GreenMon: Peer-to-Peer Cluster Management

Mayank Agrawal, Owen Kephart, Oscar Chen
Swarthmore College: CPSC 087

Spring 2016

1 Abstract

We present GreenMon, a peer-to-peer cluster management system designed to reduce power consumption. GreenMon dynamically powers nodes on and off in response to changing system usage. While this Vary-On / Vary-Off functionality has been implemented on clusters before, GreenMon is the first to take advantage of a peer-to-peer model. Benefits of a peer-to-peer model compared to a cluster with a master node include better scalability and fault tolerance. We use a pre-existing peer-to-peer resource monitoring system, PeerMon, in order to efficiently transmit system usage data throughout the network. In conjunction with this data, GreenMon's protocols utilize randomization in order to bypass any additional communication between nodes. While ideally designed for remote clusters, it is general enough to be used on any LAN cluster. Preliminary simulations show that GreenMon can reduce power consumption by eighty percent on Swarthmore College's lab cluster during a typical weekday afternoon. Further work includes removing the homogeneity assumption as well as having a more sophisticated way of determining the number of users on each node. The start we have made is already extremely promising, and we believe that further developing GreenMon can lead it become an effective way to reduce unnecessary power consumption on clusters.

2 Introduction

Over the past two decades, the rise of high-performance computing has driven the proliferation of large server farms and data centers. Crucial to the operation of such large computing resources are considerations of energy efficiency in the face of inconsistent demand, when clusters are often not used to maximum capacity. In such situations, many clusters computing centers will have many idle nodes that are performing almost no computation, and yet consume a large amount of power due to idle power usage.

An intuitive approach to reduce power consumption in underutilized clusters is VOVO (Vary-On / Vary-Off). In this model, nodes in a cluster are turned off when network usage is low and turned back on when the usage is high. This way, the number of idle nodes is kept at a minimum and the total power consumption of the cluster is minimized, while still allowing it to handle large workloads.

Previous approaches have used a hierarchical cluster, where a master node is responsible for implementing these changes. However, this master node setup has issues with both scalability and fault tolerance. As the size of the cluster increases, the master node acts as a bottleneck for the response time of the network – if too many nodes are in the system, the master node may not be able to respond to all requests efficiently. Likewise, if the master node fails, the cluster cannot operate until the node is revived or a new node is chosen to be the master node. A peer-to-peer approach, where each node is equal to the others, is able to bypass both of these issues.

GreenMon extends an existing peer-to-peer LAN network monitoring system, PeerMon, to incorporate this VOVO model. PeerMon is a low-overhead system that propagates each node's resource (CPU and memory)

utilization to other peers in the network. Thus, PeerMon enables each node to be informed of other nodes' usage as well as that of the total network. Our aim is that remote clusters running GreenMon will have a scalable and fault tolerant way to reduce power consumption by dynamically changing the size of the network in response to usage.

We will begin with a more in-depth discussion of both VOVO and PeerMon. Afterwards, we outline our solution, highlighting some of the challenges we had to overcome by working with a peer-to-peer model. We then display preliminary results that were taken on a Swarthmore College CS lab cluster. We conclude with a discussion of the results as well as suggestions on how to proceed. GreenMon has the potential to drastically reduce the power consumption of clusters, especially those with low utilization. Our working prototype is promising, and continuing to make progress will allow us to maximize its effect. We believe that GreenMon is a valuable, low-overhead cluster management system that should be utilized to provide for a greener computing experience.

2.1 Related Work

Vary-on/vary-off (VOVO) schemes have been studied extensively in various cluster-management contexts, aiming to develop strategies to dynamically resize the active cluster network in response to varying utilization. Two pioneering VOVO approaches were proposed by Pinheiro et al. and Chase et al. in 2001, both aiming to balance the load of a cluster network over the optimal number of cluster nodes.

