2001-08-01

Livelock Avoidance for Meta-schedulers

Mark J. Clement
clement@cs.byu.edu

John Jardine

Quinn O. Snell
snell@cs.byu.edu

Original Publication Citation

Livelock Avoidance for Metaschedulers, John Jardine, Quinn Snell, Mark Clement. Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC-1), San Francisco, CA, August 21.

Livelock Avoidance for Meta-schedulers

John Jardine*, Quinn Snell, and Mark Clement
Department of Computer Science
Brigham Young University
{snell, clement}@cs.byu.edu

Abstract

Meta-scheduling, a process which allows a user to schedule a job across multiple sites, has a potential for livelock. Current systems avoid livelock by locking down resources at multiple sites and allowing a meta-scheduler to control the resources during the lock down period or by limiting job size to that which will fit on one site. The former approach leads to poor utilization; the later poses limitations on job size. This research uses BYU's Meta-scheduler (YMS) which allows jobs to be scheduled across multiple sites without the need for locking down the nodes. YMS avoids livelock through exponential back-off. This research quantifies the potential for livelock, determines a suitable back-off period, and provides a structure upon which to test theoretical local schedulers. The results show that livelock exists and that a suitable exponential back-off not only avoids livelock but reduces the scheduling time for each job.

Key words: Livelock, meta-scheduling, exponential back-off.

1. Introduction

Although we have vastly more computing power today than was available with ENIAC, the problems people wish to study continue to overwhelm even the fastest supercomputers [3], [9], [12]. There are two solutions to the increasing demand for high-capacity computing resources: increase the capacity of computing resources at each computing site or create a mechanism to allow computing resources to be pooled from multiple sites. It is the later method that is studied in this research.

*John Jardine, formerly a student at Brigham Young University, is an intellectual property attorney with Merchant & Gould, 1191 Second Ave., Suite 1500, Seattle, WA 98101 USA (telephone: 206-342.6200, e-mail: jjardine@merchant-gould.com).

Meta-scheduling allows the scheduling of resources from multiple supercomputing sites. Meta-scheduling can be accomplished by requiring individual sites to lock down computing resources for specified periods of time for use by a meta-scheduler or by treating each meta-scheduler as an ordinary user and allowing resource reservations [5]. As a user, the meta-scheduler queries each site, finds an appropriate time to run a job, and makes the required reservations.

A potential for deadlock and livelock lies in the querying and reserving of resources. Deadlock is eliminated by not allowing meta-schedulers to hold resources while they wait for other resources to become available [11]. Livelock, however, can still occur.

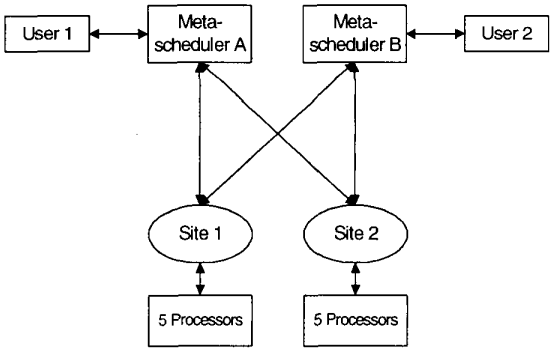

Figure 1: A simple example of Livelock

Livelock occurs when a local system changes state between a resource query and a resource reservation. For example, consider meta-schedulers A and B requesting resources on sites 1 and 2 as shown in Figure 1. Assume that both A and B are requesting 5 processors for 10 minutes on each site. If both A and B query 1 and 2 at the same time (or before either A or B makes a reservation), each is returned the same available start times. Now suppose that A is the first to reserve the resources on site 1 and B is the first to reserve the resources on site 2. A tries to reserve resources on site 2 but finds they are not available. Likewise B tries to

0-7695-1296-8/01 $10.00 © 2001 IEEE

reserve resources on site 1 but finds they are not available. Since neither A nor B can hold and wait (to avoid deadlock), both simultaneously release their reserved resources.

A and B then query sites 1 and 2 again and, assuming other users are not submitting jobs, find the same resources are available at the original time. This time A is the first to reserve resources on site 2 and B is the first to reserve resources on site 1. Although sites 1 and 2 could run the job of either A or B, neither job runs because a change of state occurs, e.g., an intervening reservation, between resource query and resource reservation. This condition could continue indefinitely and livelock has occurred.

Expand this scenario with hundreds of meta-schedulers fighting for the same resources on various sites. Theoretically, none of the meta-schedulers may actually be able to reserve all the resources they need, yet the various sites may be completely available.

