# Energy Efficient Data-Intensive Computing With Mapreduce

Thomas S. Wirtz
*Marquette University*

# ENERGY EFFICIENT DATA-INTENSIVE COMPUTING WITH MAPREDUCE

by

Thomas S. Wirtz

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

May 2013

ABSTRACT
ENERGY EFFICIENT DATA-INTENSIVE
COMPUTING WITH MAPREDUCE

Thomas S. Wirtz

Marquette University, 2013

Power and energy consumption are critical constraints in data center design and operation. In data centers, MapReduce data-intensive applications demand significant resources and energy. Recognizing the importance and urgency of optimizing energy usage of MapReduce applications, this work aims to provide instrumental tools to measure and evaluate MapReduce energy efficiency and techniques to conserve energy without impacting performance.

Energy conservation for data-intensive computing requires enabling technology to provide detailed and systemic energy information and to identify in the underlying system hardware and software. To address this need, we present eTune, a fine-grained, scalable energy profiling framework for data-intensive computing on large-scale distributed systems. eTune leverages performance monitoring counters (PMCs) on modern computer components and statistically builds power-performance correlation models. Using learned models, eTune augments direct measurement with a software-based power estimator that runs on compute nodes and reports power at multiple levels including node, core, memory, and disks with high accuracy.

Data-intensive computing differs from traditional high performance computing as most execution time is spent in moving data between storage devices, nodes, and components. Since data movements are potential performance and energy bottlenecks, we propose an analysis framework with methods and metrics for evaluating and characterizing costly built-in MapReduce data movements. The revealed data movement energy characteristics can be exploited in system design and resource allocation to improve data-intensive computing energy efficiency.

Finally, we present an optimization technique that targets inefficient built-in MapReduce data movements to conserve energy without impacting performance. The optimization technique allocates the optimal number of compute nodes to applications and dynamically schedules processor frequency during its execution based on data movement characteristics. Experimental results show significant energy savings, though improvements depend on both workload characteristics and policies of resource and dynamic voltage and frequency scheduling.

As data volume doubles every two years and more data centers are put into production, energy consumption is expected to grow further. We expect these studies provide direction and insight in building more energy efficient data-intensive systems and applications, and the tools and techniques are adopted by other researchers for their energy efficient studies.

# ACKNOWLEDGMENTS

Thomas S. Wirtz

I express my appreciation to my graduate advisor, Dr. Rong Ge. I am grateful for her effort to finds ways to incorporate my interests in the works we have performed. She has also been instrumental in mentoring and encouraging professional scholarship. Her writing skills, with english as a second language, are impressive and inspiring.

Thank you as well to my thesis committee members, Dr. Thomas Kaczmarek and Dr. Praveen Madiraju. Their effort reviewing and providing guidance and feedback were valuable to improve this thesis.

I also thank my wife, Lou, and my children Alex, Sara, Susan, Jane, Lucy, and Max. They patiently sacrificed many nights and weekends as I pursued this work. Their love and support provided time and encouragement for me to complete my work and provided welcome rest and relaxation away from the work.

Finally I would like to thank Dr. William Lobb and Dr. Timothy Creamer from the Dental School who provided many accommodations to me throughout this endeavor. It was comforting to know that my employer was so understanding and supportive of my academic studies.

TABLE OF CONTENTS

# CHAPTER 1 INTRODUCTION

## 1.1 Problem Statement

Today, data centers and servers consume enormous amounts of energy, i.e., 300 billion kWh worldwide, which accounts for 2% of total electricity use [32]. As data volume more than doubles every two years [15] and more data centers are built, energy consumed by data centers is expected to increase [18]. Nevertheless, most of the energy is inefficiently used in data centers. A significant portion of energy is consumed just to keep servers ready or move data around without performing useful computation [3] and the mean CPU utilization is 36.44% [5].

Improving energy efficiency in data centers is urgent. EPA estimated in a report [7] in 2007 that power consumption of server and data centers would double in five years if energy efficiency doesn't improve. This report indicates the "pursuit of energy efficiency opportunities in data centers remains important because of the potential for rapid growth in direct energy use in this sector and the resulting impact on both the power grid and U.S. Industries."

## 1.2 Research Challenges

Improving energy efficiency of data-intensive computing is nontrivial because efficiency is sacrificed by design to meet primary constraints. MapReduce [14] is a popular programming model for data intensive computing in data centers. With MapReduce framework, programmers can focus on application algorithm design without dealing with low-level workload distribution and management. However, these design priorities in portability and simplified programming lead to inefficient use of resources and energy.

Energy conservation for data-intensive computing requires enabling technology to provide detailed and systemic energy information and to identify the energy inefficiencies in the underlying system hardware and software. The need exists for both detailed and systemic power and energy information for data intensive computing systems and applications. Sophisticated

measurement and analysis tools must be created to capture, evaluate, and analyze energy use in data-intensive computing.

Conserving energy consumption without impacting application performance is challenging, especially for both computation- and data- intensive applications. Scientific and engineering problems are traditionally computation-intensive. An increasing trend is that these problems process ever-growing, complex data set. Improving energy efficiency for such applications requires comprehensive understanding of the correlations between performance, power, and energy and delicate balance among them.

**1.3 Research Approach and Contributions**

To address these challenges, we propose to create an energy profiling and analysis framework for date-intensive computing with MapReduce and use this framework to identify efficiency bottlenecks and effective techniques for improvement. This framework combines physical power and energy measurement and profiling, statistical and analytical power modeling, and scheduling algorithms for power and energy management.

Different from most prior energy profiling at the node level or application level, ours obtains the power and energy consumption at multiple levels from system, node to computer components and synchronizes the measurement with application execution. The profiling component also consists of software that are trained with statistical models to accurate estimate the power and energy consumption of systems and components where direct measurement is infeasible.

Provided with the comprehensive detailed performance and energy profiles, the analyzing component of the framework evaluates the performance and energy of typical execution phases and characterizes the resulting energy efficiency. This analyzing component further analyzes the effects of system and workload parameters on energy efficiency and identifies inefficient execution phases and effective methodology for improvement.

This framework also consists of optimizing component that leverages obtained knowledge from profiling and analyzing components and explores judicious resource allocation and scheduling for improvement.

The main contributions of this work include:

- We present eTune, a power and energy profiling instrument that profiles power at multiple system levels including node, core, memory, and disks. eTune is scalable to large scale systems and portable to densely packed system and is not limited by physical constraints. eTune accurately captures the system and component power consumption for data-intensive computing. With the eTune framework, it is possible to analyze the energy profiles of data intensive applications and to evaluate the effects of various hardware and software optimizations.
- We present an analysis framework for identifying and evaluating costly built-in data movements in MapReduce and propose a data movement centric approach to energy efficient MapReduce computing. We demonstrate a means of experimental investigation and reveal the unique and detailed performance and energy features of typical MapReduce data movements.
- We present an optimization technique that allocates optimal number of compute nodes according to applications's degree of parallelism for best energy efficiency. We also explore several DVFS scheduling policies and investigate the resulting energy efficiency for MapReduce framework. We find DVFS is generally effective for energy savings while DVFS policies tailored to application characteristics save most energy.

### 1.4 Organization of Thesis

This thesis covers our three areas of study - a profiling and modeling tool, the evaluation of data movement, and the optimization of CPU intensive applications. Chapter 2 provides

background information, the related work, and our contributions. Chapters 3, 4, and 5 describe each study in detail. Chapter 6 presents discussion and conclusions.

# CHAPTER 2 BACKGROUND AND RELATED WORK

## 2.1 Background

**Energy Efficiency in Data Intensive Computing** Data intensive computing poses several challenges related to energy efficiency. The large scale distributed systems of data intensive computing have complex communication and storage. The hardware is typically comprised of densely packed server blades with multicore processors. Further, when workloads are heterogeneous, the power profiles are different throughout the system. All of these factors, combined, require detailed power information, throughout the system, in order to work towards energy efficiency.

**MapReduce** Google developed MapReduce as a framework to process large data sets [14]. Several factors contribute to its popularity. It can be deployed on a wide range of systems, large and small clusters, without modification to the program. Lower cost commodity computers can be used. It provides an associated distributed file system or can accommodate other file systems. The simplified programming model automatically handles parallelism of code and distribution of data. A freely available, open-source version exists - Hadoop. We focus on this version since it is widely used. In this work, we show that many of these factors lead to its energy inefficiency.

It is helpful to have a fundamental understanding of MapReduce. In the MapReduce framework, data and processing are managed by masters and slaves as shown in Figure 2.1. The master for data, a NameNode, manages data sets stored across a distributed storage system. The data is stored in small blocks, distributed across the storage system (HDFS). Typically each block is replicated 3 times across the storage system to provide fault tolerance and maintain performance when a hardware failure occurs. The work of writing, communicating, and reading the data is performed by the DataNode slaves. The master for processing, JobTracker, accepts requests from client applications for new jobs, and assigns tasks to slave nodes, TaskTrackers. JobTrackers attempt to keep work balanced across the system and also attempt to assign work

based on data locality. TaskTrackers report their status to the JobTracker by regular heartbeat messages.

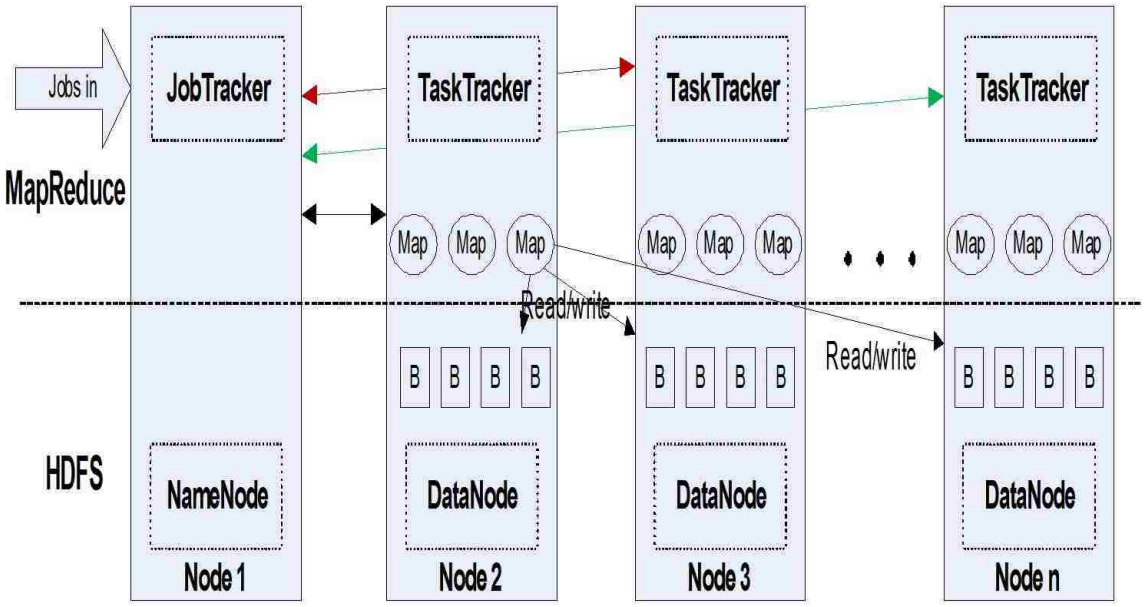



Figure 2.1: A typical deployment of Hadoop framework. The JobTracker and HDFS Na- meNode may reside on the same physical nodes, and the TaskTrackers and DataNodes are distributed on the other nodes.

TaskTrackers perform the MapReduce processes. Data exists as key/value pairs. During a map task, the key/values are input and a map function is processed creating an intermediate key/value pair. A reduce task takes the intermediate key/value pair as input, performs a reduce function, and generates the output of the job. The programmer defines the computation and processing that occurs in the map and reduce tasks. The framework takes care of reading, writing, and transferring the data throughout the system. The framework also takes care of assigning and managing tasks.

The focus of our work is the energy efficiency of the MapReduce framework. So we are interested in how changes in the framework affect energy efficiency. Several parameters alter how MapReduce operates across the entire system and ultimately alter performance. Some of the

common parameters include number of workers, number of map tasks, number of reduce tasks, HDFS replication factor, HDFS blocksize, and workload.

1. **Number of Slaves** - Typically, a slave is a node in a cluster with one or more CPUs. A slave is usually both a DataNode and a TaskTracker; this improves system performance by increasing the likelihood of local access to data by the map and reduce functions. The DataNode and TaskTracker are daemons running on the node and often run throughout the time the MapReduce system is up. It is possible to commission or decommission nodes to increase or decrease the number of nodes in a system, respectively. Too many or too few slaves negatively impact performance.
2. **Number of Map Tasks and Reduce Tasks** - As mentioned above, map tasks and reduce tasks perform the map and reduce functions. The number of map and reduce tasks to use for a particular job depend on the job and data characteristics. Some jobs which have a large workload as input may require more map tasks while other jobs requiring significant CPU processing or extensive output may require more reduce tasks. By default, each node is configured to indicate the maximum number of map tasks and reduce tasks that can run on the node. Configuration settings for each job can include the number of map and reduce tasks needed for the job.
3. **HDFS Replication Factor** - The default HDFS replication factor is 3. Under this scenario, two copies of the block are stored within the same rack, but on different nodes. When possible, a third block is maintained on a nearby rack. Mechanisms exist within the system to copy, move, and balance the blocks. While the system has a default replication for files in the DFS, it is possible for a job to set a different replication factor when creating for a file. The replication factor of 3 requires each block to be written three times which leads to decreased performance and higher power consumption.
4. **HDFS Blocksize** - The default HDFS blocksize is 64Mb. HDFS is capable of maintaining different blocksizes for files in the system. The system wide default size can

be changed in a start-up configuration file. The blocksize can also be set to a different size with a parameter setting on most jobs.

5. **Workload** - Workload is the size of the data for a job. The workload may vary within a job between the different tasks. For example, a job may have large data input, such as input every work from a file, and small data output, such as the count of the words.

## 2.2 Related Work

**Profiling and Modelling Tool** Physical power measurement, the only means for obtaining the first-hand power data for real systems and components, has been either coarse-grained or intrusive. Node and building level power are usually obtained with meters sitting between the computers AC power line and wall outlet [17], [20], [4], [28] or power sensors on chassis and sockets [40]. These devices report readings with intervals at seconds to minutes, lacking the fine granularity in both temporal and physical spaces for energy optimization. For fine-grained component level power measurement, most existing work intrusively inserts precision resistors into the DC power lines and measure the voltage drop on the resistors for power derivation [46], [20], [27], [53]. Major computer components, including processors, memory, and disks, can be isolated with such power measurements. However, intrusive measurement is not practical for blade systems or large scale systems consisting of hundreds of nodes or more.

Instead of physically measuring power, software-based estimation uses performance data to infer power. OS-reported CPU utilization is widely used to estimate system and processor power [16], [42], [25]. However, as [42] points out, CPU utilization does not accurately account for power consumption of less CPU-dominated systems, multicore processors, and aggressive power management; hardware performance events reported by performance monitoring counters (PMC) can better reflect power. Hardware PMCs are used in numerous studies to estimate power of single systems [4], [28], [13] and components including processors [27], [6], memory and cache [29], [51]. Bellosa [4] demonstrates the linear correlation between the system power and

several individual PMC measurable performance events. Papers [27], [28], [13] use multiple performance events to estimate power of uni-core processor on a single machine.

Our work comprehensively studies the energy behavior at multiple levels including system, node, and components such as processor cores, memory, disk, and fans at scale. Most existing work focus on one single system or component [42], [4], [27], [28], [13]. Second, our power models are aimed at data-intensive applications, which have significantly different energy behavior and characteristics than sequential applications on single systems. Third, our software estimates the power consumption of application at runtime, different from work that requires multiple runs of applications [27], [28], [13]. We believe that our software can greatly boost the research in energy efficient data-intensive computing through the fine grained power profiles that were untractable before.

**Evaluation of Data Movement** As the MapReduce programming paradigm is widely adopted in data centers, researchers have attempted to improve computing energy efficiency on MapReduce platforms. Restricting jobs to a portion of nodes while powering down the others [38, 49] has been an effective approach due to the dominating base power. This approach may lead to performance degradation and usually requires knowledge about the jobs including workload size and CPU requirements. Others attempt to reduce power and cooling cost of CPU, the dominating power consumer in computers. For example, several studies [22, 24], and [35] investigate temperature-aware MapReduce scheduling; [47] exploits the low processor utilization during data movements and thus reduces processor performance/power states to save energy. Recognizing the emergence of data-intensive computing and following Barroso's recommendation [3] to understand all components to achieve full potential energy savings, we seek to study the role of data movement in power consumption in MapReduce.

Various methods have been applied to data movement in MapReduce to improve energy efficiency. Chen et al [11] reduce the volume of data in motion by data compression and others [44, 45] increase data movement speed with high speed interconnects. I/O throttling and I/O

coordination [36] are exploited to reduce I/O bottlenecks in MapReduce on multi-core nodes. Multiple zones [30, 34, 8] have shown effectiveness in environments where certain types of jobs and data are executed routinely.

Usually, one of the zones is always up to host hot or immediately needed data and provides timely services while other zones for cold data can transit to deep power saving states or turn off when demands are low. While all these studies improve MapReduce energy efficiency to some extent, they don't provide a fundamental understanding of the energy profiles of data movements that is critical for efficient MapReduce system design. The study in [50] similarly presents component power. However, our work differs and we concentrate on the energy characteristics of data movement for data-intensive computing.

**Optimization of CPU Intensive Applications** While many have employed a variety of approaches to improve performance of MapReduce, a smaller number have focused on energy related to MapReduce. Chen [12] provided a framework for characterizing MapReduce performance and analyzing energy efficiency. He analyzed the common sort job and three other jobs that stress components of MapReduce - HDFS Write, HDFS Read, and Shuffle. By changing configuration parameters, he identified performance and power differences in MapReduce jobs. This work demonstrated that changes to configuration settings based on workload and type of job can lead to improvements in energy efficiency.