Pinheiro et al. [**?**] emphasized the significance of idle power usage in data center power consumption, noting that the small difference in power usage between a maximally-utilized node and an idle node creates a large power penalty for idle nodes in the system. In varying nodes on and off, there exists a trade-off between power usage and system performance; with more nodes, tasks can be more evenly distributed and achieve greater performance, but power usage also increases. Their solution used levels of acceptable (performance) degradation to calculate how many nodes could be varied on or off according to changed utilization levels.

Chase et al. [**?**] also emphasized the power-performance tradeoff, and introduce a threshold-based algorithm to manage the number of active nodes in a cluster system. If the average utilization of active nodes drops below a certain threshold, the system will turn off nodes to save power. Likewise if the average utilization surpasses some threshold, the system will turn on nodes to handle the increased system load.

More recently, Meisner et al. [**?**] proposed an instantaneously-dynamic provisioning system that transitions rapidly between high-performance active states and custom low-power idle states, and Gandhi et al. [**?**] use queueing-theoretic results to conduct optimality analysis of vary-on vary-off strategies.

PeerMon as proposed by Newhall et al. [**?**], provides the foundation for our work in implementing VOVO cluster management strategies in a peer-to-peer context. PeerMon's basic structure provides each node with a global data abstraction that stores state of the entire network. This data abstraction is maintained and updated independently by each node, and is passed around the network via each node's set of peers. Each node's view of the system is not guaranteed to be either completely up-to-date or accurate, but updated information will eventually propagate through the network as nodes select new peers to send to. Since our paper is focused primarily on GreenMon, an extension of PeerMon, we will not cover extensive implementation details here - for further information on the PeerMon architecture and its extensions, please consult the original paper.

3 Solution

GreenMon is a system designed to vary the number of powered on nodes based on the current level of cluster usage – if a cluster has a very light workload, it is wasteful to keep all of the nodes powered on

when they are doing almost no work. GreenMon allows a cluster to dynamically adapt to changing workloads, maintaining an optimal number of powered on nodes to balance energy usage and cluster performance.

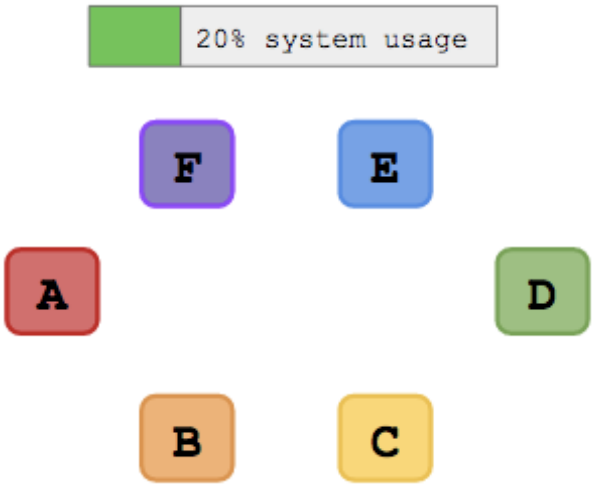

Figure 1: Default cluster under low utilization: all nodes powered on, high power usage. Most nodes are doing almost no work, and are just wasting power.

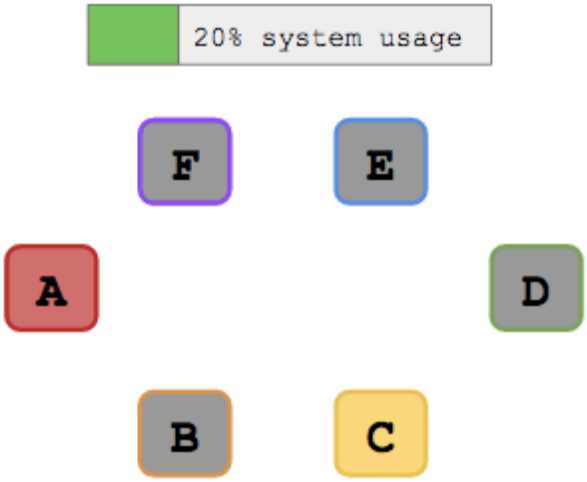

Figure 2: Cluster using GreenMon under low utilization: two nodes powered on, low power usage. Performance impact is negligible, as these nodes are capable of handling the workload.