1.1. Ethernet CSMA/CD

The Ethernet CSMA/CD network protocol deals with collisions by using exponential back-off [8], [10]. If a sender finds that there has been a collision, it waits a random exponential time and tries again to send the message. This tends to work well until the network bandwidth use is high. Then, most of the bandwidth is used in collision detection and back-off.

There is a major difference, however, between Ethernet livelock and meta-scheduling livelock. In Ethernet livelock, there is only one resource being shared and use of that resource is easy to detect. In meta-scheduling livelock, there are multiple shared resources and use of the shared resource can occur because another meta-scheduler has reserved the resource or because a local user has requested resources.

Furthermore, while livelock is theoretically possible in a meta-scheduling environment, it is unclear how much it will occur in a "real" environment. In the livelock example mentioned previously, it is assumed that each meta-scheduler will be able to reserve one of the needed resources without being able to secure the other because of an intervening reservation by another meta-scheduler. But computers running meta-scheduler algorithms will almost invariably have different compute speeds and different network conditions.

Compute speed differences will likely cause livelock to be broken with enough retries. A meta-scheduler on a faster computer will be able to perform the intersection algorithm quicker to determine where a job can be run. It will be able to send queries and process information faster. So while sometimes another meta-scheduler reserves a needed resource, eventually competing meta-schedulers get "out of sync" and one is able to reserve all required resources.

Network connections and the random nature of network traffic will also likely cause livelock to be broken with enough retries. When multiple meta-schedulers are competing for the same resources, one meta-scheduler may be able to send queries and receive replies in a network traffic lull while other meta-schedulers are hindered from doing so because of a sudden network traffic spike. Even with meta-schedulers on identical computers on the same network, it is possible that network traffic will make it possible for one meta-scheduler to complete its reservation on all needed resources before the other meta-scheduler is able to reserve even one resource.

The equation we use for back-off is $T = 2^{x \bmod n} y$ where x is a random variable, n is the number of retries that have occurred so far and y is the back-off time. When a meta-scheduler is not able to complete a reservation, it times out for this period before trying again. This corresponds closely to the equation used for CSMA/CD [1], [8], [10]. The question then becomes how necessary livelock avoidance through exponential back-off is. Is it worth using exponential back-off or should meta-schedulers rely on the compute speed and network conditions to avoid livelock?

This research shows that exponential back-off can be used to avoid livelock and reduce time to schedule a job in a meta-scheduling environment. This research will demonstrate that exponential back-off can be used to avoid livelock using the Y Meta-scheduler. The performance of this system is compared to a system that relies on compute speed and network conditions to avoid livelock.

1.2. Related work

Only three systems to date have performed significant work with advanced reservations for meta-schedule jobs. These systems are Legion, Globus and the BYU Meta-scheduler.

In the Legion meta-scheduling system, each local scheduler also acts as a meta-scheduler. When a local scheduler is given a task, it queries other local schedulers to determine if the job could be done cheaper elsewhere. Legion avoids livelock by not allowing a job to span multiple local schedulers. This puts severe limitations on Legion's capabilities of running large jobs or small jobs which require resources from multiple sites [5].

Globus is another attempt to create a meta-scheduling environment. Globus uses the locked-down node approach to avoid livelock. With this approach,

local resource managers on local sites release control of specified nodes for a certain period of time. These nodes then become available for meta-schedulers to manage [5].

There are several problems with this approach. First, locked-down nodes typically have very low utilization (in the five to twenty-five percent range). Normal utilization of non locked down nodes is much higher (eighty percent). Therefore, a supercomputer administrator can expect a significant utilization decrease by allowing meta-scheduled requests under this approach [5].

Second, this approach can only avoid livelock over multiple computing sites if each meta-scheduler is assigned its own set of locked down nodes. This, unfortunately, compounds the utilization problem mentioned above.

Finally, local administrators prefer to maintain control of local resources. When local control of resources is relinquished through lock down, local administrators can no longer enforce local policies. A local site may have a policy of allowing a user to run only so many jobs at one time, or allowing a user to have a certain number of jobs queued at one time. Additionally, to perform administrative tasks which require the locked down nodes, a local administrator must work around the lock down period.

BYU's Meta-scheduler (YMS) was built to overcome these limitations. Unlike Legion, YMS allows jobs to span multiple supercomputing sites. Unlike Globus, YMS allows local administrators to maintain control over their resources and policies. YMS does this by accessing each local scheduler as an ordinary user would, i.e. by querying each site for available resources and making a reservation.

