Scheduling Grid Workloads on Multicore Clusters to Minimize Energy and Maximize Performance

Michael Lammie, Paul Brenner, and Douglas Thain
Department of Computer Science and Engineering
University of Notre Dame

Abstract

Energy is a significant and growing component of the cost of running a large computing facility. A grid workload consisting of millions of jobs running on thousands of processors may consume millions of kilowatt hours of electricity. However, because a grid workload generally consists of many independent sequential processes, we may shape its execution to satisfy energy constraints. By varying the number and frequency of processors available, a scheduler may trade off energy against performance. In this paper, we explore energy and performance tradeoffs in the scheduling of grid workloads on large clusters. We build upon previous work by showing the interaction of intelligent job assignment, automated node scaling, and frequency scaling on multicore clusters. An unexpected result is that, even though low frequency is the most efficient mode of operating a single node, the careful application of frequency scaling can actually reduce overall energy consumption even further by reducing the number of nodes powered on.

1 Introduction

Energy is a significant and growing component of the cost of running a computational infrastructure. A grid workload consisting of millions of jobs running on thousands of processors may consume millions of kilowatt hours of electricity. Because a grid workload generally consists of many independent sequential processes, we may shape its execution to satisfy energy constraints. By varying the number and frequency of processors available, a scheduler may trade off energy against performance. In this paper, we explore energy and performance tradeoffs in the scheduling of grid workloads on large clusters. We build upon previous work by showing the interaction of intelligent job assignment, automated node scaling, and frequency scaling on multicore clusters.

More specifically, we find that utilizing frequency scaling can reduce the overhead energy cost asscoated with automated node scaling while assigning jobs to machines intelligently can reduce the total number of active machines.

We begin with a summary of the primary grid tools utilized and related work. Our model is then introduced to include grid workloads, baseline power measurements, and management policies. Performance and energy measuresments are then reported based on grid tool execution to meet policy objectives. We conclude with a short discussion of results and recommended policy implementations.

CPU Frequency Scaling (FS), also known as CPU throttling, is a technique for reducing the speed of the CPU in order to reduce the energy it consumes and the heat that it dissipates. FS and more broadly dynamic voltage and frequency scaling (DVFS) has been widely researched [11, 13, 16, 7, 17] most often in the context of power management in mobile systems where long battery life, reduced fan noise, and limited heat dissipation are high priority. A majority of modern chipsets will support some version of CPU frequency scaling. Intel chipsets utilize SpeedStep technology and AMD utilizes Cool'n'Quiet or PowerNow. With kernel support, the frequency of a CPU, and in some instances specific cores within the CPU, can be modified through software. This makes dynamic CPU frequency scaling simple to implement in a number of applications. In this work we examine dynamic FS in our grid management algorithms to balance energy efficiency and performance (throughput). Optimization decisions (intelligent job assignments) are then evaluated in coordination with automated node scaling.

Automated Node Scaling (ANS) is an effort to reduce energy consumed by under utilized machines [10, 2, 19, 6]. A mechanism is employed to dynamically calculate cluster capacity requirements based on the current state of the cluster, including the size of the queue, characteristics of currently running jobs, and historical usage information. As the name suggests, the mechanism automatically turns machines on or off in order to match these requirements and thus only active resources are consuming energy. Machines are turned on when it is deemed that the cluster cannot support the current load. Machines are powered down after remaining idle for a specific period of time. In this work we evaluate automated node scaling in conjunction with frequency scaling to determine intelligent job assignment under multiple published grid workloads.

Intelligent Job Assignment (IJA) is the third layer of optimization implemented in our set of cluster management algorithms. Numerous tools and publications have been developed to evaluate and optimize assignments [4, 20, 8, 1, 5, 18]. The focus of this work is to specifically identify energy saving assignments at target performance levels given availability of FS and ANS. An example application for such dynamic assignment: 1) Jobs which require a significant amount of time to process are submitted in bursts amidst other, less demanding jobs. 2) The burst of jobs are processed and the load on the cluster subsides. 3) The jobs which require additional time still remain on their respective machines. 4) Even though only a few jobs are being processed by the cluster, a relatively high number of machines remain turned on due to poor assignment. In this work we examine the peformance and energy metrics for policies that address assignment for similar applications.

2 System Model

This study was conducted in order to identify potential uses for CPU frequency scaling and automated node scaling in multicore clusters. The simulations described in this document were conducted within a framework which assumes that the configuration of a cluster abides by certain conditions. First, each node within the cluster is of an identical specification. As a result, each node within the cluster utilizes a CPU capable of scaling its CPU frequency to identical levels. Therefore, the maximum and minimum CPU frequencies of nodes as used in each cluster management algorithm are identical for all nodes in the cluster. Second, the energy consumed by each node under the same load is also identical. Lastly, it is assumed that all jobs submitted to the cluster are strictly CPU intensive and use 100% of the CPU cycles assigned to them. As a result, latencies related to memory, disk, and network access are ignored.

