A DECENTRALIZED SCHEDULING FRAMEWORK FOR MANY-TASK SCIENTIFIC COMPUTING IN A HYBRID CLOUD

Brian Peterson, Gerald Baumgartner, and Qingyang Wang
Division of Computer Science and Engineering
School of Electrical Engineering and Computer Science
Louisiana State University
Baton Rouge, LA 70803
{brian, gb, qywang} @csc.lsu.edu

Abstract

Cloud services are transforming many computing tasks, but the unique synchronization and communication requirements of scientific computing have caused it to lag behind in cloud adoption because of the performance variation of cloud resources and inefficient communication channels. Based on our experience with the Organic Grid, we propose a framework for a hybrid cloud that will distribute work to appropriate computing resources based on benchmark measurements and performance history to mitigate the impact of performance variation. By understanding the capacities of available computing resources, we intend to schedule work more efficiently, and measure our success in doing so. Our approach is to organize a set of computing nodes in an overlay network, to allow each node to position itself within the network to maximize its productivity, and for nodes to advertise their performance and capabilities. An application finds the resources and decides which task to run on which cloud nodes. Our simulations and experiments in the cloud demonstrate that performance variations in the cloud exist and that such a decentralized approach works best to reduce the communication burdens of the most overworked nodes, especially on networks with the highest task to node ratios.

Keywords: Decentralized Distributed Scheduler Hybrid Cloud Organic Grid

1. INTRODUCTION

Many scientific fields, such as quantum chemistry, genomics, phylogenetics, astrophysics, geophysics, computational neuroscience, or bioinformatics, require massive computational power and resources that might exceed those available on a single local supercomputer. Historically, there have been two drastically different approaches for harnessing the combined resources of a distributed collection of machines: traditional computational supercomputer grids and large-scale desktop-based master-worker grids. Over the last decade, a commercial alternative has become available with cloud computing. For many smaller scientific applications, using cloud computing resources might be cheaper than maintaining a local supercomputer. However, cloud computing has not yet been successful for high-performance computing applications.

Research on traditional grid scheduling has focused on algorithms for determining an optimal computation schedule based on the assumption that sufficiently detailed and up-to-date knowledge of the systems state is available to a single entity (the metascheduler) (Grimshaw 1997), (Berman 2003), (Abramson 2000), (Taylor 2003). While this approach results in a very efficient utilization of the resources, it does not scale to large numbers of machines, since maintaining a global view of the system becomes prohibitively expensive. Variations in resource availability and potentially unreliable networks might even make it impossible.

A number of large-scale desktop grid systems have been based on variants of the master/workers model (BOINC 2016), (SETI 2016), (Woltman 2016), (folding@home 2016), (Chien 2003), (Litzkow 1988), (Maheswaran 1999), (Heymann 2000), (Kindberg 1994), (Buaklee 2002), (Karonis 2003), (IBM 2015). The fact that SETI@home had scaled to over 5 million nodes and that some of these systems have resulted in commercial enterprises shows the level of technical maturity reached by the technology. However, the obtainable computing power is constrained by the performance of the master, especially for data-intensive applications. Since networks cannot be assumed to be reliable, large desktop grids are designed for independent task applications with relatively long-running individual tasks.

Cloud computing has been very successful for several types of applications, especially for applications that do not require frequent communication between different cloud nodes, such as MapReduce (Dean 2008) or graph-parallel (Xin 2013) algorithms. However, for applications with fine-grained parallelism, such as the NAS MPI benchmarks or atmospheric monitoring programs, it shows less than satisfactory performance (Walker 2008), (Evangelinos 2008). The main reasons are that the performance of cloud nodes is often not predictable enough, especially with virtualization, and that the communication latency is typically worse than that of a cluster (Walker 2008). While it is possible to rent dedicated clusters from cloud providers,

they are significantly more expensive per compute hour than virtual machines (VMs). With virtualization, however, the user may not know whether a pair of VMs run on the same physical machine, are in the same rack, or are on different ends of the warehouse.

VMs may never be competitive for running large high-performance computing applications with fine-grained parallelism, such as large dense matrix multiplications. For applications that can be broken into a set of smaller tasks, however, it may be possible to match the performance requirements of a task with the performance characteristics of a subset of the cloud nodes. For grids and supercomputers, this many-task computing approach (Raicu 2008) has proven very effective. E.g., Rajbhandari et al. (2014) structured the computation of a quantum chemistry tensor contraction equation as a task graph with dependencies and were able to scale the computation to over 250,000 cores. They dynamically schedule tasks on groups of processors and use work-stealing for load balancing.

A possible solution for running high-performance or many-task computing applications in the cloud is to identify the performance characteristics of the cloud nodes and the network connections between them and to map computational tasks onto subsets of the nodes with appropriate performance characteristics. Maintaining this information centrally for a large number of machines, however, is prohibitively expensive. IBM's Air Traffic Control (ATC) algorithm (Barsness 2014) attempts to solve this problem by arranging cloud nodes in groups and by letting the leader of each group (the air traffic controller) direct the compute tasks (aircraft) from a central job queue to their destination worker nodes that will then execute the task. While this approach distributes the load of maintaining performance information for worker nodes, the central job queue is still a potential bottleneck. Also, the air traffic controllers may not have enough information about the computational requirements and communication patterns of the individual tasks.

We propose a fully decentralized approach in which the applications decide which nodes to run on. Our approach relies on the nodes to advertise their performance characteristics. If performance information is available from the hypervisor or operating system, we will use it. Otherwise, the nodes periodically measure their own performance as well as network latency and throughput to neighboring nodes. This performance information is then distributed in aggregate form to neighboring nodes along an overlay network. Computational tasks can be inserted into the cloud at any point. The application queries the resource availability through an API and migrates the tasks along the overlay network to where appropriate resources are available. By letting the application decide which nodes to run on, the decision can be based on information about the communication and computation characteristics of the application that may not be available if the decision is made by leader nodes. Finally, the overlay network is periodically restructured as needed to improve the migration of tasks to appropriate cloud nodes.

Similar to real-world air traffic control, applications can enter the cloud at any point instead of through a centralized job queue. The pilot (application) communicates with air traffic control, e.g., to obtain information about the weather (resource availability) at the destination, but the final decision rests with the pilot in command. As in our approach, aircraft (tasks) travel along air routes (an overlay network) to their destination.

Our approach to high-performance and many-task computing in the cloud is based on our prior work on the Organic Grid (OG), a decentralized desktop grid prototype (Chakravarti 2005), (Chakravarti 2006). Evidence supporting our design is provided by simulations of the Organic Grid and by both simulations and cloud measurements of Barsness et al.'s Air Traffic Control algorithm. A comparison of our ATC simulation and measurements performed on machines requisitioned from the CloudLab system (CloudLab 2016) validates our simulation approach. All sets of measurements support pursuing a decentralized approach to scheduling work in the cloud.

In the next sections, we first describe the Organic Grid and ATC approaches in more detail and then present our simulation approach. We present the results of our simulations that demonstrate that our proposed scheduling approach would result in decreased communication overhead. Cloud experiments are performed and compared to the simulation result. We interpret both the cloud and simulation measurements, and what the OG and ATC comparison tells us about decentralized scheduling. Comparing the simulated and cloud ATC runs allows us both to justify the simulation methodology and present interesting measurements of cloud performance. Finally, we outline the future work in developing our framework.