There are a few major approaches to improve the energy efficiency of MapReduce.

1. Scheduling - Some have shown that changes to the scheduling of tasks can lead to improvement. Zaharia [52] uses delay scheduling in a 600-node cluster at Facebook. In Hadoop, the JobTracker attempts to assigns tasks to optimize data locality. But if there are no free nodes with access to local data, the task is assigned to another node close to the data. With delay scheduling, the JobTracker has a small delay when a free node with access to local data is not found. Zaharia achieved almost 100% task assignment with local data. Similarly, Ibrahim [26] introduced Maestro which balances map tasks across

nodes and maximizes map tasks with local data. Through this approach, performance improved as much as 34%. A common element of these approaches is the important role of access to local data for energy savings. Our work does not address improvements to scheduling, but we take up an in-depth analysis of data movement to gain a more complete understanding power and energy related to data movements.

2. Power-Down Nodes - Some have powered down nodes to accomplish energy savings. Leverich [34] introduced a covering set, a subset of nodes which contain at least one replica of data. Once the covering set is established, non-covering set nodes can be powered down. This approach achieved 51% energy savings; but performance decreased by as much as 71%. Lang and Patel [33] use an All-In Strategy (AIS) to power down nodes and overcome the performance degradation of the covering set. In AIS, all of the nodes are used to perform the work and the entire system is powered down when work is complete. AIS outperforms the covering set when there is computational complexity in the workload and when the time is low to transition from hibernation to high performance. Kaushik [31] attains energy efficiency through data classification and data placement. Data with long periods of idleness are designated cold zones. A hot zone has data with higher access and processing needs. Nodes in cold zones are put into an inactive state requiring less power; they are brought back to higher energy states to move data between the cold and hot zone as needed. This type of approach addresses the low CPU utilization inherent in MapReduce. Our works take a different approach; we investigate the effect of resource allocation, workload, and application configuration on performance and energy.
3. HDFS Data - Some focus on how data is processed to improve energy efficiency. Xie [48] improves performance efficiency by balancing data across nodes to increase data locality for map tasks. The HDFS blocks are distributed across nodes based on two algorithms. The first algorithm distributes the blocks across the cluster and the second

algorithm balances the data across under-utilized and over-utilized nodes. Efficiency is accomplished through reduction of slow IO transfer of data during MapReduce jobs. Chen [11] reports energy savings up to 60% by applying compression of data in MapReduce jobs. Compression is an effective approach since it shifts work from IO to CPU which is often underutilized in MapReduce jobs. Some of our work focuses on data movement, however we take a different approach. We study how resource allocation and application configurations affect energy efficiency.

4. System - Some focus on system components to address energy conservation. Li [35] monitors CPU temperature through sensors on the processors. The temperature is assessed in a power budget to optimize performance and energy savings. If temperature is too high relative to performance and the power budget, DVFS is used to scale down the frequency of the processor. Our work in optimization is similar in that we also throttle the processor with DVFS. But our work is different in that it provides insight into the role of computation intensive applications.

# CHAPTER 3 POWER AND ENERGY PROFILING AND MODELING

## 3.1 Introduction

In this chapter, we address the need for detailed and systemic energy information by presenting eTune, an energy analysis framework. This software-based power estimator overcomes many of the challenges of power and energy measurement in data-intensive computing. The eTune model is statistically built from performance and power measurements. Using this model and performance counters from the compute processors at runtime, eTune shows high accuracy predicting power consumption.

In this work, we build the profiling model by collecting performance and power information. We apply statistical analysis to the collected data to find a best-fit model. Then we apply the model at runtime, using data collected by performance counters. We validate the tool by comparing the tools predicted power consumption with actual meter measurements. Our results indicate that eTune is an effective and accurate measurement framework.

## 3.2 Methodology - the eTune Framework

eTune, as depicted in Figure 3.1, is comprised of three major elements: Data Acquisition, Statistical Model Inference, and Software Power Estimation. The first element, data acquisition is accomplished through collection of component power measurements and performance events of MapReduce jobs. Second, the data is used to create power-correlation models by statistical inference. Finally, the third element, the software power estimation module uses the learned models to report power at the node level.

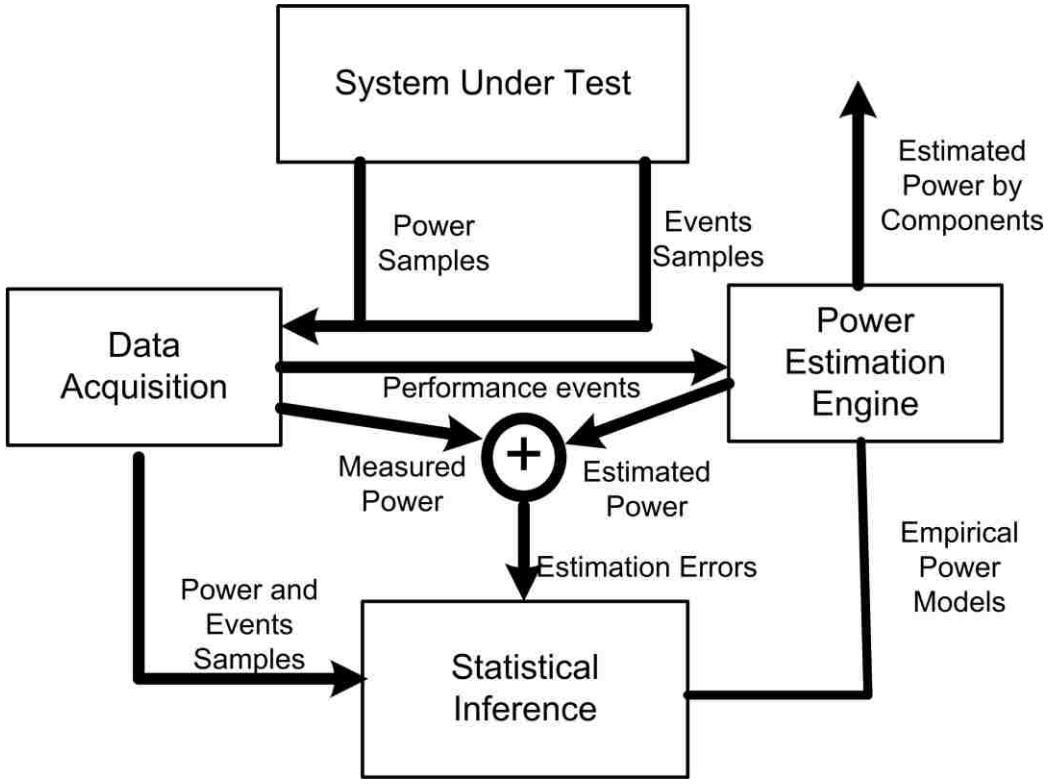



Figure 3.1: The eTune power/energy analysis framework

1. The data acquisition module collects power measurements and performance events from the system under test. Power measurements are acquired with power meters or with power sensors located on the system motherboard. The performance events comprise several streams of performance related data such as performance counters, system activity, network traffic statistics, memory, and IO statistics. The data is aligned by time stamp and saved to a central repository.
2. The statistical inference module, in offline mode, applies learning techniques to the data. Segmented multivariate linear regression is the primary technique used to build the models. The module also accounts for differences between the actual power measured by meters and estimated power.
3. The software power estimation module reports power at runtime based on the statistical models. This provides a software based measurement solution thereby eliminating a need

for a physical measurement device. It can be deployed to many nodes throughout the cluster.

The following sections provide details of these three elements.

**Data Acquisition Model** The Data Acquisition module collects three types of data: power data from meters and sensors; performance data from the operating system and profiling tools; and application-specific events from application logs.

1. Power Data Acquisition: eTune directly measures computer node power and component power with PowerPack [21]. Figure 3.2 shows a typical setup of PowerPack used in our experiments. Currently, we collect power samples from two sources.

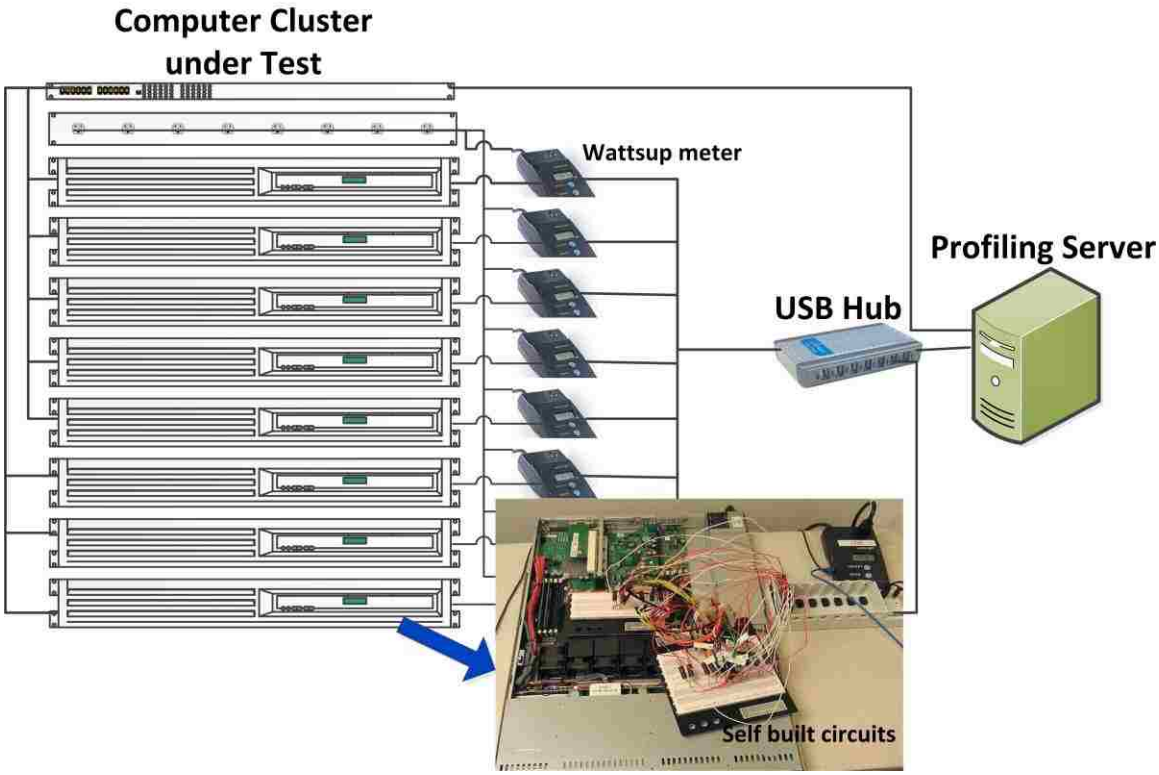



Figure 3.2: The PowerPack Power Measurements

(a) mserver: The mserver program runs on a profiling server, a Linux box that reads AC power samples from multiple WattsUp power meters plugged between power

supplies of computer nodes and power outlets. A USB hub is used to connect 8 WattsUp meters to the profiling server.

(b) NIServer: The NIServer measures component power inside a single compute node. It includes a self-built circuit board that taps a precision resistor into each individual non-ground DC power line, an NI analog input module NI9205 to measure the voltage drop on each resistor, and a LabView program to sample data from NI9205 via an USB port. The NI9205 module supports 16 simultaneous DC measurements. In our experiments, we program the sampling rate at 2K/second. These analog voltage samples are used to derive component power according to the mapping between power lines and computer components. In current configuration, we isolate the power of CPU, main memory, hard disk drives, CPU fans, and motherboard which includes network adapters and onboard video cards.

Both mserver and NIServer accept instructions from users or programs through network sockets to start, stop, and annotate the sampling process. Once set up, both servers run automatically without the need of any manual operation

2. Performance Data Acquisition: eTune collects two types of performance data.

(a) System-wide performance data: We use the nmon [23] performance monitoring tool for linux to collect system performance at 1 second intervals. The items recorded are CPU utilization, active memory usage, disk read/write bandwidth, and network bandwidth.

(b) PMC data: Modern processors are capable of monitoring performance events such as instruction fetching, cache miss, and memory access. This monitoring is non-intrusive to application execution. We employ peprof, a performance event profiling program that periodically reads from performance monitoring counters

(PMCs) on each core. peprof runs as a daemon on each node. We configure peprof with a 250ms sampling period to balance sampling granularity and profiling overhead.

3. Application-specific Events: We analyze the Hadoop Job Tracker and TaskTracker logs to identify the timing of MapReduce execution phases and data movements.

**Statistical Model Inference** The Statistical Inference module applies the power and performance data collected with the data acquisition module. Using statistical approaches, the Statistical Inference module quantifies the relations between the observable performance events and the node and component power. These models and the fine-grained runtime performance profiling create power estimations that can be used in large scale systems.

We maintain two constraints on this model. First, we use a small set of performance variables that can be obtained together at runtime. Second, we minimize runtime overhead by minimizing complexity yet maximizing accuracy.

**Software Power Estimation Module** eTune implements software power estimation into the peprof program described above. The program peprof reads the power models and model parameters from a configuration file, and then applies these models on collected performance events to compute the power consumptions of compute nodes and components including individual cores, memory and hard drive at runtime.

### 3.3 Description of Experimental Environment

We use the following hardware and software throughout our studies. In each study, we indicate configurations that vary from these general descriptions.

1. Cluster - The experiments are conducted on an 8-node power aware cluster with Gigabit Ethernet interconnection. Each node has dual AMD Opteron quad-core 2380 processors running Fedora Core 10 Linux. Each core has a 64KB L1 instruction cache, a 64KB L1 data cache, and a unified 512KB L2 cache. The four cores on the same chip share one

6MB L3 cache. The cluster supports DVFS with 4 frequencies: 0.8GHz, 1.3GHz, 1.8GHz, and 2.5GHz. Each node has one WD1600AYPS Raid Edition 7200rpm SATA hard drive.

2. MapReduce Environment - Hadoop, version 0.20.2 and version 1.0.3, is running on the cluster. One of the nodes runs NameNode and JobTracker, and the other seven nodes serve as the DataNodes and perform map and reduce tasks. Unless explicitly stated, the number of concurrent workers on each node is eight.
3. MapReduce Applications
    - Sort - The sort application distributed with MapReduce is a representative data-intensive application. This sort program simply uses the map/reduce framework to sort the input directory into the output directory. Each map task is the predefined IdentityMapper and each reduce task is the predefined IdentityReducer, both of which pass their inputs directly to the output. The full input dataset is transferred and sorted during the shuffle phase between the map and reduce tasks. Sort is a very useful benchmark for studying the shuffle phase, which exists in many MapReduce applications.
    - Matrix Multiplication - A common computational task is the multiplication of two matrices. This MapReduce implementation consists of two jobs: the first job performs the block multiplications and the second job sums up the results. In job 1, the map tasks route a copy of each A or B sub-matrix to all the reduce tasks, and the reduce tasks perform the sub-matrices multiplications. Depending on the number of reduce tasks and the number of sub-matrices, a reduce task may calculate one or more product sub-matrices. This strategy makes good use of parallelism at the expense of network traffic. In job 2, an identity map task reads from an input split, which is the output of reduce tasks in job 1, and a reduce task sums up the items for

the same C submatrix. To reduce the network traffic during the sort and shuffle phase, Combiner is used in the implementation.

- Cloudburst - This computation-intensive and data-intensive application maps reads to reference genomes. The input of the program is comprised of two multi-fasta binary files in Hadoop SequenceFile format: one containing reads and the other containing one or more reference sequences. The output is all alignments for each read with up to a user-specified number of differences including both mismatches and indels. The program has three phases: map, shuffle, and reduce. The map task emits k-mers as keys for every k-mer in the reference and all non-overlapping k-mers in the reads. During the shuffle phase the k-mers shared by the reads and the references are grouped. The reduce task extends the seeds into end-to-end alignments allowing for a fixed number of mismatches or indels.
- GridMix - This is a benchmark distributed with Hadoop. The application generates a synthetic mix of jobs to simulate typical production loads. The jobs perform a range of data-access patterns. Data is randomly generated and the benchmark submits a mixture of small and large jobs.

4. Performance Measurements - Several metrics are used to gauge performance. Run time of each job is an important measure since energy consumption is highly correlated with this factor. Other performance measurements are gathered from MapReduce job logs, data logs, and task logs. MapReduce job logs usually provide summary statistics including items such as the number of map tasks, number of reduce tasks, bytes of HDFS read, and bytes HDFS written. MapReduce also generates extensive logs for data and task activity. Appendix A shows an excerpt from the MapReduce data log. A data log is maintained for each data node. It records details about the movement of each block through HDFS. Similarly, the task log records details of each map and reduce task for a

node. The information includes creation, completion percentage of each task, and type of activity - namely copy, sort, and shuffle.

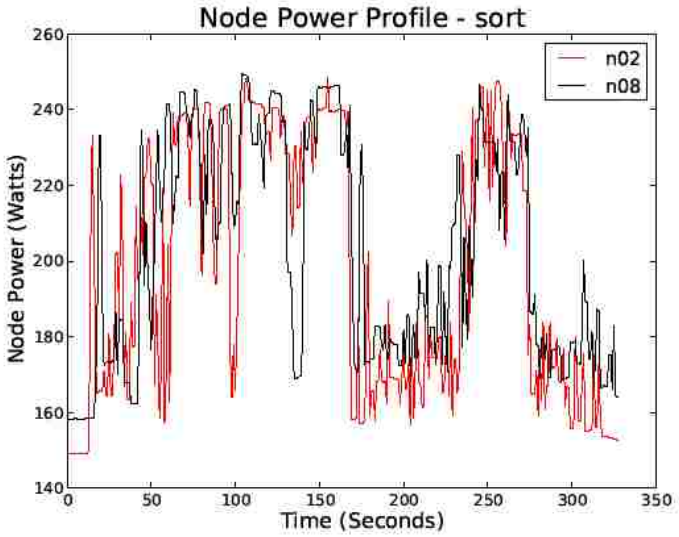
**Experimental Observations** Experimental observations provide insights in building the power model. We run the Sort benchmark on an eight-node Hadoop cluster. In the experiments, node n01 is configured as the JobTracker and the HDFS metadata server, and nodes n02-n08 as TaskTrackers and DataNodes. All nodes are enabled with nodal power measurement, and node n08 has additional component power measurement.