Each simulation is conducted using a specific cluster configuration which consists of the number of nodes, the number of cores per node, the maximum and minimum CPU frequencies of a node, energy consumption data for a single node, and the cluster management algorithm. In addition to the cluster configuration information, the simulator uses an actual grid workload archive as a basis for job submission, which determines job submission rate, distribution of jobs over time, and the amount of work to be completed for each job.

To drive the simulations, we employ trace data from the Grid Workloads Archive (GWA) [12]. We selected the four largest and most complete traces from the archive, Grid5000 [3], NorduGrid [9], DAS-2 [15], and LCG [14]. Each trace records details about every job submitted to the system over a fixed period of time. For this work, we only require the following fields: *Submit time* is used to determine when a job is to be submitted to the job queue. *ID number* is used to identify a specific job. *Run time* is used to determine the amount of work to be completed, where work is equal to the number of CPU cycles required for completion. For simplicity, it is assumed that this number is equal to the run time of a job on a processor running at the highest possible frequency for the cluster configuration. For example, assuming that the highest frequency is 2.4 Ghz, a job with a grid workload archive run time of 300 will take 300 seconds to execute on a 2.4 Ghz processor. If the frequency of the processor is reduced to 1.6 Ghz, the run time of the job will increase by a factor of 1.5 to 450 seconds.

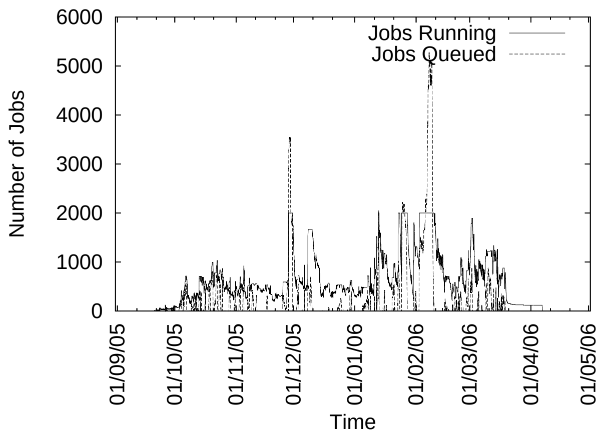

Figure 1. Burstiness of Trace Data

Grid workloads have been observed to be very bursty, and these traces are no exception. Figure 1 shows the number of jobs running and queued in a simulated cluster of 500 4-core nodes running the NorduGrid workload. It is not uncommon for hundreds or thousands of jobs to be submitted simultaneously. All cores will be busy for some time, but eventually become idle as the queue drains, until the next burst arrives. This property provides the opportunity for several energy optimizations.

Number of Busy Cores	Cycles/Joule at 1.60 GHz	Cycles/Joule at 2.40 GHz
0	0	0
1	33.3	37.5
2	59.3	61.5
3	77.4	78.3
4	94.1	90.5

Table 1. Single Node Power Efficiency

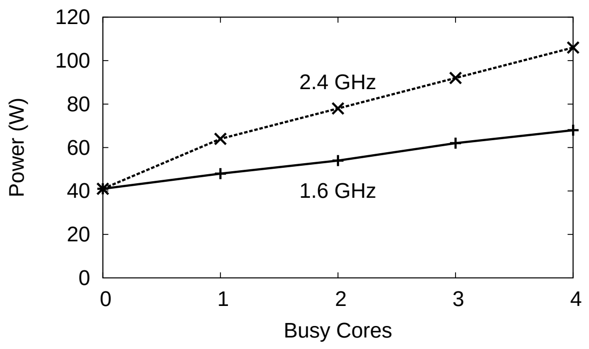

Figure 2. Single Node Power Consumption

3 Single Node Measurements

The CPU speed of a computing node is generally set to a default value equal to the maximum frequency as supported by the hardware vender. However, a majority of systems are capable of modifying the speed of its CPU on-demand while a system is still functioning. Our study utilized a sample node with an Intel Core 2 Quad processor equiped with Intel's Enhanced SpeedStep DVFS technology. The single CPU motherboard also included 4GB of RAM and a solid state flash disk. We collected the maximum and minimum supported frequency values and total energy consumption measurements (at the power plug) for this node.

To simplify exposition, we assume all cores in the CPU of a single machine are scaled to the same frequency, either the highest supported frequency or the lowest supported frequency. Therefore, power dissipation measurements were taken for each sample node at each of these frequencies. Individual cores were stressed to 100% CPU utilization one at a time until all cores were busy. Figure 2 represents the energy consumption measurements for the sample node under varying levels of stress.