2. BACKGROUND

In prior work, we have built a prototype of a decentralized desktop grid system, the Organic Grid (Chakravarti 2005), (Chakravarti 2006). The Organic Grid uses Java mobile agents as containers for transporting computational tasks to the nodes where they are executed. Nodes are initially configured to have some neighbors in the network, as shown in Figure 1. An application is structured as a set of tasks and can be submitted at any node in the network. Idle nodes periodically try to steal work from neighboring nodes. If an application is too large to be executed by a single agent in a reasonable amount of time, agents clone themselves in response to a work-stealing request and migrate to other nodes; the clones are assigned a subset of the tasks by the initiating agent. The new agents complete the tasks they were assigned and return the results to their parent. The children also, in turn, clone themselves

and send agents to available nodes and distribute tasks to them. An overlay network is, thus, constituted by the connections that are created between agents as the computation spreads. The topology of a typical overlay network is a tree with the root being the node where the original computation was injected.

It is desirable for the best-performing nodes in a subtree to be close to the root, since this minimizes the communication delay between the root and the fastest nodes and the time that these nodes need to wait for their requests to be handled. For identifying the best-performing nodes, parents are measuring the performance of their children based on the completion time of the tasks. These performance measurements are used to restructure the overlay network so that faster nodes migrate closer to the root, as shown in Figure 2. However, if too many nodes would directly communicate with the root, it could become a bottleneck. The algorithm for restructuring the overlay network, therefore, maintains an application-specific limit on the number of children any node can have. For streaming tasks to child nodes, a mechanism related to the TCP sliding window protocol is employed so that communication can be overlapped with computation. We have demonstrated that these mechanisms result in improved overall performance for two very different applications: the BLAST sequence alignment application (Chakravarti 2005) and an independent Cannon-style distributed matrix multiplication (Chakravarti 2006). While the mechanisms for restructuring the overlay network were the same for both applications, some of the parameters that yielded optimal performance, such as the number of children per node or the parameters for streaming tasks, were application-specific.

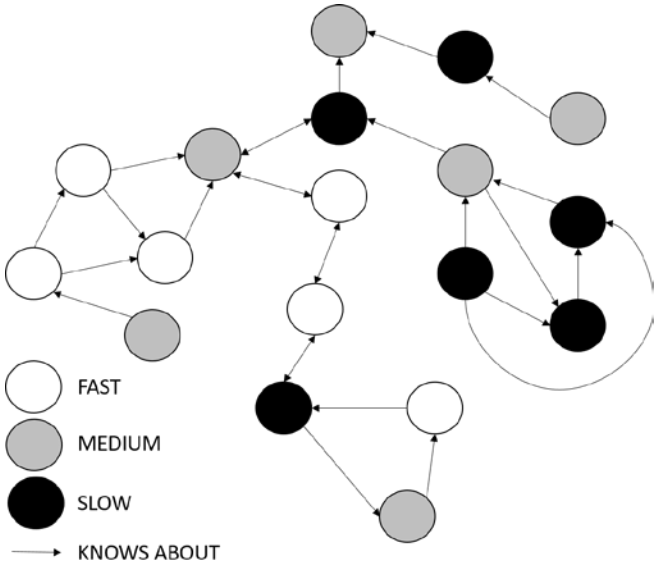

Figure 1. Organic Grid Initial Network

In the Organic Grid, the agents themselves measure the performance of their child nodes without any support from the underlying agent platform. In a cloud computing environment, this would be undesirable, since the performance information would be lost when an application terminates. We, therefore, propose that the nodes measure the performance independently of any task and provide that information to applications through an API. We also build upon the intelligently self-organizing network described in IBM's Air Traffic Controller patent (Barsness 2014). In this algorithm, groups of computational nodes are defined at the start, and provided with a leader (air traffic controller). These leaders query a central job queue, and communicate with other leaders to acquire work for their groups. A leader will choose the best task for its group, and then either request more workers from other leaders, or spawn a smaller group to size itself correctly to the task. While these groups may resize, this middleman centralization means that tasks are always only two hops away from the node that will complete them. This, however, does create a high communication burden both on the single job queue and the leader nodes that are organizing workers.

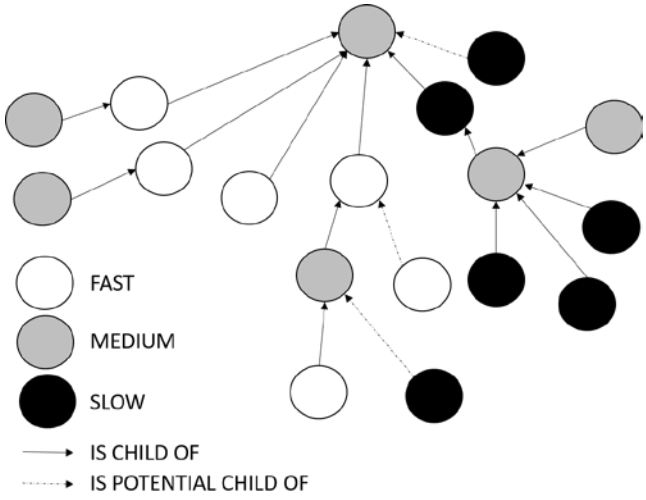

Figure 2. Organic Grid after Network Restructuring

In Figure 3, we show the organization of an ATC network, with each leader controlling a subset of worker nodes. Two jobs are currently in the queue, the smaller Job 1, and the larger Job 2. Figure 4 displays how the network changes as the jobs are distributed. Group leader A will split off a smaller network, A′ , to size itself appropriately for the smaller Job 1. Group leader B will merge with C to create a larger group to tackle the larger Job 2. Notice that the new merged or split groups maintain the intra-leader relationships that existed before the changes, so that the graph of group leaders cannot become disconnected. Additionally, a new relationship between the split groups lead by A and A′ is created. If, in the future, A's group needed to grow to take on a larger job, it would attempt first to re-merge with A′ before querying other nearby leaders, as it is assumed that the initial group layout of the network

reflects the physical organization of the system. Arranging groups in a manner that reflects hardware organization will result in groups of jobs being assigned to groups of nodes with superior interoperability.

