Configuring a MapReduce Framework for Dynamic and Efficient Energy Adaptation

Jessica Hartog, Zacharia Fadika, Elif Dede, Madhusudhan Govindaraju

Department of Computer Science, State University of New York (SUNY) at Binghamton

jhartog1@binghamton.edu, zfadika@cs.binghamton.edu, edede1@binghamton.edu, mgovinda@cs.binghamton.edu

***Abstract*—MapReduce has become a popular framework for Big Data applications. While MapReduce has received much praise for its scalability and efficiency, it has not been thoroughly evaluated for power consumption. Our goal with this paper is to explore the possibility of scheduling in a power-efficient manner without the need for expensive power monitors on every node. We begin by considering that no cluster is truly homogeneous with respect to energy consumption. From there we develop a MapReduce framework that can evaluate the current status of each node and dynamically react to estimated power usage. In so doing, we shift power consumption work toward more energy efficient nodes which are currently consuming less power. Our work shows that given an ideal framework configuration, certain nodes may consume only 62.3% of the dynamic power they consumed when the same framework was configured as it would be in a traditional MapReduce implementation.**

I. Introduction

MapReduce was originally designed for a cluster of commodity machines [1], and popular implementations such as Hadoop [2] see better performance in a homogeneous environment. Many frameworks assume that workers complete their jobs at the same time and with approximately the same cost per node. As not all clusters are homogeneous, Hadoop has a "straggler" mechanism through which it preemptively re-schedule jobs that were assigned to nodes that are not producing results as quickly as other nodes. Zaharia et al. [3] and Xie et al. [4] show this mechanism to be insufficient in heterogeneous clusters, as it struggles to balance the workload when there are several slow nodes in the cluster.

When considering energy efficiency we find that a cluster cannot be completely homogeneous. Machines vary subtly by: amount of thermal paste on the processor, fan speed, fan size, location in proximity to a cooling unit, and heat sync efficiency. These variations mean that some machines may produce higher temperatures than others even when performing the same work on the same data. This requires that some machines receive additional cooling compared to another machine with the same specifications. This translates to a larger energy demand in machines that run at higher CPU temperatures. In the converse, we know that performing additional computations and data loads requires additional work; in turn requiring additional power. A motivating factor in our work is that our homogeneous cluster is heterogeneous with respect to power consumption and in order to combat this we need a MapReduce framework that can dynamically schedule work based upon power consumption of an individual node.

This paper makes several contributions. We quantify the relationship between CPU temperature and energy consumption, and show that CPU temperature is a reliable indicator of current power consumption with respect to a single worker node. Utilizing this, we design and implement a MapReduce framework that dynamically schedules jobs using CPU temperature as a metric to estimate power use. We test various aspects of our framework for their impact on energy consumption and show that through scheduling we can reduce the amount of additional power needed by 37.7% on individual nodes when compared to our framework utilizing methods typical of other MapReduce implementations.

II. Related Work

As the MapReduce community has grown, so has the amount of work dedicated to energy efficient MapReduce implementations.

In GreenHDFS [5] the MapReduce cluster is separated into Hot and Cold zones. Nodes in the Hot zone are frequently accessed because they host popular data and consist of high power, high performance CPUs. Nodes in the Cold zone are infrequently accessed as they host unpopular data and are energy-conserving nodes. GreenHDFS uses the underutilization of nodes in the cluster to increase utilization in the Hot zone and aggressively shutdown components to combat idleness in the Cold zone, thus producing energy savings.

Leverich and Kozyrakis [6] approach conserving power in Hadoop Clusters by utilizing Hadoop's replication strategy to produce a Covering Subset (CS) of the cluster that contains at least one replica of each data-block. This allows nodes not in the CS to be disabled to conserve power. Lang and Patel [7] reinterpret this same problem, but instead of leaving the cluster online at all times with some nodes sleeping, they consider what would happen if the cluster was asleep until a job was queued. Both [6], [7] discover considerable power savings. The drawback of the approaches set forth in [5]–[7] is the coupling of the file system and the MapReduce framework. This does not allow for utilization of forthcoming green distributed file systems and results in overhead that decreases efficiency in terms of both power and turnaround time. In our previous work MARIANE [8] we further discuss the reasoning behind the de-coupling of the filesystem and the framework.

Chen et al. [9] test HDFS and how replication, block size, and file size affect energy efficiency. [9] concludes that where reliable storage systems, separate from HDFS, are deployed alongside Hadoop, replication should be set to 1, as the replication and shuffling mechanisms utilized by HDFS unnecessarily consume power in this case. In our case, we require a reliable storage system to use our framework, thus allowing for power conservation beyond what Hadoop may achieve as it wastes power managing various aspects of HDFS.