Since our system model assumes that all jobs submitted to the cluster are CPU intensive, it is easy to calcuate the energy efficiency of a specific node under ideal conditions. After analyzing the energy consumption data for the sample node, we see that there is some overhead energy associated with running a node no matter the frequency or the number of busy cores. We can extrapolate from this that the node is most energy efficient when it is running jobs on all available cores at the same time, thus reducing the significance of the overhead and distributing its cost across multiple jobs. Furthermore, a node will be running jobs on all available cores quite frequently during the course of the simulation due to the characteristics of a grid workload as described in Section 2. Jobs tend to be submitted in bursts larger than the number of cores in a single node or even in the entire cluster. Therefore, it is common that the cluster will be at capacity and the nodes within the cluster will be running jobs on all available cores.

By multiplying the frequency by the number of busy cores, and dividing by the power dissipation of the node at the designated load, we acheive a ratio equal to the amount of work completed per second divided by the amount of energy consumed per second. The result being work per joule, or cycles per joule. For example, the sample node at a high frequency and running at maxmimum capacity has an efficiency metric value equal to 90.5 million cycles per joule. A low frequency configuration running at maximum capacity produces an efficiency metric equal to 94.1 million cycles per joule. Table 1 lists energy efficiency metric values for the sample node under varying levels of stress.

The key property of this machine is the efficiency benefit gained by running the processor at a lower frequency when all cores are in use. Although our simulations are seeded with the properties of this particular machine, substituting for a machine with the same property will yield similar results.

4 Cluster Management Policies

Baseline Cluster Management Policies. The study began by constructing several baseline cluster management policies. These policies were meant as a control to identify primitive uses for frequency scaling techniques. Each policy is based on the following job submission scheme.

All machines are always powered on. One job is assigned to a single core on a single node. Machines are arbitrarily ordered from 0 to N-1, where N is the total number of nodes in the cluster. As jobs are submitted, they are pushed onto the job queue. Jobs waiting in the queue are submitted to a core in the cluster using First In First Out priority ordering. They are submitted to the lowest ordered machine (0...N-1) with an idle core. All cores in a single node are to be scaled to the same frequency, either the highest supported frequency or the lowest supported frequency depending upon which cluster management policy is in use.

HF The first policy is the high frequency policy. In this policy, the frequency of CPUs is always set to the highest supported frequency. All machines remain in this state throughout the duration of the simulation.

LF The second policy is the low frequency policy. In this policy, the frequency of CPUs is always set to the lowest supported frequency. All machines remain in this state throughout the duration of the simulation.

Scaled Cluster Management Policies. First, it is important to define *automated node scaling* as it is utilized in our cluster management policies. In our implementation, *automated node scaling* is powering machines on and off based on the number of jobs currently running in the cluster and waiting in the queue. More specifically, *automated* *node scaling* is implemented in the following manner.

Initially, all machines are powered down. Those policies which do not utilize frequency scaling turn additional machines on when the current cluster of active nodes is at capacity and additional jobs remain in the queue. Policies which utilize frequency scaling will first increase the frequency of active machines in order to increase the throughput of the cluster prior to turning additional machines on. Machines are turned off after remaining in an idle state beyond a specific threshold. For example, a threshold value of 300 seconds (5 minutes) was commonly used in our simulations. Turning a node on or off takes time. Nodes in the process of booting up are considered ”on” by the cluster manament policies. However, jobs cannot be assigned to them until they have completed the boot process.

SHF All CPUs are scaled to the highest supported frequency. As jobs are submitted, they are assigned to idle cores in the cluster. If there are no idle cores, machines are turned on such that the number of additional cores is equal to the number of jobs waiting in the queue.

SLF All CPUs are scaled to the lowest supported frequency. As jobs are submitted, they are assigned to idle cores in the cluster. If there are no idle cores, machines are turned on such that the number of additional cores is equal to the number of jobs waiting in the queue.

SSQ All CPUs are initially scaled to the lowest supported frequency. If there are jobs waiting in the queue, frequencies of machines currently turned on in the cluster are increased such that the frequency of one core is increased for each job waiting in the queue. If simply increasing the frequency of all machines powered on in the cluster cannot satisfy all jobs waiting in the queue, additional machines are turned on in order to fully satisfy the remainder of the jobs in the queue which were not accounted for by increasing the frequency of all previously turned on machines.