The core idea of GreenMon is to use and extend the underlying global data abstraction provided by PeerMon to allow all nodes to make an informed decision to change the number of powered on nodes that exist in the network. We use simple randomization protocols along with the data that PeerMon presents in order to achieve approximately correct behavior with minimal additional communication, with 'correct' behavior being a cluster with the desired average load per node.

In order to do this, we added in additional data to the hash table that PeerMon passes around. The most important of these additions was a "status" tag that encodes information about the current operating state of each node. We define three node statuses for our protocol, although more complex protocols could take advantage of a greater number of status values.

- **Alive:** This node currently has users actively using its resources.
- **Idle:** This node has no active users, but is powered on
- **Dead:** This node has been powered off through the GreenMon protocol.

Note that a node that has unexpectedly shut down (e.g. someone physically powering the node off, or the node failing), will not automatically be assigned the 'Dead' status, as it will not have had a chance to communicate with surrounding nodes to inform them that it was turning off. In these cases, we rely on the PeerMon protocol to age that data out of the system, effectively ignoring it after sufficient time has passed.

3.1 The Protocol

In our general protocol, we have each node assess the cluster usage at fairly long intervals – for our purposes we do this evaluation every three minutes, but this is an easily tunable parameter. The nodes are not guaranteed to be synchronized, so a node in the system could be doing this evaluation at any point in time. At these intervals, a node does an assessment of the current state of the network by iterating through its copy of the hash table, getting the number of live nodes and idle nodes as well as the total CPU usage of the cluster. It then uses these values to determine the average usage of machines in the cluster, which will allow it to decide if it wants to alter the number of machines in the network.

If the average CPU usage goes above a given threshold, the node will go through the power on protocol in an attempt to reach optimal average usage. If it is below a given threshold, the node will go through the power off protocol, once again attempting to adjust the number of active nodes to attain the optimal average usage. We chose 50% average CPU utilization as the optimal, with 70% being the upper bound, and 30% being the lower bound. These numbers were fairly arbitrary, and could easily be adjusted if further research indicates a more efficient set of values. Note that having this rather wide range prevents unwanted oscillatory behavior, in which average usage just above or below the optimal value would cause nodes to continually turn on and off as the average load bounces around the optimal. It also somewhat accounts for the inherent inaccuracy of the underlying PeerMon global data abstraction, as we know that the average usage will almost never be perfectly up-to-date on a given node, and as such we should not require a precise average value to be satisfied with the number of nodes currently on. To be clear, if the node determines that the average usage has dipped below the lower bound, the power off protocol will be executed, and if it determines that the average usage has risen above the upper bound, the power on protocol will be executed – otherwise, it will make no attempt at changing the network state.

3.2 Power Off

We first make the guarantee that we will never turn off a node if it has active users with tasks running on it, as this would severely interfere with cluster users, who could potentially have their jobs terminated without warning. That is, we will only turn off idle nodes. We then note that, in order to provide this guarantee, no node can be able to turn off another node directly, as each node can only be sure of its own number of users. This gets back to the idea that the data in the hash table cannot be relied on perfectly. State can change very quickly, and this information will not get always get passed through the network fast enough to be detected before a node makes its evaluation. Thus, our power off protocol only acts at a local level – each node will individually decide if it should turn off at each evaluation.

