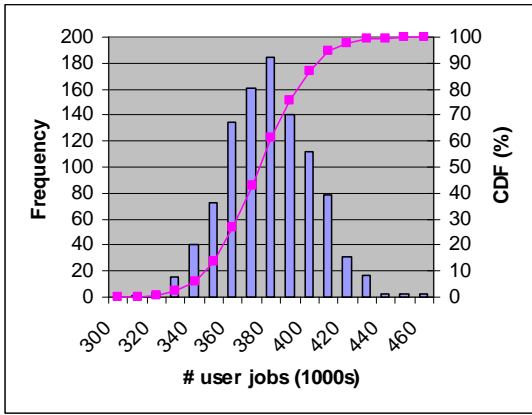


Fig 7. Four CPU user job throughput from 1000 simulation runs across 16 host clusters with job proxy migration disabled.

From the cumulative distribution function (CDF) of the simulation with job proxy migration technique disabled, we see that 90 percent of the simulation runs result in approximately 410,000 completed user jobs. Whereas from the CDF of the simulation with job proxy migration technique enabled, we see that 90 percent of the simulation runs result in approximately 460,000 completed user jobs.

In aggregate, as seen from the average throughput observed from the simulations with the job proxy migration technique enabled and disabled, an approximate 8% improvement in performance is observed when our proposed algorithm is used. In addition, if we compare the maximum throughput achieved in the simulation with job proxy enabled, and the minimum throughput achieved with job proxy disabled, an approximate 80% performance improvement in this ideal case.

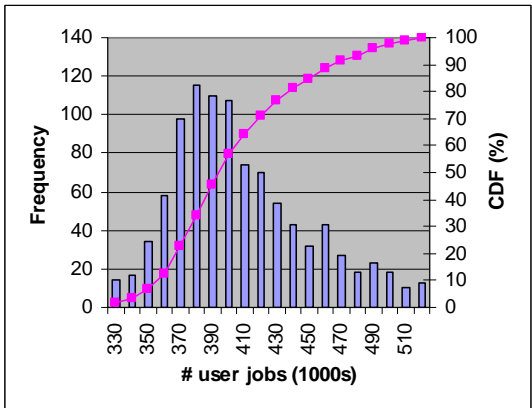


Fig 8. Four CPU user job throughput from 1000 simulation runs across 16 host clusters with job proxy migration enabled.

C. Implementation on NSF TeraGrid

To verify our simulation results, we also implemented our proposed technique in MyCluster and ran an experiment incorporating real host clusters from ANL (Argonne National Laboratory), SDSC (San Diego Supercomputing Center), NCSA (National Center for Supercomputing Application), and TACC (Texas Advanced Computing Center).

As with our simulation, the initial condition of our experiment assigned 6 job proxies, of size 32 CPUs each, to each host cluster. We then configured our consensus algorithm to migrate pending job proxies based on the liability-adjusted throughput decision metric every 5 minutes over a 24 hour period. In this experiment we also assumed all clusters have CPUs of equal FLOPS rate F .

To measure the impact of our migration algorithms, we measured how many one minute user jobs could be executed through the personal clusters created by MyCluster with and without the job proxy migration algorithm enabled.

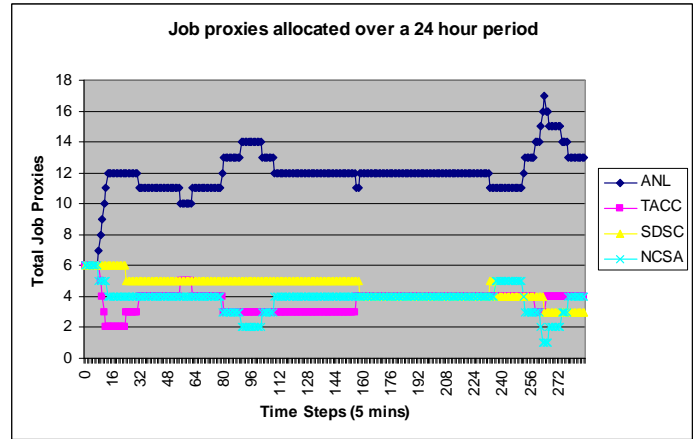


Fig 9. Job proxy distribution over a 24 hour period.