SWQ All CPUs are initially scaled to the lowest supported frequency. All CPUs remain at this frequency while there are idle cores in the cluster. Once the cluster of active nodes is at capacity, frequencies of CPUs are increased such that the throughput of the cluster is increased by the equivalent of an additional core (at low frequency) for each job waiting in the queue. If increasing the frequencies of all nodes in the cluster cannot satisfy all jobs waiting in the queue, additional machines are powered on. Machines are powered on such that one core is being activated for each job in the queue which went unaccounted for while increasing the frequencies of CPUs in the cluster.

Scaled Cluster Management Policies with Intelligent Job Assignment. In the previous cluster management policies, the next job in the queue was submitted to an idle core on the lowest ordered available machine. The following policies attempt to improve the job assignment algorithm for each of these policies. While there are a num-

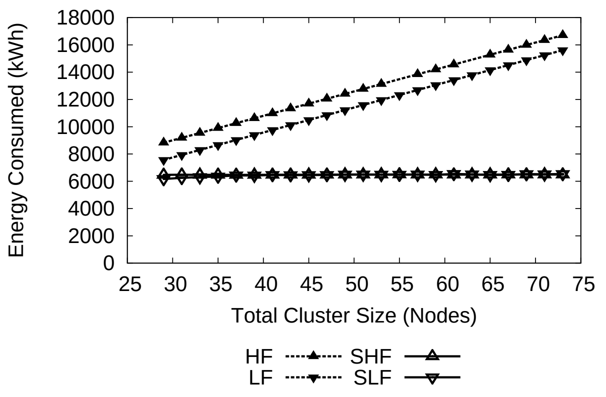

Figure 3. Grid5000: Node Scaling
The energy consumption of the Grid5000 workload with four basic policies, demonstrating that the most important policy is to turn off idle nodes.

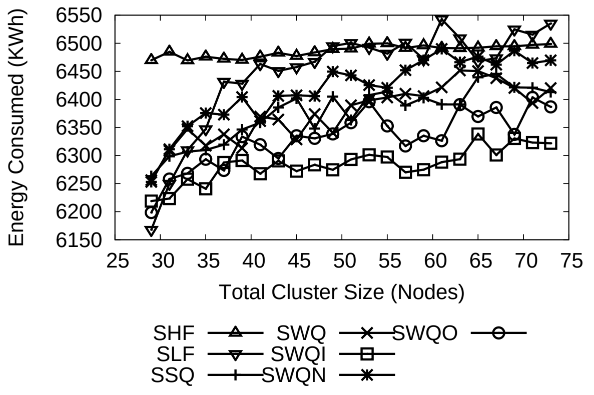

Figure 4. Grid5000: All Scaling Policies
The total energy consumed by all policies on the Grid5000 workload. For clarity, we omit SSQ, SWQ, and SWQN from the remaining results.

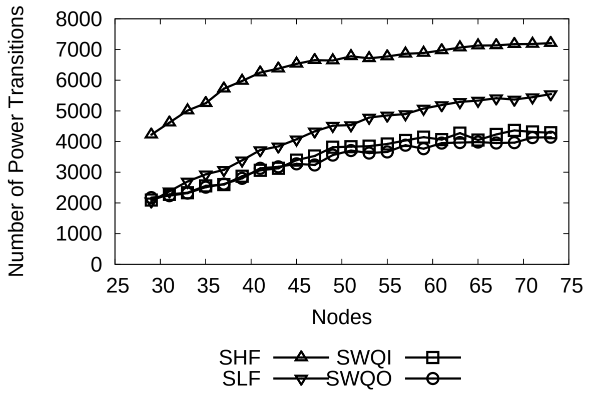

Figure 5. Grid5000: Power Transitions
Number of power transitions in Grid5000. SWQI and SWQO balance power transitions against CPU frequency.

ber of methods for optimizing assignment, we decided that the next job in the queue should be submitted to the machine scheduled to be powered on for the longest duration of time. *SWQI* assumes that the amount of work associated with each job is known at the time of submission. Using this information, we can determine which machines are to be powered on the longest at that point in time.

However, it is not realistic to assume that we know the amount of work required by a job prior to job submission. *SWQN* and *SWQO* attempt to replicate the ideal scenario by *estimating* which machine is scheduled to be turned on the longest. We decided to use the submit time of each job in the cluster as a basis for our estimation. There were initially two hypotheses for estimating which node is scheduled to be turned on the longest. The first is that the node with the most recent job submission is likely to be on the longest based on the presumption that all other jobs in the cluster were already partially completed. Therefore, it is likely that the latest job submission to the cluster will be running longer than the remaining jobs. The second hypothesis is that the machine with the oldest running job is likely to be turned on the longest. This theory was based on analysis of grid workload characteristics which show that it is not uncommon to find jobs in the queue which require substantially more work than most other jobs submitted to the cluster. Therefore, the machine with the oldest running job is likely to be turned on the longest.