Figure 3.3 shows the nodal and component power profiles of the sort benchmark. We draw the following conclusions:

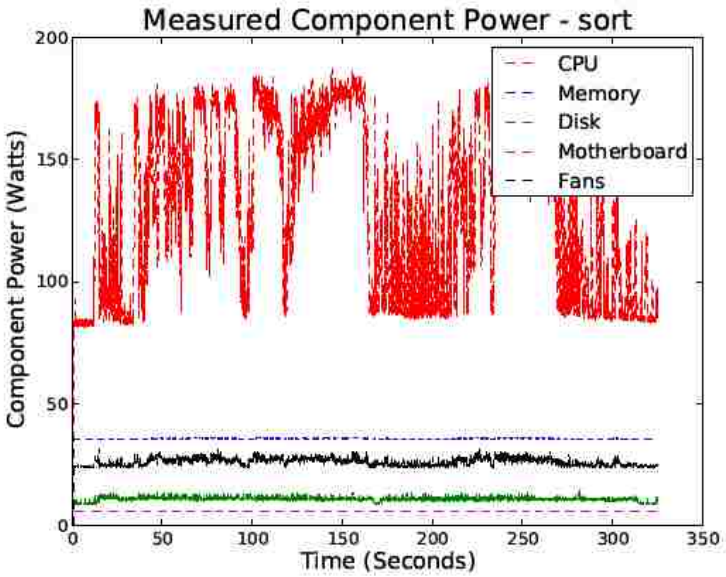
1. The power profiles of all task nodes share identical trend, rising and dropping at roughly same time points with similar highest power values.
2. The nodal power consumption varies significantly with time and execution phases.
3. CPU power dominates and varies significantly with time and execution phase in a manner similar to system nodal power.
4. The power consumptions of other components including memory, disk, motherboard, fans are relatively constant.

Figure 3.4 shows various performance profiles on node n08, the model node which is attached to the NIServer. From this Figure, we have the following findings:

1. During the Sort execution, CPU utilization varies dramatically over time. It is below 40% most of time, and yet stays around 100% for about one-sixth of time.
2. Retired instructions per cycle (uOP C and I P C ) and L1 data cache accesses per cycle (AP C DC and AP C I C ) have similar trends over time. Their peaks and valleys match those of the CPU utilization.

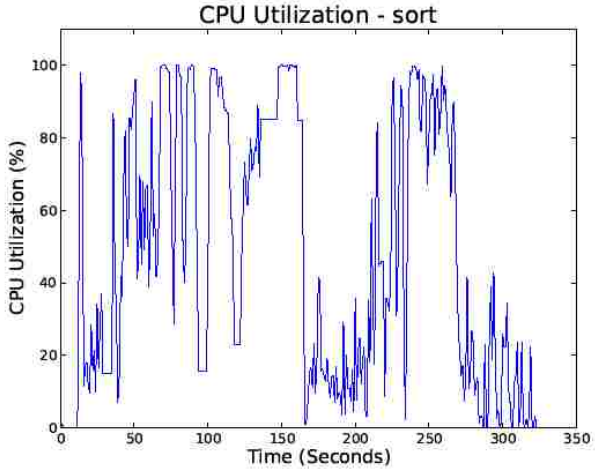



(a) System Power on Two Compute Nodes

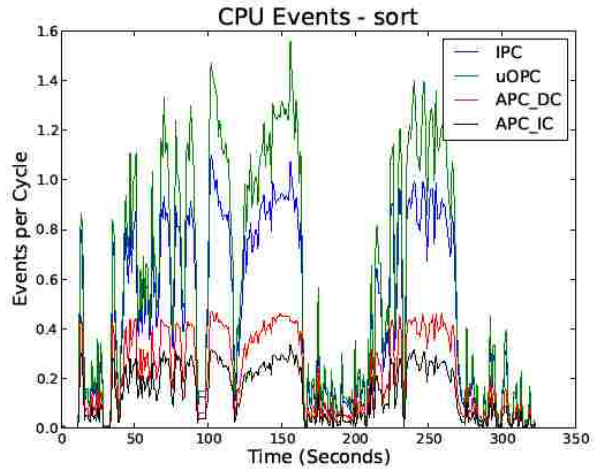



(b) Component Power on Node 08

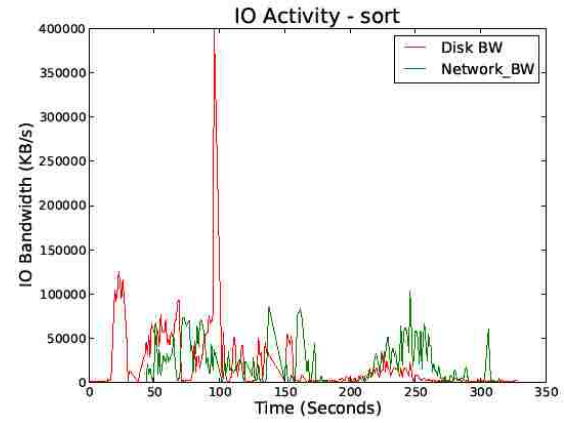
Figure 3.3: The power profiles of the Sort benchmark on an 8-node Hadoop cluster. Each node on the cluster has dual quad-core Opteron processors




(a) CPU Utilization




(b) CPU Events




(c) IO Activity

Figure 3.4: The performance profiles of the Sort benchmark on a single node (Node n08). CPU utilization and CPU events are the average value over eight cores available on the node. The disk bandwidth is summed over read and write. The same applies to the network bandwidth.

3. Disk bandwidth increases dramatically shortly after the job is launched. This corresponds to the starting of HDFS reading for map tasks. Network bandwidth increases about 30 seconds later, corresponding to the starting of the shuffle phase.
4. During the time periods with dramatic high disk bandwidth, both CPU utilization and the collected four performance events stay at a low level.

Based on the above observations, we can make two assumptions that will simplify the power models used by the eTune framework.

1. Because the power consumption of non-CPU components including memory, disk, motherboard, and fans remains relatively constant throughout the jobs, we can consider them as constant at least for MapReduce workloads under current system configuration.
2. The CPU power has strong correlations with CPU utilization and the performance events. Intuitively, it is legitimate to derive power consumption from these performance events.

**Power Model Construction**

**Model Derivation** From a top-down perspective, we model the power consumption of distributed systems at three levels: system power ($P_{sys}$), nodal power ($P_{node}$), and component power ($P_c$, where c ∈ CPU, M EM, DISK, FAN, MB, NIC ). Without considering the interconnect switches, the following equations hold.

$$P_{sys} = \sum_{n=1}^{N} P_{node,n} \tag{3.1}$$

$$P_{node} = \sum_{c \in \{CPU,MEM,DISK,FAN,MB,NIC\}} P_c \tag{3.2}$$

Based on the empirical observations for MapReduce applications on the cluster discussed in Section 3.3, we have:

$$P_c = P_{c,0} \quad \forall\, c \in \{MEM, DISK, FAN, MB, NIC\} \tag{3.3}$$

Here Pc,0 is a constant, meaning the power of a non-CPU component c doesn't change with workload.

With the above simplification, a remaining task is to model CPU and CPU core power. Previous studies have shown strong correlations between CPU power, CPU utilization, and certain set of performance events [4], [13], [16] for general purpose computing on single systems. In this work, we further study power models of CPU cores for data-intensive computing on large scale systems and investigate the selection of performance events and statistical approaches for model accuracy. As discussed in [16], [42], CPU utilization from OS statistics does not provide accurate power estimation for many cases. Therefore, we focus on modeling CPU and CPU core power using more detailed performance measures provided by hardware PMCs.

Since multi-core processors are predominant in server systems, we breakdown CPU power by individual cores using the following equation.

$$P_{CPU} = \sum_{k=1}^{K} P_{core\_k} \tag{3.4}$$

Typically, multiple cores on the same chip generally share common devices such as last level cache, bus, Translation Lookaside Buffer (TLB), etc. Each individual core has its private instruction execution function units and caches accounting for a large portion of chip footprint and the total power consumption [39]. For the cores, we study the power effects of activities in the private hardware and build a quantitative model between core power and the performance measures while evenly distributing the power consumption of the shared devices among the cores. In this study, we consider two types of statistical models for CPU core power consumption: simple linear model, and segmented linear model. The simple linear model can be written in Equation 3.5.

$$P_{CPU} = P_{core,0} + \sum_{i \in [1,I]} a_i \cdot perf_i \quad (3.5)$$

Here, Pcore,0 is the idle power, perfi and ai are the ith performance measure and model coefficient respectively, and I is the total number of performance measures introduced in the model. Theoretically, it would be beneficial to use large I and include more performance events reflecting the ongoing activities on function units, caches, branch prediction, ALU, floating point operation, and data prefetching. However, on real microprocess architectures, only a limited number of hardware counters are available to the users. In this study, we choose I = 4, which is the number of hardware counters supported by the AMD Opteron processors.

The segmented linear model splits the model into several segments with each segment being a simple linear model. To maintain the simplicity of linear regression, we use segmentation on the leading performance measure that has the strongest correlation to power. Mathematically, the resulting segmented linear model has the following form:

$$P_{core} = P^j_{core,0} + a^j_1 \cdot perf_1 + \sum_{i \in [2,I]} a_i \cdot perf_i \quad (3.6)$$

Where perf1 is the leading performance measure in the jth segment:

$$perf_1 \in [perf_1^{j-1}, perf_1^j]$$

for ∀ j ∈ [1, J ]. Here J is the total number of segments. With this segmentation regression, only the intercept and the coefficient of the leading variable change with segments, while the coefficients of other performance measures are relatively constant.

It is trivial that the aggregated CPU power model can be derived by substituting P core; k in Equation 3.4 with Equation 3.5 or 3.6. Because the aggregated CPU power can be directly measured, it is used in Section 3.4 for model validation.

Performance Events Selection We select the following performance measures as candidate variables for modeling CPU core power. Their corresponding performance events monitored by PMCs are listed in Table 3.1(a) and the derivation from performance events in Table 3.1(b).

IPC : the number of instructions per cycle. IPC is a most important performance measure [39], capturing the overall activity of pipeline function units and execution of an application on hardware.

APC : the number of accesses per cycle to instruction and data caches and memory. Memory hierarchy performance is another important performance measure [39], capturing the extra delay due to memory stall.

$\mu$OPC : the number of micro operations per cycle. $\mu$OPC models the extra power consumption of ALUs.

$\mu$FLOPC : the number of floating point operations per cycle. $\mu$FLOPC models the power consumption of floating point operations, which are heavily used in scientific and engineering simulations.

**Model Selection** We configure eTune to collect power and performance events samples for a select set of data intensive benchmarks, and output a training data set in the following form:

$$D = [t, P(t), Perf(t)]$$

Here t denotes timestamp, P(t) is a vector of power measurements at time t, and Perf(t) is a vector of performance event measurements at time t. We implement a model fitting program named modelfit. This program takes the training data set and infers the best fit power model and model parameters using both multivariate linear regression and multivariate segmented linear regression provided by the lm and segmented packages in the R software environment [41].

Table 3.1: The monitored performance events and candidate model variables. CLK cycles is collected with TSC register, and the others with general performance counters.

(a) Measurable Events

| PMC Events |
|---|
| CLK_cycles |
| Retired_instructions |
| Retired_uops |
| DC_accesses |
| Retired_FP_instructions |
| IC_accesses |
| Data_refill_L2 |
| Data_refill_sys |
| MEM_access |

(b) Performance Measures as Model Variable Candidates

| Perf. Measures | Derivation from PMC Events |
|---|---|
| IPC | Retired_instructions/CLK_cycles |
| uOPC | Retired_uops/CLK_cycles |
| uFLOPC | Retired_FP_instructions/CLK_cycles |
| APC_DC | DC_accesses/CLK_cycles |
| APC_IC | IC_accesses/CLK_cycles |
| APC_L2 | Data_refill_L2/CLK_cycles |
| APC_SYS | Data_refill_SYS/CLK_cycles |
| APC_MEM | MEM_access/CLK_cycles |

### 3.4 Results - Model Fitting and Evaluation

Hadoop, version 0.20.2, is running on the cluster. It is configured with one NameNode and one JobTracker, both running on the same physical node. The other seven nodes serve as the DataNodes and perform map and reduce tasks. Unless explicitly stated, the number of concurrent workers on each node is eight.

**Model Fitting and Evaluation** We run four MapReduce programs on the system and use the collected performance and power data to train our model. Each program has unique characteristics and together they provide a good coverage of data-intensive applications on MapReduce platforms. The applications include Gridmix - an application included in the Hadoop distribution that represents typical production loads; Sort - included in the Hadoop distribution, that stresses the shuffling phase; Matrix Multiply - A computation intensive application involving

two large matrices; and CloudBurst - a BLAST algorithm that involves a large workload and is computation intensive.

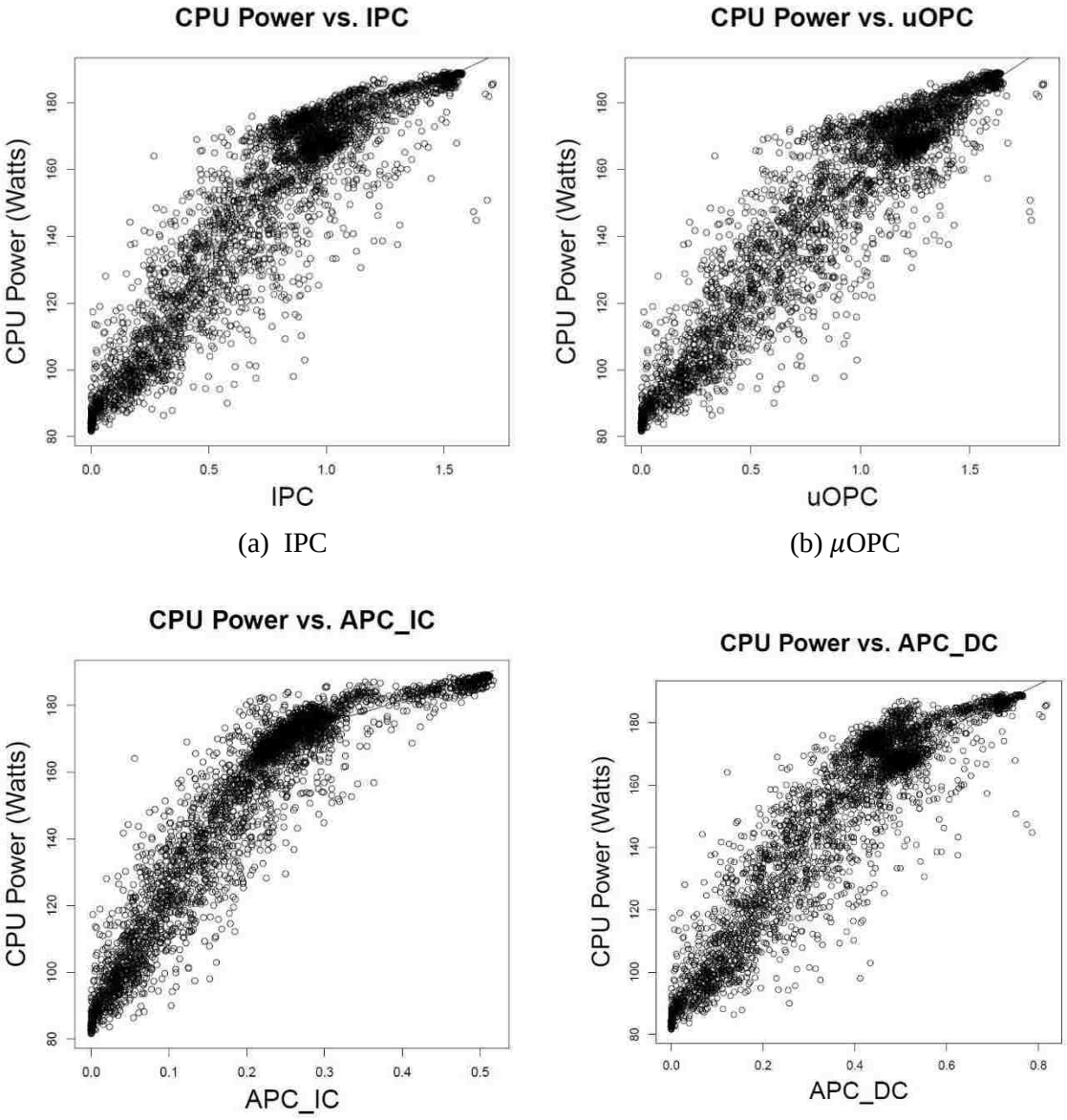
Table 3.2: The coefficients in the multivariate model fitted with the least squares method. Two break points: 0.0423 and 1.1670, are considered for $\mu$OPC in the linear model.

| | Segment I | Segment II | Segment III |
|---|---|---|---|
| $P_0$ | 83.83 | 89.83 | 152.00 |
| $a_1$ | 252.800 | 59.200 | 5.793 |
| $a_2$ | -28.3271 | | |
| $a_3$ | 14.7714 | | |
| $a_4$ | 125.1282 | | |

We collect training data consisting of aggregated CPU power and performance measures listed in Table 3.1 and apply multivariate regression to determine candidate performance measures. We rank the performance measures by their coverage ranges and values of their coefficient of determination R2. Figure 3.5 shows that the four winning candidates with strong correlations are $\mu$OPC , IP , APCDC , and APCIC . With an ordinary linear regression, $\mu$OP C shows the strongest correlation with CPU power with a standard error of 10.92 (Watts) and an R2 value of 0:90. Figure 3.5 also shows a segmented linear model with 2 break points (3 segments) that fits the data better than an ordinary linear model. Based on these results, we select the above four performance measures as model variables and use multivariate segmented linear model with 2 break points segmented by $\mu$OPC as the best-fit model. The derived model and model parameters are shown in Equation 3.7 and Table 3.2 respectively.

$$P_{CPU} = P^j_{CPU,0} + a^j_1 \cdot perf_1 + \sum_{i\epsilon 2,I} a_i \cdot perf_i \tag{3.7}$$

In Equation 3.7, $perf_i$, $i$ = 1..4 denote $\mu$OPC , IPC , APCDC , and APCIC respectively. In this model, we note that P0 = 83.83 and matches with system idle power. Meanwhile, the fact that a2 has a negative value indicates an overlap between $\mu$OPC and IPC.