Wirtz and Ge [10] analyze the use of Dynamic Voltage and Frequency Scaling (DVFS) in a homogeneous cluster in order to improve energy efficiency. They claim that a power-aware cluster is defined by the number of compute nodes and the number of processing cores per node, together with the frequency of the processor cores. We find that this simplification of the cluster fails to take into account heterogeneity and elasticity. Assuming homogeneity also precludes the possibility of different machines being added to an already existing cluster at a later date, the creation of an ad hoc MapReduce cluster, and reduces the efficiency of a shared cluster. Additionally, allowing multiple jobs to execute on the same machine can cause contention for shared resources and create slow nodes. Related to this, Chen et al. [9] discovered that slow nodes need to either be removed or assigned less work in order to reduce power consumption when the speedup provided by adding the node is not enough to offset the power penalty for adding the node. Our framework addresses both of these problems by splitting the job into m tasks, where m is significantly less than the number of worker nodes. Each worker node is then assigned one task. Faster nodes request additional tasks once their initial task is completed. If new machines are added mid-execution, they can request some of the remaining tasks from the queue.

Energy-Proportional Computing [11] states that we should consider induced energy difference since idle power dominates total power consumption, and was used as a metric for determining energy efficiency in [7], [9], [10]. Chen et al. [9] also suggest that in analyzing an energy efficient MapReduce implementation multiple metrics should be used, including, but not limited to: finishing time, energy, and power. We will use turnaround time, and power consumption when reporting the results of our experiments. [10] collected data for Matrix Multiplication, CloudBurst, and Sort to determine the energy efficiency of their framework; [9] also used Sort for this purpose. These three benchmarks serve to show that there are different types of MapReduce workloads, and so any decision regarding scheduling on a framework must work for various types of MapReduce workloads.

III. Preliminary Findings

In order to schedule for energy, we need to find a way to quantify the energy usage of a node in a relatively efficient manner. To that end, we need to find a metric through which there is a strong correlation to power consumption. Ideally this measurement would not result in a need to affix external hardware monitors (such as power meters) to each of the nodes in a cluster, as this is cost in-efficient and not practical for large clusters. We consider CPU temperature as such a metric.

To determine whether or not CPU temperature is a viable metric for our purposes, we needed to test for a reasonable correlation between CPU temperature and energy consumption. On the surface we feel that these two measurements should be correlated since more work requires more power, and work generates heat on the chip. In order to test this assumption we designed several experiments. Our experiments were carried out on a machine with the following configuration: Intel Xeon CPU E5320 @ 1.86GHz with a 8MB L2 Cache running 64-bit Linux 2.6.32.

We ran tests utilizing the Great Internet Mersenene Prime Search program mprime [12]. The aspect of mprime that we relied on for this testing was the torture testing. The torture tests stress the system in three different ways, as per the program documentation.

Test 1: Stresses the FPU with minimal testing of RAM as all data fits in the L2 cache.

Test 2: Stresses the FPU and some RAM, consumes maximum heat and power.

Test 3: A combination of tests 1 and 2, that balances the type of stress between resources, stressing the FPU and lots of RAM.

It was necessary to stress these various components of the system as we are aware that there is heterogeneity amongst MapReduce workloads. Some workloads are I/O intensive [13] while others are CPU intensive [10]. As such, in addition to using these three mprime torture tests, we developed two different methods of gathering data to help simulate variability within a given workload.

Stress: Iterate through a loop while performing floating point operations. We take 1000 total readings of CPU temperature and system power throughout, with measurements taken back to back. This simulates an intensive workload, especially when run in conjunction with mprime torture testing.

Temp: Iterate through a loop while performing floating operations, and take 100 measurements of both CPU temperature and system power. Checkpoints in the loop indicate when measurements should be taken. This produces the effect of pausing briefly between readings to allow the temperature to drop back down. This simulates running a variable intensity workload.

The results of our tests are shown in Figure 1 and help us to identify a correlation between temperature and power consumed on an individual node. The bar labeled 'test1' shows a Pearson correlation coefficient of 0.711 when considering all data gathered running Test 1 for mprime, regardless of collection method. Similarly, the bars labeled 'test2' and 'test3' show a coefficient of 0.50 and 0.81 on data gathered while running Test 2 and Test 3 for mprime respectively. The bar labeled 'simple' shows a Pearson correlation coefficient of 0.36 when considering all data collected without the stress induced by mprime. The bars labeled 'temp' and 'stress'

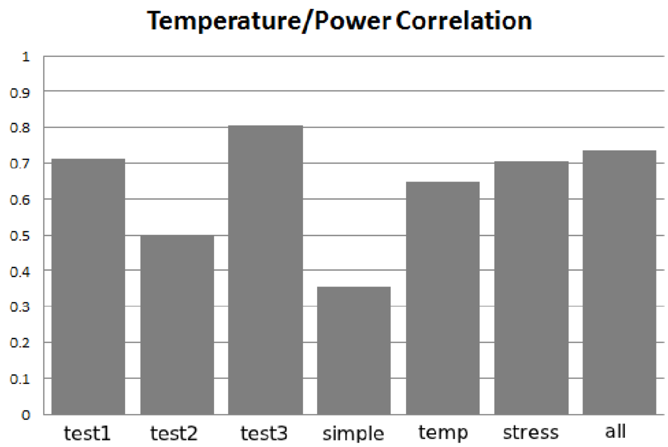

Fig. 1. This graph indicates the Pearson Correlation Coefficient on the Y-axis, with grouped data from individual tests on the X-axis.

show a correlation coefficient of 0.65 and 0.71 on all data generated using the Temp and Stress methods of gathering data respectively. The bar labeled 'all' represents the Pearson correlation coefficient of 0.73 for all data when considered as a whole.