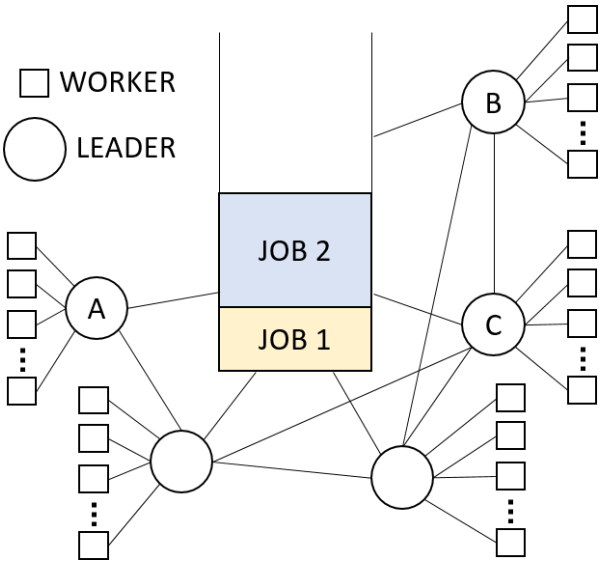

Figure 3. Air Traffic Controller Initial Network

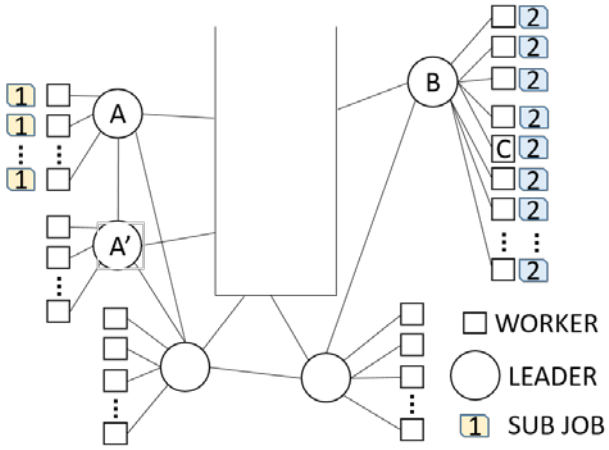

Figure 4. Air Traffic Controller after Work Distribution

3. Simulation Methodology

We have developed a simulation application to help us understand the strengths and weaknesses of different methods used to distribute work. This tool provides the measurements necessary to justify our approach to distributed computing and guide us towards how it will be implemented. It gives us a way to measure and compare the behavior of existing algorithms, and understanding what factors play the most significant role in how different approaches distribute work. We have measured simulations of both the Organic Grid (OG) and Air Traffic Control (ATC) algorithms in several different configurations to try to understand where each approach is superior and how we can develop a new system that may combine the strengths and mitigate the weaknesses of each.

A primary concern when attempting to create a simulation is showing that it is an accurate representation of the system that it simulates. We do not attempt to measure the computational time taken, we simply measure the messages sent and the burdens on individual computing nodes. By observing and categorizing the messages sent by each algorithm, we are able to compare the ways each algorithm places burden on the network. Our primary assumption is that any implementation of these algorithms on a real network would require the same types of messages in order to distribute and complete similar work. The computational time taken by the simulation, on the other hand, might be an inferior measurement because it might not match with real-world implementations of the tested algorithms. For this reason, we do not present time measurements in this preliminary investigation, we are more interested in examining how each algorithm's distribution method can be optimized than proving which is fastest

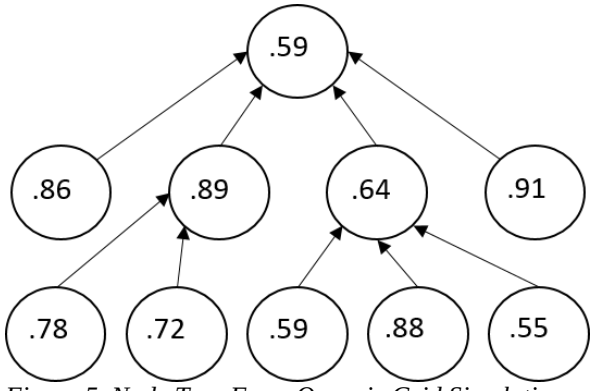

Figure 5. Node Tree From Organic Grid Simulation

Additionally, how can we be certain that what we have done will be relevant in an implementation on the cloud? The simulation attempts to accurately mimic different distributed algorithms on a common simulated set of nodes, and produces measurements to distinguish how those nodes will be burdened and how they will be utilized by each algorithm. On a cloud implementation, the differences in node performance may come from other factors, such as multiple processes competing for processor time, or delays due to communication overhead, but the simulation will still show how each distributed algorithm responds to these stressors even if the source of the stress is a simplified simulation parameter. And more importantly, the simulation can still guide us to the best experiments to run on a real implementation, and help us identify the variations on each algorithm that can result in the largest changes in performance or balance.

With the Organic Grid, we have the benefit of past prototypes against which we can compare our simulation.

The Organic Grid organizes its overlay network and moves high-performing nodes closer to the root, or source of work. In the original Organic Grid paper (Chakravarti 2006) this was done by showing a tree of “fast” and “slow” nodes, in which the “fast” nodes had migrated to near the top, leaving the “slow” nodes near the bottom, as shown in Figures 1 and 2. Our simulation produces similar results, although instead of designating only two categories of nodes, we assign each node a performance modifier as a floating point number between 0.5 and 1.0. In Figure 5 we can see the resulting overlay network from a simulated run of the Organic Grid algorithm. The nodes begin the experiment with randomly selected neighbors, and a randomly selected node is chosen to be the source of work. Notice that the root, or source of work, cannot be switched out, and therefore remains a relatively slow node. However, its immediate children are very high performing nodes. The ordering is not completely based on performance, as not enough time has elapsed to force every high performing node higher in the network, and there is a limit to the number of immediate children any node can have. Comparing this to the previous Figure 2 demonstrates that the simulation can accurately reproduce the behavior of the Organic Grid algorithm.

Defending the simulations of the Air Traffic Controller algorithm is more difficult, as we are referring to a patent and not a prior implementation. There are a range of possible implementations discussed in IBM’s patent (Barsness 2014). We simulate the basic system as described in Section II, monitoring communication as it takes place through leader nodes, while the leader nodes utilize a decentralized network to reorganize themselves and their worker nodes appropriately based on the sizes of jobs added to the job queue. To accurately compare the ATC and OG, these nodes have the same performance modifiers, however because the ATC does not utilize this information when reorganizing the network, it has no visible effect on the algorithms actions, only the speed at which work is completed We monitor messages from the job queue, among the leader nodes as they organize themselves, and from leader to worker nodes as jobs are assigned.

Messages in our simulated network are organized into information, job, and result types. We are most interested in job messages, which move work from node to node through the network, and the result messages, which move the results of completed work back to the point at which that work was inserted. A message that moves computation or results from one location to another is in all likelihood larger than a simple status or heartbeat message, and unlike a request for work, will occur more often as the network becomes more heavily burdened. It is, therefore, not just a good measurement of how well the algorithm distributes work, but by examining which nodes participate in which messages, it is also a good measurement for estimating which nodes are disproportionally burdened by the communication requirements of a distributed algorithm.

Grouping messages by type also allows us to distinguish signal from noise. For example, as both algorithms we examine are work-stealing approaches, there will be an increasing amount of unfulfilled requests for work as a system becomes less burdened. These requests are much less interesting than work transfers, and their overhead will increase when the system does not have other meaningful work to do. Finally, by examining the participants in specific messages, we can analyze which nodes in the system are playing a disproportionally large role in managing information. While we remove statistical outliers in most of our measurements, it is useful here to examine individual participants that are consistent outliers. We examine the top three most overburdened nodes under each configuration, organized by the type of message, to determine if any interesting patterns arise.