In order to decide if any given node should turn off, we use randomization to create an extremely simple protocol. The data collected from the hash table gives us a desired number of nodes to turn off, as well as the number of currently idle nodes in the system. Dividing the nodes we wish to turn off by the number of idle nodes in the system gives us a probability that this specific node will turn itself off. For example, if there ten idle nodes in the system, and we want five to turn off, each individual idle node will have a 50% probability to turn itself off. While this sometimes turn off too many or too few nodes, it has an expected value of correctness, and this randomization error will approach zero as the cluster grows. One foreseeable edge case would be a situation in which the number of idle nodes is less than the number of nodes we wish to turn off. In this case, we get a probability for each idle node to turn off that is greater than one. However, this edge case still retains the desired behavior, as all the nodes that are safe to turn off will turn off.

Once a node decides to turn off, it will send out a final copy of its hash table to its neighbors, in which it changes its status to dead, and makes a call to *poweroff*, which will physically turn off the machine. This ensures that other nodes in the system will become aware of the fact that this machine has turned off voluntarily (and as such is a viable target for powering on later). We also add in a special clause to this protocol that prevents a node from turning off if it has the highest IP in the network and the cluster would otherwise desire to turn off all the nodes in the system. This is because you must have at least one node that is actively running the GreenMon client in order for other machines to turn on. If all the machines in the network shut off, someone would have to manually power a node back on, which would be far from optimal.

3.3 Power On

Clearly, a node cannot power itself on, as the GreenMon client cannot run on a powered off computer. In order to remotely power on dead nodes, we make use of a functionality called Wake-On-LAN. True to its name, this allows for nodes to send a "magic packet" to powered off machines and have them turn on in response. This means we can have nodes easily power each other on with a simple *etherwake [MAC Address]* call.

Our power on protocol uses randomization in much the same way as the power off protocol. We use the hash table to generate a number of nodes that we wish to turn on, and this time count the number of nodes that are not dead. It then divides the number of nodes that we wish to turn on by the number of non-dead (i.e. alive or idle) nodes to generate a number of nodes that it wishes to turn on. If this number is less than one, we simply treat this number as a probability – if there are ten live nodes in the system and we wish to turn on five, then each node will have a 50% chance to turn on a node (which will be randomly selected from a list of dead nodes). However, if the system has ten nodes and we wish to turn on 25 nodes, each node will want to turn on 2.5 nodes. We handle this case by saying that each node will definitely turn on nodes until this value drops below one, at which point we revert back to the randomization system. In this case, each node would be sure to turn on at least two nodes, and then would have a 50% chance of turning on a third node. This scheme was designed to minimize variance – it ensures that a fairly large proportion of the desired nodes will be turned on correctly, even if the randomization step is particularly unlucky.

4 Results

After debugging and verifying GreenMon's basic functions (updating of, varying nodes on/off according to total network load) on a small 3-node test set, we turned to evaluating GreenMon's effectiveness as a power-aware node provisioning service for cluster networks. However, running GreenMon and evaluating its main functionality in our larger development environment is a non-trivial problem for two reasons: 1) The Swarthmore CS lab cluster must remain highly available both to remote and physical logins for students working on CS projects, preventing us from actually turning off workstations in response to high network loads, and 2) directly measuring the power usage of the full cluster network requires electricity usage monitors on each of the nodes, which we do not have access to. Under these circumstances, we turned to an

alternative evaluation solution: simulation.

Instead of running the full GreenMon daemon, we run GreenMon in a special simulation mode. In sim mode, each node periodically logs current information about the state of the system with critical statistics such as total load in the system and the protocol-defined difference in number of nodes needed to achieve optimal power usage efficiency. When a node is "turned off" in simulation mode, it merely sleeps its sender thread until a special "wake-up" packet (similar to a Wake-on-LAN magic packet) is sent to it by another machine. With each of these events logged and timestamped, each machine produces a comprehensive record of what GreenMon's activity would have been during a live run. We can then parse the output of these logs to obtain evaluable information, such as the amount of time nodes were turned off and the total amount of load present in the cluster network.