(a) IPC (b) $\mu$OPC

(b) APC_IC (d) APC_DC

Figure 3.5: The correlation between CPU power and model variable candidates.

**Model Accuracy** - We compare estimated CPU power against direct measurements for all four benchmarks and show the results in Figure 3.6. Overall the model estimations match well with the actual measurements for all four benchmarks running with various Hadoop configurations.

The average estimation error falls within a range of ±5%. This observation confirms the validity of the eTune modeling framework. In contrast, power estimations with CPU utilization is less accurate. Take Gridmix application as an example, the average estimation error using CPU utilization is 14.1%. Since Gridmix reflects the workload of production MapReduce systems, a large range of errors indicates CPU utilization is not an accurate power indicator for data intensive computing.

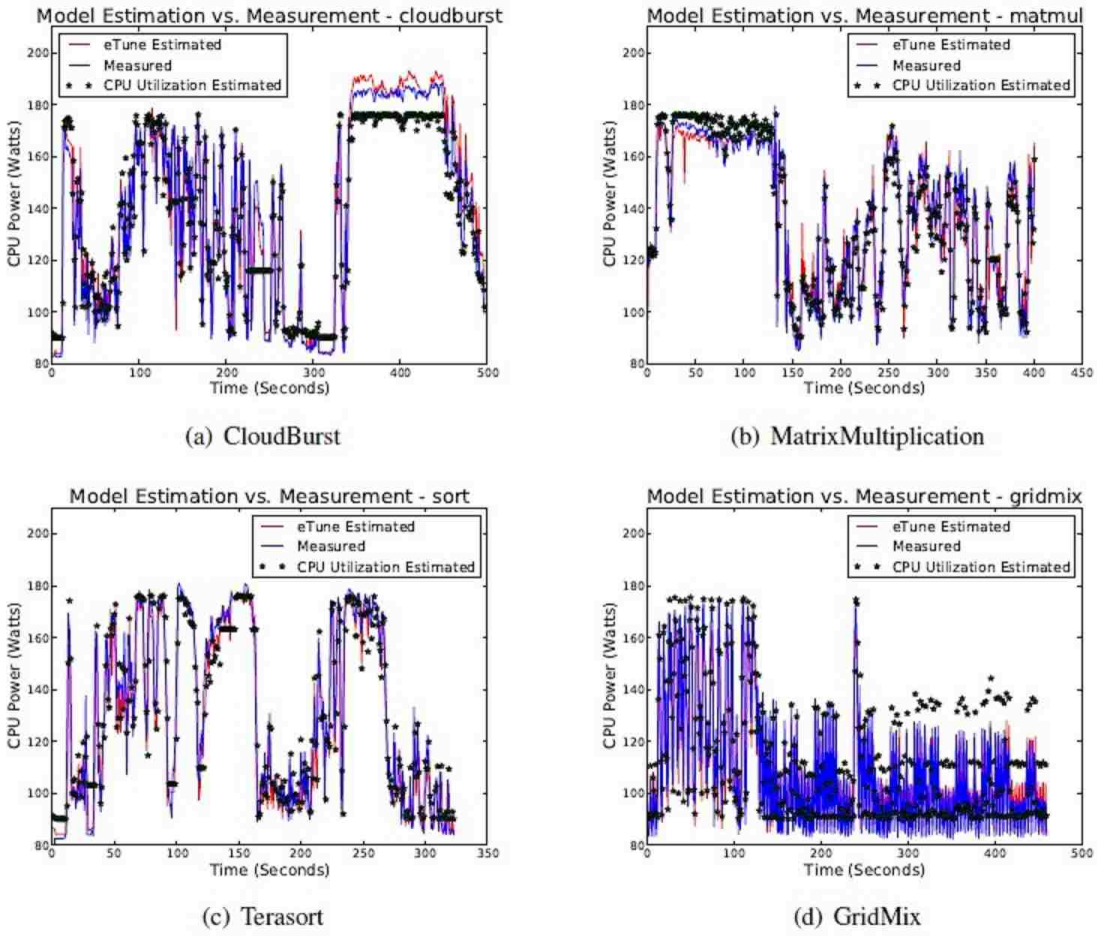



Figure 3.6: The comparisons between estimated power and directly measured power.

### 3.5 Summary

We apply multivariate regression to training data comprised of CPU Power and performance measures. We identify the top four performance measures and find a multivariate segmented

linear model with 2 break points as the best-fit model. The model estimations match well with actual power estimations with average estimate error with ±5%. This profiling software can be utilized to collect and profile fine-grained energy characteristics of the data-intensive MapReduce applications.

# CHAPTER 4 ENERGY EFFICIENCY EVALUATION AND ANALYSIS

## 4.1 Introduction

Having developed a sophisticated software tool to profile energy behavior in data-intensive applications, we implement eTune in this chapter to investigate system-level and component-level energy characteristics. Our study focuses on the data movements of MapReduce applications. This work performs in-depth analysis of data movements in typical MapReduce applications. We investigate the variability of energy efficiency with changes to common MapReduce configuration parameters - numbers of map and reduce tasks, workload size, and HDFS blocksize. We further examine energy and power in data movement by isolating the three major data tasks in MapReduce jobs - HDFS Write, HDFS Read, and shuffle. We apply our findings to a synthetic workload to demonstrate potential energy savings.

## 4.2 Description/Methodology

**Data Movement Patterns** Data movement is a process during which data is transferred from one place to another in a computing platform. Several different data movements occur with MapReduce jobs. One type of data movement occurs as blocks of a file are moved within a node as input for a map task. Another type occurs as input for a map task moves across the network. Another type is the movement of intermediate key-pairs, output from map tasks, from HDFS to local disks. And another type is the movement of data from the local disks between reduce tasks to assemble the final output. We apply three criteria to enhance the usability of our findings. First, we study data movement that is common in MapReduce applications.

- **HDFS Read**. File splits are read from HDFS stable storage disks on data nodes to map functions on task nodes and processors by input readers.

- **Local BufferDisk Write**. Map output data, which are intermediate (key, value) pairs, are written to local buffers and disks from processors by map function on task nodes. Reduce output data are similarly written to local buffers and disks by reduce function.
- **All-to-All Shuffle**. Map output data are exchanged and sorted between all task nodes during the shuffle stage, which follows map stage and precedes reduce stage. This shuffle is in all-to-all fashion across network as each task node pulls data from all other nodes. It involves memory/disk read and write because the initial source and final destination of the shuffled data are buffers and disks on task nodes.
- **HDFS Write**. Reduce output data are written to distributed HDFS stable storage devices on data nodes from processors on task nodes by output writer.

A second criteria is that the data movements should significantly impact performance and energy consumption. Studies have shown that each of HDFS read and write accounts for about 12% of total I/O traffic each [37], and shuffle involves even more data volume and incurs more time due to the all-to-all communication across network [37], [45].

A third criteria is that the data movement should be easy to isolate and measure. The information about all four data movement patterns can be extracted from MapReduce system logs. However, local buffer/disk write is difficult to isolate as it occurs inside map function. Based on the above criteria, we focus our study on the following three data movement patterns: HDFS read, shuffle, and HDFS write.

**Experimental Environment Platform** We use the hardware platform described in 3.3 with Hadoop version 1.0.3. HDFS replication factor 1 is used, as many of our experiments use a single node, to eliminate extraneous activity and power due to communication with nodes not involved in map and reduce tasks. Each job is repeated 5 times and average performance and power are reported.

**Isolation of Data Movement** We use synthetic benchmarks to isolate the data movements and to eliminate interference from each other. Both HDFS read and HDFS write are isolated with the

TestDFSIO I/O benchmark distributed with Hadoop. TestDFSIO reads and writes a specified number of files, whose size can be set through command-line arguments. Shuffle is isolated with a modified version of the RandomWriter application distributed with Hadoop. The original application only involves HDFS write by generating random records instead of reading from disks. We set the OutputFormat parameter of its reduce function as Null-OutputFormat to avoid HDFS write. The resulting application spends a majority of time shuffling records between map tasks and reduce tasks.

**Data Collection** eTune is used to collect nodal and component power data. It also collects and processes system-wide performance data, architecture-level performance events, and application events via the Hadoop JobTracker and TaskTracker logs.

### 4.3 Results

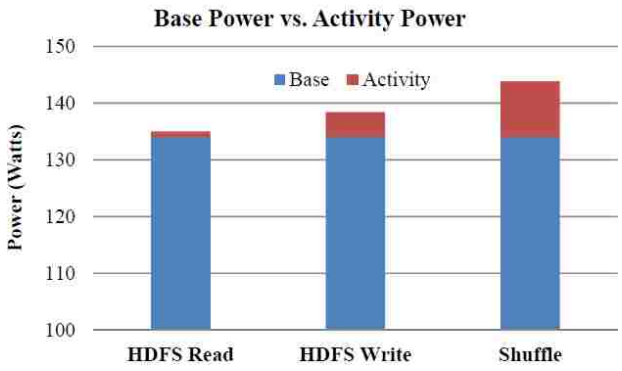



Figure 4.1: Base power and activity power of HDFS read, HDFS write and shuffle. Base power is the power consumption when no user job is running, and activity power is extra power incurred by MapReduce jobs.

**Overall Power Profiles** Figure 4.1 shows node power consumption of these three data movements. The power numbers are averaged over more than 100 experiments, each lasting at least several minutes. Node power comprises base power and activity power. The former represents the power consumption when no user job is running, and the latter represents the

additional power incurred by user jobs. Among these three data movements, HDFS read incurs the least activity power while shuffle incurs 880% more activity power. Base power is 134 watts and dominates for all three data movements and accounts for 90% or more of total system power consumption. To mitigate the inefficient use of base power, multiple hardware and software technologies have been explored and they include energy efficient components [2], server consolidation [43], and deep energy-saving hardware states [1].

Figure 4.2 further shows the breakdown of node power to computer components for each data movement. For all three data movements, CPU power comprises the largest portion of system power, followed by memory, fan, and disk. Most of the power difference between these three data movements is from CPU. The power consumption of memory and disks vary minimally among the three data movements. For example, disk power ranges from 10.31 watts to 10.94 watts, or a 6.1% difference, and fan power ranges from 26.75 watts to 28.0 watts, a 4.7% difference. We also observe that as CPU consumes more power, fans need to run at higher speed to cool down CPUs and consume more power.

Table 4.1 presents the components' activities that are associated with components' power. The power consumption of all components show linear association to the corresponding activities. For example, higher CPU power is caused by higher CPU utilization, and higher memory power comes from higher active memory. Among these components, CPU shows the strongest linear relation between its power consumption and activity.

Table 4.1: Component activities for HDFS read, HDFS write and shuffle.

| | HDFS Read | HDFS Write | Shuffle |
|---|---|---|---|
| CPU utilization(%) | 18.2 | 30.8 | 64.7 |
| Act. Memory (GB) | 1.06 | 1.10 | 1.63 |
| Disk Read (MB/s) | 24.59 | 0.026 | 0.003 |
| Disk Write (MB/s) | 0.16 | 53.58 | 31.54 |

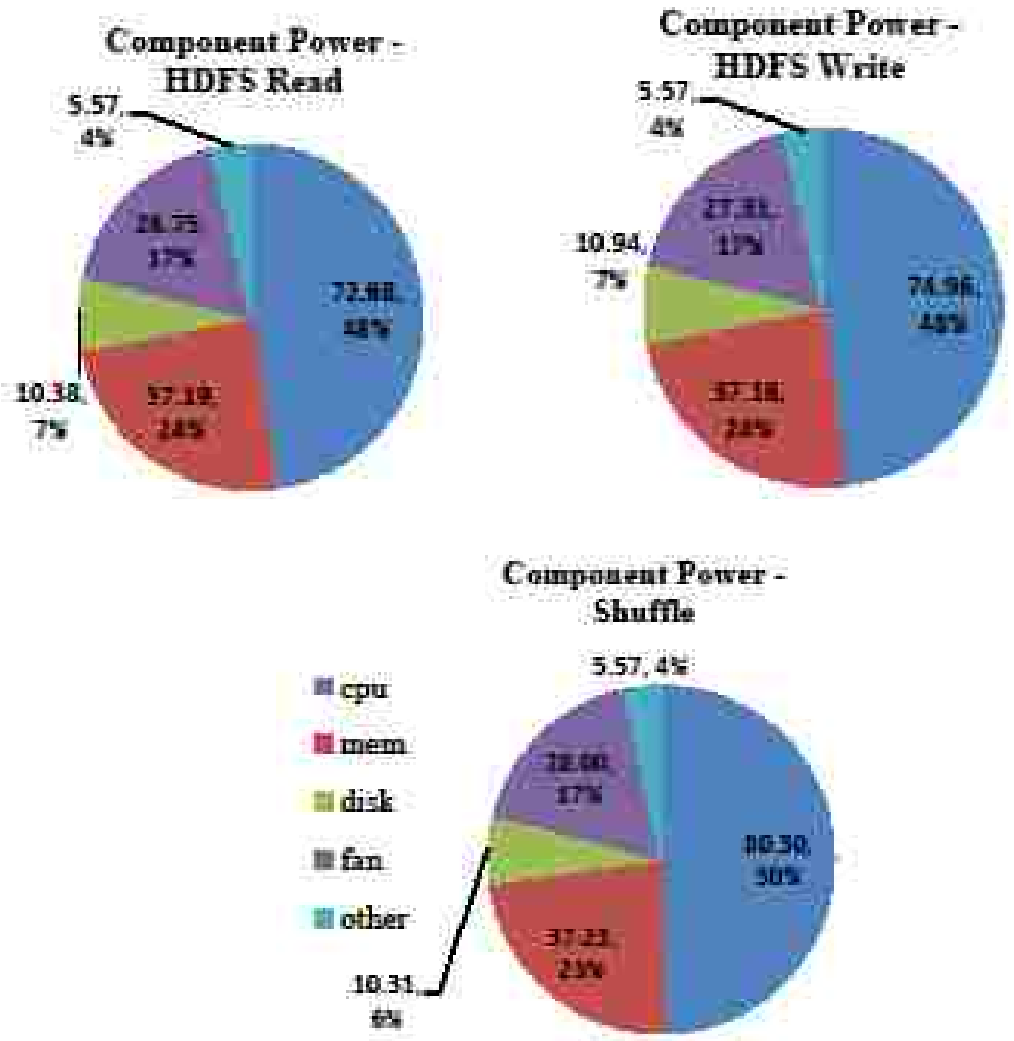
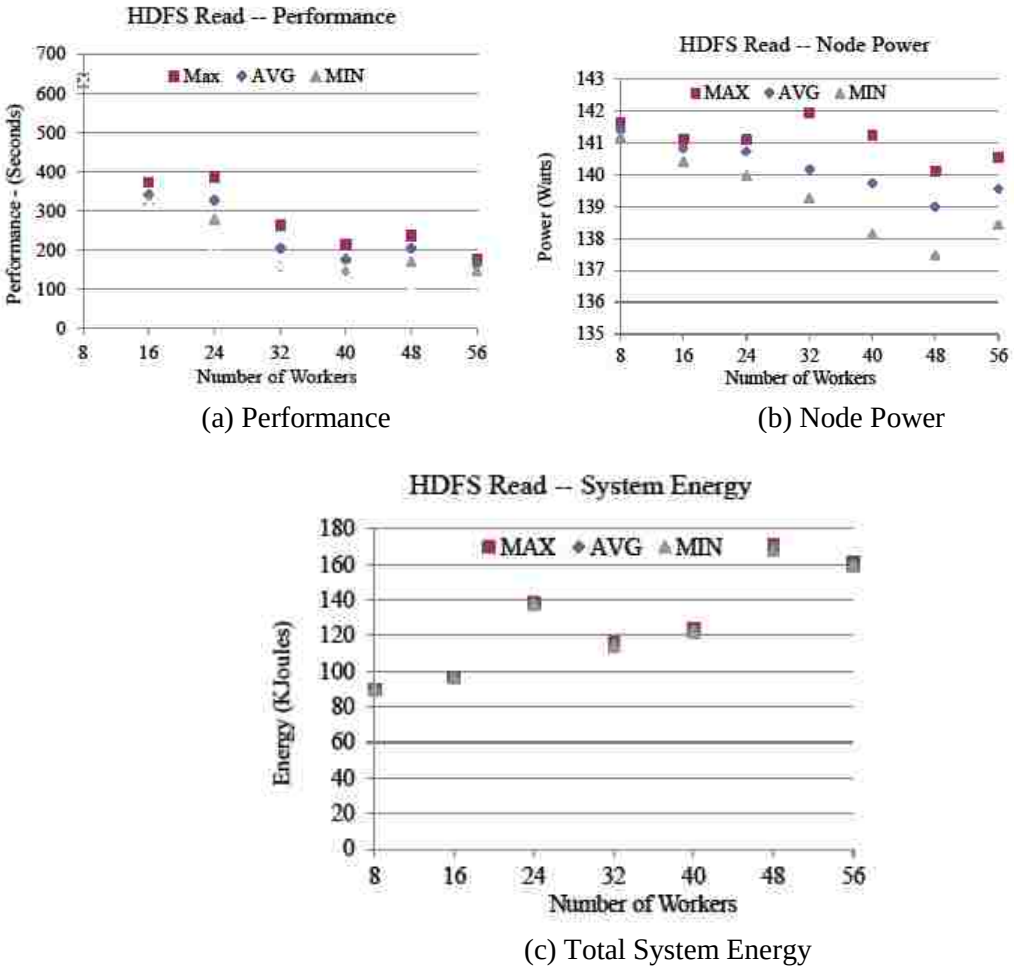



Figure 4.2: Node power breakdown for HDFS read, HDFS write, and shuffle. CPU power makes up the largest portion of node power, and is followed by memory and fan power. CPU also contributes most of the power difference between the data movements.