SWQI This policy improves upon the job assignment mechanism of *SWQ*. We assume that the amount of work required by a job is known at the time of submission. With this information, we know how much work still needs to be completed for each job in the cluster. This allows us to identify which machine with an available core is scheduled to be on the longest. The next job in the queue is then assigned to a core on this machine. This policy is, in general, unrealistic since it is nearly impossible to specifically know the run time of a job at the time of submission.

SWQN This policy attempts to improve upon the job assignment of *SWQ*. Rather than submitting a job to an idle core on the lowest ordered machine, a job is assigned to the machine running the most recently submitted job.

SWQO This policy attempts to improve upon the job assignment mechanism of *SWQ*. Rather than submitting a job to an idle core on the lowest ordered machine, a job is assigned to the machine with the oldest running job.

5 Simulation Results

We begin by examining one workload – Grid5000 – in some detail. After establishing some basic properties, we consider the energy and performance of all four workloads.

Figure 3 illustrates the difference in total energy consumed between the baseline (HF and LF) and scaled policies (SHF and SLF) on the Grid5000 workload. The figure demonstrates that HF consistently consumes more energy than LF. This can be explained by examining the efficiency metrics for the sample node from Section 3, where we determined that a node is more efficient at a low frequency. In addition, the amount of energy being consumed increases according to the number of nodes in the cluster. Since these policies do not utilize automated node scaling, this is to be expected. There is a significant amount of energy being consumed by idle machines, and as the number of nodes in the cluster grows, so does the amount of wasted energy. By utilizing automated node scaling in the enhanced management policies, we eliminate most of this overhead energy.

We studied the behavior of all policies across all workloads, but determined that *SWQI* and *SWQO* achieved the best efficiency and performance overall. For completeness, we show the energy consumption of all policies on Grid5000 in Figure 4. For clarity, the remaining figures only show *SHF*, *SLF*, *SWQI*, and *SWQO*.

Figure 4 shows that SHF consumes more energy than any other policy. This can be explained in part by referencing the efficiency metric calculated for the sample node in Section 3. We determined that the sample node was less efficient at a frequency of 2.4 GHz than at 1.6 Ghz. As a result, the SHF policy consumes more energy throughout the duration of each simulation, and the SLF policy consumes approximately 3.3% less energy than the SHF policy across all grid workload archives.

More interstingly, however, we see an additional decrease in energy consumption by each of the frequency scaling policies in all but extremely over-burdened cluster configurations. Figure 5 explains this by showing the number of power transitions. In order to power a node on or off, energy is required during boot and shutdown. This energy is considered overhead energy, since it is not directly used to process a job submitted to a cluster. Therefore, as the number of transitions of nodes on and off increases, the amount of wasted energy in the form of overhead increases. The SHF and SLF policies require a relatively large number of transitions compared to the number of transitions required by the frequency scaling policies. Frequency scaling policies (*SWQI* and *SWQO*) increase throughput by increasing the frequencies of CPUs in the cluster before powering additional nodes on. Additional machines are only turned on when all active machines have already been scaled to a higher frequency. As a result, less energy is consumed by turning nodes on and off less frequently.

Figure 6 shows the energy consumption for four policies on all four workloads. Across all workloads, *SWQI* and *SWQO* provide less energy consumption, except in the case of a very small number of nodes, where *SLF* performs better, because nodes run continuously. In addition, we can see that the practical *SWQO* closely approximates the ideal