From these tests, we can see that the 'test2' and 'simple' bars show the least correlation between power and CPU temperature. This indicates that as the amount of work done by a map or reduce task increases, so does the accuracy with which the CPU temperature can predict power consumption. Some of our results, namely 'test2' and 'simple', show a weaker correlation than expected. We then derived several reasons that the correlation may be weakened:

(1) The cooling system in place on an individual machine requires power to operate, and by expelling additional power, the temperature of the CPU may drop, thus having the opposite of the effect we first anticipated.

(2) External cooling systems (i.e. air conditioning, and neighboring computer's cooling systems) require no power to operate, and yet drop the CPU temperature, thus having the effect of reducing temperature while power consumption may remain the same.

(3) When a node's work comes to completion, the power consumption is immediately reduced as the CPU is executing many less commands. However, the higher temperature that was generated as a result of the workload takes time to dissipate, thus further weakening the correlation between power and CPU temperature.

The data from our experiments indicates that there is a positive relationship between power consumption and CPU temperature; as the CPU temperature goes up, so does the power consumed. We find that the relationship between power and temperature grows stronger when we stress the node while taking readings. With a correlation coefficient of 0.734, our results have confirmed our hypothesis that CPU temperature and power consumption are related for some classes of applications. To that end, we will use CPU temperature in an effort to schedule jobs on worker nodes. Note that in future work, we intend to weigh various other metrics and their relationship to power consumption in an effort to find a better scheduling mechanism. Using these preliminary findings, we worked to design a MapReduce framework that would allow us to cleverly schedule jobs on worker nodes.

IV. Design

Our design is based upon previous work with MARIANE [8] and MARLA [14], both utilize a shared filesystem to distribute jobs across the network. Works such as GreenHDFS [5] and the contributions of Chen et al. [9] indicate that an energy-related weakness of HDFS is the way in which work is distributed and replicated. While distribution of jobs based upon data and replication of data makes Hadoop the de facto standard for MapReduce implementations, we believe that this strong coupling between implementation and file system has the potential to inhibit energy efficiency. As such, the flexibility of choosing the most energy efficient Distributed Filesystem (DFS) currently available was a must-have for our framework, and in future work we will test various file systems for energy efficiency.

We approach this problem with the realization that even the most homogeneous clusters have some elements of heterogeneity, especially with respect to energy efficiency. Aside from the traditional specifications reported for a node there are many characteristics of a node that affect a node's temperature as well as its energy efficiency. These characteristics include, but are not limited to: proximity to external cooling and heating elements, amount of dust in the case, fan speed and size, amount and distribution of thermal paste on the chip, and efficiency of the heat sync. Because of this realization, another feature we wanted in our framework was the ability to react to such heterogeneity. In order to do this, we borrow aspects from MARLA [14] and allocate more than one job per node. This way, the faster (and/or more energy efficient nodes) can take on additional tasks and leave the slower (and/or less efficient nodes) to cool off and consume less power.

In order to meet these design goals we begin with an NFS distributed filesystem. Following this, we have set up job allocation in the following way: First, split the input into a user-defined number of tasks. In the future, we hope to have completed enough testing to find the optimal number of splits without having to rely on user discretion. Then distribute one job to each node in the cluster. Alongside the previous step, the master node starts a thread that polls each node's temperature to verify it is below a certain threshold. Currently the temperature threshold is also user-defined. It may be necessary in the future to set up individual node thresholds because some chipsets have a higher operating temperature than others. Finally, once a node has completed its first task, if its temperature is below the user-defined boundary temperature, we allocate another task to that node. However, if the temperature exceeds the boundary temperature, this node simply waits until either its temperature is low enough to run another task, or the job completes.

This process repeats as long as there is work to be done. Once all work has been assigned, the fault tolerance module begins work and distributes jobs without regard to temperature of a node. Note that this trade-off is made to preserve fault tolerance; without it a task may get stuck waiting for nodes to cool down and the job may never finish. Since there are user-defined parameters in this implementation, we will first discuss the impact of each of these parameters, beginning with the boundary temperature, then moving on to number of tasks assigned to the cluster.

V. Implementation

We designed our MapReduce framework so as to exploit the implicit heterogeneity of some MapReduce clusters, and in order to accomplish this we relied on two user-defined parameters. The first such parameter is the boundary temperature, which is used to determine the temperature at which a node should no longer be considered for rescheduling. The second such parameter is the number of tasks that are created from a single MapReduce job; with each task corresponding to a subset of the input file. User-defined parameters lend to variability in the performance results of our MapReduce implementation. We first discuss the impact of each of these parameters, beginning with the boundary temperature, then moving on to the number of tasks assigned to the cluster.