**Detailed Energy Characteristics and Scalability** We investigate the scalability of energy efficiency of each data movement and their variations with workload and system parameters including number of workers, data size, and HDFS block size.

**HDFS Read**

**Number of Workers** The number of workers determines the degree of parallelism in HDFS read. In our experiments, we change the number of task nodes and thus the number of processors involved in HDFS read across the Hadoop system. The total data volume is fixed at 14GB and the number of files is fixed at 7, each with 2GB data.




(a) Performance (b) Node Power

(c) Total System Energy

Figure 4.3: HDFS Read with various numbers of workers.

Figure 4.3 shows performance, single node power and total system energy of HDFS read under various numbers of task nodes, each consisting of eight workers. Overall, execution time

decreases as the number of workers increases. The speedups gained from using 16 workers and 32 works are significant. Particularly, the speedup is roughly 2X when worker count increases from 8 to 16. This 2X speedup implies 1) the number of splits and map tasks are evenly divided to the workers and 2) a majority of HDFS read is still from local disks. Performance only slightly increases when worker count increases from 16 to 24, which indicates more splits are accessed from remote disks. There are no obvious performance gains by using more than 40 workers due to the overhead of process creation and management.

Power consumption on a single node is very steady with a minimal decrease as the number of workers increases. This indicates less workload is assigned to a single node as more task nodes are involved. However, the total system energy, which is the sum of the product of node power and time over all task nodes, increases with the number of workers. It almost doubles when the number of workers increases from 8 to 48. The energy consumption using 3 task nodes or 24 processors is peculiarly high. This spike occurs because neither performance or power improves by increasing worker count from 16 to 24. A similar spike occurs at 48 workers.

**Workload Size** Energy efficiency of HDFS read is determined by multiple factors including workload size, hardware bandwidth and the number of involving MapReduce tasks. Evaluating the variables of energy efficiency can guide us in configuring workload and system parameters and estimating minimum energy requirement. Here we evaluate the scalability of energy efficiency with data size, as shown in Table 4.2. The number of workers is 8 on one node and the block size is the default value 64MB.

Table 4.2: HDFS read performance, power, and energy efficiency with various data sizes. The last column indicates the normalized energy efficiency based on 2.3KJoules/GB, the best one achieved at 3.5GB workload.

| Workload (GB) | Perf. (MB/s) | Base Power (Watts) | Act. Power (Watts) | Energy Eff. (KJ/GB) | Norm. Eff. % |
|---|---|---|---|---|---|
| 1.75 | 32.5 | 134.00 | 6.14 | 4.37 | 53% |
| 3.5 | 61.19 | 134.00 | 6.80 | **2.30** | 100% |
| 7.0 | 41.13 | 134.00 | 5.48 | 3.39 | 68% |
| 14.0 | 43.21 | 134.00 | 5.83 | 3.24 | 71% |
| 28.0 | 33.10 | 134.00 | 4.79 | 4.19 | 55% |

As data size increases from 1.75GB to 3.5GB, performance almost doubles, activity power slightly changes, and energy efficiency roughly doubles. This is because the number of map tasks doubles in response to the doubling data size. However, overall energy efficiency drops as workload size further increases, mainly due to the reduced achieved overall bandwidth. At 28GB data it halves from the maximum. More data leads to contention between multiple I/O streams and more overhead to manage processes.

**Block Size** Block size determines the size of file splits and granularity of MapReduce tasks. A finer granularity may lead to balanced workload but also incur more overhead in managing MapReduce tasks. Table 4.3 shows the effects of HDFS block size on performance and energy efficiency of HDFS read. The total data size is fixed at 7GB and one task node is used.

Table 4.3: HDFS read performance, power, and energy efficiency with various HDFS block sizes. The last column indicates the normalized energy efficiency based on the best energy efficiency 2.30KJoules/GB.

| BlockSize (MB) | Perf. (MB/s) | Base Power (Watts) | Act. Power (Watts) | Energy Eff. (KJ/GB) | Norm. Eff. % |
|---|---|---|---|---|---|
| 16 | 45.34 | 134.00 | 5.78 | 3.08 | 75% |
| 32 | 50.87 | 134.00 | 6.59 | 2.76 | 83% |
| 64 | 41.13 | 134.00 | 5.48 | 3.39 | 68% |
| 128 | 40.23 | 134.00 | 5.65 | 3.47 | 66% |
| 256 | 55.21 | 134.00 | 6.69 | 2.55 | 90% |

Though there isn't an obvious trend, block size does greatly affect performance and energy efficiency by up to 24%. The highest efficiency occurs at 256MB block size, and the lowest efficiency occurs at 128MB block size. Block size only slightly changes power.

**HDFS Write**

**Number of Workers** Figure 4.4 shows performance, power and energy efficiency of HDFS write with varying number of workers. In these experiments, the number of files is 7 and the total data volume is 14GB. Overall, HDFS writes speed up with more workers but performance gain diminishes with 40 or more workers. Execution time almost halves as worker count increases from 8 and 16. Node power visibly changes with worker counts. It reaches the highest at 8 workers (144.4 watts) and drops to the lowest at 32 workers (141.7 watts). System consumes the lowest energy with 8 workers while it delivers the best energy-performance tradeoffs with 40 workers.

**Workload Size** Table 4.4 presents the effects of workload size on local HDFS write on a single task node. Performance monotonically increases with workload size at a similar rate

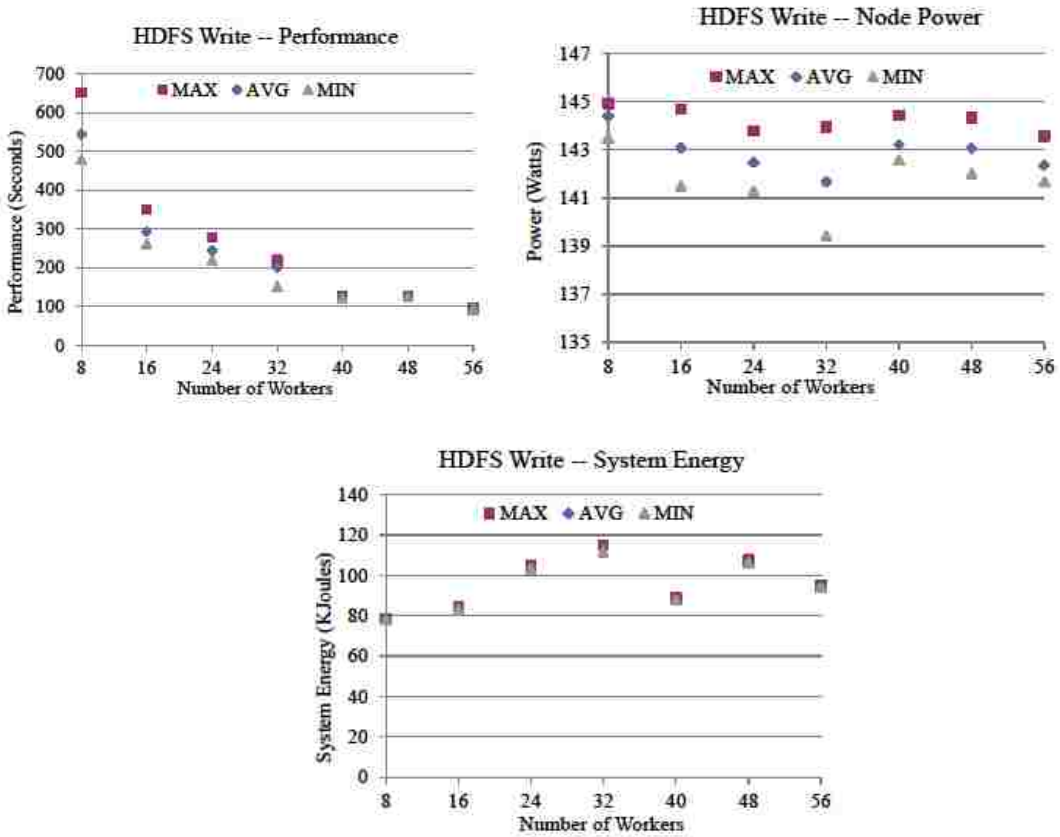



Figure 4.4: HDFS write with various numbers of workers.

except at 28GB. The corresponding I/O write bandwidth reaches maximum at 14GB, and then drops. Node power increases by up to 8-10 watts on top of base power. Energy efficiency has a similar trend as performance. It rises from the lowest value and then reaches the peak at 14GB workload size, and then decreases. This trend is explained by the various buffers used in the HDFS platform and disk devices. As workload size is small, these buffers prevent HDFS write from blocking by temporarily storing the data. Once workload size exceeds the buffer threshold, HDFS write blocks and overhead of managing these buffers incurs.

**Blocksize** Table 4.5 presents the effects of HDFS block size on HDFS write on one task node. The total data amount is 7GB. Performance significantly changes with block size, the trend is unclear, though. The best performance occurs at 32MB block size while the worst performance occurs at 128 MB. Node activity power can rise up to 9 watts on top of base power. The best energy efficiency occurs at 32MB and 256 block size.

Table 4.4: HDFS write performance, power, and energy efficiency with various data sizes. The last column indicates the normalized energy efficiency based on the best energy efficiency 2.56KJoules/GB, that is the best for HDFS write and achieved at 14GB workload.

| Workload (GB) | Perf. (MB/s) | Base Power (Watts) | Act. Power (Watts) | Energy Eff. (KJ/GB) | Norm. Eff. % |
|---|---|---|---|---|---|
| 1.75 | 29.97 | 134.00 | 8.26 | 4.75 | 54% |
| 3.5 | 34.94 | 134.00 | 1.12 | 3.87 | 66% |
| 7.0 | 40.37 | 134.00 | 9.34 | 3.55 | 72% |
| 14.0 | 55.42 | 134.00 | 8.09 | **2.56** | 100% |
| 28.0 | 52.06 | 134.00 | 8.67 | 2.74 | 94% |

Table 4.5: HDFS write performance, power, and energy efficiency with various block sizes. The last column indicates the normalized energy efficiency based on the best energy effi- ciency 2.56KJoules/GB achieved at 14GB workload.

| BlockSize (MB) | Perf. (MB/s) | Base Power (Watts) | Act. Power (Watts) | Energy Eff. (Joules/GB) | Norm. Eff. % |
|---|---|---|---|---|---|
| 16 | 44.36 | 134.00 | 8.44 | 3.21 | 80% |
| 32 | 50.95 | 134.00 | 9.53 | 2.82 | 91% |
| 64 | 40.37 | 134.00 | 1.12 | 3.35 | 77% |
| 128 | 40.32 | 134.00 | 7.97 | 3.52 | 73% |
| 256 | 50.14 | 134.00 | 9.45 | 2.86 | 90% |

**MapReduce Shuffle**

**Number of Workers** Figure 4.5 shows the effects of number of workers when the total data volume is fixed at 6GB. Execution time greatly drops as more workers are used. The speedup is 3X when worker count increases from 8 to 16, and 2X when worker count increases from 16 to 24. However, speedup is marginal once more than 40 workers are used. Node power rises on top of base power during the shuffle. Activity power is about 10 watts with 8 workers and 13.5 watts with 24 workers. Total system energy dramatically drops with worker count and is minimum with 24 workers. More than 24 workers leads to higher total system energy consumption.

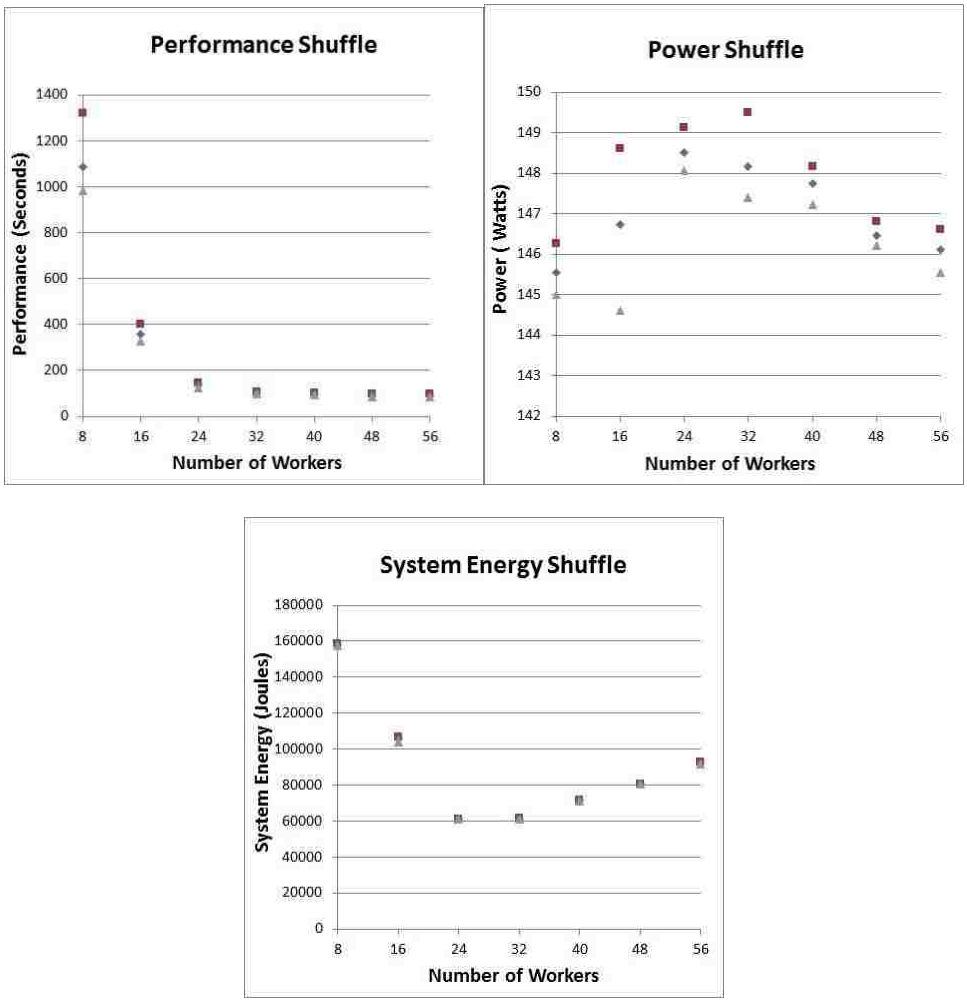



Figure 4.5: Shuffle with various numbers of workers.

**Workload Size** Performance dramatically changes as workload size increases, as shown in Table 4.6. Bandwidth almost doubles with workload size. Bandwidth reaches the maximum when data size is 1.5GB. At this point the platform saturates. Bandwidth drops though, as data size further increases. This is due to the overhead of buffer and process management. Node power rises about 8-13 watts on top of base power during shuffle, and reaches the highest with 1.5GB workload data. There is a linear association between node activity power and achieved bandwidth: the higher the bandwidth is, the larger node power is. The resulting energy efficiency is low at 0.375GB and reaches the highest at 1.5GB.

Table 4.6: Shuffle performance, power, and energy efficiency with various data sizes. The last column indicates the normalized energy efficiency based on the best energy efficiency 9.15KJoules/GB achieved at 1.5GB workload.

| Workload (GB) | Perf. (MB/s) | Base Power (Watts) | Act. Power (Watts) | Energy Eff. (KJ/GB) | Norm. Eff. % |
|---|---|---|---|---|---|
| 0.375 | 5.70 | 134.00 | 8.89 | 25.07 | 37% |
| 0.75 | 9.8 | 134.00 | 11.27 | 14.76 | 62% |
| 1.5 | 16.2 | 134.00 | 14.26 | **9.15** | 100% |
| 3.0 | 11.64 | 134.00 | 12.29 | 12.57 | 73% |
| 6.0 | 6.22 | 134.00 | 11.99 | 23.46 | 39% |

**Blocksize** Table 4.7 shows the performance, power, and energy efficiency under various block sizes when the total data amount is 1.5GB. In general, block size only slightly impacts performance, power, and energy efficiency. This is expected because no HDFS I/O is involved during shuffle phase.