Previous literature suggests that idle node power usage is about 70% that of active node power usage – however, current standards of idle power usage have improved considerably. Initial measurements of a single CS lab workstation's idle power usage indicated $\tilde{1}7\%$ idle power usage, but could not be replicated. In lieu of a stable idle power usage estimate, we cite a third-party source [**?**], where similar machine configurations to yield an idle/max power usage ratio of about 30%-40%. We take this to represent the idle power usage of nodes in our cluster network, which we use to calculate to power savings.

To gather data, we ran GreenMon in simulation mode on all workstations in the Swarthmore CS main lab (28 total nodes) over a 78-hour period from Friday morning to Monday afternoon. This test set was most representative of a typical remote-access cluster computing resource, with minimal physical logins due to a lack of scheduled lab/class hours. In analyzing log output, we take load to scale linearly with power usage, and we calculate power savings for each node as follows:

Total power saved = (Total powered-off time)$\times$(Idle power usage)

Percentage power saved = $\frac{\text{total power saved}}{\text{(total power usage) + (total power saved)}}$

The following graph displays aggregated percent power saved over a range of idle power usage percentages:

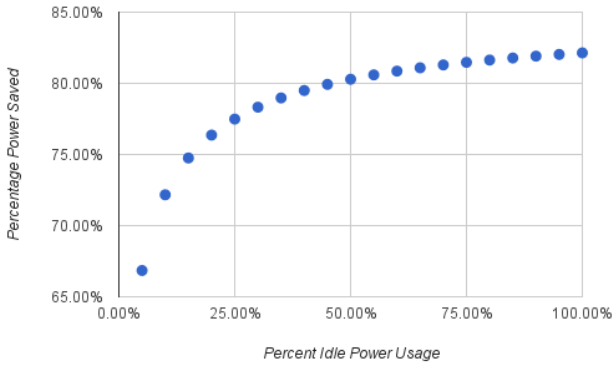

Figure 3: Percent power saved by various idle power usages

Over the 78-hour experimental period in the Swarthmore CS lab cluster, GreenMon node provisioning achieved about 78% - 80% power savings under our assumption of 30% - 40% idle power usage. These results neglect a couple important quantities: 1) the amount of background standby power resulting from leaking electricity (a phenomenon found in all electronic devices) and 2) the power cost of the process of turning nodes on and off, which is significantly higher idle node usage. However, these initial power consumption results in GreenMon peer-to-peer VOVO node provisioning are significant and sufficiently promising enough to warrant discussion and inquiries into future work.

5 Discussion

While the results we have so far are very promising, we must also point out that they do not fully reflect how GreenMon would perform on its target system. Ideally, GreenMon would be running on a large, high usage, remote cluster with thousands of nodes, where we could directly measure the power savings. However, our experiments were run on the Swarthmore CS lab cluster, which is essentially the opposite of our ideal system. We ran GreenMon on a set of 28 nodes, which we were not allowed to physically power off, as any bugs in our code would have the potential to be disastrous. Additionally, even a flawless system would interfere with many users who are not used to having to physically power machines on when in the lab.

This forced us to spend considerable time creating a GreenMon extension that simulated powering off instead, logging whenever it started or stopped. This introduced a host of new problems, as there were no provisions to prevent people from logging in (physically or remotely) to nodes that were categorized as dead by the system. In order to deal with such events (which ended up being quite common), we had the node constantly check for new logins, and simulate powering itself back on whenever one occured. This behavior interferes with our data, as it can cause new nodes to power on, even if there are already too many nodes on in the cluster.

In addition, the overall system usage of the CS lab cluster is much lower than an actual high performance cluster. Users are mostly doing text-editing, with computationally expensive tasks only rarely run. This would lead our GreenMon implementation to assume that it is ok for almost all the machines to be powered off. While this is actually a reasonable result for this specific cluster, it means that our results do not really generalize to all systems, where we expect a much higher percentage of nodes to be on. Thus, our result showing that we can cut power usage by almost eighty percent does not generalize well to other clusters, and should not be taken out of context. However, this result does indicate that the current Swarthmore CS lab cluster does waste a significant amount of power, something that could be prevented through GreenMon.