VI. Experimental Setup

We collect power data for one node in our cluster using a Watts Up? .Net power meter. We do this as we make local decisions regarding energy consumption and so as few as one node may have any power consumption changes. In an effort to determine the actual realized power savings, we run tests designed so that the single node takes on opposite sides of saving and consuming more energy. As described in [9], [15] the energy level of the Master node is not measured, because the Master node contributes approximately the same amount of energy to the cluster, regardless of the cluster's size. These works also do not report power consumption on behalf of the network switch because when a cluster is not isolated such results could corrupt the data if other machines are utilizing the network.

All nodes in the cluster ran 64-bit Linux 2.6.32. Our Master node had an Intel Xeon CPU 5150 @ 2.66GHz with a 4MB L2 Cache.

Similarly all but one worker node also ran on an Intel Xeon CPU 5150 @ 2.66GHz with a 4MB L2 Cache.

Our exception was the metered worker node, which had an Intel Xeon CPU E5320 @ 1.86GHz with a 8MB L2 Cache.

All nodes in the cluster have the lm-sensors [16] package installed in order to be able to determine the CPU temperature of each node. It is important to note that although 80% of the worker nodes have the same configuration, two of the non-metered workers ran with a higher average CPU temperature than the others. One such node averaged approximately 100°C, and the other such node averaged approximately 115°C. The data is shared between nodes using NFS, hosted on a local server, but the framework makes no specific assumptions and any shared filesystem could be used. Looking forward, this kind of flexibility is necessary as research in the area of green distributed file systems is in progress [17], [18]. As breakthroughs are made, our system will be adaptable and able to realize the changes necessary to find power savings. Each of our experiments used the traditional WordCount application. The average of all iterations of a given experiment is reported.

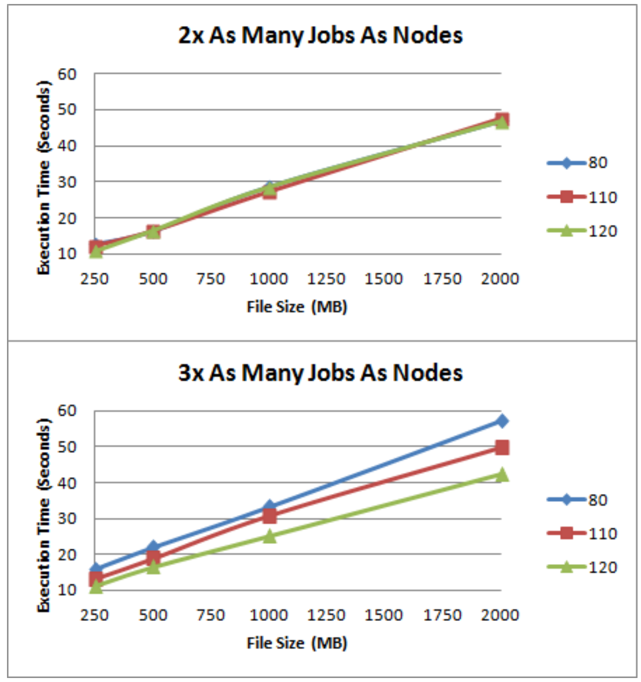

Fig. 2. Execution Time in Seconds is presented on the Y-axis and File Size in Bytes on the X-axis. The graphs are differentiated by the number of tasks with respect to the number of workers. This shows that as the number of tasks increases, the boundary temperature plays a more important role in determining execution time.

VII. Parameter Exploration

In this section we describe the results of the experiments performed on the cluster as described in Section V.

A. Boundary Temperature Sensitivity

Based upon the way our framework is designed, the temperature that is used to decide whether or not a node is able to take on more work is a parameter that can limit performance. In light of this, we perform all tests with three different temperatures as our boundary temperature. As was described in Section V, 20% of the worker nodes had a tendency to run at a higher temperature than the other 80%. As such, the boundary temperatures we selected were 80, 90, 100, 110, 120, and 130°C. However, for the sake of brevity, we will only discuss data from 80, 110, and 120°C as they correspond to 60%, 80% and 100% reschedulable workers and the results for 90 and 100°C, mimic those of 80°C, similarly for 120 and 130°C.

The graphs in Figure 2 show that as the boundary temperature decreases, the execution time increases in nearly all

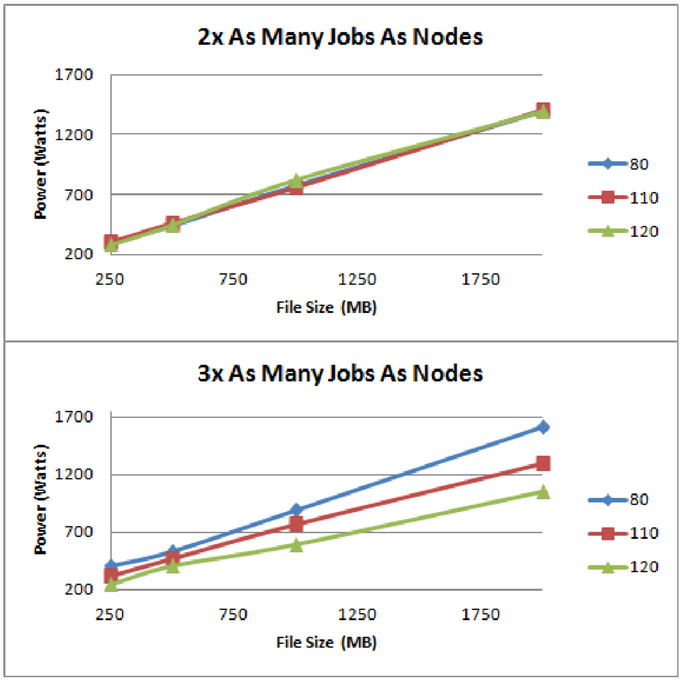

Fig. 3. The total change in watts of energy consumed is on the Y-axis plotted versus the file size in Bytes is on the X-axis. In conjuncion with 2 shows that execution time is a determining factor in total change in power consumption.