In order to compare algorithms in different environments, many simulations were performed with several different values for relevant variables. The ones we compare here are based on the number of nodes in the network, (Node Count or NC in the figures), the number of jobs added to the system (Job Count or JC), and the amount of time each piece of a job takes to complete (Job Time or JT), which we measure in milliseconds. These three variables are shared by the two algorithms we simulate and make the best points of comparison, both for these algorithms and for any future algorithms we may wish to simulate and measure. While we do not scale the number of nodes in our simulation as far as we will eventually scale our implementation, the varying size measurements still serve to show measurable differences between the OG and ATC methodologies.

4. Simulation Results

The most relevant results of the simulation concern the job messages in the network. Figure 6 shows the total number of job messages that were sent by each algorithm in each network in the eight environments simulated.

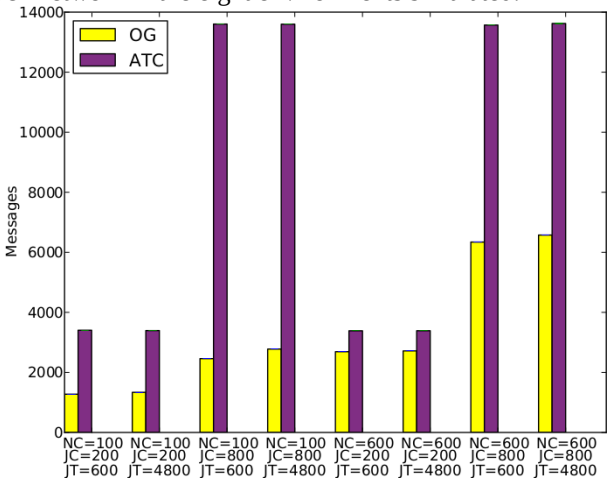

Figure 6. Simulation Job Messages Total

This includes the messages for when a subtask is moved from one node to another. There are many fragments of work for each job added, which is why for a job count of 800 one can see as many as 14000 messages under the ATC algorithm. Figure 6 shows that the number of job messages sent by the ATC algorithm is in all situations higher than the number sent by the Organic Grid algorithm. This is because in the ATC algorithm all jobs are relayed through leader nodes, while in the Organic Grid each node can both compute and relay jobs.

The ATC algorithm cannot over-allocate jobs because each leader has complete knowledge of its own group. This makes the ATC algorithm better at immediately leveraging available computing nodes. We can see that, in general, the Organic Grid improves as the number of jobs and nodes in the network increases. This suggests that adopting a decentralized strategy that contained some deliberately organized grouping with defined leaders, such as the one in the ATC, would allow us to more quickly leverage large numbers of computing nodes. However, this advantage is less evident as more nodes and work are added to the network. It is least evident in the fifth and sixth cases, which represents the simulation with the highest number of jobs and the lowest number of computational nodes, or the most-overloaded network. Still, groupings such as those defined by the ATC algorithm likely confer additional advantages when they collect nodes that work better as a group. An ideal solution should allow for nodes to be collected into groups when this produces superior results, but utilizing the grouping strategy in all situations would limit both the types of computing resources that could be managed by the network and the types of jobs that the network was capable of managing.

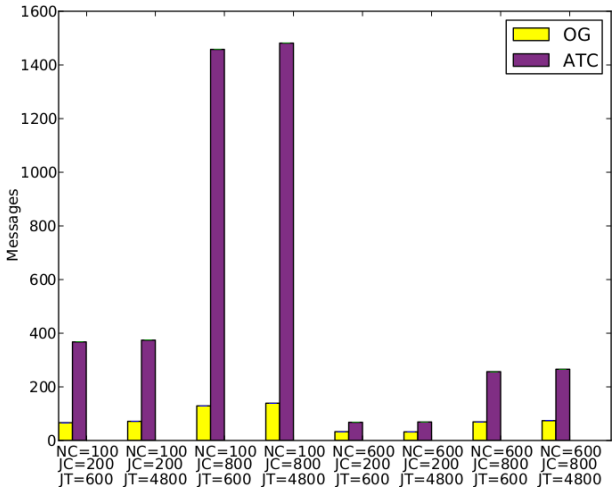

Figure 7. Simulation Job Messages on 2nd Most Overloaded Nodes

Figure 7 shows that leader nodes in the ATC algorithm can bear a disproportionally high communication load and, therefore, could become a bottleneck. Since in all cases the most-overloaded node in the ATC algorithm was the node containing the centralized job queue, we compared the number of job messages for the second-most-overloaded nodes. In six out of eight cases, the ATC approach results in a significantly larger communication burden placed on leader nodes than the Organic Grid. Only when run with the large computational pool and the smaller workload did both algorithms place similar burdens on the most used nodes. As this is the least overworked network, the problem of evenly balancing the workload becomes less difficult. Figure 7 also shows that in the case where the Node Count is 100 and the Job Count is 800, which results in the most-overloaded network, the ATC algorithm places the heaviest load on the leader nodes in the network and shows the worst performance relative to the Organic Grid.

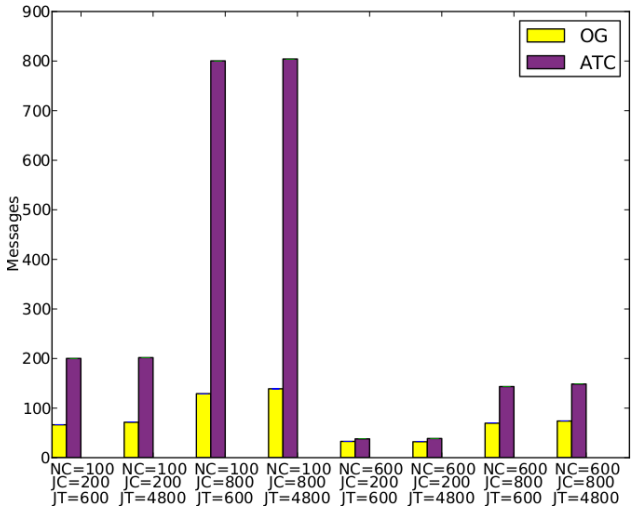

Figure 8. Simulation Result Messages on 2nd Most Overloaded Nodes

This pattern is repeated in Figure 8, a measurement of the second-most-overloaded nodes handling messages containing job results to be collected. This suggests that the Organic Grid approach may be superior at handling an overloaded network. If we adopt an approach that borrows from both methods, this suggests a further question. Is there a work-load point at which a leader node should fragment its own group in order to avoid becoming a bottleneck? Such a multi-layered fragmenting approach might be superior, but the Organic Grid already possesses this multi-layered aspect. On the other hand, an improved Organic Grid might vary the maximum number of children allowed to a single parent based on the load on an individual node or portion of the network, since lightly loaded nodes could afford to manage more children than heavily loaded nodes. This would result in an overlay network in a lightly loaded case that would be a flatter and shallower tree, while a more heavily loaded network might more optimally self-organize as a taller, narrower one.

The patterns we observed in the second-most-overloaded nodes were repeated in the most-overloaded and third-most overloaded nodes as well. However, a different implementation of the ATC might partially decentralize the job queue, or utilize a system designed to make that bottleneck irrelevant to the work throughput. Therefore, by looking at the second most-overloaded nodes, we can show the effects on group leader nodes, which is what we are most interested in, as they are a fundamental difference between the ATC and the OG approaches.