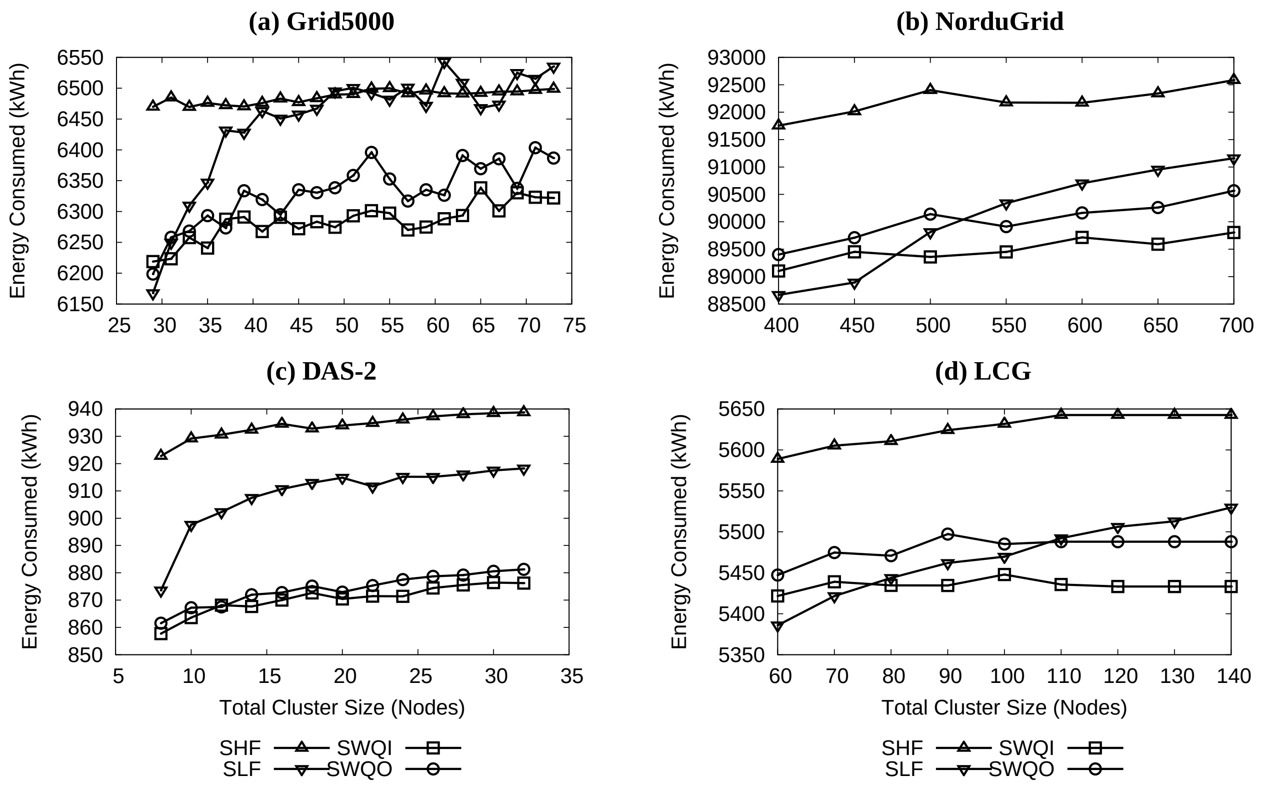

Figure 6. Energy Consumed by Grid Workloads on Clusters

The energy consumed for each workload on clusters of varying sizes using four policies. (Lower numbers are better.) As expected, scaled high frequency (SHF) consumes more energy than scaled low frequency (SLF). However, the adaptive policies (SWQI and SWQO) consume even less than SLF, because they minimize the number of idle cores and use high frequency selectively to avoid the cost of powering on additional machines.

SWQI by choosing to submit the next job in the queue to the available machine with the longest running job.

An individual user is most interested in how quickly their own jobs will be processed by the cluster. From this perspective, turnaround time is the primary metric for determining performance and is defined as both the time spent idling in the queue and the total run time of individual jobs. From the simulation results in Figure 7, SHF provides the best overall turnaround time.

This is to be expected considering that machines are always running at the maximum frequency supported by the machine. Additional machines are instantly powered on when jobs begin to accumulate in the queue, providing relatively instant relief for those jobs waiting in the queue. On the other hand, SLF consistently provides the worst turnaround time of all frequency scaling policies. This is primarily due to running all nodes at the minimum frequency supported by the machine.

The remaining policies, those which utilize frequency scaling, provide turnaround times within the band between SHF and SLF. In highly over-burdened cluster configurations, turnaround time approaches that of the SHF policy. This occurs when the cluster cannot effectively process the bursts of jobs being submitted. Jobs are consistently waiting in the queue, raising the frequencies of machines in the cluster according to the specific frequency scaling policy in use. The run times of jobs decrease due to the increase in frequency of machines in the cluster, resulting in decreased average total turnaround time.

When the cluster is under-burdened at cluster configurations containing a larger number of nodes, a higher number of cores are available to process the bursts of job submissions. Fewer jobs spend time waiting in the queue, and as a result, a fewer number of machines increase in frequency. Therefore, the run time and total turnaround time of these jobs approach levels similar to that of the LF policy.

Baseline policies will provide comparable performance. However, the size of the cluster never changes, because all machines are always powered on. As a result, jobs submitted to clusters using a baseline policy never need to spend time in the queue waiting for additional machines to be powered on. In addition, jobs will have a longer average

