NATIONAL TECHNICAL UNIVERSITY OF ATHENS

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

MICROPROCESSORS AND DIGITAL SYSTEMS LABORATORY

Performance Monitoring and Workload Characterization for Big Data and Cloud Based Applications on the Intel SCC Manycore Platform

Diploma Thesis

Andreas - Lazaros Georgiadis

Supervised By

Dimitrios Soudris, Associate Professor

Athens, April 2015

Acknowledgements

I would like to wholeheartedly thank Professor Dimitrios Soudris for giving me the opportunity to carry out my diploma thesis under his supervision. This diploma thesis has been a unique opportunity for me so as to be introduced in the process of scientific research and Mr. Soudris has always provided me with motivation and inspiration to pursue this goal.

I would also like to thank Senior Research Associate Sotirios Xydis for the continuous guidance he has provided me with throughout the development of this thesis and for all the knowledge he has shared with me, the help he has offered and his constant engagement. I would also like to thank Dimitrios Rodopoulos and Ioannis Giannakopoulos for the valuable assistance they offered me when I asked.

Finally, I want to thank all the people that stood beside me throughout the years of my studies in NTUA. I want to thank my friends for all the experiences we have had during these years and especially my family, my parents and my brother for constantly supporting me in achieving my goals.

Contents

Abstract

The scope of this Diploma Thesis is to explore several performance, power consumption and scalability aspects of the execution of Big Data and Cloud Based workloads on the Intel Single-chip Cloud Computer Manycore Platform, which differentiates from typical cluster topologies, since it integrates 48 cores on a single chip. The applications we study are implemented using the MapReduce framework on top of the Hadoop Distributed File System. For the purpose of this analysis we have developed a runtime monitoring infrastructure which utilizes Ganglia, a monitoring tool for large clusters.

Chapter 1 initially states the importance of studying Cloud Computing and Big Data Applications and presents some basic aspects of the concepts this diploma thesis deals with. This chapter concludes with the contribution this thesis attempts to make in the field of scale-out applications and many-core systems.
Chapter 2 describes recent research findings in the related fields of scale-out workloads and performance and power monitoring of the Intel SCC that have provided the background and inspiration for this diploma thesis.
Chapter 3 describes the architecture of the Intel SCC in detail, emphasizing on aspects of the platform whose understanding is crucial for application behavior characterization.
Chapter 4 presents a detailed analysis of the Hadoop Distributed File System and the MapReduce framework, by discussing key implementation aspects and providing guidelines of how to configure an HDFS cluster installation and tune the execution of MapReduce jobs.
Chapter 5 provides a detailed description of the tools that have been used and developed so as to deploy and launch Hadoop Clusters on the Intel SCC. The Runtime Environment setup and the Hadoop Cluster installation processes are described and explained in detail.
Chapter 6 presents the Runtime Monitoring Framework we have developed for the Intel SCC. The Ganglia Cluster topology we have configured for the Intel SCC is analyzed and the process of collecting, storing and visualizing runtime metrics is explained.
Chapter 7 describes and explains the experimental analysis we have conducted for four MapReduce applications when they run on the Intel SCC. Our investigation is focused on the behavior of those applications for varying input sizes, HDFS cluster topologies and frequency settings for the cluster nodes.
Chapter 8 concludes the findings of this diploma thesis and presents suggestions for future work.

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing is emerging as a dominant computing platform for providing scalable online services to a global client base. Today's popular online services (e.g. web search, social networking and business analytics) are characterized by massive working sets, high degrees of parallelism and real-time constraints. These characteristics set scale-out applications apart from desktop (SPEC), parallel (PARSEC) and traditional commercial server applications.

In the context of digitalized information explosion, more and more businesses are analyzing massive amount of data - so called big data - with the goal of converting big data to "big value". Typical data analysis workloads include business intelligence, machine learning, bio-informatics and ad hoc analysis. The business potential of the data analysis applications in turn is a driving force behind the design of innovative data center systems, both hardware and software.

The explosion of accessible human generated information necessitates automatic analytical processing to cluster, classify and filter this information. The MapReduce paradigm has emerged as a popular approach to handling large-scale analysis, farming out requests to a cluster of nodes that first perform filtering and transformation of the data (map) and then aggregate the results (reduce).

This introductory chapter initially presents a synopsis of the concepts of distributed file systems and the MapReduce framework, that this thesis is going to deal with in the following chapters. Consecutively, it provides a high-level description of the Intel SCC architecture, the big data applications that have been ported on the Intel SCC and characterized and the Ganglia monitoring system, which has been leveraged so as to extract all the necessary metrics that enable us to track the performance of these applications. It concludes with the contribution that this thesis attempts to make in the fields of scale-out applications and manycore systems.

1.1 The Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) has been developed by Apache, as part of the Apache Hadoop Core project. It was initially built so as to provide infrastructure for the Apache Nutch Web Search Engine project. It was predominantly inspired by the Google File System (GFS), a proprietary distributed file system which was developed in Google Labs by Google. GFS was designed so as to provide efficient and reliable access to data using large clusters of commodity hardware. Therefore, HDFS and GFS share some common principles. The design and the implementation of HDFS is based on some key assumptions and goals.

First, since the filesystem consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines, it is guaranteed that some of the components are not functional for any given time and will not recover from their current failures. Therefore, since component failures are the norm rather than the exception, constant monitoring, error detection, fault tolerance and automatic recovery must be integral to the system.

Second, applications that run on HDFS have large datasets, meaning that a typical file in HDFS is gigabytes to terrabytes in size. Thus, HDFS is tuned to support large files. It provides high aggregate data bandwidth and scales to hundreds of nodes in a single cluster. It can potentially support tens of millions of files in a single cluster instance.

Third, the supported access pattern for the files that are stored in HDFS is mutation by appending new data rather than overwriting existing data or writing data at a random offset. This assumption greatly simplifies coherency issues and places the focus of performance optimization on the append operation. In addition, data reads are sequential in most cases.

Fourth, applications that run on HDFS need streaming access to their datasets. They are not general purpose applications that run on general purpose file systems. As a result, HDFS is designed for batch processing rather than interactive use by users. The emphasis is on high throughput of data accesses rather than low latency of data accesses. In order to achieve this functionality, several hard requirements that are imposed by POSIX and are not needed for applications that are targeted for HDFS have been traded to increase throughput rates.

Fifth, HDFS provides interfaces for applications to move themselves closer to where the data is located because of the fact that a computation requested by an application is much more efficient if it is executed near the data it operates on, especially when the size of the data is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running.

Sixth, HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

1.2 The MapReduce Framework

MapReduce is a software framework for easily writing applications which process vast amounts of data in parallel on large clusters of commodity hardware in a reliable, fault tolerant manner. The MapReduce framework that has been implemented by Apache, is designed to run on top of an HDFS cluster deployment. The datasets that are processed by MapReduce jobs can potentially scale to several terabytes. MapReduce jobs can utilize clusters that consist of hundreds or even thousands of nodes.

A MapReduce job usually splits the input data set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. The map tasks process key/value pairs to generate a set of intermediate key/value pairs and the reduce tasks merge all intermediate values associated with the same intermediate key, so as to produce the final key value pairs, which are the output of the MapReduce job.

The input and the output of a MapReduce job are stored in HDFS. This decision allows the framework to effectively schedule tasks on the nodes where the data is already present, resulting in very high aggregate bandwidth across the cluster.

The MapReduce framework takes care of the details of partitioning of the input data, scheduling the program's execution across a set of machines , handling machine failures and handling the required inter machine communication. Therefore, it provides a level of abstraction that hides the messy details of parallelization, fault tolerance, data distribution and load balancing, allowing programmers to express the simple computations that they are trying to perform.

1.3 Energy Inefficiencies of Hadoop Clusters

Typically, the energy efficiency of a cluster can be improved in two ways: by matching the number of active nodes to the current needs of the workload, placing the remaining nodes in low-power standby modes and by engineering the compute and storage features of each node to match its workload and avoid energy waste on oversized components. Unfortunately, MapReduce frameworks have many characteristics that complicate both options.

MapReduce frameworks implement a distributed data-store comprised of the disks in each node, which enables affordable storage for multi-petabyte datasets with good performance and reliability. Since high data availability is demanded, even idle nodes

remain powered on to ensure this requirement is met. Therefore, while significant periods of inactivity are observed, the need for data availability prohibits the shutting down of idle nodes and a significant amount of the power that is consumed is wasted on idle CPU cycles [5]. Figure 1.1 depicts the distribution of lengths of system inactivity periods across a cluster during a multi-job batch workload, comprised of several scans and sorts of 32 GB of data. Figure 1.2 shows the average CPU utilization across a cluster when sorting 128 GB of data.

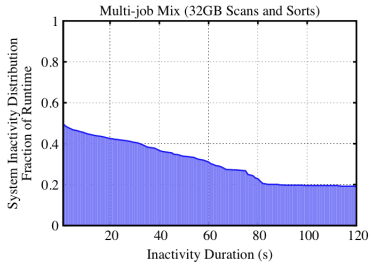


Figure 1.1: Distribution of the lengths of system inactivity periods, [5]

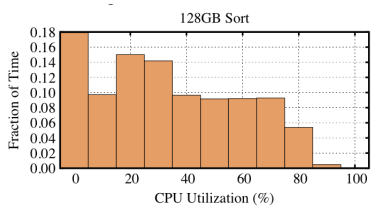


Figure 1.2: Average CPU utilization, [5]

As a consequence of the above, the energy efficiency of Hadoop clusters is an area to be searched. This diploma thesis makes an attempt to investigate performance and power consumption tradeoffs of MapReduce workloads. The Intel SCC provides a fine-grained power management API which enables us to perform frequency and voltage perturbations on subsets of cores of the SCC board. Later in this thesis, we utilize this power management API so as to explore power saving opportunities that can result from statically scaling down the frequency of nodes that are expected to have low CPU utilization during the execution of a MapReduce job.

1.4 The Intel SCC Manycore Platform

The Single-chip Cloud Computer is a 48-core Intel Architecture (IA) many-core exprerimental processor prototype. It is a research chip, which was built in Intel Labs so as to study many-core CPUs, their architectures and the techniques to program them. The Intel SCC was created as part of Intel's Tera Scale Computing Research Program, which is a worldwide effort to advance computing technology for the next decade and beyond. The program is investigating how to increase the performance and capabilities

of current computers.

The research regarding the Intel SCC has the following goals:

- ⋆ To demonstrate a shared memory message-passing architecture for a large number of cores and to experiment with its programmability and scalability.
- ⋆ To design and explore the performance and power characteristics of an on-die 2D mesh fabric.
- ⋆ To explore benefits and costs of software-controlled dynamic voltage and frequency scaling for multiple cores.

The SCC is the second generation processor design that resulted from Intel's Tera-Scale research. The first was Intel's Teraflops Research Chip; it had 80 non-IA cores. The second is the SCC; it has 24 tiles and two cores per tile. The SCC core is a full IA P54C core and hence can support the compilers and OS technology required for full application programming. Figure 1.3 shows a stylized view of the SCC chip. The 24 tiles of the SCC board are arranged in a XxY = 6x4 array. There is a router associated with each tile. The tiles are connected by a fully synchronous mesh fabric with rigorous performance and power requirements. The SCC has multiple voltage and frequency domains, some configurable at startup, others that may be dynamically varied for application-controlled fine grained dynamic power and performance management. The SCC has four on-die memory controllers capable of addressing a total of up to 64 GB of external memory. It also has a small amount of fast local memory located in each tile. Message-passing support is provided that uses shared regions of local memory or off-die main memory. The SCC has a new memory type and a new processor cache instruction to facilitate memory management.

The entire system is controlled by a Board Management Microcontroller (BMC) that initializes and shuts down critical system functions. It is commonly connected by a PCI-express cable to a PC acting as a Management Console (MCPC). The Management Console is a 64-bit PC running some version of Linux. Intel Labs provides software that runs on the Management Console to manage the SCC chip. Key features of this software are the ability to load a Linux image on each core or a subset of cores, to read and modify SCC configuration registers and to load programs on the SCC cores. Running Linux on the SCC cores is the most common configuration, but it is not mandatory.