cases. When the temperature of a node exceeds the boundary temperature, the node is considered to be temporarily dead. A boundary temperature that is set too close to the idle temperature of the CPUs in the cluster could potentially make it such that very few nodes are eligible for obtaining a second task after completing their first task. This also means that there is potential for a live lock to occur. In order to eliminate this possibility, some knowledge of the cluster to which this framework is deployed can provide a guaranteed means of preventing such conditions. Ideally, if the idle temperature of all nodes was known prior to deployment then the boundary temperature can be set so that there is an ample temperature range between an active node and an inactive node. To further evidence the importance of properly setting user-defined variables, we note the increase in execution time as boundary temperature decreases. We can also see that as we reduce the ratio of tasks to worker nodes the effects of changes to the boundary temperature are reduced. Note in Figure 2 where there are three times as many jobs as there are workers, the improvement in turnaround time increases as the file size increases. In the case where there are only two times as many jobs as there are workers, the improvement in turnaround time is minimal at best, and does not show a marked improvement as file size increases.

The graphs in Figure 3 show the exact same trend as the turnaround time graphs in 2. This means that fine-tuning the boundary temperature to the cluster can save a significant portion of power. Note that these trends are not absolute. In our experiments the node that we tested for power took additional jobs only when the temperature was too high on other nodes,

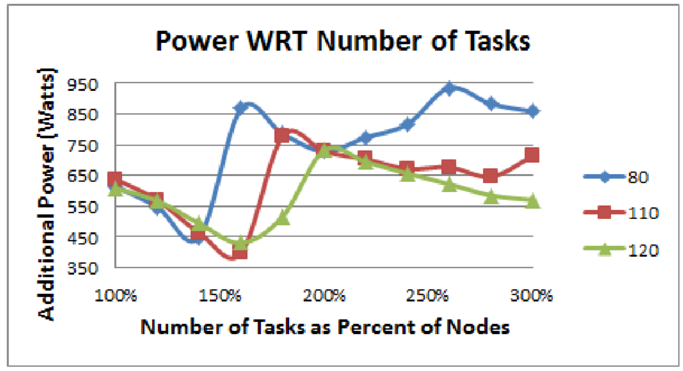

Fig. 4. This graph has the total change in watts of energy consumed on the Y-axis and the number of jobs available as a percent of the number of worker nodes on the X-axis. The results of three different boundar temperatures are displayed.

despite it being a slow node. Consequently, this node was not chosen when there were a smaller number of available jobs and cooler, faster nodes could accept them instead. In order to illustrate this we consider the results presented in Figure 4. Note that this figure indicates that the power consumed varies based upon how many taks are spawned and what percentage of the nodes will be deactivated after their first task.

B. Data Split Sensitivity

From Figure 4, it can be determined that part of our heterogeneous cluster's success at processing MapReduce data quickly is dependent on how well we split our data. To further analyze this we will consider how much the file size depends on how many pieces we can split it into. Since there is some amount of overhead associated with stopping and starting a MapReduce job in any framework, we will consider how the splitting of various size files into multiple pieces impacts our overall throughput. In Figure 5 we see data for execution time based upon number of splits for a 500MB and a 1GB file. While this same test was performed on a file of 250MB and 2GB as well, this data is omitted because the 250MB file has a similar fit to the 500MB file, and the 2GB file has a similar fit to the 1GB file. Instead we will analyze the 500GB and 1GB files, and we will discuss their similarities and differences.

Both graphs in Figure 5 show two intersecting points, the first one is at a task to node ratio of one. This value represents when there are exactly as many tasks as there are nodes, which is the traditional ratio used by MapReduce frameworks. The next point of intersection is at a task/node ratio of two. This intersection was predictable, as it is probable that the delay between the fastest and slowest nodes finishing jobs is less than the time it takes to execute the task, thus resulting in each node getting two tasks. An interesting observation is that when the number of tasks is three times the number of nodes, we do not see an intersection point. One possible explanation for this can be seen when comparing this to the intersection that occurred at the task/node ratio of two. At this point we saw that each node was given two tasks to complete and we determined that this likely happened as the delay between the