Table 4.7: Shuffle performance, power, and energy efficiency with various block sizes. The last column indicates the normalized energy efficiency based on the best energy efficiency 9.15KJoules/GB achieved at 64MB block size.

| BlockSize (MB) | Perf. (MB/s) | Base Power (Watts) | Act. Power (Watts) | Energy Eff. (KJ/GB) | Norm. Eff. % |
|---|---|---|---|---|---|
| 16 | 15.76 | 134.00 | 14.10 | 9.38 | 98% |
| 32 | 14.97 | 134.00 | 14.31 | 9.41 | 97% |
| 64 | 16.20 | 134.00 | 14.26 | 9.15 | 100% |
| 128 | 15.76 | 134.00 | 13.95 | 9.88 | 93% |
| 256 | 15.79 | 134.00 | 14.12 | 9.40 | 97% |

**Data Center Applications** Chen et al [10, 9] generated synthetic workloads from Facebook production traces to complement benchmarks for realistic evaluations of MapReduce workloads in data centers. We apply our findings to one synthetic workload and analyze its energy characteristics. This workload is a representative sample of jobs over a 24 hour period on a 3000-machine cluster. It comprises 24,442 jobs with a diversity of data sizes and data movements, as shown in Figure 4.6. 10.0% of the jobs process more than 28GB bytes each, and together account for 97.6275% of data volume of HDFS Read. In contrast, 32.9% of the jobs process less than 1MB data each, together accounting for less than 0.0001% of data volume. 33.8% of the jobs either read significantly more data than write or do the opposite, and a large number of jobs (34.1%) do not have shuffle phase. We apply the energy characteristics of data movements measured from our platform to this synthetic workload. Our analysis shows that this workload would annually consume 346,550 Kwh energy for HDFS read, 293,251 Kwh energy for HDFS

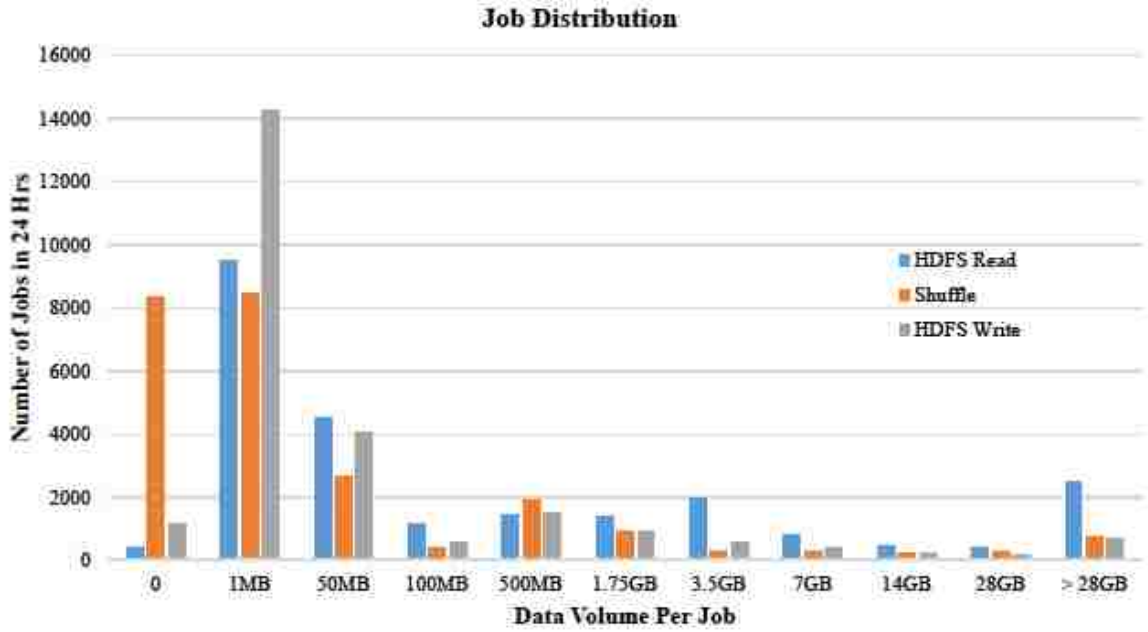



Figure 4.6: A synthetic workload generated from Facebook production traces.

write, and 138,516 Kwh energy for shuffle with the default 64MB HDFS block size. 33.1% energy could be saved if 256MB HDFS block size is configured for these jobs.

### 4.4 Summary

We use eTune to gather fine-grained information about data movement with MapReduce. We further isolate the three major phases of data movement in MapReduce - HDFS Read, HDFS write, and shuffle. As such, we are able to report characteristics of each phase and how each phase reacts to some of the common MapReduce parameter configurations.

- HDFS read incurs the least system activity, followed by HDFS write, and finally shuffle.
- Base power accounts for 90% or more of total system power consumption.
- For HDFS read, the poor speedup as multiple nodes are used reinforces two common characteristics of MapReduce. 1) there is a detrimental performance and power effect for non-local read of input and 2)IO is a major bottleneck in MapReduce and negatively affects performance.
- When resource allocation remained stable, there was an ideal workload size for MapReduce. Our normative efficiency measure showed that the peak performance occurred at a smaller workload size for HDFS read (3.5GB) compared with the peak for HDFS write (14GB).
- There appears to be a linear association between node activity power and achieved bandwidth.
- The only components with major variation between HDFS read, HDFS write and shuffle are CPU power and CPU fan. CPU power and Active memory account for almost 75% of power consumption.

# CHAPTER 5 ENERGY EFFICIENCY OPTIMIZATION

## 5.1 Introduction

In the previous chapters, we developed a software tool to profile energy behavior in data-intensive application in data centers and we detailed the power and energy characteristics of data movements within MapReduce applications. In this chapter, we identify methods to optimize MapReduce applications. We focus on a growing set of MapReduce applications - computation intensive computing. We want to determine the impact of higher computation intensity affects energy efficiency. We use an experimental approach to study how resource allocation and DVFS scheduling will affect energy efficiency for MapReduce applications. We apply resource management and system configuration to find optimization opportunities. The resource management control we apply is the number of workers for the application. We use DVFS (Dynamic Voltage and Frequency Scaling) scheduling since these jobs have a computation intensive component.

## 5.2 Methodology

MapReduce Benchmark Applications We include three MapReduce benchmark applications in our experiments that span the spectrum of data intensive and computation intensive applications. The CloudBurst benchmark is both computation intensive and data intensive. The Matrix Multiplication benchmark is also computation intensive, but compared to CloudBurst, its data set is significantly smaller. To reveal the system behavior of the shuffling phase in many MapReduce applications, we also include the Sort benchmark in the Hadoop distribution, which is data intensive but not computation intensive.

**Energy Management Parameter Space** The performance and energy of MapReduce applications are affected by two major factors: the number of concurrent workers, i.e., the number

of worker nodes n times the number of workers per node c, and f , the processor frequency on each work node.

**The number of concurrent workers** In this work, we execute each benchmark with multiple settings, where each setting is identified by a unique number of concurrent workers. The concurrency is determined by the number of worker nodes allocated and the number of concurrent tasks on each node. To maximize the performance and efficiency, we use all 8 processor cores (i.e., c = C = 8) on each node during benchmark runs. The concurrency ranges from 8 with 1 worker node to 56 with 7 worker nodes. We use hadoop-daemon.sh to control the TaskTracker on each compute node and give a delay of 15 minutes to allow Hadoop to recognize the active/inactive node. We repeat the experiments 5 times in each setting and use average performance and energy in the analysis. To ensure no extra disk and network I/O is introduced for the varying number of concurrent workers, data replication is set to 8 on our 8-node cluster. With this replica setting, each node has a copy of the required data in the local storage disk and accesses the data locally. For CloudBurst and Sort, the data is replicated prior to the job execution. For Matrix Multiplication, the data is generated on the fly.

**The processor frequency** The key of DVFS scheduling is to identify the workload phases and then adapt the processor frequency to match the computational demand of each phase. In this work, we analyze and identify the workload phases and corresponding performance and energy use by tracing system activities. Specifically, we trace CPU utilization, memory access, disk IO bandwidth, and network bandwidth on the worker nodes. We consider three DVFS scheduling policies:

- Fixed policy: a single processor frequency is used for all cores across the worker nodes during the entire execution.
- Adaptive I policy: based on workload phase heuristics observed from MapReduce application performance traces, we insert DVFS scheduling codes into MapReduce programs to adjust processor frequency during its execution. Specifically, this policy uses

maximum processor frequency inside the map and reduce functions, and uses minimum processor frequency otherwise. Thus, the computations in the map and reduce tasks are with faster cores while I/O accesses are with slower cores for power reduction. The actual deployment of this policy on the Hadoop System is at job level. This is because Java based MapReduce framework lacks the capability to identify the specific physical core associated with a map/reducer task. Particularly, we set the affinity of the TaskTracker daemons to core 0 on each node, and fix its frequency at maximum speed. Then we apply the DVFS scaling to the remaining seven cores on each worker node.

- Adaptive II policy: This policy is performance-constraint and bounds the performance loss within a user specified value. The performance loss is relative to the performance at highest fixed processor frequency. In this work, we set the allowable performance loss 5%. With this constraint, a low processor frequency might not be scheduled for execution phases even if the resulting power reduction is much more than the performance loss. CPUMiser [19] implements this policy. CPUMiser uses hardware performance counters to collect fine grain CPU activity information, and uses such information to predict the performance and identify target processor speed periodically at runtime. CPUMiser runs on each node in the cluster and adapts the processor frequency of each core to applications' demand.

**Evaluation Metrics** We use execution time (T ) as performance metric and total system energy (E) for energy metric. We also introduce two other metrics in our analysis. The first one is work-induced energy $E^{WI}$ , defined as:

$$E^{WI} = E - T \cdot P_{idle} \tag{5.1}$$

The rationale of using work-induced energy in addition to total system energy lies in the fact that in today's data centers, idle power dominates system power consumption, accounting for up to

60% of the system power under load. Meanwhile, motivated by the concept of energy proportional computing [3], which essentially assumes zero idle power, many techniques are being developed to significantly reduce the idle power. Thus, we believe work induced energy provides a direct indication of energy demand by the applications and workloads.

The second metric is energy-performance efficiency, defined as the ratio of performance per Joule, or

$$eff_n = \frac{\frac{1}{T_n}}{E_n^{WI}} / \frac{\frac{1}{T_1}}{E_1^{WI}} = \frac{T_1}{T_n} / \frac{E_n^{WI}}{E_1^{WI}} = \frac{PerformanceSpeedup}{RelativeEnergy} \quad (5.2)$$

The metric $eff_n$ measures how performance per Joule scales with the number of processor cores within the context of energy-proportional computing. $eff_n = 1$ indicates constant performance per Joule, or performance grows with the number of worker nodes at the same speed as energy consumption. $eff_n > 1$ indicates performance grows faster than energy consumption.

**5.3 Results**

**The Effects of the Number of Concurrent Workers** Matrix Multiplication: As shown in Figure 5.1, the execution time decreases when the number of concurrent workers increases. Due to parallel overhead, a maximum relative speedup of 3.3, instead of an ideal speedup of 7, is achieved when n = 56. While total system energy increases significantly when more worker nodes are used for parallel programs due to system idle power, work-induced energy only increases slightly. The energy-performance efficiency increases with n and achieves the maximum when n = 48. By allocating 48 concurrent workers on 6 nodes, we can achieve 3X speedup with 6.6% extra work-induced energy, or 2.8X efficiency using the metric defined in Equation 5.2.

To explain the above observation, we trace the CPU utilization, network and disk accesses during the execution. Figure 5.1(c) shows two apparent low CPU utilization

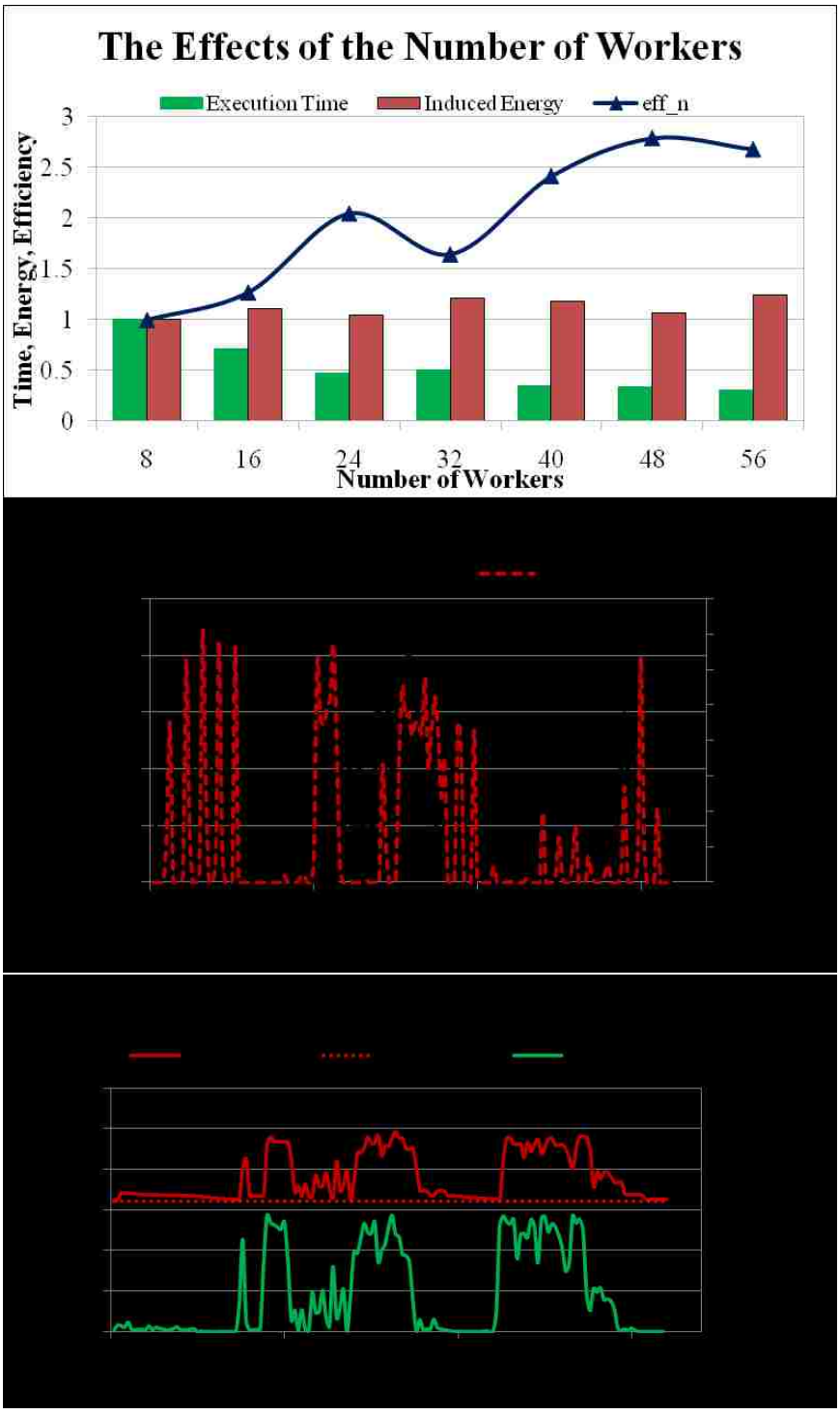



Figure 5.1: The variations of performance and energy with the number of concurrent workers for Matrix Multiplication. (a) the normalized performance, energy, and efficiency against 8 workers, (b) the I/O traces and (c) the power traces and CPU utilization when n=48 and f=2.5GHz.

phases during the execution. The first matches the distribution of input data for the first MapReduce job and the second corresponds to the finishing of the first MapReduce job and the setting up for the second MapReduce job. There is a short period of low CPU utilization during the first job execution when the map tasks finish and the shuffle occurs. As the reduce task is computation intensive, a high CPU utilization is sustained for the second MapReduce job. Complementing CPU utilization, three I/O intensive phases are observed in Figure 5.1(b). The first phase corresponds to the job initialization, and the last two correspond to the first and second MapReduce job respectively.

The power trace in Figure 5.1(c) highlights how total power and idle power of a single node vary during the execution. The work-induced power is the difference between the total power and the idle power. The idle power is about 160 Watts and dominates the total power, even when the CPU utilization is close to 100%. The idle power is about twice the maximum work induced power when matrix multiplication program executes. This observation indicates effective power reduction technologies should consider reducing system idle power as a top priority. The work induced power curve follows the same trend as CPU utilization. This figure also implies that within this experimental environment, the majority of work-induced power comes from CPU activity, and the memory and I/O activity only slightly change the total node power.

CloudBurst: As shown in Figure 5.2a, Cloudburst achieves super-linear speedup with the number of concurrent workers because more data can be accessed in memory versus from disks with larger number of workers. With 48 concurrent workers, Cloudburst achieves a maximum speedup of 12X and a minimum work-induced energy 0.7X, resulting in an optimal efficiency value of 17.4. In contrast to Matrix Multiplication, CloudBurst has better scalability in both performance and energy. Thus allocating more resources for CloudBurst is preferred.

The system activity traces provided in Figure 5.2(b)-(c) and MapReduce log files indicate there are two MapReduce jobs in this benchmark; each job consisting of a map, a shuffle, and a reduce phase. The first job accounts for 90% of the total execution time and the CPU utilization is

high during most of map and reduce phases, except in the middle and the end of map tasks where CPU utilization oscillates around 20%. The I/O traces further reveal that network traffic and disk I/O accesses are high within the map and reduce phases. In addition, there are short periods with low CPU and I/O activities between two MapReduce jobs or different phases. These traces indicate that even though CloudBurst is computation intensive, its MapReduce implementation involves significant disk and network accesses and warrants energy efficiency optimization.

Sort: Unlike the above two benchmarks, Sort does not scale well with the number of cores. As shown in Figure 5.3(a), while the execution time gradually decreases when more cores are used, the maximum speedup is still less than 2. On the other hand, work-induced energy gradually increases with the number of concurrent workers. Sort also delivers its best efficiency at n = 48.

System activity traces in Figure 5.3(b)-(c) reveal that disk and network accesses are very active during most of the execution period. These heavy I/O activities are responsible for a lower CPU utilization than previous two benchmarks.

**The Effects of Processor Frequency** While the analysis in the previous section demonstrates that resource allocation is an effective approach to improve both performance and efficiency, it also points out that there are significant I/O activities within MapReduce applications. Provided that DVFS is a practical energy saving technology for non-CPU bound applications, we discuss how different DVFS scheduling policies presented in Section 5.2 perform for MapReduce applications in this section.