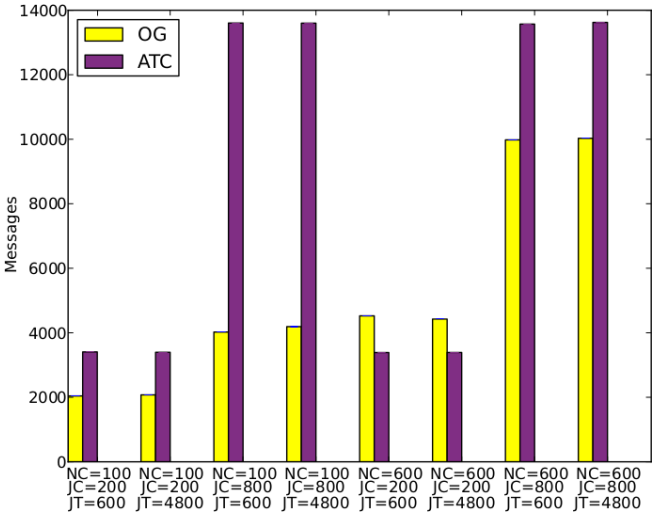

Figure 9. Simulation Job Messages Total, Organic Grid Steals More Jobs

In IBM's patent (Barsness 2014), the possibility of leader bottlenecks was discussed, a potential solution that was mentioned was to simply increase the number of leader nodes in the system. This would put multiple nodes in charge of a group. However this addition might introduce further complications when making decisions for this group, or when managing resources. A multilayered approach has the benefit of allowing one individual node to be the final manager for a group of nodes, without necessarily having knowledge of each single node that it manages.

After examining our first results we looked at what other possible variables could be changed to allow algorithms to perform more optimally. Two possible approaches to change the Organic Grid algorithm were considered, changing the initial structure of the network to be a more strongly connected graph and changing the job stealing algorithm to steal more work from parent nodes. The first approach did not result in a meaningful difference from our previous results. However the second approach did produce a definite change in a few of our metrics. Figure 9 shows the total jobs messages within the network with this altered approach. If the Organic Grid increases the work stolen, then it predictably moves more jobs, overtaking the ATC algorithm in the total number of job messages for the case with a Job Count of 200 and a Node Count of 600, which is the most underloaded network simulated. However, Figure 10 shows that while the most overloaded nodes carry a heavier burden than in the previous scenario in Figure 7, the Organic Grid still does not perform worse than the ATC algorithm. Since the ATC steals work based on the size of node groups, the amount of work it steals cannot be further optimized.

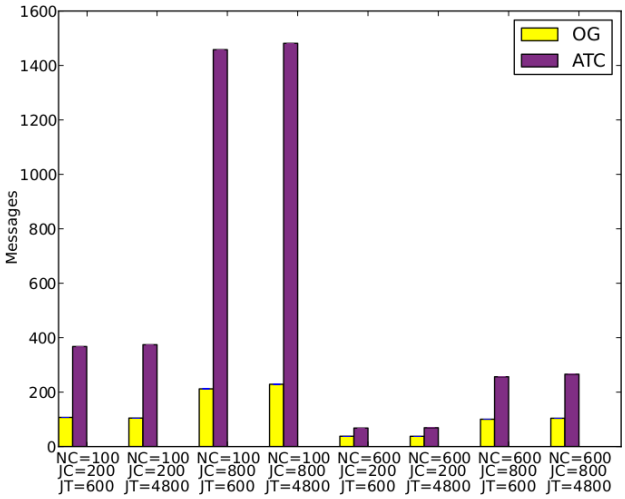

Figure 10. Simulation Job Messages 2nd Most Overloaded Nodes, Organic Grid Steals More Jobs

A crucial decision for a decentralized scheduler is enabling a node to determine the type and amount of work it should take from an individual work source. Since nodes in a decentralized network do not possess perfect information, and since any information we send has a cost, identifying what information is most relevant to balanced job distribution is critical. Our initial approach in the Organic Grid was to base the number of jobs pulled from a parent on the number of children possessed by a node. This would allow nodes with existing children to steal more work so that their entire subtree would have more work to pull. However, the fact that we can see an improved workload balance when the amount of work stolen is flatly increased suggests that this may not be the ideal metric to use. Since the network is dynamically altered over time the number of child nodes possessed by any node is in flux, and pulling a larger amount simply to force the network to distribute work faster in our simulation improves workload balance. Once a node possessing work has saturated its immediate neighbors, the only way it will obtain more children is by one of its existing children being promoted to the root. Spreading work in larger pieces also more quickly builds a larger and deeper tree, as each node gives more work immediately to the first child or neighbor who requests it. This gives the Organic Grid algorithm a larger network to begin optimizing. In a combined system the approach could change as the overlay network as a whole matures, moving

from a riskier strategy, stealing more work in order to generate more decentralized knowledge about node performance, to a more conservative strategy that is optimized based on the knowledge generated by a mature network.

We argue that ideally the best aspects of both systems should be combined. Our proposed model utilizes node grouping where appropriate, but allows a more robust system for leader nodes than the ATC, and lets work enter from any point in the network. If we extend the Organic Grid's multi-layered tree overlay network into a node leader system, we gain several advantages. One interesting difficulty with the Organic Grid is for a node to discover, using the limited knowledge available to it, how much work it should pull, keeping in mind that it will also be the path by which its children, and their children, receive work. If nodes are gathered in groups it will be possible to create a somewhat smaller overlay tree that contains more information about the total number of computational nodes available. This should combine the Air Traffic Controller's advantage in quickly disseminating work, with the Organic Grid's complete decentralization and multilayered architecture that is more appropriate to a variety of tasks, and is better able to handle an overloaded network in a balanced manner.

Both methods provide a mechanism by which the system can be restructured. In the ATC algorithm, leader nodes join or split their groups in order to best fit the available job. In the Organic Grid, the overlay network is restructured based on measuring node performance. In a combined approach, we would maintain the Organic Grid's performance-based restructuring, but additionally allow both node groups and individual nodes to participate in the network. Dividing node groups may still be necessary, but it will no longer be required to gather nodes into artificial groups. Node gathering can be done by the overlay network structure, which can maintain smaller groups below one super leader. This would avoid a potential pitfall of the ATC approach, that over time the original groups can be dissolved not because of performance, but simply based the size of past workloads. This has the disadvantage of decoupling the overlay network from the actual hardware arrangement, which could cause performance penalties. In the proposed combined approach we could ensure that node groups, while fragmented, can remain neighbors and will not need to be merged into other unrelated groups that might result in inferior performance over the entire network. Additionally, if the initial configuration is really not optimal, a measurement based restructuring can discard it.