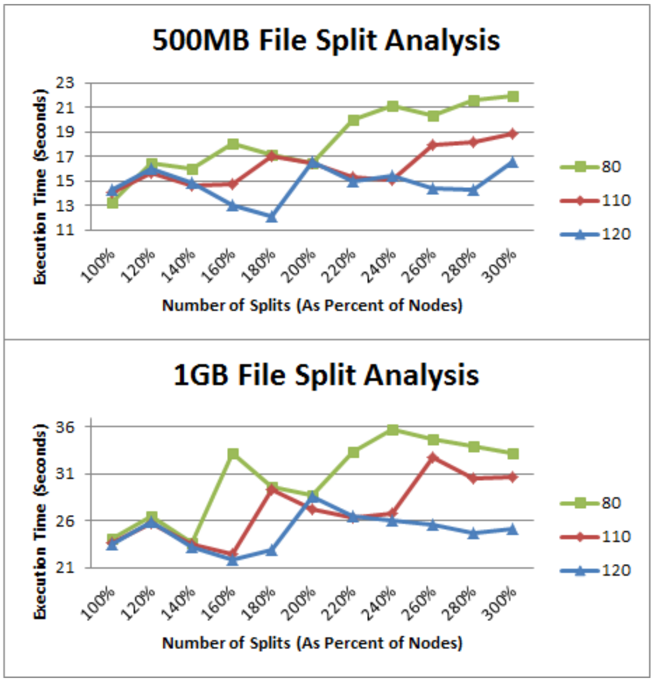

Fig. 5. Here, the X-axis displays the number of jobs as a percentage of the nodes in the cluster while Y-axis of both graphs represent the execution time of the MapReduce job in seconds. The results of three different boundary temperatures are displayed.

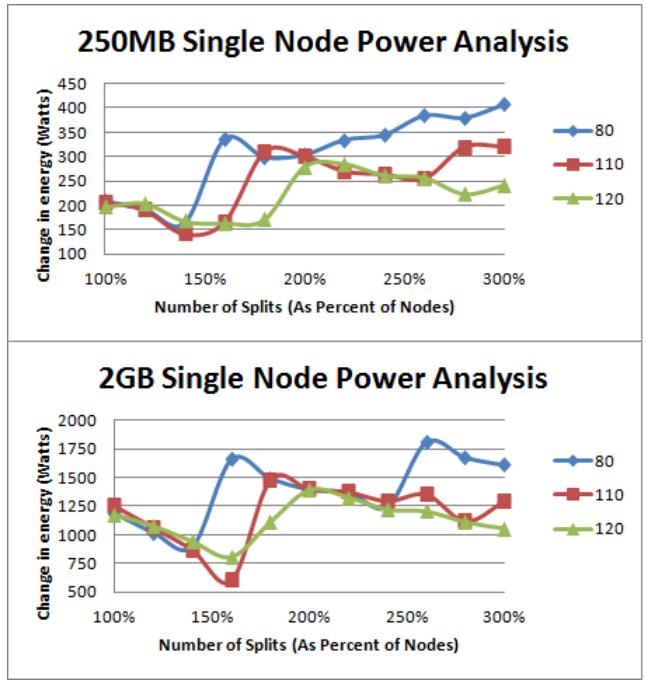

Fig. 6. The total change in watts of energy consumed is presented on the Y-axis and the number of jobs available with respect to the number of worker nodes in the cluster on the X-axis. The data for three different boundary temperatures in our framework are presented.

fastest and slowest nodes completing a job is less than the the time taken to execute one job. However, at the task/node ratio of three, the tasks are shorter because there are more of them, and it is now more likely that the delay between the fastest and slowest nodes completing a job is more than it takes to execute one job. Note here that our cluster is only moderately heterogeneous, and that more heterogeneity would show a vast change in the effectiveness of a given task/node ratio. In the future, we would like to assign a formulaic degree of heterogeneity to a cluster, and determine the number of tasks necessary to effectively balance power and speed.

We can also see from both sets of graphs that the threshold temperature value is a contributing factor to execution time as the number of splits increases. This makes sense as a MapReduce job can complete only as fast as the slowest node completes the job. For this reason a decrease in the boundary temperature results in an increase in the execution time. Note that when the boundary is low enough, several nodes are effectively removed from the cluster and so the remaining nodes become slowed down by having to do a larger percentage of the work. However, our cluster is only moderately heterogeneous and we hope that with a larger, more heterogeneous setup, lower boundary temperatures may not as dramatically shift execution time.

One feature that defines each of these graphs is the relative smoothness of each of the temperature plots. In the 500MB graph, the 110°C plot has the least variation in the slope of its various segments, whereas in the 1GB graph, the 120°C plot has the least variation in the slope of its segments. The sum of the variance in the slopes of all segments in the 500MB file is 194.5, whereas the same value for the 1GB file experiments is 731.9. This means that as the input size grew by a factor of two, the variance grew by 376.3%, nearly a factor of four. With respect to the 500MB file experiments, we see that our results are more consistent (independent of the number of file splits) if we leave some of our thermally excited nodes out of rescheduling. The data from the 1GB file experiments tells us that as file size increases the number of splits becomes increasingly important. The results of subsection VII-A together with subsection VII-B indicate that as the number of tasks increases and the data size grows the framework becomes less dependent on the boundary temperature, indicating that the framework is dynamically adapting to the cluster configuration.