A key missing ingredient of YMS has been a livelock avoidance mechanism. This research into livelock avoidance is the next step in making YMS a robust meta-scheduler. Once this research and changes to YMS have been made, universities like BYU and the University of Utah and other supercomputing sites will be able to more efficiently combine their computing resources to collaborate on research projects.

2. Experimental Design

A large number of experiments were performed to determine the need for exponential back-off and the impact on performance from varying back-off parameters. Figure 2 shows the experimental setup used in this research. The Maui scheduler can be used to gather scheduling request information from production supercomputing sites. This trace data is read by the Coordinator and sent through the Submit Job Host to the Meta-schedulers on different physical machines. These Meta-schedulers then contacted the local scheduler (the production Maui scheduler run in simulation mode) through a socket connection to attempt to schedule the tasks in the trace file. Up to 64 Meta-schedulers and 64 Local schedulers were run in order to determine the impact of back-off. Average values were computed from many runs of each configuration.

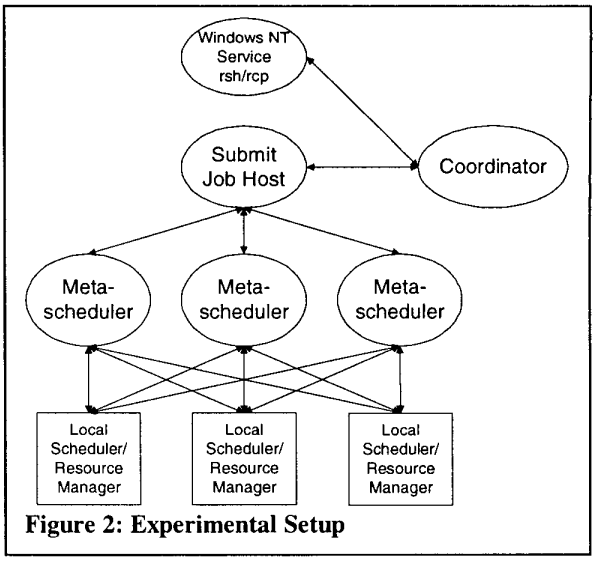

Figure 2: Experimental Setup

2.1 Order for reservations on local schedulers

The algorithm meta-schedulers use and the order in which meta-schedulers make reservations on local schedulers can make the difference between livelock and no-livelock situations. In this section, first a simple meta-scheduling algorithm will be given. Then, depending on which order the meta-schedulers make reservations on the local schedulers, how this algorithm avoids or causes livelock will be examined. Finally, the order of making reservations used in this research will be given.

Consider a meta-scheduler which schedules jobs according to the following algorithm:

- Sequentially query each local scheduler for available time ranges;
- Determine the soonest time a job can run; and
- Sequentially make, as needed, a reservation on each local scheduler for a chosen time. If at any time, resources are not available at the chosen time, stop the reservation process, release all resources, and start over. If the Meta-scheduler fails to be able to reserve resources after 25 times, the request fails.

Let Meta-scheduler A have local scheduler 1 as its first local scheduler and local scheduler 2 as its second

local scheduler. Let Meta-scheduler B have the same ordering of local schedulers. Finally, assume that simultaneously both Meta-schedulers A and B receive jobs to schedule which require all the resources of local schedulers 1 and 2.

Whichever meta-scheduler reserves first on local scheduler 1 will also be able to reserve on local scheduler 2. This occurs because of the sequential nature of the reservation algorithm outlined above. Specifically, the requirement to abort a reservation at any step in which a reservation cannot be made combined with the fact the both Meta-schedulers A and B are trying to reserve first on local scheduler 1 and then on local scheduler 2 cause the first to reserve on local scheduler 1 to complete scheduling successfully.

Now assume that Meta-scheduler B reserves in reverse order on local schedulers 1 and 2. Now, a livelock potential exists in which Meta-scheduler A may be able to always reserve on local scheduler 1 first and Meta-scheduler B may be able to always reserve on local scheduler 2 first. In this case livelock will occur.

One then could ask: why not require all meta-schedulers to reserve resources from local schedulers in a canonical way? One such way would be to have all meta-schedulers translate the local scheduler's IP address to integers and then reserve resources from the local schedulers in ascending IP numerical order.

There are several problems with this approach. First, users running meta-schedulers may prefer a certain ordering of local schedulers. For example, most users would prefer to reserve resources on a local scheduler in close physical proximity, i.e. at the same site, if possible over one located far away. Local proximity does not always correspond to ascending IP address.