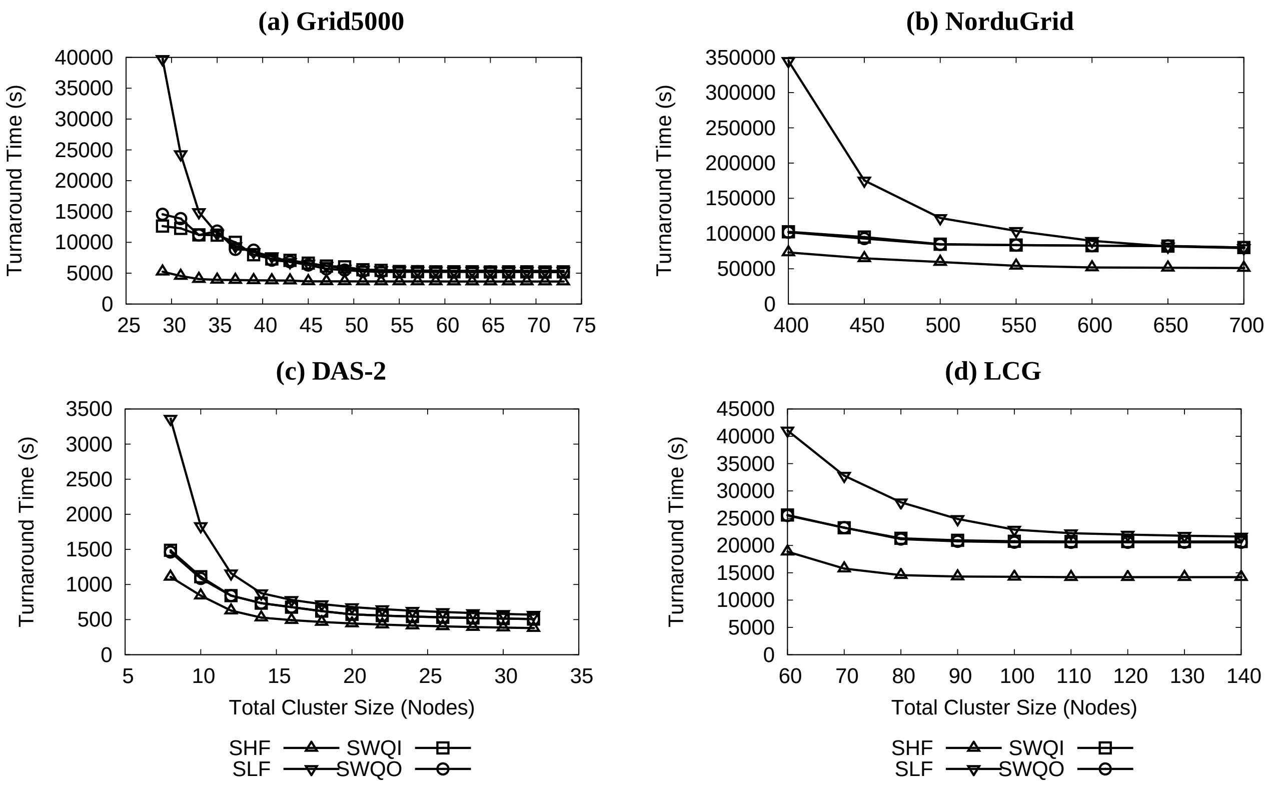

Figure 7. Turnaround Time of Grid Workloads on Clusters
The total time to run an entire workload to completion, for clusters of varying sizes using four policies. (Lower numbers are better.) As expected, scaled high frequency (SHF) provides much better performance than scaled low frequency (SLF). The adaptive policies (SWQI and SWQO) provide intermediate performance at improved energy efficiency, as shown in Figure 6.

run time, because all nodes in the cluster are running at a low frequency while the cluster is under capacity. Any variation in the performance between baseline and scaled cluster management algorithms is insignificant compared to the energy gain acheived by the scaled policies. Therefore, the scaled cluster management policies provide a much greater energy efficiency than the baseline policies when considering energy-performance tradeoffs.

If the jobs contained within a grid workload archive vary greatly in work required and distibution, we may see a saw tooth effect on the trend lines in the simulation results. Under these circumstances, jobs which require a significant amount of work are submitted along with jobs requiring a relatively insignificant amount of work. When the jobs requiring an insignificant amount of work are completed, other jobs waiting in the queue are submitted to the idle cores. Once the cluster processes a large portion of a specific burst of job submissions, the jobs which required a significant amount of work may still be running. The amount of overhead energy required to complete these jobs depends upon the way in which these jobs are submitted to individual nodes. Increasing the number of available nodes in the cluster may result in a better assignment of jobs, where large jobs are pseudo-randomly co-located on the same machine, thus reducing overhead energy.

Discussion of Assumptions. In order to produce these results, we assumed that jobs were CPU intensive. We then correlated CPU cycles with the amount of work required by each job. In actuality, other factors such as latencies related to memory, disk, and network accesses will affect performance. As a result, performance does not scale perfectly with frequency as we assumed in this study.