Figure 5.4 shows the performance, energy, and efficiency when the three DVFS scheduling policies are applied to the benchmarks running with 56 concurrent workers. The first four groups correspond to fixed policy with 4 different frequencies: 2.5 GHz, 1.8 GHz, 1.3 GHz, and 0.8 GHz. Adaptive I inserts DVFS control into the benchmark source code. Adaptive II uses CPUMiser to schedule the core frequencies.

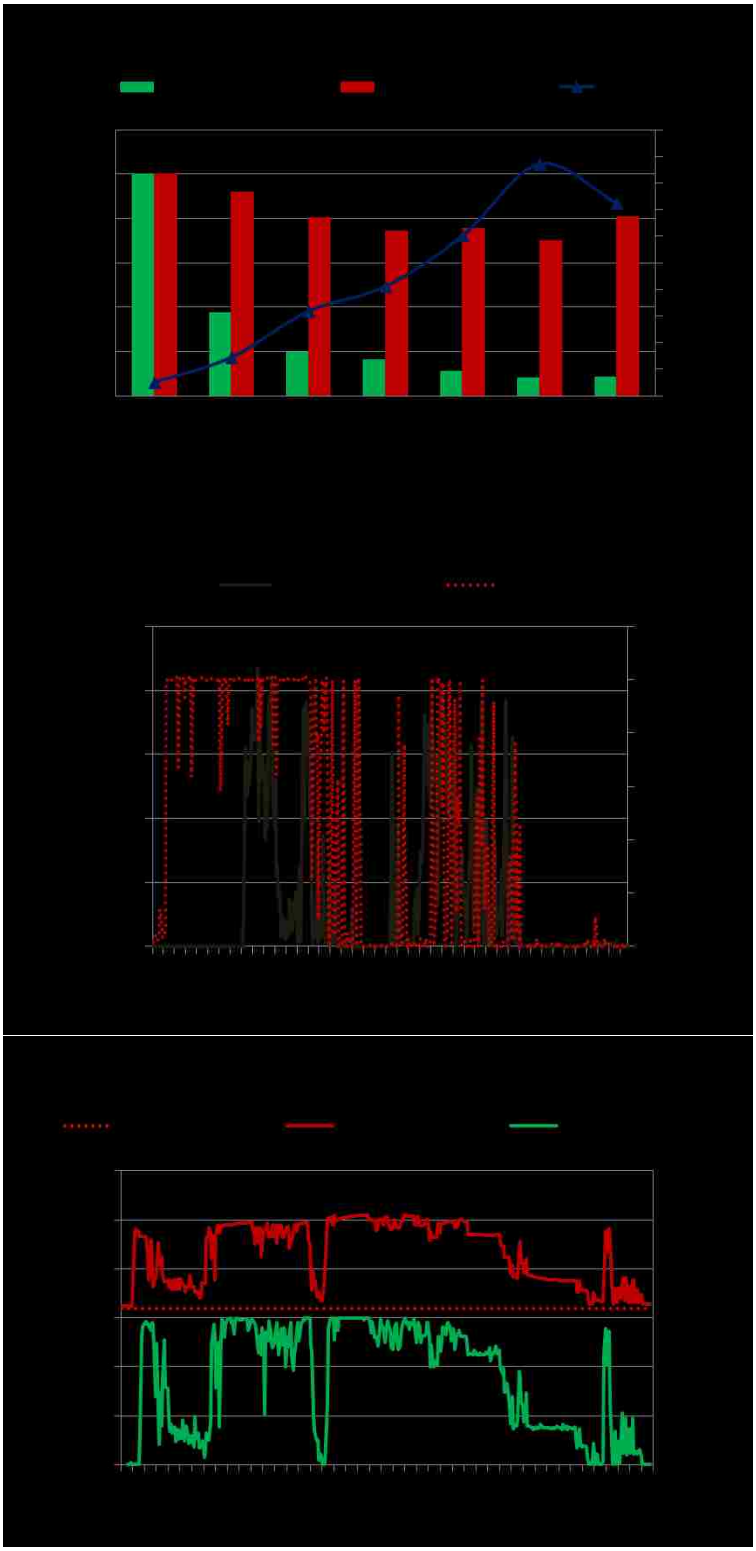


Figure 5.2: The variations of performance and energy with the number of workers for CloudBurst. (a) the normalized performance, energy, and efficiency against 8 workers, (b) the I/O traces and (c) the power traces and CPU utilization when n=48 and f=2.5GHz.

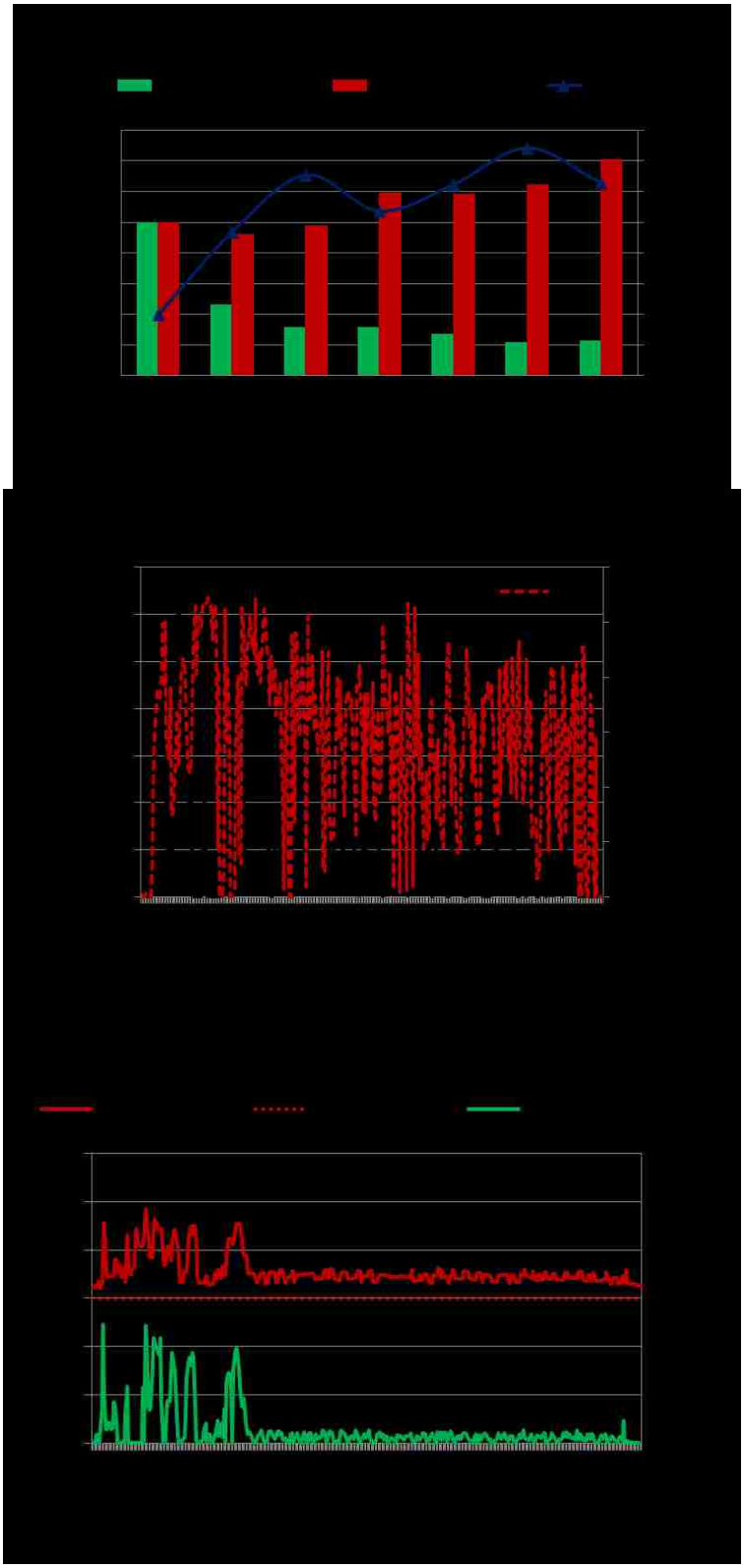


Figure 5.3: The variations of performance and energy with the number workers for Sort. (a) the normalized performance, energy, and efficiency against 8 workers, (b) the I/O traces and (c) the power traces and CPU utilization when n=48 and f=2.5GHz.

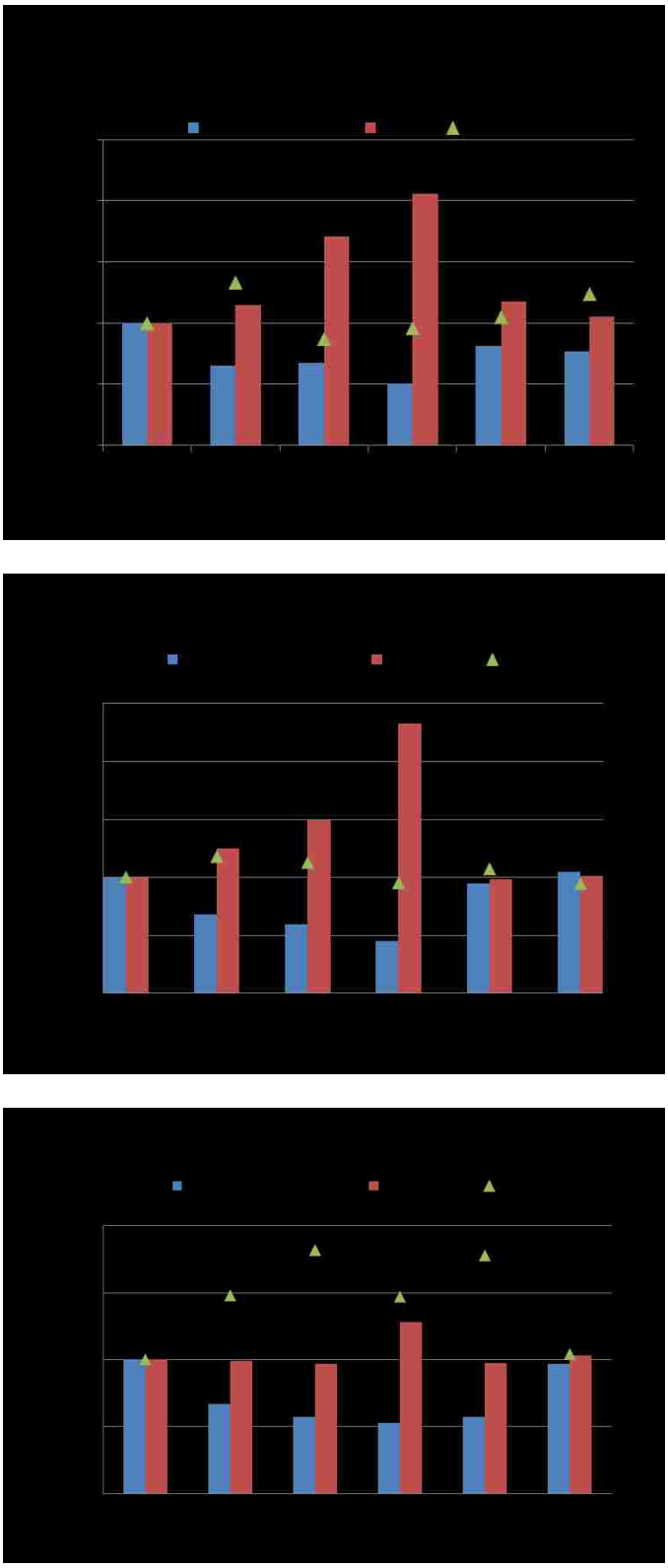


Figure 5.4: The effects of various DVFS policies for Matrix Multiplication, Cloudburst and Sort.

Fixed Policy : Overall, for all three benchmarks, a best efficiency has been observed when running the benchmarks at a fixed frequency, though the optimal frequency differs from code to code. For Matrix Multiplication, the optimal frequency is 1.8 GHz, at which there is 35% work-induced energy saving at the cost of 15% performance degradation, resulting in an efficiency number of 1.33. For CloudBurst, 1.8 GHz also results in a best efficiency of 1.18, with 32% savings of work-induced energy at the cost of 24% performance loss. A more interesting result happens for Sort. At 1.3 GHz, it achieves an efficiency number of 1.33 with a 35% work-induced energy saving and a 4% performance gain. A performance gain from lower processor frequency has also been observed for NPB sorting benchmarks IS in our earlier work [30]. We believe this is a result of better matching between processor and system bus speeds. However, this explanation is not confirmed yet and we are still investigating it.

While the results of fixed policy are promising, there are two major issues with it. First, it requires extensive performance and energy profiling. Second, the performance decrease is usually significant except for some rare cases such as the Sort benchmark.

Adaptive I policy : With sufficient internal information about the workload, we expect the adaptive I policy to result in better efficiency improvement. However, the experiments show mixed results. For Matrix Multiplication, this policy reduces the work-induced energy by 19% at an expense of 17% performance degradation. For CloudBurst, it delivers a similar performance at 2.5 GHz and reduces the work induced energy by 5%, which is equivalent to 3% total system energy saving. For Sort, the resulting performance and energy are similar to those achieved at 1.3 GHz.

Adaptive II policy : Unlike the adaptive I policy, CPUMiser is implemented as a system software and adapts the processor frequency automatically without requiring code changes or performance profiling. Another unique feature of CPUMiser is that its performance control

prevents some unacceptable cases such as large energy saving at the cost significant performance slowdown.

The experimental results match our expectations. For Matrix Multiplication, the adaptive II policy reduces the work-induced energy by 23% with a 5% performance loss, improving the efficiency number by 23%. CPUMiser does not save energy for CloudBurst because lowering processor frequency would adversely degrade performance. For Sort, CPUMiser delivers a same performance as 2.5GHZ fixed policy with 4% induced energy reduction.

Figure 5.5 presents power traces of the three benchmarks with fixed 2.5 GHz and Adaptive II policies. The power traces with Adaptive II policy are identical to those at 2.5 GHz for Matrix Multiplication and CloudBurst, except some shift due to lower processor frequency and lower power consumption for idle or non-CPU intensive phases. For Sort, CPUMiser schedules processor frequency to lower values to save energy. The traces also reveal that as CPUMiser seeks performance oriented energy savings, it works best for current systems with large idle power but might not be the best for future energy-proportional computing systems.

**5.4 Summary**

We apply resource management and three DVFS strategies to determine ways to optimize computation intensive applications. We examine three different workloads - one high data and computation intensive, one high computation and mid data intensive, and one data intensive but nor computation intensive. For each type of job, the best energy efficiency occurred with 48 workers. However the rate of energy efficiency varied greatly with the best efficiency occurring in the high data, high computation workload.

Although a fixed DVFS policy had best energy efficiency, there were significant performance decreases. Similarly, our adpative I policy had good energy efficiency at the expense of performance. The adaptive II policy showed good energy performance improvement with minimal performance loss for the high computation, mid data intensive application. The adaptive

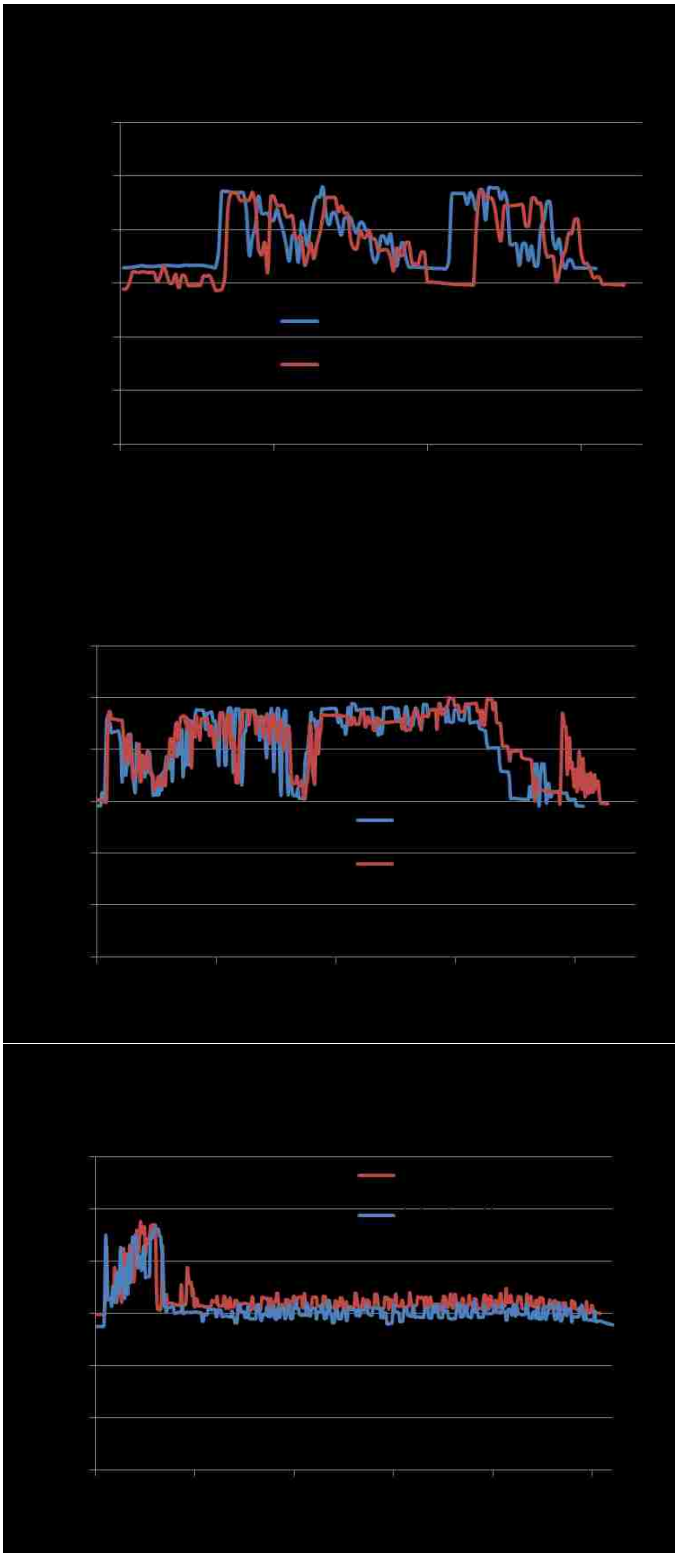


Figure 5.5: The power traces under fixed 2.5GHz and Adaptive II DVFS scheduling policies for Matrix Multiplication, Cloudburst and Sort.