Fig 9 shows the job proxy migration at work over the 24 hour period at the four participating host cluster sites. The graph shows the ANL host cluster acquiring additional job proxies over the 24 hour period for a maximum allocation of 17 job proxies. The graph also shows the NCSA host cluster losing job proxies over the 24 hour period for a minimum allocation of 1 job proxy.

Fig 10 shows the number of one minute serial user jobs completed through the personal clusters created when job proxy migration is enabled and disabled in the experiment. In this experiment, we observed an approximate 15% improvement in throughput when the job proxy migration algorithm was enabled compared to when it was disabled.

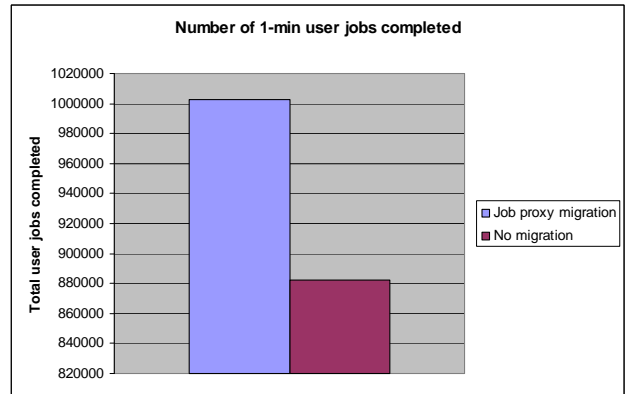


Fig 10. Number of one minute serial jobs completed through MyCluster created personal cluster with/without job proxy migration.

V. RELATED WORK

Singh et. al. [13] has looked at provisioning resource slots using a multi-objective Genetic Algorithm (MOGA). Their approach attempts to minimize the allocation cost and maximize the application performance. However the approach assumes that host clusters can advertise the available slots for provisioning to the application. This is often not the case on many clusters where best effort scheduling is used.

Raicu et. al. [15] implemented the Falkon system for fast execution of short tasks. The system includes a dynamic *provisioner* component for creating and destroying executors (similar to our job proxies) for executing short tasks. They have implemented policies (e.g. one-at-a-time, all-at-once, etc.) that determine how many CPU resources are acquired at a time, and policies for determining when CPU resources should be retired (i.e. 15 secs, 60 secs, etc.). However, unlike the proposed provisioning policy described in this paper, they do not use the host cluster throughput information as a decision metric for adjusting the provisioning decision over time.

Provisioning through advanced reservations has also been extensively studied over the years [14][16][17][18]. However many HPC clusters do not support user-settable advanced reservations because they have been shown to increase queue wait-times, and decrease cluster utilization [19].

Finally there has been much work on scheduling using prediction services like Network Weather Service [20], queue wait-time predictors [21][22] etc. Our method does not use prediction methods, nor rely on the accuracy of these methods. We calculate and use the current, and historical, throughput performance of a cluster to make our provisioning decisions.

VI. CONCLUSION

This paper describes a method for mapping job proxies to host clusters over time with the objective of improving user job throughput. The provisioning method is unique in using a decision metric which incorporates the throughput of a cluster, adjusted for its already pending job liability, and filtered to eliminate anomalous transient spikes in our calculations. The algorithm is further shown to increase the user job throughput by an average of 8%, and up to 80%, in simulation runs. The algorithm is also shown to increase the user job throughput by 15% in a real-world experiment.

ACKNOWLEDGMENT

This paper is based on work supported by the National Science Foundation under grants #0721931 and #0503697.