1.5 The Benchmark Suites

This diploma thesis utilizes benchmarks that have been introduced as part of the Cloudsuite and DCBench benchmark suits. Both Cloudsuite and DCBench employ machine learning algorithms implementations that are included in Apache Mahout. Apache Mahout is a scalable machine learning library which is implemented on top of distributed systems. The Apache Mahout version we use includes implementations

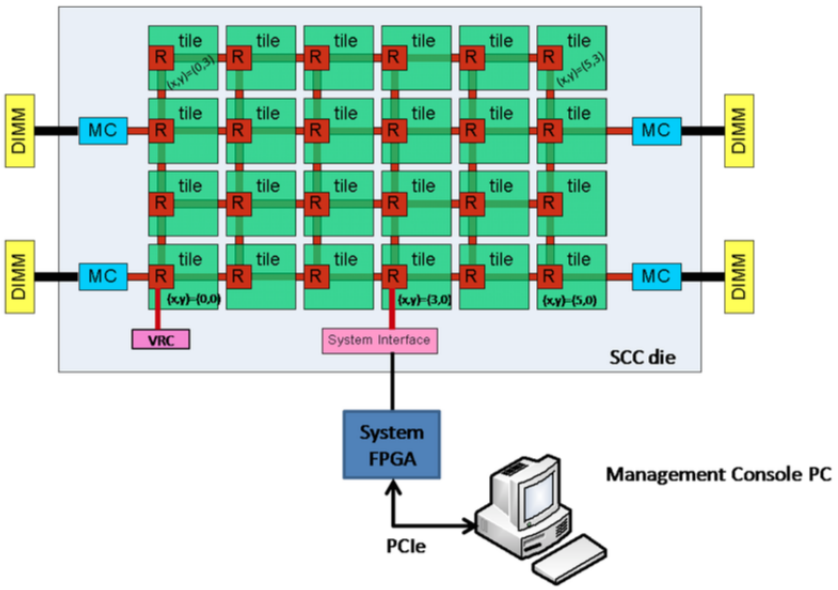


Figure 1.3: Intel SCC Top-Level Architecture

of core machine learning algorithms such as clustering, classification and collaborative filtering using the MapReduce framework, on top of an HDFS deployment.

Cloudsuite is a benchmark suite for emerging scale-out applications. It consists of eight applications that have been selected based on the popularity in today's datacenters. The benchmarks are based on real-world software stacks and represent real world setups. The application categories that are covered by Cloudsuite are Data Analytics, Data Serving, Data Caching, Graph Analytics, Media Streaming, Software Testing, Web Search and Web Serving. In this diploma thesis, we examine the data analytics benchmark, which provides an implementation of the Bayesian Classification algorithm using the MapReduce framework, which is derived from Apache Mahout.

DC Bench is a benchmark suite for representative workloads that are found in modern datacenters. DC Bench offers implementations of applications that are based on diverse programming models which run in large distributed environments employing state-of-art techniques. DC Bench offers implementations for the following datacenter workloads : Base Operations (e.g. Wordcount), Classification, Clustering, Recommendation, Association Rule Mining, Segmentation, Warehouse Operations, Feature Reduction, Vector Calculation, Graph Mining, Services and Interactive Real-Time Applications. Those applications are implemented using either the MapReduce or the MPI paradigm. In these diploma thesis, we examine three benchmarks that are provided by DC Bench, which are Wordcount, K-Means Clustering and Frequent Pattern Growth, that belong to the Base Operations, Clustering and Association Rule Mining categories respectively. The implementation for K-Means Clustering and Frequent Pattern Growth are provided by the Apache Mahout library, whereas Wordcount is derived from the Hadoop Examples that accompany every HDFS installation.

1.6 The Ganglia Monitoring System

In order to capture critical per core metrics such as CPU utilization and network traffic, we utilize the monitoring infrastructure that is provided by Ganglia, which is a tool for large cluster monitoring. Ganglia is architecturally composed by three daemons : gmond, gmetad and gweb. Operationally, each daemon is self-contained, but all three are architecturally cooperative (Figure 1.4).

gmond is responsible for collecting the metrics that are specified in each configuration file in each host of the cluster. gmond instances share with each other the state of the node they reside on, so that each gmond instance knows the current value of every metric recorded by every other node in the Ganglia cluster. This communication takes place with UDP datagrams, through either multicast or unicast channels. An XML-format dump of the entire cluster can be requested from a remote poller from any single node in the cluster running gmond, on port 8649.

gmetad periodically polls gmond nodes and stores the metrics that it receives in round-robin databases using RRDtool. Since each gmond node that is polled provides the values for all the metrics that are collected in the entire cluster, gmetad needs to poll only one gmond node per gmond cluster.

gweb is Ganglia's frontend visualization UI. gweb is implemented in PHP and exposes the data that is stored in the RRD databases by gmetad. It typically runs under the Apache Web Server. gweb gives easy and instant access to any metric from any host in the network. It graphically summarizes the grid using graphs that combine metrics by cluster and provides sane click-throughs for increased specificity.

1.7 The Contribution of this Diploma Thesis

This diploma thesis provides a detailed description of the tools that have been used and developed so as to build Hadoop Clusters on the Intel SCC, with respect to technical problems that have been encountered and physical limitations that the platform introduces. In addition, it presents a run-time monitoring framework for the Intel SCC, which enables us to capture per-core metrics such as CPU utilization and network traffic as well as aggregate metrics for the entire SCC board such as power consumption and board temperature. This infrastructure is subsequently utilized so as to characterize four MapReduce applications (Wordcount, Bayes Classification, K-Means Clustering and Frequent Pattern Growth) in terms of performance, scalability and power consumption when they run on SCC hardware.

The behavior of these applications is studied for varying input sizes so as to explore the scalability of those applications in terms of input size when they run on the SCC platform. This approach enables us to determine the input size that scalability breaks for each application, thus the completion of a MapReduce job is impossible and to

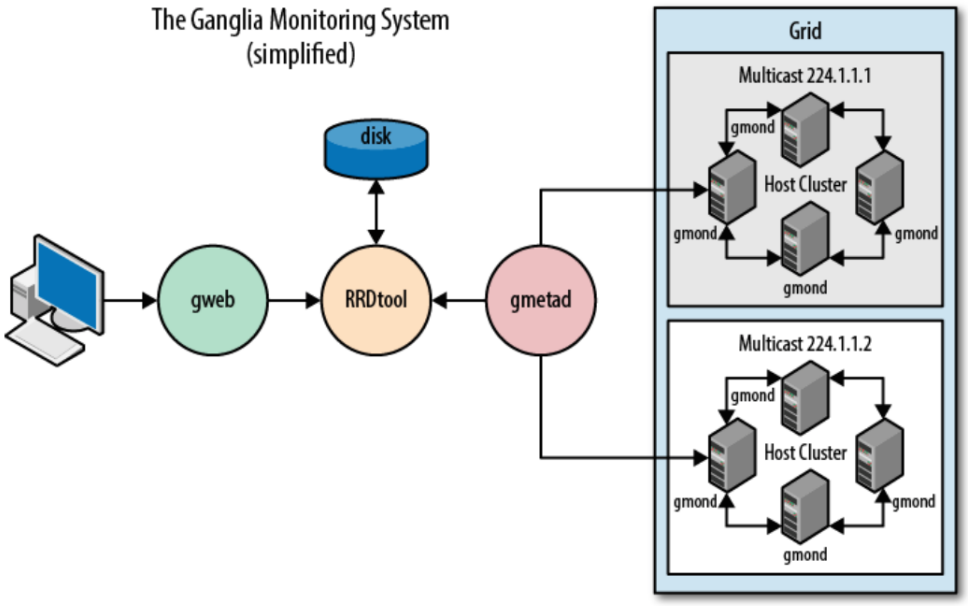


Figure 1.4: Ganglia Architecture

correlate this information with physical limitations of the SCC platform.

In addition, the performance of these applications is examined for diverse cluster deployments that utilize different amount of SCC cores, resulting in different cluster topologies on the SCC board. For cluster topologies that leverage only some of the cores of the SCC board, the remaining cores operate at the minimum frequency, so as to avoid power being wasted on idle cycles. This strategy gives us the opportunity to draw conclusions regarding the scalability of those applications in terms of the number of cores they employ, so as to define the point at which phenomena such as network congestion or I/O bandwidth saturation become bottlenecks, resulting in suboptimal performance and utilization of the on-die resources.

Furthermore, performance and power consumption tradeoffs are investigated so as to spot possible power and energy saving opportunities that those applications might conceal. This goal is achieved by configuring different groups of cores (frequency islands) to run at different frequencies. The target of our is investigation is to realize if statically scaling down the frequency of cores that are expected to have low CPU utilization throughout the execution of a MapReduce job significantly degrades performance, thus canceling the benefits of power consumption saving.

Chapter 2

Related Work

This chapter presents a detailed analysis and presentation of recent research findings related to the fields that this diploma thesis intends to cover. It is divided into two sections. In the first section findings regarding the field of scale-out workloads are presented. In the second section, research results in the field of performance and power monitoring of the Intel SCC are described.

2.1 Scale-Out Workloads

Cloud computing is emerging as a dominant computing platform for delivering scalable online services to a global client base. Today's popular online services, such as web search, social networks and video sharing are all hosted in large data centers. With the industry rapidly expanding, service providers are building new data centers, augmenting the existing infrastructure to meet the increased demand. However, while demand for cloud infrastructure continues to grow, the semiconductor manufacturing industry has reached the physical limits of voltage scaling [12, 13], no longer able to reduce power consumption or increase power density in new chips. Physical constraints have therefore become the dominant limiting factor for data centers, because their sheer size and electrical power demands cannot be met.

Recognizing the physical constraints that stand in the way of further growth, cloud providers now optimize their data centers for compute density and power consumption. Cloud providers have already began building server systems specifically targeting cloud data centers, improving compute density and energy efficiency by using high-efficiency power supplies and removing unnecessary board-level components such as audio and graphics chips [14, 15].

Today's volume servers are designed with processors that are essentially general-purpose. These *conventional* processors combine a handful of aggressively speculative and high clock frequency cores supplemented by a large on-chip cache. Recently, *tiled* processors have emerged as competition to volume processors in the scale-out server space [16]. Recognizing the importance of per-server throughput, these processors use a

large number of relatively simple cores, each with a slice of the shared LLC, interconnected via a packet-based mesh interconnect. Lower-complexity cores are more efficient than those in conventional designs [17]. Additionally, the many-core architecture improves throughput compared to conventional chips and memory and I/O bound scale out workloads. Despite the differences in the chip-level organization, the technology scaling trends of tiled processors are similar to conventional designs; each technology generation affords more tiles, which increases the core count, cache capacity and interconnect resources.

In the context of processors for scale-out applications, both architectures make suboptimal use of the die area. As recent research examining scale-out [10] and traditional server workloads [18] has demonstrated, large caches, such as those found both in conventional and tiled designs, are inefficient due to limited reuse at the LLC resulting from vast data footprints of these applications. In fact, large LLC configurations have been shown to be detrimental to performance, as they increase the fetch latency of performance-critical instructions whose footprint exceeds the capacity of first-level caches. Moreover, recent work has identified significant over-provisioning in conventional server chip's core capabilities, on-die interconnect, and memory bandwidth.

Micro-architectural studies of scale-out workloads have proved a large mismatch between the demands of the scale-out workloads and today's predominant processor micro-architecture [10]. It has been demonstrated that:

- ⋆ **Scale-out workloads suffer from high instruction-cache miss rates.** Instruction caches and associated next-line prefetchers found in modern processors are inadequate for scale-out workloads (Figure 2.1).

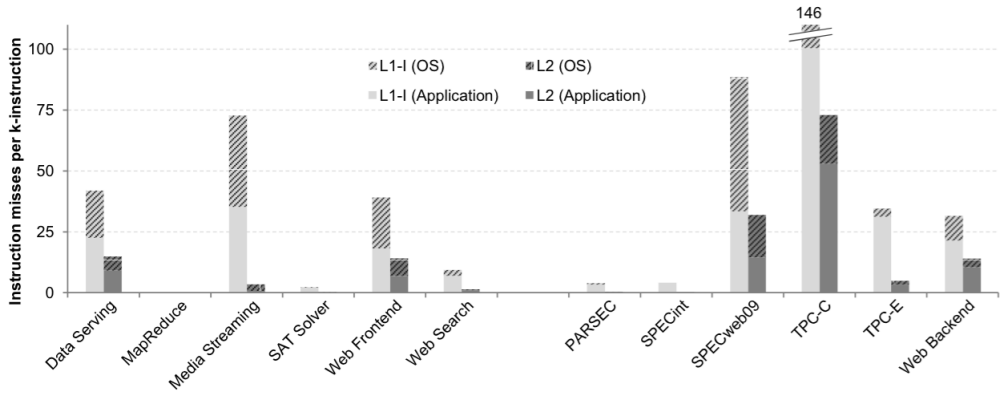


Figure 2.1: L1-I and L2 instruction cache miss rates for scale-out workloads compared to traditional benchmarks, [10]

- ⋆ **Instruction- and memory-level parallelism in scale-out workloads is low**. Modern aggressive out-of-order cores are excessively complex, consuming power and on-chip area without providing performance benefits to scale-out workloads (Figure 2.2).