Second, and more importantly, not every meta-scheduler will have access to the same set of local schedulers. For example, meta-scheduler A may have access to local schedulers 1 and 3 while meta-scheduler B has access to local schedulers 2 and 3. Even if each meta-scheduler was to reserve in ascending order, Meta-scheduler A would start with local scheduler 1 and Meta-scheduler B would start with local scheduler 2. Then, the first to reserve on its first local scheduler is not guaranteed reservation on its second local scheduler.

For purposes of this research, to simulate the randomness of meta-scheduler users, each with individual preferences, meta-schedulers reserved on available local schedulers in a random order. In addition, to maximize livelock potential, all meta-schedulers tried to reserve on all local schedulers.

3. Experimental Results

The results of this research indicate that livelock can cause significant problems in the performance of scheduling systems. A significant number of scheduling failures and retries occur if exponential back-off is not used. With 32 Meta-schedulers, a significant number of requests (12-14%) failed after 25 attempts as shown in Figure 3.

Figure 4 shows the average number of retries that occur for different numbers of meta-schedulers. Most of these requests eventually succeed, but incur a large delay through having to retry. Figure 5 shows the time required to schedule a job without exponential backoff. Notice that with 32 Meta-schedulers it can take as long as 175 seconds to schedule a job when exponential back-off is not used.

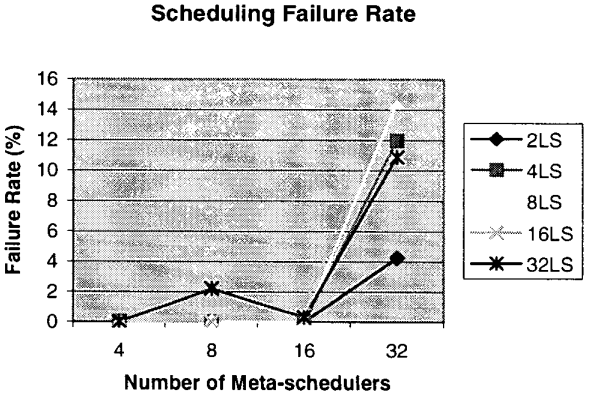

Figure 3: Failure rate for Maui Scheduler without exponential back-off.

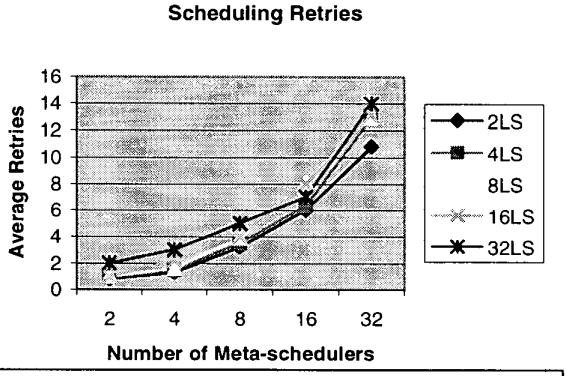

Figure 4: The average number of retries for each request using Maui without exponential back-off

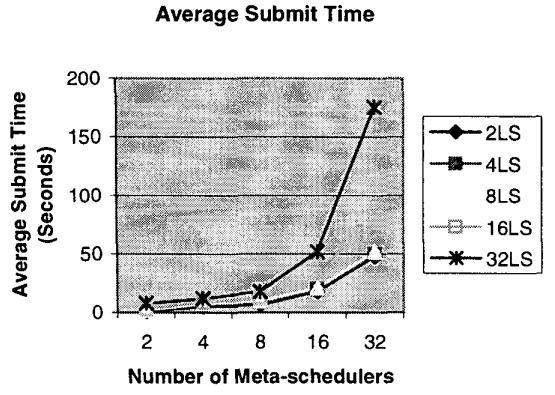

Figure 5: Submit time for Maui Scheduler without exponential back-off

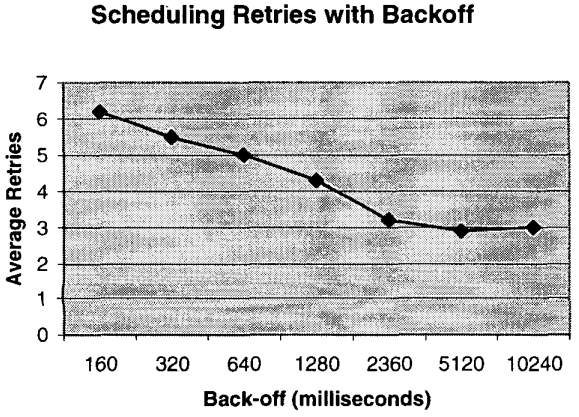

Figure 6: Average retries required per scheduling attempt as back-off is changed with 32 meta-schedulers and 4 Maui schedulers.