II policy did not take effect for the high data, high computation workload due to the performance safeguard.

As we analyzed the data, we observed that even when CPU utilization is 100%, idle power still dominates total power. Therefore, reducing system idle power is a necessary and major approach for energy efficiency in MapReduce.

# Chapter 6 Discussion and Conclusion

## 6.1 Summary

This thesis details three areas of our investigation of energy efficiency with data intensive computing of MapReduce. Our studies include the development of a software tool, eTune, to assist in the scientific, fine-grained measurement and profiling of data intensive applications in distributed environments. Then, from information collected with eTune, we perform an in-depth examination of how the various data movements in MapReduce impact energy performance. Finally, we work to identify ways to optimize MapReduce in computation intensive applications. Some of the major contributions of this work include:

- eTune - a software tool for measuring and profiling energy in data intensive applications. MapReduce creates some unique challenges for measuring energy performance. There is a need for measurement tools that can be easily and inexpensively used in large, distributed environments; that can collect fine-grained information related to energy; and that are unobtrusive to the applications under study. We have developed eTune, a software tool that addresses these needs by providing a multivariate, segmented regression model based on data collected through system performance counters and application performance data. We have validated the tool with power meters on several MapReduce applications. eTune more accurately estimates power than CPU utilization.
- Detailed energy characteristics of data movements within MapReduce. Data movement within MapReduce contributes to poor energy performance. This involves movement of data between local disk and the distributed file system and between nodes as data is moved from the output of map tasks to the input of reduce tasks. It is important to understand how the many aspects of data movement affect MapReduce. We perform an in-depth analysis of data movement for the three data movement phases in MapReduce - HDFS read, HDFS write, and shuffle. We demonstrate that the various MapReduce

activities possess unique and distinct power profiles. Upon stressing each of the data movement phases, we found that HDFS read consumed the least power, followed by HDFS write and then shuffle. Examination of the power components revealed variation across the three phases for CPU and the CPU fan. It also revealed that CPU and active memory comprise approximately 75% of the power consumption. The energy measurements provide a way to identify the optimal performance and energy for a job.

- Optimization of energy efficiency in computation intensive MapReduce applicatoins. As MapReduce is utilized more in scientific applications, it is important to understand how to optimize energy efficiency in computation intensive applications. We study two approaches to improve energy efficiency with applications that have different levels of data and computation intensity. We provide a metric, energy performance efficiency, to assess energy performance. The metric is useful to identify the number of workers to optimize energy efficiency. Also, with high CPU utilization in these computation intensive applications, we evaluate three DVFS strategies. We find that performance constrained DVFS scheduling strategies improve energy efficiency.
- Dominance of idle power. A few features of energy and power in MapReduce were evident throughout our studies. First, we confirmed the findings of several others that MapReduce is not an energy efficient framework. In many of our experiments, we observed that speedup of MapReduce was much less than ideal. Most important, through our fine-grained analysis, we were able to document the idle power in MapReduce jobs. We found that base power accounts for 90% or more of total power.

**6.2 Discussion**

Despite the dominance of base power, there is a clear need to address activity power and work induced energy. We provide some measurement tools and methods to evaluate energy efficiency for these. Measuring energy at several levels - system, node, and CPU - provides

insight for ways to optimize data-intensive applications. More can be done to improve our understanding of activity power and work induced energy. Future work might utilize eTune on many nodes throughout a cluster to document the variability of power profiles during the execution of a single job. Do certain aspects of MapReduce, such as data movement or job characteristics, have more stable power profiles across nodes? This leads to a more general question. Can we identify ways for managers of data centers or for programmers to use MapReduce more efficiently?

Our use of the Facebook production traces revealed a high degree of diversity and a wide range of MapReduce jobs with respect to the size of data read, written, and shuffled. This raises a few interesting points. First, MapReduce was designed to process large amounts of data. In this synthetic workload, though, a significant portion of the jobs process 1MB of data or smaller. It is important to study the energy impact of these small jobs. Our work focused exclusively on the slaves. For these small jobs, it may be necessary to include the data and task masters in the analysis since a large portion of the work for the job may occur at that level. Second, with a mixture of jobs, it is important to use job level configuration settings. Data center managers will not obtain energy efficiency with MapReduce by relying on a system-wide setting for blocksize or simply using the maximum number of workers available.

There appear to be two major challenges in working towards energy efficiency in data centers. First, it appears that job performance is the primary, and perhaps the sole guide. A second challenge is the variability and change in the workload and applications. It is difficult to determine parameter settings in such a changing environment. Ideally, easy to use tools that classify application characteristics could help. We think the application of the energy efficiency equation could identify optimum conditions for energy efficiency. This objective measure would provide a quantitative guide for parameters and for performance goals.

In several of our experiments, the node power and system energy results fluctuated as the parameter increased. Several factors contribute to this variance. First, tasks within MapReduce

are independent and are not uniformly distributed. Also, our studies showed that with specific resources, there is an ideal workload size and HDFS blocksize. We speculate that blocksize is an important and primary factor to determine for MapReduce jobs. So, while MapReduce can operate with minimal configuration on a wide variety of platforms, it is prudent to establish the optimal blocksize. We found that a large blocksize was most efficient for both HDFS read and HDFS write operations with a large workload. Additional research is needed to establish the optimal blocksize; whether a single blocksize is ideal for all workloads; and whether the optimal blocksize is the same for HDFS read and HDFS write operations. It is likely that the optimal blocksize will vary on different platforms, so some tools to easily identify the ideal configuration would be beneficial.

### 6.3 Conclusion

While idle power consumes the majority of power in MapReduce applications, focus on activity power is still valuable. Both system designers and data center operators benefit from the deeper understanding of energy characteristics in these jobs. The sophisticated tools and general power trends identified in these studies can be applied to optimize energy.

## BIBLIOGRAPHY

[1] Advanced Configuration and Power Interface Specification. http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf.

[2] Intel Delivers the World's First 6-Watt Server-Class Processor. http://newsroom.intel.com/community/intel_newsroom/blog/2012/12/11/ intel-delivers-the-worlds-first-6-watt-server-class-processor.

[3] L. Barroso and U. Holzle. The Case for Energy-Proportional Computing. Computer, 40(12):33–37, 2007.

[4] F. Bellosa. The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems. In Proceedings of 9th ACM SIGOPS European Workshop, Kolding, Denmark, 2000.

[5] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya, et al. A taxonomy and survey of energy-efficient data centers and cloud computing systems. Advances in Computers, 82(2):47–111, 2011.

[6] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In 27th International Symposium on Computer Architecture, pages P. 83–94, Vancouver, BC, 2000.

[7] R. Brown et al. Report to congress on server and data center energy efficiency: Public law 109-431. 2008.

[8] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy Efficiency for Large-Scale Mapreduce Workloads with Significant Interactive Analysis. In Proceedings of the 7th ACM european conference on Computer Systems, pages 43–56. ACM, 2012.

[9] Y. Chen, S. Alspaugh, A. Ganapathi, R. Griffith, and R. Katz. Statistical Workload Injector for MapReduce (SWIM). https://github.com/SWIMProjectUCB/SWIM/wiki/Workloads-repository.

[10] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The Case for Evaluating MapReduce Performance Using Workload Suites. In Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on, pages 390–399. IEEE, 2011.

[11] Y. Chen, A. Ganapathi, and R. H. Katz. To Compress or Not to Compress - Compute vs. IO Tradeoffs for Mapreduce Energy Efficiency. In Proceedings of the first ACM SIGCOMM workshop on Green networking, Green Networking '10, pages 23–28, New York, NY, USA, 2010. ACM.

[12] Y. Chen, L. Keys, and R. Katz. Towards Energy Efficient MapReduce. Technical Report Technical Report No. UCB/EECS-2009-109, University of California at Berkele, 2009.

[13] G. Contreras and M. Martonosi. Power Prediction for Intel XScaleOR Processors Using Performance Monitoring Unit Events. In Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on, pages 221–226. IEEE, 2005.

[14] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 51(1):107–113, 2008.

[15] P. Dubey. Recognition, mining and synthesis moves computers to the era of tera. Technology@ Intel Magazine, pages 1–10, 2005.

[16] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. ACM SIGARCH Computer Architecture News, 35(2):13–23, 2007.

[17] J. Flinn and M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In the Second IEEE Workshop on Mobile Computer Systems and Applications, 1999.

[18] J. Gantz and D. Reinsel. Extracting Value from Chaos. White Paper, IDC, 2011. [19] R. Ge, X. Feng, W.-c. Feng, and K. W. Cameron. Cpu miser: A performance-directed, run-time system for power-aware clusters. In Parallel Processing, 2007. ICPP 2007. International Conference on, pages 18–18. IEEE, 2007.

[20] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. IEEE Transactions on Parallel and Distributed Systems, 99(1), 2010.

[21] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. Parallel and Distributed Systems, IEEE Transactions on, 21(5):658–671, 2010.

[22] I. n. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks. In Proceedings of the 7th ACM european conference on Computer Systems, EuroSys '12, pages 57–70, New York, NY, USA, 2012. ACM.

[23] N. Griffiths. nmon performance: A free tool to analyze aix and linux performance, 2003.

[24] J. Hartog, Z. Fadika, E. Dede, and M. Govindaraju. Configuring a MapReduce Framework for Dynamic and Efficient Energy Adaptation. In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, pages 914 –921, june 2012.

[25] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr, and R. Bianchini. Energy conservation in heterogeneous server clusters. In Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming, pages 186–195. ACM, 2005.

[26] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu. Maestro: Replica-aware map scheduling for mapreduce. In Cluster, Cloud and Grid Computing (CCGrid), 012 12th IEEE/ACM International Symposium on, pages 435–442. IEEE, 2012.

[27] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In the 36th annual IEEE/ACM International Symposium on Microarchitecture, page 93, 2003.

[28] R. Joseph, D. Brooks, and M. Martonosi. Live, runtime power measurements as a foundation for evaluating power/performance tradeoffs. In Workshop on Complexity-effective Design, Goteborg, Sweden, 2001.

[29] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. Irwin, and A. Sivasubramaniam. vEC: Virtual Energy Counters. In Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, pages 28–31. ACM, 2001.

[30] R. Kaushik, T. Abdelzaher, R. Egashira, and K. Nahrstedt. Predictive Data and Energy Management in GreenHDFS. In Green Computing Conference and Workshops (IGCC), 2011 International, pages 1–9. IEEE, 2011.

[31] R. T. Kaushik, M. Bhandarkar, and K. Nahrstedt. Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system. In Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, pages 274–287. IEEE, 2010.

[32] J. Koomey. Growth in Data Center Electricity Use 2005 to 2010. Oakland, CA: Analytics Press. August, 1:2010, 2011.

[33] W. Lang and J. M. Patel. Energy management for mapreduce clusters. Proceedings of the VLDB Endowment, 3(1-2):129–139, 2010.

[34] J. Leverich and C. Kozyrakis. On the Energy (in) Efficiency of Hadoop Clusters. ACM SIGOPS Operating Systems Review, 44(1):61–65, 2010.

[35] S. Li, T. Abdelzaher, and M. Yuan. TAPA: Temperature Aware Power Allocation in Data Center with Map-Reduce. In Green Computing Conference and Workshops (IGCC), 2011 International, pages 1 –8, july 2011.

[36] S. Ma, X. Sun, and I. Raicu. I/O Throttling and Coordination for MapReduce. Technical report.

[37] R. McDougall. Analyzing Hadoop's Internals with Analytics. http://cto.vmware.com/analyzing-hadoops-internals-with-analytics/.

[38] P. Paraskevopoulos and A. Gounaris. Optimal Tradeoff between Energy Consumption and Response Time in Large-Scale MapReduce Clusters. In Informatics (PCI), 2011 15th Panhellenic Conference on, pages 144 –148, 30 2011-oct. 2 2011.

[39] D. A. Patterson and J. L. Hennessy. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, San Fancisco, CA, 3rd edition, 2003. CPI formulas found in pages 35-38.

[40] P. Popa. Managing server energy consumption using ibm powerexecutive. white paper (May 2006), 2006.

[41] R Development Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

[42] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. In Proceedings of the 2008 conference on Power aware computing and systems, pages 3–3. USENIX Association, 2008.

[43] C. Subramanian, A. Vasan, and A. Sivasubramaniam. Reducing Data Center Power with Server Consolidation: Approximation and Evaluation. In High Performance Computing (HiPC), 2010 International Conference on, pages 1 –10, dec. 2010.

[44] S. Sur, H. Wang, J. Huang, X. Ouyang, and D. Panda. Can High-Performance Interconnects Benefit Hadoop Distributed File System. In Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC). Held in Conjunction with MICRO, 2010.

[45] Y. Wang, X. Que, W. Yu, D. Goldenberg, and D. Sehgal. Hadoop Acceleration Through Network Levitated Merge. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, page 57. ACM, 2011.

[46] A. Weissel and F. Bellosa. Process Cruise Control-Event-Driven Clock Scaling for Dynamic Power Management. In Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2002), Grenoble, France, 2002.

[47] T. Wirtz and R. Ge. Improving MapReduce Energy Efficiency for Computation Intensive Workloads. In Second International Green Computing Conference and Workshops (IGCC), pages 1–8. IEEE, 2011.

[48] 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 Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, pages 1–9. Ieee, 2010.

[49] W. Xiong and A. Kansal. Energy Efficient Data Intensive Distributed Computing. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2011.

[50] Y. Xu, Z. Luan, Z. Cheng, D. Qian, N. Zhang, and G. Guan. Predictive Data and Energy Management under Budget. In Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on, pages 80–87. IEEE, 2012.

[51] W. Ye, N. Vijaykrishnan, M. T. Kandemir, and M. J. Irwin. The Design and Use of Simplepower: a Cycle-Accurate Energy Estimation Tool. In Design Automation Conference, pages 340–345, 2000.

[52] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In Proceedings of the 5th European conference on Computer systems, pages 265–278. ACM, 2010.

[53] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling Hard-Disk Power Consumption. In Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pages 217–230. USENIX Association, 2003.

# APPENDIX A EXCERPT FROM MAPREDUCE LOG FILE

2012-11-28 20:01:36,392 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /10.1.255.254:50010, dest: /10.1.255.247:50873, bytes: 1043, op: HDFS READ, cliID: DFSClient attempt 201211281826 0019 m 000119 1, offset: 10752, srvID: DS-41921281-10.1.255.254-50010-1351729320491, blockid: blk -2466018376943290026 9840, duration: 729512

2012-11-28 20:01:52,192 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk 1594602514789743968 9852 src: /10.1.255.254:43642 dest: /10.1.255.254:50010

2012-11-28 20:01:52,344 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk 7510101801503309833 9857 src: /10.1.255.254:43646 dest: /10.1.255.254:50010

2012-11-28 20:01:52,373 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk -271840664644927180 9857 src: /10.1.255.254:43645 dest: /10.1.255.254:50010

2012-11-28 20:01:52,790 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk -4979687340524957654 9861 src: /10.1.255.254:43651 dest: /10.1.255.254:50010

2012-11-28 20:01:52,883 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk -2936618566191555453 9863 src: /10.1.255.254:43652 dest: /10.1.255.254:50010

2012-11-28 20:01:53,063 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk 3538453300647792441 9870 src: /10.1.255.254:43654 dest: /10.1.255.254:50010

2012-11-28 20:01:53,237 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk -5822080454651759526 9872 src: /10.1.255.254:43656 dest: /10.1.255.254:50010

2012-11-28 20:01:53,954 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk 2893539232139696470 9879 src: /10.1.255.254:43658 dest: /10.1.255.254:50010

2012-11-28 20:01:54,725 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk -5759061036097675437 9894 src: /10.1.255.254:43662 dest: /10.1.255.254:50010

2012-11-28 20:01:55,043 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk 2067467499973467856 9898 src: /10.1.255.254:43664 dest: /10.1.255.254:50010

2012-11-28 20:01:57,269 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /10.1.255.254:43642, dest: /10.1.255.254:50010, bytes: 67108864, op: HDFS WRITE, cliID: DFSClient attempt 201211281826 0019 r 000031 0, offset: 0, srvID: DS-41921281-10.1.255.254-50010-1351729320491, blockid: blk 1594602514789743968 9852, duration: 4567527500

2012-11-28 20:01:57,269 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk 1594602514789743968 9852 terminating 2012-11-28 20:01:57,276 INFO

org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk -7220754479178278721 9918 src: /10.1.255.254:43665 dest: /10.1.255.254:50010

2012-11-28 20:01:57,295 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /10.1.255.254:43645, dest: /10.1.255.254:50010, bytes: 67108864, op: HDFS WRITE, cliID: DFSClient attempt 201211281826 0019 r 000010 0, offset: 0, srvID: DS-41921281-10.1.255.254-50010-1351729320491, blockid: blk -271840664644927180 9857, duration: 4906758483

2012-11-28 20:01:57,296 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk -271840664644927180 9857 terminating

2012-11-28 20:01:57,333 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk -4568061548608895496 9918 src: /10.1.255.254:43666 dest: /10.1.255.254:50010

2012-11-28 20:01:57,458 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /10.1.255.254:43646, dest: /10.1.255.254:50010, bytes: 67108864, op: HDFS WRITE, cliID: DFSClient attempt 201211281826 0019 r 000024 0, offset: 0, srvID: DS-41921281-10.1.255.254-50010-1351729320491, blockid: blk 7510101801503309833 9857, duration: 5105083758