Although we have not done any direct scalability analysis as of yet, we believe that we have not made any changes that would significantly hinder the scalability of our system in comparison to PeerMon's, which has been shown to scale quite well in terms of network, RAM, and CPU usage, even with systems of several thousand nodes [**?**]. In addition, GreenMon relies on the fault tolerance of PeerMon to remove inactive nodes from the hash table after sufficient time has passed. In this way, we abstract away the scalability and fault tolerance concerns, and just rely on our protocols to perform adequately well in regular use. Lastly, our randomization protocol should perform better as the number of nodes in the network increase. The amount of nodes that are shut down or powered back on will converge to the optimal amount.

6 Future Work

The main purpose of turning on nodes in response to higher system usage is to reduce the total utilization of each node. However, this requires the use of a load balancer, and our experiment on the Swarthmore College cluster did not use one. We believe that GreenMon is the most effective when paired with a load balancer.

Otherwise, turning on nodes is detrimental since they have a high chance of remaining idle. PeerMon was originally presented with smarterSSH, an application that distributes ssh logins throughout the cluster to keep utilization constant among nodes. We envision GreenMon interacting with a load balancer such as smarterSSH or even a more sophisticated load balancer that is able to transport running jobs (and not just users) to different nodes. This way, CPU utilization is relatively equal among nodes and no individual node experiences performance degradation.

Currently, GreenMon is implemented in PeerMon's source code. We would like to eventually expand out GreenMon to be a client interacting with PeerMon and thus keep PeerMon's source code relatively clean. We want PeerMon to be as low-overhead as possible and having a separate client enables us to use PeerMon independently of GreenMon. Ideally, GreenMon will be a smaller part of the PeerMon 'family,' where the base PeerMon interacts with clients such as GreenMon and smarterSSH. There are source code changes that must remain in PeerMon, such as storing MAC addresses and node status. However, the protocols are able to be exported out.

One major assumption we make in our paper is that each node is homogenous. In the future, some data we should factor into our algorithm is the number of cores on each machine and the architecture of each machine. Our algorithm aims to reach a certain number of cores for the given load and then estimates the number of nodes to turn on/off by using the average number of cores per node. What we could do instead is try to reach this optimal core threshold by prioritizing turning on/off certain nodes. For example, if we only need twelve more cores, we should turn on a four-core node and an eight-core node instead of two eight-core nodes. This problem seems to reduce to the subset sum problem, which is NP-Hard. While perhaps too computationally expensive for GreenMon, we still may be able to come up with a better heuristic than the current. Similarly, we assume each node is equally power efficient. We should clearly want to prioritize more efficient machines over less efficient machines, give all else being equal. Thus, future versions of GreenMon should not treat each node as homogenous and have preferences on which ones to turn on and off.

One distinction relevant distinction in GreenMon is that between node failure and node shutdown. The question is whether they should be treated the same or not. Our implementation currently treats them differently. We will only attempt to power back on nodes that have been turned off through the GreenMon protocol. Thus, the only way for a failed node to come back into the system is to restart the GreenMon daemon, which may require human intervention. Despite the fact that sending on wake-on-LAN signals to a failed node should not have any negative effects, we do not attempt this in our current prototype. We do not want failed nodes to be part of the protocol in turning on to meet system usage requirements since we cannot guarantee that they will actually turn back on. It thus may cause us to underestimate the nodes to turn on. An alternative approach worth exploring could be having certain nodes send wake-on-LAN messages to failed nodes periodically to try to reboot them. This method is only effective if failed nodes are able to reconnect through powering back on. Evaluating whether this is the case as well as how often node failure is in the cluster is important to decide whether GreenMon should play a role in trying to recover failed nodes.