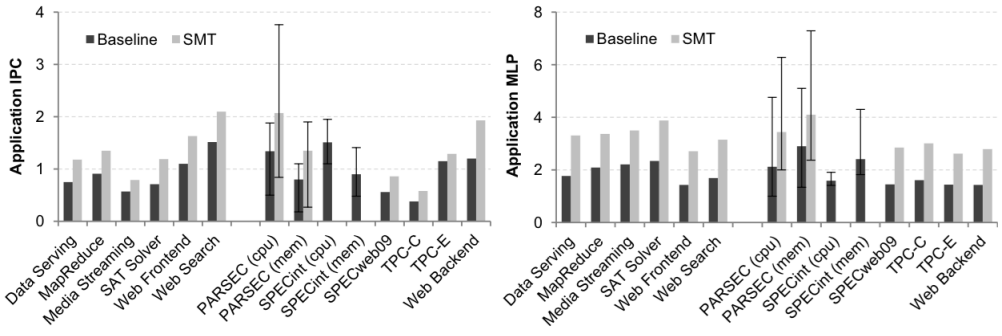


Figure 2.2: Instruction and Memory - Level Parallelism for scale-out workloads compared to traditional benchmarks, [10]

⋆ **Data working sets of scale-out workloads considerably exceed the capacity of on-chip caches.** Processor real-estate and power are misspent on large last-level caches that do not contribute to improved scale-out workloads performance (Figure 2.3).

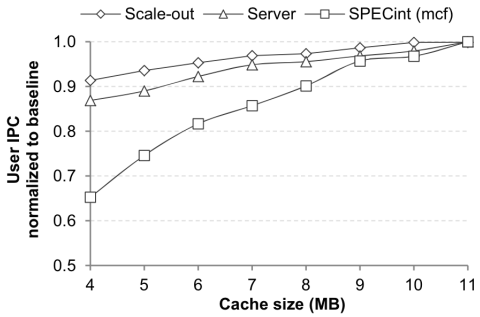


Figure 2.3: Performance sensitivity to LLC capacity for scale-out workloads, [10]

⋆ **On-chip and off-chip bandwidth requirements of scale-out workloads are low.** Scale-out workloads see no benefit from fine-grained coherence and high memory and core-to-core communication bandwidth (Figure 2.4).

Based on those findings, methodologies for designing scalable and efficient scale-out processors have been proposed [11]. Those studies have verified that smaller caches than can capture the dynamic instruction footprint of scale-out workloads, afford more die area for the cores, without penalizing per core performance. Moreover, it has been demonstrated that while the simpler cores found in tiled designs are more effective than conventional server cores for scale-out workloads, the latency incurred by the on-chip interconnect in tiled organizations lowers performance and limits the benefits of integration, as additional tiles result in more network hops and longer delays.

Performance Density, defined as throughput per unit area is used to quantify how effectively an architecture uses the silicon real-estate. Proposed design methodologies

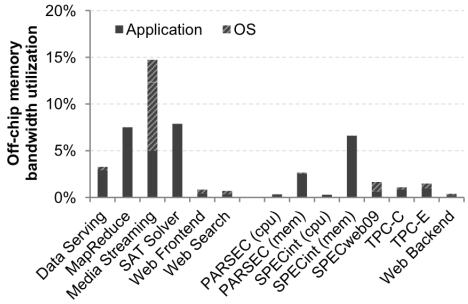


Figure 2.4: Average off-chip memory bandwidth utilization for scale-out workloads, [10]

[11] derive a performance density optimal processor building block called a *pod*, which tightly couples a number of cores to a small LLC via a fast interconnect. As technology scales to allow more on-chip cores, those methodologies calls for keeping the design of the pod unchanged, replicating the pod to use up the available die area and power budget. A key aspect of the Proposed architecture is that pods are stand-alone servers, with no inter-pod connectivity or coherence.

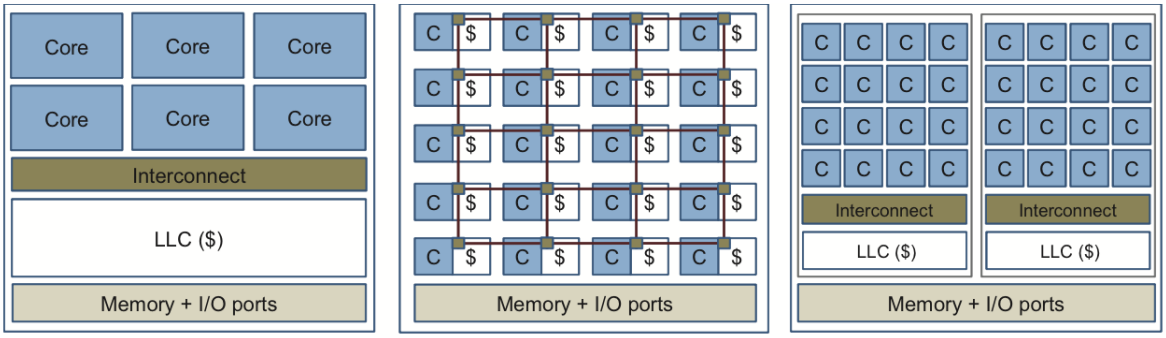


Figure 2.5: Comparison of Conventional, Tiled and Scale-Out architectures, [11]

With the use of analytic models and cycle-accurate full-system simulation of a diverse suite of representative scale-out workloads, it is demonstrated that:

- ⋆ The core and cache area budget of conventional server processors is misallocated, resulting in a performance density gap of 3.4× to 6.5× against an optimally-efficient processor.
- ⋆ The distributed cache architecture in tiled designs increases access latencies and lowers performance, as manifested in a performance density gap of 1.5× to 1.9× versus an optimally-efficient processor.
- ⋆ Performance density can be used to derive an optimally efficient pod that uses a small (i.e. 2-4 MB) last-level cache and benefits from a high core-to-cache area ratio and simple crossbar interconnect.

- ⋆ Replicating pods to fill the die results in an optimally-efficient processor which maximizes throughput and provides scalability across technology generations. For example, in the 20 nm technology, scale-out processors improve performance density by 1.5× - 6.5× over alternative organizations.

Efforts have also been made towards improving the energy efficiency of MapReduce frameworks like Hadoop [5]. It has been shown that Hadoop has the global knowledge necessary to manage the transition of nodes to and from low-power modes. Hence, Hadoop should be, or cooperate with, the energy controller for a cluster. It has also been shown that it is possible to recast the data layout and task distribution of Hadoop to enable significant portions of a cluster to be powered down while still fully operational. Energy can be conserved at the expense of performance, as there is a trade-off between these two.

In order to enable the disabling of storage nodes without affecting data availability, a new invariant has been proposed for use during block replication: at least one replica of a data-block should be stored in a subset of nodes referred as the *covering subset*. The premise behind a covering subset is that it contains a sufficient set of nodes to ensure the immediate availability of data, even were all nodes not in the covering subset to be disabled. The experimental evaluation of this proposition has clearly demonstrated that while data availability is preserved, energy savings come with a deleterious impact on performance. However, it is argued that nodes tend to contribute less to performance than they do in energy consumption. Figure 2.6 depicts the performance and energy consumption trade-offs that are observed while running web data sort and web data scan MapReduce jobs on a 36-node cluster with a covering subset of 9 nodes.

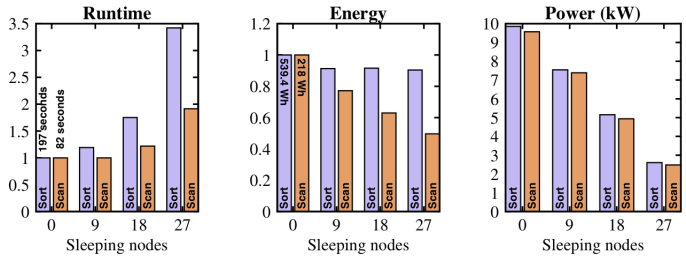


Figure 2.6: Runtime, Energy Consumption and Average Power Consumption for the 32 GB Sort and 32 GB Scan workloads as nodes are disabled, [5]

This diploma thesis attempts to explore the behavior of scale-out workloads, which have been implemented using the MapReduce framework, when they are executed on the Intel SCC, which is many core platform, integrated on a single chip, as opposed to the traditional cluster topology organization.

2.2 Performance Analysis and Power Consumption Monitoring on the Intel SCC

There has been a continuous change over the past years in CPU design and development towards both power-aware hardware architectures as well as many-core processors. The Intel SCC is a highly configurable many-core chip that provides unique opportunities to optimize run time, communication and memory access as well as power and energy consumption of parallel programs.

Significant efforts have been made to analyze and characterize the performance behavior of the chip under various power settings, mappings of processes to cores and memory controllers as well as different techniques for data exchange between cores through benchmarking [19]. Conclusions from those studies have shown that:

- ⋆ Data exchange based on shared memory is slower compared to using a message passing scheme, which utilizes the on-chip SRAM of the Intel SCC, called the Message Passing Buffer. The performance advantage of communication using message passing compared to shared memory varies from 3.26× to 9.06×. The reason for the comparatively low performance of shared memory communication lies in the time required to copy the data between private and shared memory at the sender and the receiver core. In addition, communication time increases linearly with the number of cores that are involved regardless of the size of the message that is sent. Figure 2.7 shows the communication time required to complete a broadcast operation using shared memory and message passing data exchange for various data sizes with increasing number of cores.

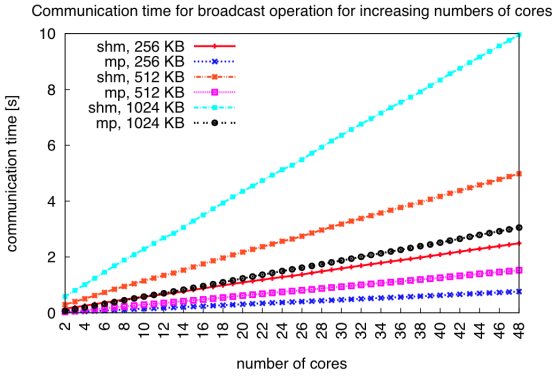


Figure 2.7: Communication time required to complete a broadcast operation using shared memory and message passing data exchange, [19]

- ⋆ Contrary to popular belief, lowest energy consumption is not achieved for the fastest execution time but rather for a medium frequency-voltage setting, depending on the program being executed. Figure 2.8 depicts the experimental results that where derived from the execution and run-time monitoring of the

BT and *LU NAS parallel benchmarks*, *NAS problem classes A and B* and *Gadget2* simulator in terms of overall energy consumption [23, 24]. It is evident that lowest energy consumption is reached for a core frequency of 400 or 320 MHz, depending on the application. As a consequence, the benefits of core clock frequency and voltage scaling depend on the actual program executed and whether it is computationally-bound.

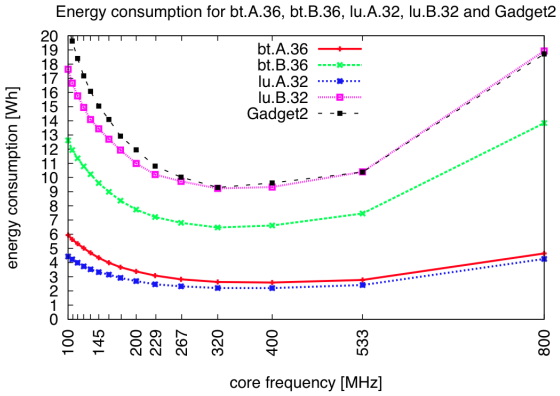


Figure 2.8: Energy consumption for all possible core clock frequencies, [19]

⋆ In order to improve the memory access behavior it is more beneficial to increase the clock frequency of both, mesh network and memory controllers, compared to just increasing the clock of one of the two entities. Furthermore, parallel memory access that involves all cores shows performance degradation of up to 14.2% compared to serial memory access. The tendency is a lower performance for higher distances between cores and their memory controllers, as well as for a higher number of cores accessing the memory controller simultaneously. Figure 2.9 shows the memory bandwith observed when running the Stream Benchmark, which is a well known memory-intensive benchmark [25] on the Intel SCC on a single core, for varying core distances from the corresponding memory controller. Figure 2.9 presents the memory bandwidth degredation that occurs when the benchmark is executed concurrently in more than one cores, resulting in an increasing number of cores accessing the same memory controller simultaneously.

Intel provides a customized programming library for the SCC, called RCCE, that allows for fast message passing between the cores. RCCE operates on an application programming interface (API) with techniques based on the well-established message passing interface (MPI). The use of MPI in a large many-core system is expected to change the performance-power trends considerably compared to today's commercial multi-core systems. Furst and Coskun in [20] develop a system monitoring software and benchmarks specifically targeted at investigation the impact of message passing on the performance and the power consumption of the Intel SCC.