Considering only a single job on a single machine, the latencies (in terms of real time) incurred while processing the job are not likely to significantly change if the CPU frequency changes. Therefore, only the CPU-bound component of a job is effected when the CPU frequency is scaled resulting in a less significant performance difference between jobs on high and low frequency machines.

In addition, contention for resources also affects the performance of multiple jobs running on the same machine. However, our policies dictate that a machine is only at a high frequency when every core of the cluster is in use. Therefore, a machine at a high frequency will see a greater contention for resources, and as a result, the machine will not be as efficienct as a low frequency machine. Since we classified all machines running in the cluster as most efficient at lower frequencies and maximum capacity, based on the power curve of a single machine in Figure 2, inclusion of resource contention will not significantly affect the results.

Lastly, it should be noted that Section 3 described efficiency as Cycles per Joule. In reality, wasted cycles should also be considered when calculating machine efficiency. However, we did not include latencies and resource contention in this metric. If we had, low frequency machines at maximum capacity would again be more efficient than high frequency machines at maximum capacity since fewer CPU cycles are wasted in the process, thus reinforcing the key characteristic of machines in the cluster.

While adding a bit of realism to our model may affect the precise numbers, it is unlikely to change the character of the results.

6 Conclusions

In this paper, we have shown the effect of three techniques on the power consumption and performance of large grid workloads. Automated node scaling powers off idle nodes, at the cost of some latency to power them back on. Intelligent job placement seeks to minimize the number of machines in use by co-locating long running jobs. Dynamic frequency scaling is employed to increase system throughput instead of activating nodes that will be underutilized. Together, these techniques can be used to reduce the energy consumption of grid workloads on large clusters.

7 Acknowledgements

This work was partially supported by a Federal NWICG grant. Computational resources were made available by the Center for Research Computing and the Department of Computer Science and Engineering at the University of Notre Dame. We thank the Grid5000, DAS-2, NorduGrid, and LCG teams for the availability of their archives.

References

[1] W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, and F. Zini. Optorsim: A grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications*, 17:403–416, 2003.

[2] R. Bianchini and R. Rajamony. Power and energy managment for server systems. *Computer*, 37, 2004.

[3] R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20, 2006.

[4] R. Buyya and M. M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *CoRR*, cs.DC/0203019, 2002.

[5] H. Casanova. Simgrid: a toolkit for the simulation of application scheduling. In *Cluster Computing and the Grid*, 2001.

[6] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116, New York, NY, USA, 2001. ACM.

[7] K. Choi, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *Int. Symp on Low Power Electronics and Design*, 2004.

[8] C. Dumitrescu and I. Foster. Gangsim:a simulator for grid scheduling studies. In *Cluster Computing and the Grid*, volume 2, May 2005.

[9] P. Eerola, B. Kónya, O. Smirnova, T. Ekelöf, M. Ellert, J. R. Hansen, J. L. Nielsen, A. Wäänänen, A. Konstantinov, J. Herrala, M. Tuisku, T. Myklebust, F. Ould-Saada, and B. Vinter. The nordugrid production grid infrastructure, status and plans. In *GRID '03: Proceedings of the 4th International Workshop on Grid Computing*, page 158, Washington, DC, USA, 2003. IEEE Computer Society.

[10] T. Heath, B. Diniz, E. V. Carrera, W. M. Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 186–195, New York, NY, USA, 2005. ACM.

[11] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *ISLPED*, August 2007.

[12] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema. The Grid Workload Archive. *Future Generation Computing Systems*, 24, July 2008.

[13] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power managment policies: Maximizing performance for a given power budget. In *MICRO - International Symposium on Microarchitecture*, 2006.

[14] M. Lamanna. The lhc computing grid project at cern. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 534(1-2):1 – 6, 2004. Proceedings of the IXth International Workshop on Advanced Computing and Analysis Techniques in Physics Research.

[15] H. Li, D. Groep, and L. Walters. Workload characteristics of a multi-cluster supercomputer. *In Job Scheduling Strategies for Parallel Processing, Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, (ed.), Springer-Verlag, Lect. Notes Comput. Sci*, 3277, 2004.

[16] A. Mallik, J. Cosgrove, R. P. Dick, G. Memik, and P. Dinda. Picsel: Measuring user-perceived performance to control dynamic frequency scaling. In *ASPLOS*, 2008.

[17] A. Mallik, B. Lin, G. Memik, P. Dinda, and R. Dick. User-driven frequency scaling. *IEEE Computer Architecture Letters*, 5, 2006.

[18] J. Moore, J. Chase, and P. Ranganathan. Making scheduling cool: Temperature-aware workload placement in data centers. In *USENIX*, 2005.

[19] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. *Compilers and Operating Systems for Low Power*, 5, 2003.

[20] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *High Performance Distributed Computing*, 1999.