The problem of identifying idle users is still an issue that GreenMon needs to continue to work on. A requirement for GreenMon to shut off a node is that the node be idle, i.e. have no active users. On the Swarthmore College lab cluster, this problem is not straightforward. The naive approach is to find out who is logged in using the *users* command. This outputs the users that are physically logged in or ssh'ed in, but it does not include those with active screen sessions. We then experimented with commands that allowed us to find users with screen sessions and other active jobs, but these outputs also included idle users. That is, there would be users from weeks ago who did not properly log out and were thus considered to still be using the machine. One approach we tried was calculating the total number of these "ghost users" by examining how many users were logged in when GreenMon started up. If the number of users ever increased above this benchmark, it meant that the node was active. There are two problems with this method. One is that it assumes no one is logged in when GreenMon started. The second issue is that it does not account for users

that log in after GreenMon starts up and then become "ghost users." While this problem is not necessarily as relevant on remote clusters, it is important to solve if GreenMon is going to be used on a cluster that has a higher number of physical logins. One approach may be to measure how old the users are and any user that was last active over a certain amount of time is considered to be a ghost user. Another approach may be monitoring the most intense processes over a certain time period. If none of these processes are from a user, then the node could be considered idle. We have not attempted either of these two suggestions. We implement the naive solution for our results in simulation mode, but have experimented with some of the earlier discussed approaches. We acknowledge that a more advanced heuristic is necessary before GreenMon begins active use on a large cluster.

While our results look promising, it is important to also test GreenMon on a remote cluster. Remote clusters will have a higher utilization on average. Likewise, the power versus performance tradeoff is much more important to deal with on remote clusters. Remote clusters can have long queues of jobs to complete, thus reducing incentives to shutting down nodes. In addition, tests on a remote cluster will give a better understanding of what certain parameters of the GreenMon protocol should be. For example, what should the optimal amount of time to take between turning on nodes be in order to quickly respond to load spikes throughout the system? We expect GreenMon to still be useful on high-performance, remote clusters, but we would like to get actual evidence before we actively advocate for it.

7 Conclusions / Meta-Discussion

In the creation of GreenMon, we ran into many roadblocks involving permissions. The functionality that we required in order to turn nodes on and off required sudo privileges, and as such, the GreenMon daemon needed to be run as root. While Jeff was extremely helpful throughout this whole process, it was somewhat inconvenient to have to ask for his help every time we wished to restart GreenMon, as only he had the power to push updates to all the machines in our cluster. This meant that making tiny changes and recompiling was not a feasible workflow. The permissions issue even locked out our ability to use common debugging tools such as valgrind or gdb. While debugging and testing with limited access was certainly the hardest part of our project, we found that adding more information to PeerMon's hashtable was one of the easiest things to accomplish. PeerMon was structured well enough that it was very simple to just copy the pattern laid out by other data fields, and alter them slightly to fit our needs. We believe this speaks to PeerMon's promise as an easily extendable piece of software - it would be trivial to add in even more fields, such as GPU usage or number of processes, if those were found to be relevant factors.

In general, we stayed pretty faithful to the core principles our original project proposal. However, our original proposal did not conatin a strong idea of how to create the protocols - in fact we did not actually finish creating these protocols until nearly two weeks in. Our original plan for these protocols was quite vague, and involved some sort of distributed locking. This would have added some additional communication overhead, as well as severely complicated the code-writing process. It also seems that this type of protocol would not be able to provide the guarantees that the current randomization protocol would (i.e. you have an expected value of the optimal number of powered on nodes after a single iteration). As a whole, we are very happy with our randomization protocol design, and believe it could be useful outside of the scope of this class.

8 Acknowledgements

We would like to thank both Tia Newhall and Jeff Knerr for their help and support throughout this project. Tia provided us with the PeerMon implementation, and making changes required for GreenMon were intuitive. Additionally, she was a helpful resource in terms of giving feedback and suggesting further extensions. Jeff gave us special permissions to both simulate and test GreenMon on machines in the CS lab cluster. He

was extremely flexible in his time and responded quickly to any requests we had.