This experimental evaluation that is offered by this study is based on the execution

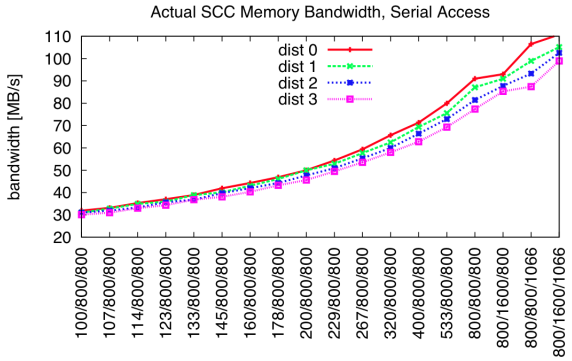


Figure 2.9: Single core memory bandwidth for all distance and frequency possibilities, [19]

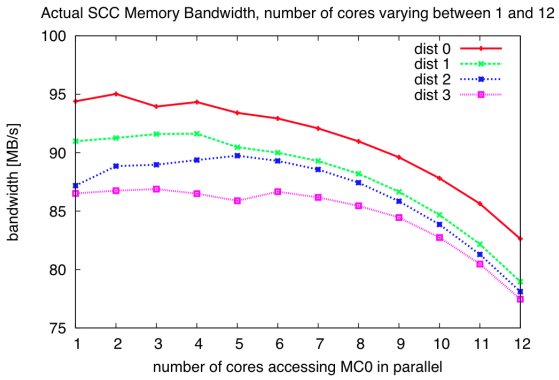


Figure 2.10: Memory bandwidth degradation for increasing numbers of cores acessing the same memory controller, [19]

and performance monitoring of the *Share*, *Shift*, *Stencil*, *Pingpong* and *Bcast* benchmarks which are provided by Intel. The main conclusions that have been drawn are the following:

- ⋆ High IPC workloads suffer from execution time increase, as the distance of the communicating cores grows. Figure 2.11 depicts the execution time and average IPC of the *Stencil* benchmark when it runs on a pair of cores, as the distance between those cores increases. *Stencil* is an application that is characterized by its high IPC.

- ⋆ Memory intensive applications present significant delays as the number of cores that are concurrently executing them increases, due to memory contention. Figure 2.12 presents the execution time and average IPC of the *Share* benchmark with local communication as the number of pairs that are executed concurrently increases. *Share* is known to present memory intensive behavior.

- ⋆ Applications that heavily utilize broadcast messages suffer from significant execution time delays, after the number of the cores participating in the broadcast

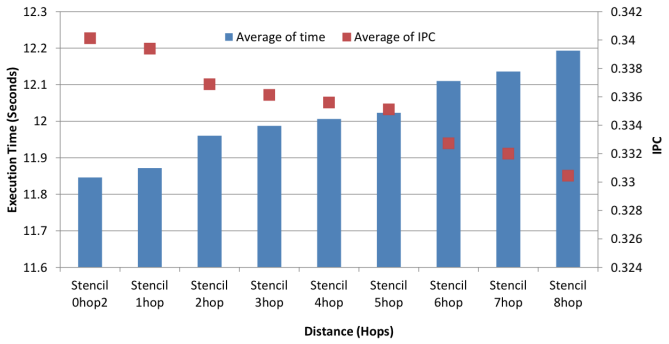


Figure 2.11: Execution time and average IPC for *Stencil* as distance between cores increases, [20]

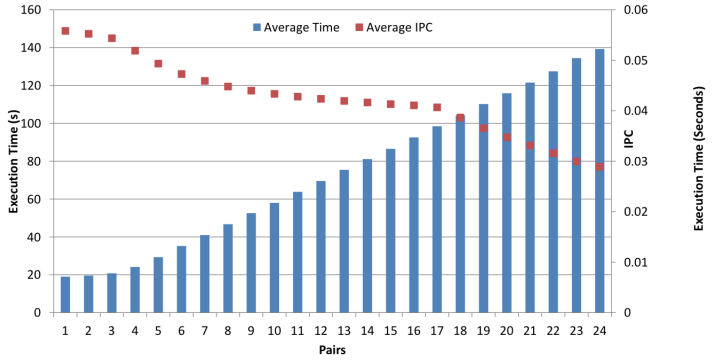


Figure 2.12: Execution time and average IPC for *Share* as number of pairs executed concurrently increases, [20]

increases beyond a certain count. Figure 2.13 shows the execution time and the average IPC observed at the execution of the *Bcast* benchmark, as the core count participating in the broadcast increases. Clearly, for core counts greater than 8, network contention becomes a bottleneck causing an overall performance drop. *Bcast* is evidently a network intensive application because of the significant number of messages it sends and receives.

⋆ Applications characterized by a high number of memory accesses but low IPC tend to present low power consumption. On the contrary, applications with high IPC consume more power. Figure 2.14 presents a comparison of the IPC and power consumption between *Share*, *Shift*, *Sencil* and *Pingpong* when they are executed with local communication, using all 24 pairs of cores. Evidently, power consumption is highly correlated with the applications IPC. Moreover, memory intensive applications do not benefit from running on a larger number of cores, since high delays because of memory contention keep the execution time and thus the overall energy consumption high. However, high IPC applications can greatly benefit of an increased core count, since the execution time drop compensates for the increased power consumption, leading in an overall energy consumption

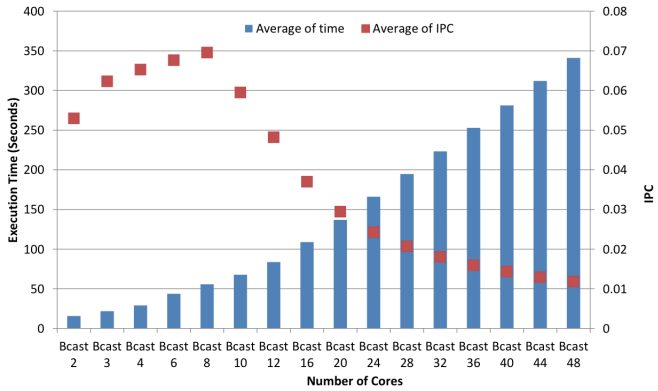


Figure 2.13: Execution time and average IPC for *Bcast* with respect to number of cores, [20]

reduction.

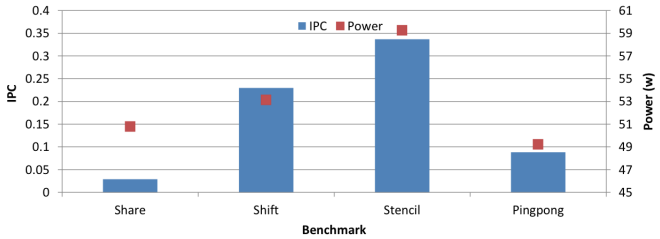


Figure 2.14: Power consumption and IPC for *Share*, *Shift*, *Stencil* and *Pingpong*, [20]

Efficient broadcasting is essential for good performance on distributed or multiprocessor systems. RCCE implements broadcasting in a traditional way: sending $n-1$ unicast messages, where n is the number of cores participating in the broadcast. This implementation hinders performance as the number of cores participating in the broadcast increases and the data being sent to each core is large. In addition, in the RCCE implementation the broadcasting core is blocked from doing any useful work until all cores receive the broadcast.

Matienzo and Jerger in [21] explore several broadcasting schemes that take advantage of the resources of the SCC and the RCCE library. Their best broadcast scheme achieves a 35× speedup over the RCCE implementation. They also demonstrate that this broadcasting scheme significantly reduces the time spent in communication in some benchmarks. This study presents two approaches towards implementing more efficient broadcast schemes. The first approach is to utilize cores that have already received the broadcast. The original broadcasting core is responsible for only sending the message to a few processors and the cores that have received the message are responsible for forwarding the message to the other cores, which happens in parallel. The second strategy is to utilize concurrent accesses to a specific memory location, which contains

the message to be broadcasted. Figure 2.15 depicts the average broadcast latency that is observed of the broadcast schemes that were implemented for increasing message size. It is evident that all of them but one outperform the RCCE implementation.

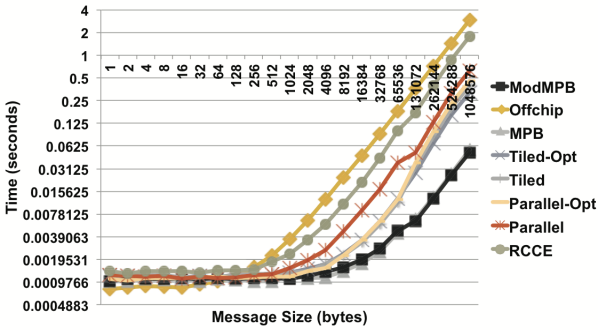


Figure 2.15: Broadcast latency for varying message size, [21]

Dynamic frequency and voltage scaling (DVFS) techniques have been widely used for meeting energy constraints. Single-chip many-core systems bring new challenges owing to the large number of operating points and the shift to message passing interface from shared memory communication. Bartolini et al. evaluate the impact of frequency scaling on the performance and power of many-core systems with MPI at [22]. They provide an extensive analysis quantifying the effects of frequency perturbations on performance and energy efficiency. Their experimental results show that run-time communication patterns lead to significant differences in power/performance trade-offs in many-core systems with MPI.

In this study, performance aspects of the execution of *Share*, *Shift*, *Stencil*, *Pingpong*, *NPB* and *Bcast* applications on the SCC are measured. Those applications are deployed on pairs of cores for different numbers of hops between them. Each cores runs at either 533 MHz or 166 MHz, so as to measure the impact of frequency scaling. The metrics that are extracted from these experiments are the execution time, the chip power and energy consumption, the instructions per second and the message density. It is verified that memory intensive benchmarks with low IPS such as *Pingpong* and *Share* have lower sensitivity to scaling down of frequency, compared to high IPS, CPU intensive or network intensive benchmarks such as *Shift* and *Stencil*. All applications benefit from both cores running at the same frequency since they are based on bidirectional communication, apart from Bcast, whose communication pattern is unidirectional. Figure 2.16 summarizes those findings.

The metrics that are collected by those experiments are used so as to train neural networks who attempt to predict the execution time of an application for a given frequency configuration. It is verified that message density and frequency of communication significantly improve the accuracy of those predictors. As a result, communication patterns and message densities should be included in DVFS performance optimization

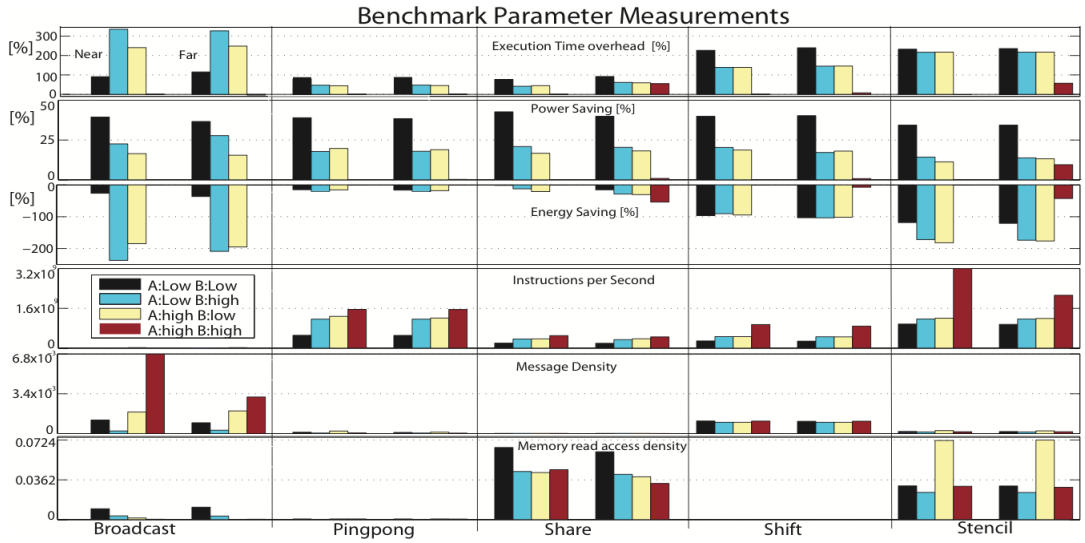


Figure 2.16: Sensitivity of the Intel SCC benchmarks to frequency scaling, [22]

on many-core systems with MPI.

This diploma thesis expands the research that has been carried out on the Intel SCC regarding performance and power consumption analysis to the field of scale-out workloads, which are characterized by significantly different runtime behavior, compared to traditional PARSEC benchmarks.

Chapter 3

The Intel SCC Architecture

This chapter provides a detailed description of the functional units of the Intel Single-chip Cloud Computer.