REFERENCES

- [1] (2008) TeraGrid website. [Online]. Available: <http://www.teragrid.org>
- [2] E. Walker, J. P. Gardner, V. Litvin, and E. L. Turner, "Personal Adaptive Clusters as Containers for Scientific Jobs", *Cluster Computing*, vol. 10(3), Sept 2007.

- [3] E. Walker, J. P. Gardner, V. Litvin, and E. Turner, "Creating Adaptive Clusters in User-Space for Managing Scientific Jobs in a Widely Distributed Environment", in *Proc. of IEEE Workshop on Challenges of Large Applications in Distributed Environments (CLADE'2006)*, Paris, July 2006.
- [4] (2008) MyCluster TeraGrid User Guide website. [Online]. Available: <http://www.teragrid.org/userinfo/jobs/mycluster.php>
- [5] Edward Walker, and C. Guiang, "Challenges in Executing Large Parameter Sweep Studies in Widely Distributed Computing Environments", in *Proc. of IEEE Workshop on Challenges of Large Applications in Distributed Environments (CLADE2007)*, Monterey, CA, June 2007.
- [6] Edward Walker, D. J. Earl, M. Deem, "How to Run a Million Jobs in Six Months on the NSF TeraGrid", in *Proc. of TeraGrid'07*, Madison, WI, June 2007.
- [7] (2008) Condor website. [Online]. Available: <http://www.cs.wisc.edu/Condor/>
- [8] M. Litzkow, M. Livny, and M. Matka, "Condor – A Hunter of Idle Workstations", in *Proc. of the International Conference of Distributed Computing Systems*, pp. 104–111, June 1988.
- [9] (2008) Portable Batch System website. [Online]. Available: <http://www.openpbs.org>
- [10] (2008) Sun Grid Engine website. [Online]. Available: <http://gridengine.sunsource.net/>
- [11] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *Intl. Journal Supercomputing Applications*, vol. 11(2), pp. 115–128, 1997.
- [12] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System", *Communications of the ACM*, vol. 21(7), July 1978.
- [13] G. Singh, C. Kesselman, and E. Deelman, "A Provisioning Model and its Comparison with Best-Effort for Performance-Cost Optimization in Grids", in *Proc. IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 2007.
- [14] G. Singh, C. Kesselman, and E. Deelman, "Adaptive Pricing and Resource Reservations", in *Proc. 8th IEEE/ACM International Conference on Grid Computing (Grid2007)*, 2007.
- [15] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Falkon: a Fast and Light-weight task execution framework", in *Proc. IEEE/ACM Supercomputing 2007*, Nov 2007
- [16] A. S. McGough, "Making the Grid Predictable through Reservations and Performance Modeling", *The Computer Journal*, vol. 48(3), 2005.
- [17] M. Wicczorek, "Applying Advance Reservation to Increase Predictability of Workflow Execution on the Grid", in *Proc. 12th Workshop on Job Scheduling Strategies for Parallel Processing*, 2006.
- [18] T. Roblitz and J. Wendler, "Elastic Grid Reservations with User-Defined Optimization Policies", in *Proc. Workshop on Adaptive Grid Middleware*, 2004.
- [19] M. Margo, K. Yoshimoto, P. Kovatch, and P. Andrews, "Impact of Reservations on Production Job Scheduling", in *Proc. 13th Workshop on Job Scheduling Strategies for Parallel Processing*, 2007.
- [20] R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", *Future Generation Computer Systems*, vol. 15(5-6), 1999.
- [21] J. Brevik, D. Nurmi, and R. Wolski, "Predicting Bounds on Queuing Delay for Batch-Scheduled Parallel Machines", in *Proc. ACM Symp. on Principles and Practice of Parallel Programming*, New York, 2006.
- [22] A. Downey, Predicting Queue Times on Space-Sharing Parallel Computers", in *Proc. 11th Int. Parallel Processing Symposium*, 1997
- [23] J. Kay, and P. Lauder, "A fair share scheduler", *CACM*, vol. 31(1), 1988.