When exponential back-off is used, the performance of the Meta-scheduling system is significantly improved. Figure 6 shows the results of varying the back-off time for 32 Meta-schedulers and 4 local schedulers. Figure 7 compares the number of retries with and without exponential back-off. This difference increases as larger machine configurations are used.

4. Conclusions

This research set out to study if livelock in meta-scheduling was a problem which needed to be avoided and if it could be done through using exponential back-off. The results show that livelock does occur and that it can be avoided with exponential back-off. Furthermore, this research shows that the time to meta-schedule jobs which do not eventually livelock can be decreased while the compute time spent meta-scheduling the job also decreases.

YMS has been enhanced to include exponential back-off. With this enhancement, Brigham Young University and the University of Utah, and other universities like them are closer to being able to combine their computing resources to collaborate on research. The infrastructure allows jobs to be scheduled which require more resources than either University has access to.

One may ask what the likelihood is that 64 meta-schedulers would simultaneously request resources from 2 or more local schedulers. In the Internet future, every computer on every desk may be brought into a meta-scheduling environment donating unused compute cycles to research. When this happens, livelock avoidance will not just be a research topic; it will be required to keep the computational grid from locking up.

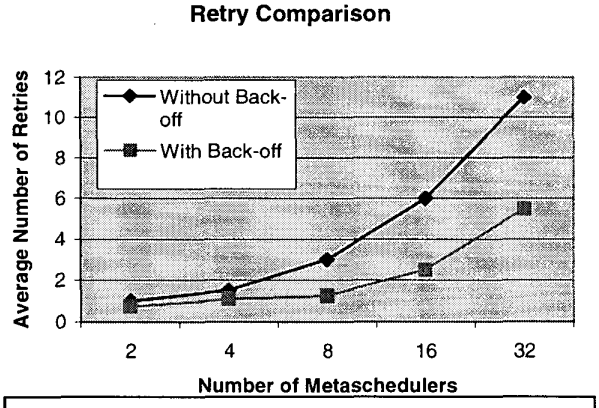

Figure 7: Average retries required per scheduling attempt as back-off is changed with 32 meta-schedulers and 4 Maui schedulers.

5. References

[1] Boggs, David; Mogul, Jeffrey; Kent, Christopher. (1988). Measured capacity of an ethernet: myths and reality in *Proceedings of the SIGCOMM '88 Symposium*, pages 222-234, August 1988.

[2] Chervenak, Ann; Foster, Ian, Kesselman; Carl; Salibury, Charles; Tuecke, Steven. (1999). The data grid: towards an architecture for the distributed management and analysis of large scientific datasets. *The Globus Project*. See http://www.globus.org/

[3] *Revenge of the Supercomputers*. See http://www.cnet.com/specialreports/0-6014-7-1554681.htm.

[4] Foster, Ian; Kesselman, Carl; Lee, Craig, Lindel, Bob; Nahrstedt, Klara; Roy, Alain. (1999). A distributed resource management architecture that supports advance reservations and co-allocation in *International Workshop on Quality of Service, 1999*. See http://www-fp.mcs.anl.gov/qos/qos_papers.htm.

[5] Gregory, Chad. (2000). *The Effect of Meta-scheduling and Advance Reservations on Supercomputing Performance*. Brigham Young University master's thesis, 2000.

[6] Grid Forum, The. (2000). See http://www.gridforum.org/.

[7] Hoo, Gary; Johnston, William; Foster, Ian; Roy Alain. (1999). QoS as middleware: bandwidth reservation system. In *Proceedings of the 8th IEEE Symposium on High Performance Distributed Computing*. See http://www-fp.mcs.anl.gov/qos/qos_papers.htm.

[8] Metcalf, R.; Boggs, D.; (1976). Ethernet: distributed packet switching for local computer networks. *Communications of the ACM* 19(7):395-403.

[9] O'Brien, Bill. (2000). *Revenge of the Supercomputers*. See http://www.cnet.com/specialreports/0-6014-7-1554681.htm.

[10] Peterson, Larry; Davie, Bruce. (1996). *Computer Networks a System Approach*. San Francisco, CA: Morgan Kaufmann Publishers, Inc.

[11] Snell, Quinn; Clement, Mark; Jackson, David; and Gregory, Chad. (2000). The performance impact of advance reservation meta-scheduling to be published in 6th *Workshop on Job Scheduling Strategies for Parallel Processing*. See http://www.ipdps.org/2000_workshops.html.

[12] Weik, Martin H. (1961). *The ENIAC Story*. See http://ftp.arl.mil/~mike/comphist/eniac-story.html.