3.1 The SCC Core Layout

The Intel SCC consists of 48 cores, which are organized in 24 tiles. The 24 tiles are connected through a Network on a Chip (NoC) with each other and with other functional units of the platform, such as the memory controllers. There are three IDs associated with each core, the processor ID, the tile ID and the core ID. Figure 3.1 depicts the way the 48 cores are laid on the SCC board, with their tile ID (blue), processor ID (red) and (x, y) coordinates. The tile ID of each core is calculated from its coordinates as `0xyx`. The decimal number of the tile ID is thus $16 * y + x$. The core ID of each core is either 0 or 1 and identifies the core within the boundaries of a specific tile. The processor ID is equal to $tile\_id * 2 + core\_id$. Each core is also identified by a unique hostname, which is the concatenation of the word '`rck`' and the processor ID. As a consequence, core hostnames range from `rck00` to `rck47`.

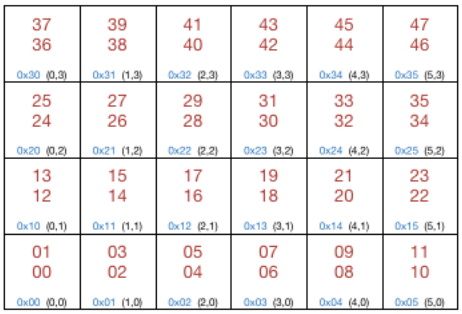


Figure 3.1: Intel SCC Core Layout

3.2 The SCC Tile

Figure 3.2 shows an overview of an individual tile. In this section, the functional units that are part of each tile are described.

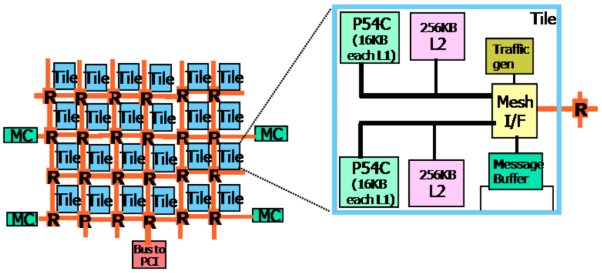


Figure 3.2: Intel SCC Tile Overview

3.2.1 P54C IA Core

The core is a P54C Pentium design that has been altered to increase the L1 data and instruction cache size to 16 KB each. These caches are 4-way set associative with pseudo-LRU replacement policy. Additionally, the original front side bus-to-cache controller interface (M-unit) has been integrated into the core. The P54C ISA (instruction set architecture) was extended with a new instruction (`CL1INVMB`) and a new memory type (MPBT) introduced to facilitate the use of message data. All accesses to MPBT data bypass the L2 cache. The new instruction was added to invalidate all L1 cache lines typed as MPBT. These changes were added to facilitate maintaining coherency between caches and message data. Finally, a write combine buffer was added to the M-unit to accelerate the message transfer between cores.

3.2.2 L2 Cache

Each core has its own private 256 KB L2 cache and an associated controller. During a miss, the cache controller sends the address to the Mesh Interface Unit (MIU) for decoding and retrieval. Each core can only have one outstanding memory request and will stall on missed reads until data are returned. On missed writes, the processor will continue operation until another miss of either type occurs. Once the data has arrived, the processor continues normal operation. Tiles with multiple outstanding requests can be supported by the network and memory system. The L2 cache is a 4-way set associative with a pseudo-LRU replacement policy. It is write-back only. It is not write-allocate.

3.2.3 Message Passing Buffer (MPB)

In addition to the traditional cache structures, a message passing buffer (MPB) capable of fast R/W operations has been added to each tile. This 16 KB on-chip SRAM buffer provides the equivalent of 512 full cache lines of memory. Any core or the system interface can write or read data from these 24 on-die message buffers. One of the intended uses of the MPB is message passing.

3.2.4 DDR3 Memory Controllers

The four memory controllers provide a maximum capacity of 64 GB of DDR3 memory. The Intel SCC we have used for our experiments is configured with a total of 32 GB of system memory, as stated in section 3.4.1. This memory physically exists on the SCC board. Each memory controller supports two unbuffered DIMMs per channel with two ranks per DIMM. The supported DRAM type is DDR3-800 x8 with 1 GB, 2 GB or 4 GB capacity, leading up to 16 GB capacity per channel. The DDR3 protocol includes automatic training, calibration and compensation as well as periodic refresh of the DRAM. Memory accesses are processed in order, while accesses to different banks and ranks are interleaved to improve throughput. The memory controllers can either operate at 800 MHz or 1066 MHz. The memory controllers' frequency is determined during platform initialization and cannot be changed during normal operation.

3.2.5 Look Up Table (LUT)

Each core has a lookup table (LUT) which is a set of configuration registers that map the core's physical addresses to the extended memory map of the system. Each LUT contains 256 entries, one for each 16 MB segment of the cores 4 GB physical memory address space. Each entry can point to any memory location (private memory, message passing buffer, configuration registers, system interface, SCC power controller or system memory). On an L2 cache miss, the MIU looks through the LUT to determine where the memory request should be sent.

3.2.6 Mesh Interface Unit (MIU)

The Mesh Interface Unit (MIU) contains the following:

- ⋆ Packetizer and De-Packetizer
- ⋆ Command interpretation and address decode/lookup
- ⋆ Local configuration registers
- ⋆ Link level flow control and Credit Management
- ⋆ Arbiter

The packetizer/depacketizer translates the data to/from the agents and to/from the mesh. The Data, Command and Address Buffers provide queuing for flit organization. Specifically, the MIU takes a cache miss and decodes the address, using the LUT to map from the core address to system address. It then places the request to the appropriate queue. The queues are the following:

- ⋆ Router → DDR3 request
- ⋆ Message Passing Buffer access
- ⋆ Local Configuration Register access

For traffic coming from the router, the MIU routes the data to the appropriate local destination. The link level flow control ensures flow of data on the mesh using a credit-based protocol. Finally, the arbiter controls tile element access to the MIU at any given time via a round robin scheme.

The tile configuration registers provide a method for applications to control the operating modes of various tile hardware elements. Table 3.1 presents the configuration registers of the MIU and they desired operations that they are designed for. Each register is mapped to the core address space through the LUT and can be referenced using memory-mapped I/O.

The Tile ID register contains the tile's (x,y) coordinates. The Core Configuration registers are dedicated to each core of the tile and are writable by each core and the System Interface unit. The GCU configuration register is dedicated to the global clocking unit and is writable by all cores and the System Interface as well. The test-and-set registers enable communication protocols (such as message passing) in a multi-processor environment. The LUT registers contain the LUT entries of each core. The L2 Cache Configuration registers controls the sleep and power behavior of the L2 cache. The Sensor Registers allows enabling and checking the thermal sensors in the core.

3.2.7 Traffic Generator

The traffic generator is a unit used to test the performance capabilities of the mesh by injecting and checking traffic patterns and is not used in normal operation.

3.3 The SCC Mesh

The on-die 2D mesh network has 24 packet-switched routers connected in a 6×4 configuration and is on its own power supply and clock source. This enables power-performance tradeoffs to ensure that the mesh is delivering the required performance while consuming minimal power. The SCC Mesh can either operate at 800 MHz or 1.6 GHz. The mesh frequency is determined during platform initialization and cannot be changed during normal operation.

Register Name	Desired Operation	Register Offset	Valid Data Bits
LUT register core 1 (256 8-byte entries)	Read/write LUT1	0x1000	22
LUT register core 0 (256 8-byte entries)	Read/write LUT0	0x0800	22
Atomic Flag Core1 LOCK1	Read/write test-and-set Core 1 atomic	0x0400	1
Atomic Flag Core0 LOCK0	Read/write test-and-set Core 0 atomic	0x200	1
Tile ID register MYTILEID	Read Tile ID	0x0100	11
Global Clock Unit (GCU) GCBCFG	Read/write GCU	0x0080	26
Sensor Register	Read Thermal Sensor value	0x0048	26
Sensor Register SENSOR	Read/write Thermal Sensor control	0x0040	14
L2 Cache Configuration 0 L2CFG0	Read/write L2 Cache 0	0x0020	14
L2 Cache Configuration 1 L2CFG1	Read/write L2 Cache 1	0x0028	14
Core Configuration 0 GLCFG0	Read/write Core 0 config	0x0010	26 (top 14 read only)
Core Configuration 1 GLCFG1	Read/write Core 1 config	0x0018	26 (top 14 read only)

Table 3.1: Intel SCC Configuration Registers

3.3.1 Router (RXB)

The RXB is the next generation router for future many-core 2D mesh fabrics. It has the following design targets:

- ⋆ Wide Links : 16 B data + 2 B side band
- ⋆ High Frequency : 2 GHz @ 1.1 V P1266
- ⋆ Low Latency : No load latency = 4 cycles including link traversal
- ⋆ Multiple Message Classes : Two Message Classes 1 Request (Message class 0) + 1 Response (Message class 1)
- ⋆ Multiple Virtual Channels (VCs): 1 VC reserved per Message Class (VC6 for request and VC7 for response), six VCs in free pool for a total point of eight VCs
- ⋆ Dynamic Power Management : sleep, clock gating, voltage control etc.

3.3.2 Packet Structure and Flit Types

The different agents of the mesh fabric communicate with each other at packet granularity. A packet consists of a single flit or multiple flits (up to three) with header, body and tail flits. Control flits are used to communicate control information such as credits.

3.3.3 Flow Control in SCC

Flow control in SCC is credit-based for the routers of the mesh.

- ⋆ Each router has eight credits to give per port
- ⋆ A router can send a packet to another router only when it has a credit from that router
- ⋆ Credits are automatically routed back to the sender when the packet moves on to the next destination

Most of the other agents use on-off signal-based flow control. The exception is the MIU which is the main traffic controller in the tile and uses a request/grant protocol to control access of the tile agents to the router.

3.3.4 Error Checking

Error checking is done end-to-end, primarily through parity bits on mesh packets. Parity checks on packets are done on the following fields : route field, commands and data. Parity generation is done at the mesh interface (MIF) of the MIU. No automatic error correction is attempted. Error signals are sent to agents if a parity error is detected. In such cases, a retry mechanism is used by the agents.

3.4 The SCC System Memory

3.4.1 System Memory Map

Each of the SCC's four memory controllers provides access to from 4 GB to 16 GB of main memory, depending on the density of the DIMMs used, for a total of up to 64 GB. The Intel SCC used for our experiments has been configured with 8 GB per memory controller, resulting to a total 32 GB of system memory. Each core has 32 address bits capable of addressing only 4 GB, so system address Lookup Tables map addresses from the core physical addresses to system physical addresses. Memory addresses can be mapped in a manner that shares all, some or none of the system memory among cores. The boundaries between the shared and private space are dynamically programmable, giving some flexibility in the partitioning of tasks between cores.

All I/O accesses are passed through the system interface and on to the board FPGA.

The 4 GB core address space is divided into 256 16 MB pages, for system address translation. Each page has an entry in the LUT that provides routing and address translation information. The LUT is programmed at boot time. However, no restrictions are placed on LUT re-programming during normal operation.

3.4.2 Memory Address Translation

The SCC Lookup Table (LUT) unit performs the address translation from core address to system address. Two LUTs, one for each core are used to translate all outgoing core addresses into system addresses. Figure 3.3 illustrates address translation. During address translation, the upper 8 bits of the core address are used to index one of the 256 LUT entries. A 22-bit output bus is distributed as follows : 10 bits for the upper 10 bits in the new memory address, 8 bits for the tile destination ID, 3 bits for the subdestination ID and 1 bit for MIU bypass. The subdestination ID is used to identify the specific component of the destination tile that the packet should be routed at. Different values correspond to the tile MPB, CRB (configuration register) and the four ports of the router of the destination tile (east,west,north,south).

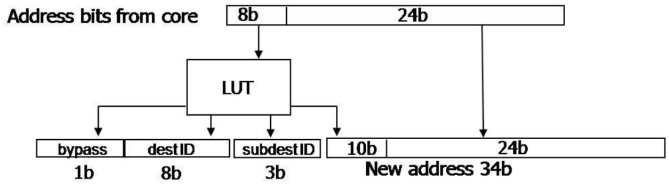


Figure 3.3: Intel SCC Address Translation

3.5 The SCC Power Management API

3.5.1 Voltage and Frequency Islands

SCC cores are divided into six voltage islands, each containing a 2×2 array of tiles; each island has a total of eight P54C cores. Each island has a separate power supply. The voltage islands are also called voltage domains. Clocking is at an even finer granularity with each tile on SCC able to have its own operating frequency. The voltage and frequency islands enable parts of SCC to be turned off or dialed down to a lower frequency to minimize power consumption. Figure 3.4 illustrates the voltage and frequency domains on the SCC.

The mesh has its own clock and power supply with all router stops on the same clock and power supply. The power consumption of the mesh can thus be controlled independently of the cores and vice versa. Thus, the entire mesh can be thought as a single voltage/frequency island.