5. CLOUD EXPERIMENTAL METHODOLOGY

For validating the simulation results, we have run experiments on the CloudLab platform (CloudLab 2014). CloudLab is designed to allow for repeatable experiments performed in the cloud. By defining a profile that describes the machines used and using randomly generated but preserved machine configurations, we can create a variety of experiments and maintain an experimental log that will allow these experiments to be reliably replicated (Ricci 2015). We used a machine template running Docker 1.6.2 on Ubuntu 12.04 LTS. These machines were provisioned on the APT Utah Cluster, which possesses two classes of nodes. The following information on the available hardware is from the CloudLab Utah hardware description (Apt Utah 2016):

Name	CPU	RAM
r320	1x Xeon E5-2450 processor	16 GB
c6220	2x Xeon E5-2650v2 processors	64 GB

Each node is connected to three network interfaces, a public Internet facing control network, a "flexible fabric" network, and a "commodity fabric" network (Apt Utah 2016).

The cloud experiment utilized nodes that were requisitioned from CloudLab in a common manner. While the simulation artificially created performance distinctions between different nodes, that was not the case in the cloud simulation. Distinctions in performance are due to differences in the hardware, in the load on that hardware, on the communication latency, and on differences in workload on the participating nodes. Even when all nodes are pulled from the same system, there may be some measurable variation in performance. Measurements of node performance are taken and compared to judge whether these similar nodes will perform measurably differently.

Our experimental platform is a Python application that uses the Python-twisted libraries for networking, packaged in a Docker container. Each running Docker instance maintains a server program that listens for incoming messages and a thread of execution to generate messages to send to other instances. This program is implemented primarily in a base class called SimMgr. This class is extended by other classes to specialize into an ATCSimMgr and ExpSimMgr. ATCSimMgr has the specialized logic required to take on the roles of nodes in an ATC system, including the Queue, Controller, and Worker nodes. It also contains extensive local logging, for both debugging and experimental measurement purposes. We will refer to Docker instances running ATCSimMgr nodes as experiment nodes. ExpSimMgr allows a Docker instance to manage and report on an entire experiment as an experimental monitor. These nodes contain information about every experimental node in an experiment, they dictate roles to experimental

nodes, and once a given experiment is completed they contact the nodes to gather and collect their individual log information. Logs are written each time useful messages are sent or received. These collected logs are the basis for the cloud-based measurements presented below.

Experimental nodes listen for the following messages:

- Send Experimental Logs
- Terminate
- Add Batch Job
- Batch Job Done
- Add Sub Job
- Sub Job Done
- Set Type
  - Controller, Queue, or Worker
  - Also informs controllers of queue location
- Add Sub Nodes (assigns workers to controller)

The experimental monitor listens for the following messages:

- Terminate
- Receive Logs
- Receive experimental node
- Batch Job Done

We simulated work in the form of 200 batch jobs, each of which contains 18 parallel portions. Each of the parallel portions range in time from 100 to 200 milliseconds. These 200 batch jobs are added to the queue node and given on a first-come first-served basis to controllers that request work. A controller then allocates work for each of its worker nodes. Once a worker completes a piece of work, it reports that to the controller, and once the controller completes a batch job, it reports to the queue. Once all jobs in the queue have completed, the experiment is over and the logs are collected.

In the cloud application, we bundled messages when communicating from a controller to the queue, which results in a reduced number of messages for the queue. This can be seen as an improvement over the original simulation methodology, however it does not necessarily translate into a reduced communication load overall, since the same information must be transmitted. Depending on the algorithm implemented, and how much information returned to the point of origin could be reduced in size, the burden on the queue could grow or shrink. However the burden on the controllers, which is more interesting, is primarily based on the number of workers that it manages. Although the overall burden will be determined by the nature of the work, the disproportionate burden will be determined by the structure of the network.

6. CLOUD RESULTS

The cloud experiment is an implementation of the ATC algorithm that was tested in the original simulation. The cloud experiment was very similar to the smaller scale simulation experiment. This experiment was run on requisitioned cloud machines, and used TCP-IP for interprocess communication, where the original simulation was run on a single machine and simulated all messages. Both experiments used simulated work, with a major job consisting of a certain number of distributable sub-jobs, each of which takes a defined amount of time.

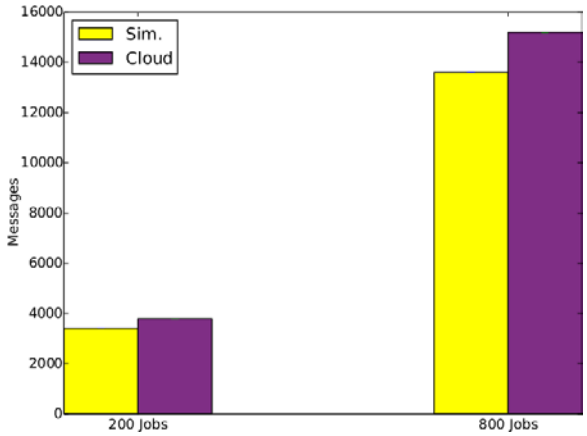

Figure 11. Simulation and Cloud Experiment Job Message Total Comparison

The cloud experiment measurements operate on a different implementation of the ATC algorithm than the one used in the simulation, therefore the numbers generated are not identical. However the same patterns are evident in the cloud experiment that were seen in the simulation. This experiment produced a similar, although not exactly the same, number of total job messages as the 100 nodes, 200 jobs and 100 nodes, 800 jobs configurations from the simulation measurements, as seen in Figure 11. The cloud experiment used 100 workers, five controllers, a single queue, and 200 or 800 jobs. The controllers are set by the algorithm and do not dynamically shift during the experiment. The five most overburdened nodes are all controller nodes, with the queue as the sixth. The controller nodes in the cloud experiment are more heavily burdened than those in the simulation experiment. Either experiment demonstrates the large messaging burden placed on controller nodes, as seen in Figure 13. Both of these experiments, therefore, point to the need for a more decentralized method for work scheduling as work scales up on Cloud systems.

In Figure 12 we compare the job message burdens on the controllers in the cloud and the simulation. The columns marked “1st” refer to the most-burdened controller node, those marked “2nd” refer to the second-most-burdened controller node. This comparison shows that both systems display a consistent pattern. The simulation experiments produced much smaller error bars, as the simulation is in general very consistent. In the cloud the actual performance

variations on real hardware produced a much larger 95% confidence interval

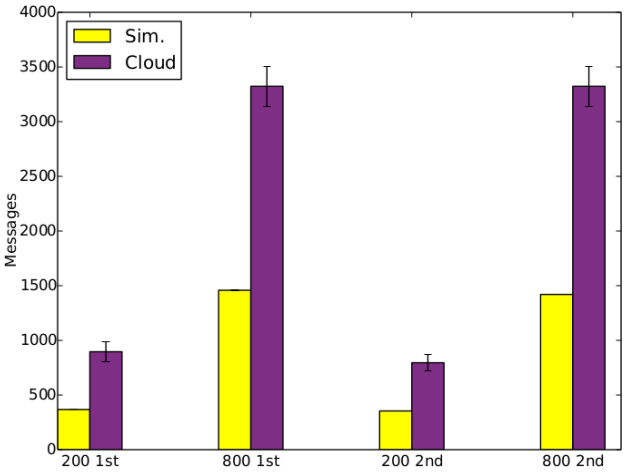

Figure 12. Simulation and Cloud Job Message Burden Complete

When we compare the simulation experiment and the cloud experiment, we notice some differences but the patterns are similar. In Figure 11 we can see a shift in our measurements moving from simulation to cloud. However, Figure 12 shows that in both tests we measure a larger messaging burden on the controller nodes. We can see that the algorithm performed largely the same in the cloud environment as it did in the simulation. In Figure 11 the cloud experiments, as the simulation experiments, were very consistent in the compared measurement of absolute job messages, hence the essentially invisible error bars. This is largely due to the implementation of the algorithm being quite similar, and the scope of the problem being as similar as possible when migrating to a new system and implementation.