C. Energy Savings

As was discussed in Section V for our experiments we collected data on a single worker node as we made local decisions regarding job scheduling and hope that such decisions will translate to global power savings. While our present experiments do not consider the power consumption of the entire cluster, we have plans for future experiments that consider the power consumption problem at various granularities including individual nodes, a proper subset of worker nodes, all worker nodes and the entire cluster. At present, these experiments serve to act as a proof of concept that our decisions regarding temperature adjust the temperature and power consumption of individual worker nodes, with cluster power savings being quantified in future work.

Consider the results of our power analysis on a single node as presented in Figure 6. We present our data for only the most extreme cases within the bounds of our experiments with file sizes of 250MB and 2GB, and only the additional power (total power - idle power) is presented for reasons discussed in [11]. The trends we see in the presented data apply to the 500MB and 1GB file sizes as well. As with several other trends, we see that both the boundary temperature and the number of splits play an important role in the realized energy savings. We can see from this node, that as it is not used for rescheduling when the number of jobs is low, power is saved over the traditional one task per node approach presented in MARIANE [8]. We see in the 2GB file case that when the boundary temperature is 80°C there are two peaks of power consumption, and these peaks correspond to when the node is rescheduled once and twice. When the node is not rescheduled and the tasks become shorter (the number of splits increases) our power consumption demands decrease. We see similar trends in the other boundary temperatures as well, where the data reaches two local minimums and two local maximums, with the minimums located just before an extra task is scheduled, and the maximums occurring just after. Note that as the number of tasks increases, the disparity between the minimum and maximum is decreased. Another noteworthy trend is that compared to the data using the number of splits and the execution time, more data points have similar values amongst the various temperature boundaries when change in power is considered, especially as the file size increases. In the 250MB example, there are 3 cases where the range of the values graphed at each temperature falls less than 10% of the average of those values; this is true of both execution time and power. In the 2GB example, there are 4 cases where the range of the values graphed at each temperature falls less than 10% of the average of those values with respect to execution time; where there are 6 such cases with respect to Watts, an 18.18% increase in similar values. This trend confirms that boundary temperature does affect both the job's execution time, and the single node's power consumption. We also see that the single node's consumption is only increased when the boundary temperature forces the node to take on additional tasks. Our future work will more completely explore this relationship. Our results indicate that there is some balance achievable between power savings and execution time by adjusting the boundary temperature, namely that even though execution time varies with boundary temperature, we see that the power consumption is 18.18% more consistent.

VIII. Comparison With MARLA

As performance is necessary for any successful MapReduce framework, we will take this section to discuss the effects that scheduling for energy awareness has on performance. It is expected that energy savings will result in performance loss. However, as was pointed out in [7] the formula for power consumption is $Power = Energy * Time$. Due to this fact, the longer a machine is active, the more power it consumes. This means that even while scheduling for energy, performance still remains a priority. In this section we will discuss the performance impacts of setting various elements of our framework, and compare this framework to another framework MARLA, which is also based upon MARIANE. The primary difference between our framework and MARLA is that we have scheduled this framework for energy awareness, whereas MARLA schedules for performance in heterogeneous clusters alone. Comparing our framework to MARLA is sufficient as MARLA has been compared to Hadoop and Mariane in our previous work [14]; showing improved performance in heterogeneous clusters.

Fig. 7. Here the number of splits as a percentage of nodes in the cluster is on the X-axis, with execution time in seconds on the Y-axis. The results for MARLA and our framework with a boundary of 110°C are displayed.

Recall from Section III that a framework that works well in a heterogeneous environment will be better suited to energy adaptive scheduling, as no cluster is truly homogeneous. Our results in Figure 7 indicate that our only performance loss over MARLA occurs when the number of splits and the runtime length of each split precludes some nodes from being rescheduled due to their temperature variations. For this reason, we can see why it takes our framework longer than it takes MARLA to complete jobs in some instances. Recall from the previous section that this same scenario also changes the power consumption of an individual node. Note that if our framework is properly tuned to have the appropriate boundary temperature for the given cluster, we can realize turnaround times on par with those discussed in [14]. Since we know that our execution times are similar to those of MARLA, we can see that power savings on the cluster level may be achieved.

We saw that the results of Figure 7 indicate that the turnaround times of our framework are similar in many instances to those of MARLA. Thus, if we realize power savings on a single node between our framework and MARLA, we should be able to realize power savings throughout the cluster