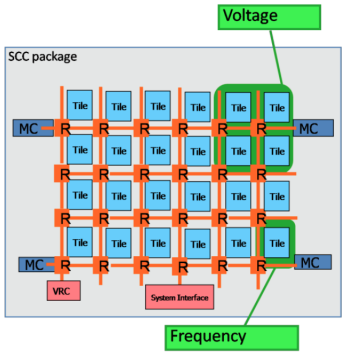


Figure 3.4: Intel SCC Voltage and Frequency Islands

3.5.2 The Global Clock Unit (GCU) Configuration Register

The Global Clock Unit (GCU) Configuration Register regulates the operating frequency of each tile. The router clock is set at either 800 MHz or 1.6 GHz. When the router clock is 800 MHz, the memory clock is also at 800 MHz. When the router clock is 1.6 GHz, the memory clock is either 800 MHz or 1066 MHz. When the router frequency is 800 MHz, the default tile frequency is 533 MHz. When the router frequency is 1.6 GHz, the default tile frequency is 800 MHz.

By writing bits 25:08 of the GCU, the tile frequency can be changed. The value that has to be written to those bits of the GCU depends on the desired tile frequency as well as the frequency of the mesh. Due to Intel SCC hardware limitations, we have noticed that it is possible to perform frequency perturbations only when the router operates at 800 MHz and the memory at 800 MHz. Table 3.2 shows the binary value that has to be written to the 25:08 bits of the GCU so as to achieve the specific tile frequency.

3.5.3 The SCC Power Controller (VRC)

The SCC Power Controller (VRC) enables each core of the platform to adjust the voltage of each voltage island. The VRC has its own destination target in the core's memory map and thus its own entry in the LUT. A core or the system interface can write to this memory location, and it will be decoded as a command for the VRC. This command is then routed to the VRC across the mesh and executed. The VRC accepts the command, adjusts the voltage and then sends an acknowledgement back to the tile so that it knows the command completed successfully.

The VRC can set the voltage of a voltage domain to any value between 0 V and 1.3 V, with a 6.25 mV step. However, depending on the frequency settings of the tiles of the voltage domain, a minimum voltage level is required so as to ensure stable operation. Table 3.3 states the minimum voltage that is required for safe operation for all possible frequency settings.

Tile Frequency (MHz)	GCU Config Setting [25:08]
800	00 0111 0000 1110 0001
533	00 1010 1000 1110 0010
400	00 1110 0000 1110 0011
320	01 0001 1000 1110 0100
266	01 0101 0000 1110 0101
228	01 1000 1000 1110 0110
200	01 1100 0000 1110 0111
176	01 1111 1000 1110 1000
160	10 0011 0000 1110 1001
145	10 0110 1000 1110 1010
133	10 1010 0000 1110 1011
123	10 1101 1000 1110 1100
114	11 0001 0000 1110 1101
106	11 0100 1000 1110 1110
100	11 1000 0000 1110 1111

Table 3.2: Tile Frequency Settings for Router Clock of 800 MHz

3.5.4 Changing The Tile Frequency

Each core can access its own configuration registers as well as those of other cores using memory-mapped I/O. Memory-mapped I/O is performed in standard Linux using the `mmap()` function. The base address for the configuration registers for the tile at (x=0, y=0) is `0xe0000000`. The configuration registers for each tile are offset by `0x01000000` from `0xe0000000` as you travel along the x axis. Following this convention, the base address for the tile at (x=1, y=0) is `0xe1000000`, that for the tile at (x=2, y=0) is `0xe2000000`, etc. The tile after (x=5, y=0) is (x=0, y=1), etc. Continuing with this method, the base address for the final tile at (x=5, y=3) is `0xf7000000`. The base address `0xf8000000` is a special one. When a core specifies this base address, it specifies its own base address.

The program shown below (`setFreq800.c`) sets the frequency of a specific tile to 800 MHz (for router frequency at 800 MHz). The device `/dev/rckncm` is used to specify the file descriptor of the file to be mapped. This filed is mapped to a memory page using `mmap()`. The Global Clocking Unit Configuration Register is accessed by specifying an offset of `0x80` from the base address of the Configuration Registers of the specific core (`0xf8000000`). The bits 25:08 of the GCU are set to `0x070e100`, which corresponds to a tile frequency of 800 MHz. Finally, the page is unmapped using `munmap()`.

`setFreq800.c :`

```
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

Tile Frequency (MHz)	Minimum Voltage (V)
800	1.16250
533	0.85625
400	0.75625
320	0.69375
267	0.66875
229	0.65625
200	0.65625
178	0.65625
160	0.65625
145	0.65625
133	0.65625
123	0.65625
114	0.65625
107	0.65625
100	0.65625

Table 3.3: Minimum Voltage Levels for Safe Operation

```
#include <stdlib.h>
#define CRB_OWN   0xf8000000
#define GCBCFG  0x80

main() {
    typedef volatile unsigned char* t_vcharp;
    int PAGE_SIZE, NCMDeviceFD;
    // NCMDeviceFD is the file descriptor for
    // non-cacheable memory (e.g. config regs).
    unsigned int result;
    t_vcharp MappedAddr;
    unsigned int alignedAddr, pageOffset, ConfigAddr;
    ConfigAddr = CRB_OWN+GCBCFG;
    PAGE_SIZE  = getpagesize();
    if ((NCMDeviceFD=open("/dev/rckncm", O_RDWR|O_SYNC))<0) {
        perror("open");
        exit(-1);
    }
    alignedAddr = ConfigAddr & (~(PAGE_SIZE-1));
    pageOffset = ConfigAddr - alignedAddr;
    MappedAddr = (t_vcharp) mmap(NULL, PAGE_SIZE, PROT_WRITE|PROT_READ,
        MAP_SHARED, NCMDeviceFD, alignedAddr);
    if (MappedAddr == MAP_FAILED) {
        perror("mmap");exit(-1);
    }
    result = *(unsigned int*)(MappedAddr+pageOffset) & 0xff;
    result += 0x070e100;
```

```
    *(unsigned int*)(MappedAddr+pageOffset) = result;
    munmap((void*)MappedAddr, PAGE_SIZE);
}
```

This program is cross-compiled with `icc` on the MCPC so as to produce the `setFreq800` executable that will run on SCC hardware. The executable is executed using the `pssh` command. The `pssh` command is used so as to load the executable on the SCC cores. The resulting executable has to be placed in a subdirectory of the `/shared` directory, so that it can be accessed by the cores. The `/shared` directory is mounted on all the cores as a network file system. The `pssh` command has to be executed as follows:

```
pssh -h hosts.txt -p 1 -P -t -1 /shared/ageo/setFreq800
```

The file `hosts.txt` specifies the cores that this program will be run. If for example, this file contains the following two lines, then the program is executed by cores `rck00` and `rck02` and will set the frequency of cores `rck00`, `rck01`, `rck02` and `rck03` to 800 MHz.

`hosts.txt :`

```
rck00 root
rck02 root
```

The `-p` switch defines the number of concurrent threads that will be executed by the `pssh` command. The `-t` switch specifies the timeout in seconds, which in this case is `-1`, which means that the execution never times out. The `-P` switch specifies that the program prints the output as it is received.

3.6 The Management Console

The Management Console is a PC that communicates with the SCC platform over a PCI Express bus. The PCIe bus connects to the system FPGA interface on the SCC board which connects to the System Interface on the SCC itself. The MCPC runs a stable version of Ubuntu Linux. The **sccKit** software that is provided by Intel, enables users to boot Linux on the SCC cores, read and write core memory locations and registers, monitor performance etc.

Figure 3.5 illustrates how the SCC and the MCPC are connected. The figure shows two Ethernet cables coming from the SCC chassis. The MCPC also has two Ethernet cables and to NICs. The `eth0` cable connects to the Internet through a public IP. The `eth1` cable connects to the Board Management Microcontroller (BMC). The BMC is an ARM processor that is responsible for initializing and shutting down critical system functions.

The sccKit software offers a variety of functions that are available from the command line so as to configure and monitor the SCC platform. In the remainder of this section, the commands offered by sccKit are described. Appropriate examples are given as well.

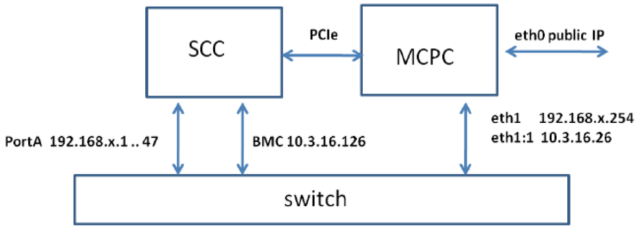


Figure 3.5: Connection of the Intel SCC to the MCPC

3.6.1 sccBoot

`sccBoot` enables users of the platform to boot Linux on the SCC cores and to check that the cores have been successfully booted as well. When run with the `-L` switch, `sccBoot` boots Linux on all SCC cores. When run with the `-l` switch, followed by a core ID or a range of core IDs, then `sccBoot` boots Linux on the specified core or range of cores. For example, in order to boot Linux on cores `rck00` and `rck01`, the following command should be executed:

```
sccBoot -l 0..1
```

In order to check that Linux has been successfully booted on all cores, `sccBoot` should be run with the `-s` switch. This commands pings all cores on the SCC board and returns the processor IDs of the cores that were successfully reached.

3.6.2 sccPerf

`sccPerf` opens a graphical user interface (GUI) that visualizes the status of the SCC board. This GUI shows the CPU utilization of each core, the overall CPU utilization and the overall power consumption of the platform. The performance meter window is shown at figure 3.6.

3.6.3 sccDump

`sccDump` provides the ability to read data from the off-chip DRAM (using the `-d` switch), a Message Passing Buffer (using the `-m` switch), a Core Configuration Register (using the `-c` switch) or the System Interface (using the `-s` switch). The following example shows the output that is received when `sccDump` is executed with the `-c` switch, followed by the hexadecimal identification of the bottom left tile (that includes cores `rck00` and `rck01`). The output of this command lists the values of all the configuration registers of the specific tile, as well as all the LUT entries of both cores of the tile.

```
ageo@mitsos:~$ sccDump -c 0x00
INFO: Packet tracing is disabled...
INFO: Initializing System Interface (SCEMI setup)....
INFO: Successfully connected to PCIe driver...
INFO: Welcome to sccDump 1.4.1 (build date Jun 28 2011 - 16:02:28)...
```

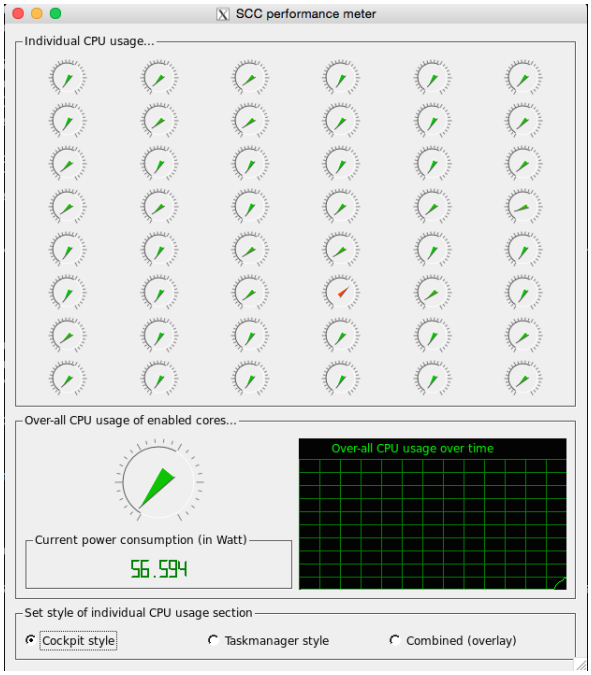


Figure 3.6: Intel SCC Performance Meter

```
============================================================
Dumping CRB registers of Tile 0x00
============================================================
GLCFG0   = 0x00348df8
GLCFG1   = 0x00348df8
L2CFG0   = 0x000006cf
L2CFG1   = 0x000006cf
SENSOR   = 0x00002554
GCBCFG   = 0x0070e1f0
MYTILEID = 0x00000005
LOCK0    = 0x00000001
LOCK1    = 0x00000001
------------------------------------------------------------
Restoring locks: LOCK0 and LOCK1
============================================================
Dumping LUTs of Tile 0x00
Format: Bypass(bin)_Route(hex)_subDestId(dec)_AddrDomain(hex)
============================================================
LUT0, Entry 0x00 (CRB addr = 0x0800): 0_0x00_6(PERIW)_0x000
LUT0, Entry 0x01 (CRB addr = 0x0808): 0_0x00_6(PERIW)_0x001
         . . . .
LUT0, Entry 0xfe (CRB addr = 0x0ff0): 0_0x95_1(CORE1)_0x014
LUT0, Entry 0xff (CRB addr = 0x0ff8): 0_0x00_6(PERIW)_0x1f4
LUT1, Entry 0x00 (CRB addr = 0x1000): 0_0x00_6(PERIW)_0x029
LUT1, Entry 0x01 (CRB addr = 0x1008): 0_0x00_6(PERIW)_0x02a
         . . . .
```

```
LUT1, Entry 0xfe (CRB addr = 0x17f0): 1_0xc5_0(CORE0)_0x24b
LUT1, Entry 0xff (CRB addr = 0x17f8): 0_0x00_6(PERIW)_0x1f5
============================================================
```

3.6.4 sccBmc

`sccBmc` is used for initializing the SCC platform and for sending commands to the BMC. The platform can be initialized when running the `sccBmc` command with the `-i` switch. The `-i` switch has to be accompanied by one of the following configurations, which determine the tile, mesh and memory frequency respectively:

```
Tile533_Mesh800_DDR800
Tile800_Mesh1600_DDR1066
Tile800_Mesh1600_DDR800
Tile800_Mesh800_DDR1066
Tile800_Mesh800_DDR800
```

The `sccBmc` command can also be executed with the `-c` switch, so as to connect to the BMC and execute the specific command. The following example shows the output of the execution of the `status` command at the BMC, which displays information regarding the current board status, such as voltage levels and board temperature. This functionality of the `sccBmc` command has been significantly utilized in the monitoring infrastructure we have developed.