While Figure 11 may be used to compare the cloud approach to the simulation approach, the burdens on the individual overburdened nodes are a more interesting measurement. In the simulation, all performance distinctions were due to artificial differences in performance defined by the simulated environment. In the cloud, we are in the position to test whether we will see performance distinctions emerge from machines that should be requisitioned from one uniform system without artificial distinctions in performance. In fact, the experiment confirms the need to handle variation in performance of cloud hardware.

Beyond validating the simulation, the cloud experiment is specifically useful for pointing out that even cloud hardware on a common system can show performance variation. Figure 13 shows that there is a clear difference between the burden placed on the different controllers. While there is some variation in the measured burdens, and the 95% confidence intervals of the most-overloaded and second-most-overloaded controllers does overlap, this is not the case for the most- and least-overloaded controller nodes. There is a definite measurable difference in burden on different controllers. This lends support to the idea that a distributed scheduler, particularly one that measures the performance of the nodes on which work is scheduled, would be useful.

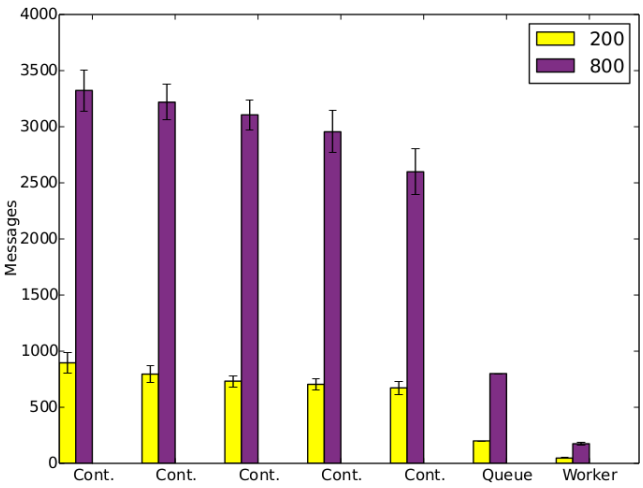

Figure 13. Cloud Messages on Most Overloaded Nodes

7. SUMMARY

We have presented a framework for running high-performance and many-task applications in the cloud. Since cloud resources often have unpredictable performance characteristics, we have proposed to periodically measure the performance of cloud nodes and network connections to allow computational tasks to be placed on the most appropriate (sets of) nodes. Since centrally maintaining such detailed performance information for a large network is prohibitively expensive, we have argued that some degree of decentralization is needed. We have evaluated two candidate decentralized approaches, our Organic Grid and IBM's Air Traffic Control framework, by simulating their network traffic.

Our simulations have shown that group leaders such as those used by the ATC framework can be very beneficial in efficiently allocating work to many child nodes. However, when networks are more heavily loaded that same strategy can result in the leader nodes becoming saturated in trying to organize much more computation than can be easily handled. The Organic Grid's decentralized methods are not as effective at moving work quickly, but the burden of work distribution and management is spread more evenly through the network. Managing computation in the cloud requires utilizing a scalable approach that can dynamically adapt to variations in cloud hardware performance, and our simulation results show that our proposed cloud framework will be able to provide that scalability.

While our simulations have been run with limited numbers of nodes and jobs and have only measured communication events, they nonetheless have shown clear trends in the expected performance of the ATC and OG approaches to distributing computational tasks. They are providing guidance in designing a better framework for deploying high-performance and many-task applications in the cloud.

We have also run further experiments in the cloud to verify both the simulation's accuracy and the existence of performance variation between cloud nodes. We did not attempt to replicate every simulated experiment in the cloud, as we want to move towards building complete functional systems as a next step, not continuing to build more and more sophisticated simulations. However, since the general patterns of the simulation have also been observed on the cloud platform, we are more confident in using the simulation to guide our future progress. Both the simulation and cloud experiments help to make the case that we need a more sophisticated, decentralized work scheduling model based on concepts from both the ATC algorithm and the Organic Grid.

We are currently working on developing a prototype based on the results of our measurements. The specific conclusions that will guide the prototype design are the subject of the remainder of this section.

Having group leaders can improve the efficient assignment of groups of tasks to groups of nodes, but it can also result in those same leaders becoming bottlenecks. While using an overlay network instead of a central job queue performs and scales better overall, it may be beneficial to use group leaders similar to the ones in the ATC framework near the leaves of the overlay network.[1]

There are many other characteristics of jobs and computational nodes that might produce interesting information to guide job scheduling. Developing an extensible system would allow an intelligent platform in a decentralized network to utilize many different parameters to judge the correct method to match work to processors. We need to ensure that rather than limiting our system to only the attributes that we identify as useful, our system can allow the growth, identification, and use of as-yet unknown parameters, the measurement and monitoring of which could result in better solutions to the distributed scheduling problem. While we may not be able to build a system that can adjust fully to the unknown, we intend to make it as extensible as possible. Since our approach does not rely on any specific cloud vendor or technology, it could be used to construct a hybrid cloud that overlays potentially multiple academic and commercial clouds as well as spare desktop machines and local supercomputers.

Additionally, it will be necessary to look at a variety of methods to move tasks through the network. A basic starting point is simply moving atomic tasks onto specific machines, but many cloud models are built on virtual machines that may migrate between hardware locations while processes are running. Work done on self-organizing overlay networks of VMs (Ganguly 2006) has demonstrated the possibility of extending our approach to handle VM migration but also the extreme cost in communication latency between migrating virtual machines. Work has been done to optimize migration itself (Hacking 2009), but if our work is fine-grained enough it may be necessary to use a more fine-grained method for work migration such as using mobile agents with strong mobility (Chakravarti 2003) as containers for groups of tasks.

We are currently developing a distributed scheduler, which we are implementing and testing on the same CloudLab platform that we tested the ATC on. The individual distributed nodes will contain the ability to monitor the performance of a specific piece or pieces of hardware and are responsible for presenting that information to adjacent nodes organized in a graph of computing resources. They will utilize both node performance and workload information to schedule tasks on the network. Eventually, we plan to add the ability to utilize information about specific task requirements as a third variable to use for more specific scheduling. At each step we are performing measurements in order to determine the usefulness of each piece of information in making scheduling decisions. We intend to determine the number of job movements done by the network when utilizing different scheduling strategies, as well as the distance traveled by a job and whether any unnecessary movement of work occurred. All of these measurements can be used to determine the relative value of each piece of information when making scheduling decisions. Additionally, if we allow weighting the different pieces of information in scheduling decision, we can approach optimal weights, at least at specific points in time, for the use of any available information in the scheduling problem.