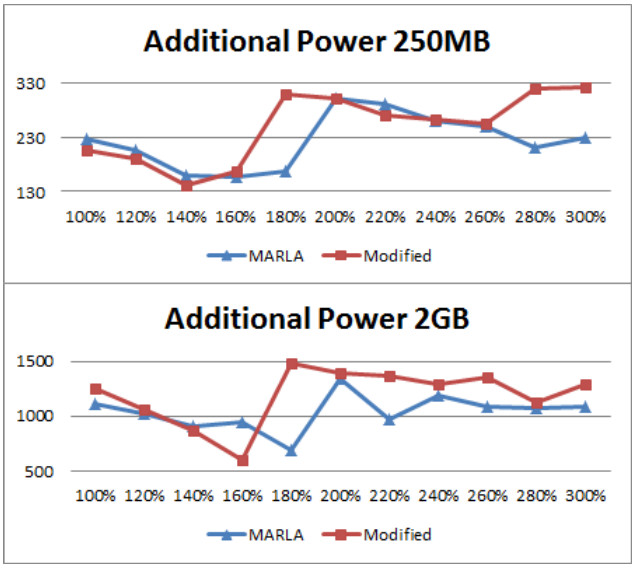

Fig. 8. Here, the number of splits as a percentage of nodes in the cluster is on the X-axis, and the change in energy consumption in watts is on the Y-axis. The results for MARLA and our framework with a boundary of 110°C are displayed.

given the right configuration of our parameters. We present our results in Figure 8. When the file size is small, e.g. in the case of the 250MB file size, our power consumption closely follows the trends and values of the unmodified MARLA framework. Note that the changes in power consumption occur in our framework occur before they do in MARLA as we eliminate one node from the cluster as our boundary temperature was 110°C. In the best configuration, we see that our node consumes only 88.85% of the additional power that it consumed while running on MARLA. Note also that when the file size is large we see a much smoother trend of power consumption in our framework than we do in MARLA. While our framework continues to attain its minimum and maximum power consumption with a smaller number of tasks than MARLA, we see that in the best configuration our node consumes only 63.96% of the additional power consumed while running MARLA. So we can see 11.15% and 36.04% power savings in non-idle power is realized for the 250MB and 2GB files respectively. As a result we can say that our framework provides a method for dynamically scheduling MapReduce applications for energy.

IX. Conclusions and Future Work

In our work we designed and implemented a MapReduce framework whose scheduling is dynamic and energy aware. Our paper offers the following contributions:

- Established a positive correlation between CPU temperature and power consumption.
- Designed and implemented a MapReduce framework that is able to utilize the correlation between power and CPU temperature for scheduling.
- Tested various user-defined characteristics of our framework in an effort to determine the effect each of them has on the success of our framework with respect to both turnaround time and power consumption of an individual node.
- Shown potential for a MapReduce framework that can schedule in an energy-aware manner without having to rely on expensive power hardware attached to each node.

In our future work we plan to:

- Test this framework on larger clusters, both heterogeneous and homogeneous, for performance as well as power consumption.
- Study the optimal way to split input given a cluster's heterogeneity, such that we provide the most power savings.
- Test other scheduling metrics independent from and along side this approach.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] Apache Hadoop. [Online]. Available: http://hadoop.apache.org

[3] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *OSDI*, 2008, pp. 29–42.

[4] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *IPDPS Workshops*, 2010, pp. 1–9.

[5] R. T. Kaushik and M. Bhandarkar, "Greenhdfs: Towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, Vancouver, BC, Canada, 2010, pp. 1–9.

[6] J. Leverich and C. Kozyrakis, "On the energy (in)efficiency of hadoop clusters," in *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, 2010, pp. 61–65.

[7] W. Lang and J. M. Patel, "Energy management for mapreduce clusters," in *Proceedings of the VLDB Endowment*, vol. 3, no. 1, 2010, pp. 129–139.

[8] Z. Fadika, E. Dede, M. Govindaraju, and L. Ramakrishnan, "Mariane: Mapreduce implementation adapted for hpc environments," *Grid Computing, IEEE/ACM International Workshop on*, vol. 12, 2011.

[9] Y. Chen, L. Keys, and R. H. Katz, "Towards energy efficient mapreduce," Electrical Engineering and Computer Science Department, University of California at Berkeley, Tech. Rep. UCB/EECS-2009-109, 2009.

[10] T. Writz and R. Ge, "Improving mapreduce energy efficiency for computation intensive workloads," in *Green Computing Conference and Workshops (IGCC), 2011 International*, 2011, pp. 1–8.

[11] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[12] PrimeNet Benchmarks (GIMPS). [Online]. Available: http://www.mersenne.org/

[13] M. C. Schatz, "Cloudburst: Highly sensitive read mapping with mapreduce," in *Bioinformatics*, vol. 25, no. 11, 2009, p. 1363 1369.

[14] Z. Fadika, E. Dede, J. Hartog, and M. Govindaraju, "Marla: Mapreduce for heterogeneous clusters," vol. 12, 2012.

[15] Y. Chen, A. S. Ganapathi, A. Fox, R. H. Katz, and D. A. Patterson, "Statistical workloads for energy efficient mapreduce," Electrical Engineering and Computer Science Department, University of California at Berkeley, Tech. Rep. UCB/EECS-2010-6, 2010.

[16] Lm-sensors- Linux Hardware Monitoring. [Online]. Available: http://lm-sensors.org/

[17] T. Kosar and M. Livny, "A framework for reliable and efficient data placement in distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 65, no. 10, pp. 1146 – 1157, 2005.

[18] Z. Zong, M. Briggs, N. O'Connor, and X. Qin, "An energy-efficient framework for large-scale parallel storage systems," in *Parallel and Distributed Processing Symposium*, 2007, pp. 1–7.