```
ageo@mitsos:~$ sccBmc -c status
INFO: openBMCConnection(10.3.16.126:5010): You are participant #2
INFO: Welcome to sccBmc 1.4.1 (build date Jun 28 2011 - 16:01:43)...
INFO: Result of BMC command "status":
I?C access is switched to BMC
Power Status = 0xCF3F, ON

Standby supplies:
  5V0PWR:    5.002 V (Primary)
  1V8SB:     1.800 V (Secondary)
  3V3PWR:    3.260 V     -"-

Primary supplies:
  3V3IN:     3.360 V
  5V0IN:     5.054 V
  12V0R1:    11.972 V
  12V0R2:    11.999 V

Secondary supplies:
  1V0:       1.018 V    1.590 A
  1V1VCCA:   1.104 V    2.480 A
  1V1VCCT:   1.096 V    4.229 A
  1V5:       1.522 V    6.241 A
  1V65:      1.666 V
  1V65ADJ:   1.652 V
```

```
  1V8PHY:    1.796 V
  2V5:       2.480 V
  3V3:       3.316 V    2.350 A
  3V3SCC:    3.304 V   15.842 A

Tertiary supplies:
  OPVR VCC0: 1.0928 V
  OPVR VCC1: 1.1014 V
  OPVR VCC2: 1.1862 V
  OPVR VCC3: 1.0874 V
  OPVR VCC4: 1.1089 V
  OPVR VCC5: 1.0931 V
  OPVR VCC7: 1.0984 V

Temperatures:
  Board:      34 ?C
  FPGA:       43 ?C

Fan speed:
  FPGA:       108 RPM (Needs real conversion to RPM!)
  SCC:        148 RPM

Misc.:
  FPGA status: 0xC7
  Lane Good LED is off
  L0: normal operation
  CPLD status: 0x47
  PLL is locked.
  PLL lock lost is cleared.
```

3.7 The SCC Linux

The MCPC contains an Intel-provided Linux image that runs on the SCC cores. This sections discusses two aspects of the SCC Linux whose understanding is critical for characterizing MapReduce workloads that run on the SCC : the TCP/IP stack and the Network File System that is mounted on the cores.

3.7.1 The TCP/IP Stack

Each SCC Linux instance has two virtual network interfaces : **mb0** and **emac0**.

- **emac0** is used for communication between the cores and the MCPC. The IP address of this interface is `192.168.3.x` where `1 < x < 48`, depending on the processor ID. Packets that are directed towards this interface, are sent to the Gigabit Ethernet Switch that connects the SCC with the MCPC, so as to reach the MCPC, whose IP address is `192.168.3.254`. Communication between cores is not possible through this interface.

- **mb0** is used for communication between the cores of the SCC. The IP address of this interface is `192.168.0.x` where `1 < x < 48`, depending on the processor ID. Packets that are directed towards this interface are sent to the Message Passing Buffer of the receiving core, i.e. the destination IP address is translated to a physical MPB address. That is, the communication between cores takes place entirely within the boundaries of the SCC Mesh and does not exit the SCC board.

3.7.2 The Network File System

The directory `/shared` on the MCPC is NFS - mounted on the cores. As a consequence disk I/O takes place through the `emac0` interface and is directed to the MCPC, where it is stored in its physical storage.

Chapter 4

The Hadoop Distributed File System and the MapReduce Framework

This chapter provides a detailed analysis of the Hadoop Distributed File System and the MapReduce framework. Each of the following two sections covers several implementation aspects of HDFS and MapReduce respectively and concludes with a list of configuration and runtime parameters that users and cluster administrators can specify with respect to the HDFS cluster installation and the execution of MapReduce jobs.

4.1 The Hadoop Distributed File System

This section describes several aspects of the Hadoop Distributed File System (HDFS) in detail. Particular emphasis is placed on the master/slave architecture of HDFS, data replication and data reliability and availability, which are some of the key features offered by HDFS. The section concludes with the runtime and configuration parameters that can be used so as to customize an HDFS cluster installation.

4.1.1 The NameNode and the DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single **NameNode**, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of **DataNodes**, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file systems clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode. Figures 4.1 illustrates the organization of the

HDFS architecture.

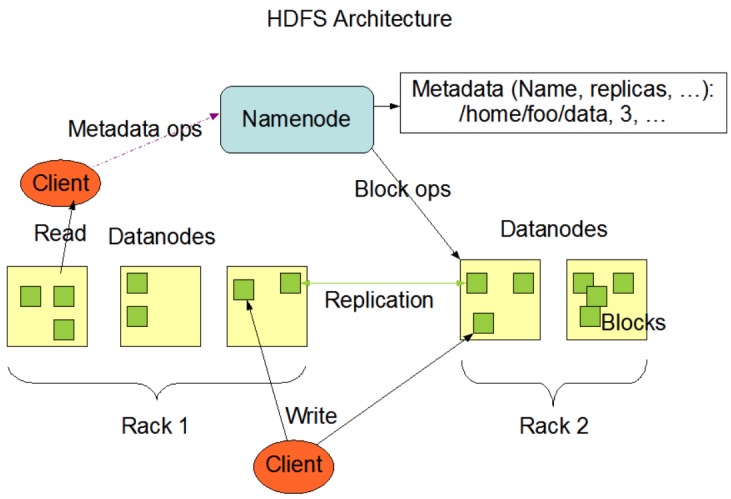


Figure 4.1: The HDFS Architecture

The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

4.1.2 The File System Namespace

HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories. The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file. HDFS does not yet implement user quotas. HDFS does not support hard links or soft links. However, the HDFS architecture does not preclude implementing these features.

The NameNode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the NameNode. An application can specify

the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. This information is stored by the NameNode.

4.1.3 Data Organization

HDFS is designed to support very large files. Applications that are compatible with HDFS are those that deal with large data sets. These applications write their data only once but they read it one or more times and require these reads to be satisfied at streaming speeds. HDFS supports write-once-read-many semantics on files. A typical **block size** used by HDFS is 64 MB. Thus, an HDFS file is chopped up into 64 MB chunks, and if possible, each chunk will reside on a different DataNode.

A client request to create a file does not reach the NameNode immediately. In fact, initially the HDFS client caches the file data into a temporary local file. Application writes are transparently redirected to this temporary local file. When the local file accumulates data worth over one HDFS block size, the client contacts the NameNode. The NameNode inserts the file name into the file system hierarchy and allocates a data block for it. The NameNode responds to the client request with the identity of the DataNode and the destination data block. Then the client flushes the block of data from the local temporary file to the specified DataNode. When a file is closed, the remaining un-flushed data in the temporary local file is transferred to the DataNode. The client then tells the NameNode that the file is closed. At this point, the NameNode commits the file creation operation into a persistent store. If the NameNode dies before the file is closed, the file is lost.

Suppose the HDFS file has a replication factor of three. When the client application local file accumulates a full block of user data, the client retrieves a list of DataNodes from the NameNode. This list contains the DataNodes that will host a replica of that block. The client then flushes the data block to the first DataNode. The first DataNode starts receiving the data in small portions (4 KB), writes each portion to its local repository and transfers that portion to the second DataNode in the list. The second DataNode, in turn starts receiving each portion of the data block, writes that portion to its repository and then flushes that portion to the third DataNode. Finally, the third DataNode writes the data to its local repository. Thus, a DataNode can be receiving data from the previous one in the pipeline and at the same time forwarding data to the next one in the pipeline. Thus, the data is pipelined from one DataNode to the next.

4.1.4 Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number

of replicas of a file. The **replication factor** can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

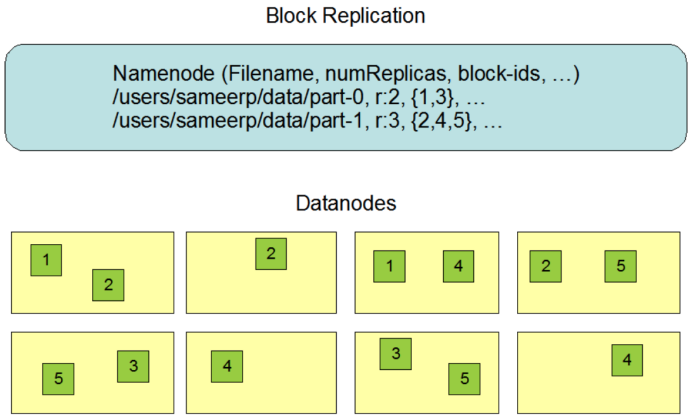


Figure 4.2: HDFS Data Replication

The NameNode makes all decisions regarding replication of blocks. It periodically receives a **Heartbeat** and a **Blockreport** from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.

Large HDFS instances run on a cluster of computers that commonly spread across many racks. Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.

For the common case, when the replication factor is three, HDFSs placement policy is to put one replica on one node in the local rack, another on a node in a different (remote) rack, and the last on a different node in the same remote rack. This policy cuts the inter-rack write traffic which generally improves write performance. In addition, since the chance of rack failure is far less than that of node failure, this policy does not impact data reliability and availability guarantees.

To minimize global bandwidth consumption and read latency, HDFS tries to satisfy a read request from a replica that is closest to the reader. If there exists a replica on the same rack as the reader node, then that replica is preferred to satisfy the read request. If an HDFS cluster spans multiple data centers, then a replica that is resident in the local data center is preferred over any remote replica.

On startup, the NameNode enters a special state called **Safemode**. Replication of data blocks does not occur when the NameNode is in the Safemode state. The NameNode receives Heartbeat and Blockreport messages from the DataNodes. Each block

has a specified minimum number of replicas. A block is considered safely replicated when the minimum number of replicas of that data block has checked in with the NameNode. After a configurable percentage of safely replicated data blocks checks in with the NameNode (plus an additional 30 seconds), the NameNode exits the Safemode state. It then determines the list of data blocks (if any) that still have fewer than the specified number of replicas. The NameNode then replicates these blocks to other DataNodes.

4.1.5 The Communication Protocols

All HDFS communication protocols are layered on top of the **TCP/IP** protocol. A client establishes a connection to a configurable TCP port on the NameNode machine. It communicates through the **ClientProtocol** with the NameNode. The DataNodes communicate with the NameNode using the **DataNode Protocol**. A **Remote Procedure Call (RPC)** abstraction wraps both the Client Protocol and the DataNode Protocol. By design, the NameNode never initiates any RPCs. Instead, it only responds to RPC requests issued by DataNodes or clients.

4.1.6 The Persistence of File System Metadata

The HDFS namespace is stored by the NameNode. The NameNode uses a transaction log called the **EditLog** to persistently record every change that occurs to file system metadata. For example, creating a new file in HDFS causes the NameNode to insert a record into the EditLog indicating this. Similarly, changing the replication factor of a file causes a new record to be inserted into the EditLog. The NameNode uses a file in its local host OS file system to store the EditLog. The entire file system namespace, including the mapping of blocks to files and file system properties, is stored in a file called the **FsImage**. The FsImage is stored as a file in the NameNodes local file system too.

The DataNode stores HDFS data in files in its local file system. The DataNode has no knowledge about HDFS files. It stores each block of HDFS data in a separate file in its local file system. When a DataNode starts up, it scans through its local file system, generates a list of all HDFS data blocks that correspond to each of these local files and sends this report to the NameNode: this is the Blockreport.

4.1.7 Data Availability and Reliability

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

A **network partition** can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and

does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

It is possible that a block of data fetched from a DataNode arrives corrupted. This **corruption** can occur because of faults in a storage device, network faults, or buggy software. The HDFS client software implements **checksum checking** on the contents of HDFS files. When a client creates an HDFS file, it computes a checksum of each block of the file and stores these checksums in a separate hidden file in the same HDFS namespace. When a client retrieves file contents it verifies that the data it received from each DataNode matches the checksum stored in the associated checksum file. If not, then the client can opt to retrieve that block from another DataNode that has a replica of that block.

The FsImage and the EditLog are central data structures of HDFS. A corruption of these files can cause the HDFS instance to be non-functional. For this reason, the NameNode can be configured to support maintaining multiple copies of the FsImage and EditLog. Any update to either the FsImage or EditLog causes each of the FsImages and EditLogs to get updated synchronously. When a NameNode restarts, it selects the latest consistent FsImage and EditLog to use.

4.1.8 File and Block Deletion

When a file is deleted by a user or an application, it is not immediately removed from HDFS. Instead, HDFS first renames it to a file in the `/trash` directory. The file can be restored quickly as long as it remains in `/trash`. A file remains in /trash for a configurable amount of time. After the expiry of its life in `/trash`, the NameNode deletes the file from the HDFS namespace. The deletion of a file causes the blocks associated with the file to be freed. Note that there could be an appreciable time delay between the time a file is deleted by a user and the time of the corresponding increase in free space in HDFS.

When the replication factor of a file is reduced, the NameNode selects excess replicas that can be deleted. The next Heartbeat transfers this information to the DataNode. The DataNode then removes the corresponding blocks and the corresponding free space appears in the cluster. Once again, there might be a time delay between the completion of the `setReplication` API call and the appearance of free space in the cluster.

4.1.9 The HDFS Command Line API

HDFS provides a command line interface called **FS Shell** that lets a user interact with the data in HDFS. The following examples demonstrate how a user can create the directory `/testdir` and view the contents of the `testfile.txt` file, which is under the `/testdir` directory in HDFS using the FS Shell. Those commands should be executed from the parent directory of the HDFS installation.