Our eventual goal is building a distributed system for scheduling many-task computing problems. We are working on target problems defined by task graphs, to allow for a mixture of dependent and independent jobs, as well as jobs that may have differing hardware requirements and expected run times. We predict that some of these tasks will require tight communication between several nodes to perform optimally, while some will be more easily handled by nodes that require little to no communication. This type of problem, with a mixture of high performance requirements and tasks that can be distributed to lower performing nodes provides an ideal testing ground for a hybrid cloud platform unifying disparate resources. Going forward we plan to work with a specific application that seeks to gain an

[1] We could think of those leaders as the approach and tower controllers who organize the flow of traffic in the vicinity of an airport, while the overlay network is used for traveling between airports, as in actual air traffic control.

understanding of chemical properties by generating a task graph based on a tensor operation.

8. REFERENCES

D. Abramson, J. Giddy, and L. Kotler. (2000) High performance parametric modeling with Nimrod/G: Killer Application for the Global Grid? *In Proc. Intl. Parallel and Distributed Processing Symp.*, pages 520–528, May 2000.

Apt Utah technical information. (2016) http://docs.cloudlab.us/hardware.html. Retrieved 2016.

E.L. Barsness, D.L. Darrington, R.L. Lucas, and J.M. Santosuosso. (2014) Distributed job scheduling in a multi-nodal environment. http://www.google.com/patents/US8645745, February 4 2014. US Patent 8,645,745.

Berkeley Open Infrastructure for Network Computing (BOINC). http://boinc.berkeley.edu/. Retrieved 2016

F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. (2003) Adaptive computing on the Grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):369–382, 2003.

D. Buaklee, G. Tracy, M. K. Vernon, and S. Wright. (2002) Near-optimal adaptive control of a large Grid application. In *Proceedings of the International Conference on Supercomputing*, pages 315–326, June 2002.

Arjav J. Chakravarti, Gerald Baumgartner, and Mario Lauria. (2005) The Organic Grid: Self-organizing computation on a peer-to-peer network. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 35(3):373–384, May 2005.

Arjav J. Chakravarti, Gerald Baumgartner, and Mario Lauria. (2006) Selforganizing scheduling on the Organic Grid. *International Journal on High-Performance Computing Applications*, 20(1):115–130, January 2006.

Arjav J. Chakravarti, Xiaojin Wang, Jason O. Hallstrom, and Gerald Baumgartner. (2003) Implementation of strong mobility for multi-threaded agents in Java. *In Proceedings of the International Conference on Parallel Processing*, pages 321–330. IEEE Computer Society, October 2003.

Andrew A. Chien, Brad Calder, Stephen Elbert, and Karan Bhatia. (2003) Entropia: architecture and performance of an enterprise desktop Grid system. *J. Parallel and Distributed Computing*, 63(5):597–610, 2003.

Cloudlab. (2014) https://www.cloudlab.us/, Retrieved 2014.

Jeffrey Dean and Sanjay Ghemawat. (2008) MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107– 113, 2008.

Constantinos Evangelinos and Chris N. Hill. (2008) Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2. In *The 1st Workshop on Cloud Computing and its Applications* (CCA, 2008).

folding@home. http://folding.stanford.edu. Retrieved 2016

Arijit Ganguly, Abhishek Agrawal, P Oscar Boykin, and Renato Figueiredo. (2006) WOW: Self-organizing wide area overlay networks of virtual workstations. In *High Performance Distributed Computing, 2006 15th IEEE International Symposium* on, pages 30–42. IEEE, 2006.

Andrew S. Grimshaw and W. A. Wulf. (1997) The Legion vision of a worldwide virtual computer. *Comm. of the ACM*, 40(1):39–45, January 1997.

Stuart Hacking and Benoˆıt Hudzia.(2009) Improving the live migration process of large enterprise applications. In *Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing, VTDC '09*, pages 51–58, New York, NY, USA, 2009. ACM.

Elisa Heymann, Miquel A. Senar, Emilio Luque, and Miron Livny. (2000) Adaptive scheduling for master-worker applications on the computational Grid. In *Proc. of the First Intl. Workshop on Grid Computing*, pages 214–227, 2000.

IBM. (2015) World Community Grid. http://www.worldcommunitygrid.org/.

N. T. Karonis, B. Toonen, and I. Foster. (2003) MPICH-G2: A Grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing*, 63(5):551–563, 2003.

T. Kindberg, A. Sahiner, and Y. Paker. (1994) Adaptive parallelism under Equus. In *Proceedings of the 2nd International Workshop on Configurable Distributed Systems*, pages 172–184, March 1994.

Michael Litzkow, Miron Livny, and Matthew Mutka. (1988) Condor — a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, June 1988.

Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra A. Hensgen, and Richard F. Freund. (1999) Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Proceedings of the 8th Heterogeneous Computing Workshop*, pages 30–44, April 1999.

Ioan Raicu, Ian T. Foster, and Yong Zhao. (2008) Many-task computing for Grids and supercomputers. In *Proceedings of the 2008 Workshop on Many-Task Computing on Grids and Supercomputer*s (MTAGS 2008), pages 1–11, Austin, TX, November 2008. IEEE.

Samyam Rajbhandari, Akshay Nikam, Pai-Wei Lai, Kevin Stock, Sriram Krishnamoorthy, and P. Sadayappan. (2014) A communication-optimal framework for contracting distributed tensors. In *Proceedings of SC14, The International Conference on High Performance Computing, Networking, Storage, and Analysis*, New Orleans, LA, 16–21 November 2014.

Robert Ricci, Gary Wong, Leigh Stoller, Kirk Webb, Jonathon Duerig, Keith Downie, and Mike Hibler. (2015) Apt: A platform for repeatable research in computer science. *SIGOPS Oper. Syst. Rev.*, 49(1):100–107, January 2015.

SETI@home. http://setiathome.ssl.berkeley.edu. Retrieved 2016.

Ian Taylor, Matthew Shields, and Ian Wang. (2003) *Grid Resource Management*, chapter 1 — Resource Management of Triana P2P Services. Kluwer, June 2003.

Edward Walker. (2008) Benchmarking Amazon EC2 for high-performance scientific computing. LOGIN, 33(5):18–23, October 2008.

G. Woltman. GIMPS: The great internet Mersenne prime search. http://www.mersenne.org/prime.htm.

Reynold S. Xin, Joseph E. Gonzalez, Michael J. Franklin, and Ion Stoica. (2013) GraphX: A resilient distributed graph system on Spark. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, pages 2:1–2:6, New York, NY, USA, 2013. ACM.

Authors

Brian Peterson received his Ph.D. in computer science from Louisiana State University. His research interests include high performance cloud applications, large scale decentralized systems, and automated program vulnerability analysis.

Gerald Baumgartner received his Ph.D. in computer science from Purdue University. He is currently an associate professor at the Division of Computer Science and Engineering at Louisiana State University. His research interests include compiler optimizations, the design and implementation of domain-specific and object-oriented languages, cloud computing middleware, data mining of social network graphs, and testing tools for object-oriented and embedded systems programming.

Qingyang Wang received his Ph.D. in computer science in 2014 from the Georgia Institute of Technology. He is currently an assistant professor at the Division of Computer Science and Engineering at Louisiana State University. His research interests include distributed systems and cloud computing, and performance and scalability analysis of large-scale Web applications.