```
bin/hadoop dfs -mkdir /testdir
bin/hadoop dfs -cat /testdir/testfile.txt
```

The **DFSAdmin** command set is used for administering an HDFS cluster. These are commands that are used only by an HDFS administrator. The following example demonstrates how to force the NameNode to exit the SafeMode:

```
bin/hadoop dfsadmin -safemode leave
```

HDFS APIs are also available for the **Java** and **C** programming languages, so as to enable client applications interact with files stored in HDFS. HDFS also offers a **Browser Interface** that enables users navigate the HDFS namespace and view the contents of its files.

4.1.10 Configuring an HDFS Cluster

Each HDFS cluster deployment is configured by a big set of parameters. A default value is specified for each configuration parameter, which can be overriden so as to customize the HDFS installation. The HDFS deployment configuration is controlled by four configuration files : `core-default.xml`, `core-site.xml`, `hdfs-default.xml` and `hdfs-site.xml`. Those files are loaded in the classpath of each HDFS daemon (NameNode and DataNodes) at runtime. The runtime environment of an HDFS cluster is set up by the `hadoop-env.sh` configuration script.

core-default.xml and **core-site.xml** contain information that regards global properties of an HDFS installation, such as the endpoint URI that consists of the host and the port of the file system. Additional information that is determined in those files regards I/O properties such as error checking and rack topology configuration. The default values of those parameters are specified in `core-default.xml`. Default parameter values overrides should be included in `core-site.xml`. Table 4.1 provides a description of configuration parameters that can be defined in `core-site.xml` accompanied by example values.

hdfs-default.xml and **hdfs-site.xml** contain information such as the local file system directories where HDFS metadata and file data blocks should be stored by the NameNode and the DataNodes. In addition, those files determine the default block size and replication factor of the distributed file system. The default values of those parameters are specified in `hdfs-default.xml`. Default parameter values overrides should be included in `hdfs-site.xml`. Table 4.2 provides a description of configuration parameters

Configuration Parameter	Description	Example Value
`fs.default.name`	`The name of the default file system, in the form of and endpoint URI.`	`hdfs://192.168.0.1:54310`
`hadoop.tmp.dir`	`A base for other temporary directories.`	`/home/ageo/tmp_dir`
`topology.script.file.name`	`The script name that determines the allocation of cluster nodes to HDFS racks.`	`/home/ageo/hadoop-topology.sh`
`io.skip.checksum.errors`	`If true, when a checksum error is encountered while reading a sequence file entries are skipped, instead of throwing an exception.`	`true/false`

Table 4.1: Configuration Parameters Defined in `core-site.xml`

that can be defined in `hdfs-site.xml` accompanied by example values.

hadoop-env.sh determines overrides for environment variables that are related to the HDFS installation, such as the Java Home directory and the Java Heap Size. Table 4.3 provides a description of runtime parameters that can be defined in `hadoop-env.sh` accompanied by example values.

4.2 The MapReduce Framework

This section describes several aspects of the MapReduce framework in detail. A MapReduce **Job** usually splits the input dataset into independent chunks which are processed by the **Map** tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the **Reduce** tasks. Typically both the input and the output of the job are stored in HDFS. The framework takes care of scheduling and monitoring tasks and the re-execution of the failed tasks. This section concludes with a list of configuration parameters that can be specified so as to tune the execution of MapReduce jobs.

4.2.1 The JobTracker and the TaskTrackers

The MapReduce framework consists of a single master **JobTracker** and one slave **TaskTracker** per cluster-node. The JobTracker is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The TaskTrackers execute the tasks as directed by the JobTracker.

Minimally, applications specify the input/output locations and supply map and re-

Configuration Parameter	Description	Example Value
dfs.replication	Default block replication. The actual number of replications can be specified when the file is created.	1
dfs.block.size	The default block size for new files.	4194304 (4 MB)
dfs.name.dir	Determines the local filesystem directory, where the NameNode should store the name table (fsimage).	/home/ageo/hdfsnames
dfs.data.dir	Determines the local filesystem directory, where the DataNode should store file data blocks.	/home/ageo/hdfsdata

Table 4.2: Configuration Parameters Defined in `hdfs-site.xml`

Configuration Parameter	Description	Example Value
JAVA_HOME	Home directory of the Java installation	/opt/ibm-jdk-bin-1.6.0.8_p1
HADOOP_HEAPSIZE	Maximum Java heap size in MB for Hadoop Daemons	128
HADOOP_SSH_OPTS	Extra SSH options	-p 1234 -l root
HADOOP_ROOT_LOGGER	Hadoop logging level	ERROR,console

Table 4.3: Runtime Parameters Defined in `hadoop-env.sh`

duce functions via implementations of appropriate interfaces and/or abstract-classes. These and other job parameters comprise the job configuration. The **Hadoop Job Client** then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the Job Client.

The MapReduce framework operates exclusively on `<key, value>` pairs, that is, the framework views the input to the job as a set of `<key, value>` pairs and produces a set of `<key, value>` pairs as the output of the job, conceivably of different types.

Figure 4.3 illustrates how the MapReduce framework can be leveraged in order to count the occurences of each word in an input document. Initially, the document is split into

three different `InputSplits`, which are provided to each Map task as `<key,value>` pairs of the form `<byte offset, line string>`. Each Map task tokenizes each line string in order to generate intermediate `<key,value>` pairs for each word, which contain the found word as a key and `1` as value, `<word,1>`. During the shuffle stage, the intermediate `<key,value>` pairs are sorted and grouped based on the intermediate key and each `<key,list of values>` pair is provided to each Reducer task. The Reducer tasks sum the `1`'s that are contained in each list of values and output the number of occurences of each word in the input document.

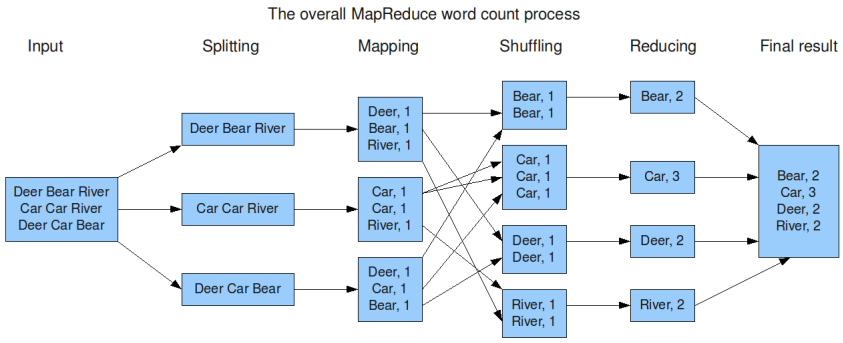


Figure 4.3: MapReduce Wordcount

4.2.2 The Mapper Function

The Mapper function maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. The framework calls the `map()` method for each key/value pair in the `InputSplit` for that task. The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job. Thus, the number of maps is driven by the total size of the inputs, that is, the total number of blocks of the input files.

Applications can use the `Reporter` to report progress, set application-level status messages and update `Counters`, or just indicate that they are alive. All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to the Reducer(s) to determine the final output. Users can control the grouping by specifying a `Comparator`. The Mapper outputs are sorted and then partitioned per Reducer. The total number of partitions is the same as the number of reduce tasks for the job. Users can control which keys (and hence records) go to which Reducer by implementing a custom `Partitioner`. Users can optionally specify a `Combiner`, to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer. The intermediate, sorted outputs are always stored in a simple (key-len, key, value-len, value) format.

4.2.3 The Reducer Function

The Reducer function reduces a set of intermediate values which share a key to a smaller set of values. The number of reducers for the job can be configured by the user. The Reduce phase of a MapReduce Job has 3 primary stages: shuffle, sort and reduce. During the **Shuffle** stage, the framework fetches the relevant partition of the output of all the mappers, via HTTP. During the **Sort** stage the framework groups Reducer inputs by keys, since different mappers may have output the same key. The shuffle and sort phases occur simultaneously; while Map outputs are being fetched they are merged. During the **Reduce** stage, the `reduce()` method is called for each `<key, (list of values)>` pair in the grouped inputs.

The output of the reduce task is typically written to HDFS. Applications can use the `Reporter` to report progress, set application-level status messages and update `Counters`, or just indicate that they are alive. The output of the Reducer is not sorted. It is possible to set the number of Reduce tasks to zero if no reduction is desired. In this case the outputs of the Map tasks go directly to the corresponding HDFS output path. The framework does not sort the map outputs before writing them out to HDFS.

The `Partitioner` controls the partitioning of the keys of the intermediate Map outputs. The key (or a subset of the key) is used to derive the partition, typically by a *hash function*. The total number of partitions is the same as the number of the Reduce tasks of the job. That is, the `Partitioner` controls to which of the Reduce tasks the intermediate key (and hence the record) is sent to for reduction.

4.2.4 Job Configuration

The `JobConf` entity represents a MapReduce job configuration. `JobConf` is the primary interface for a user to describe a MapReduce job to the Hadoop framework for execution. The framework tries to faithfully execute the job as described by `JobConf`. `JobConf` is typically used to specify the `Mapper`, `Combiner` (if any), `Partitioner`, `Reducer`, `InputFormat`, `OutputFormat` and `OutputCommitter` implementations. `JobConf` also indicates the set of input files and where the output files should be written.

Optionally, `JobConf` is used to specify other advanced facets of the job such as the `Comparator` to be used, files to be put in the `DistributedCache`, whether intermediate and/or job outputs are to be compressed (and how), debugging via user-provided scripts, whether job tasks can be executed in a *speculative* manner, maximum number of attempts per task, percentage of tasks failure which can be tolerated by the job etc.

4.2.5 Task Execution and Environment

The TaskTracker executes the `Mapper/Reducer` task as a child process in a separate JVM. The child-task inherits the environment of the parent TaskTracker. The user can specify additional options to the child JVM via the `mapred.{map|reduce}.child.java.opts`

configuration parameter in the `JobConf`.

The TaskTracker has its local directory, `${mapred.local.dir}/taskTracker/`. When the job starts, the TaskTracker creates the localized job directory `$user/jobcache/$jobid/`, which is relative to the previous directory. The localized job directory contains the following subdirectories/files:

- `work/` : The job-specific shared directory. The tasks can use this space as scratch space and share files among them.
- `jars/` : The jars directory, which has the job jar file and expanded jar. The `job.jar` is the application's jar file that is automatically distributed to each machine. It is expanded in the `jars/` directory before the tasks for the job start.
- `job.xml` : The `job.xml` file, the generic job configuration, localized for the job.
- `$taskid/` : The task directory for each task attempt. Each task directory contains the following subdirectories/files.
  - `job.xml` : A `job.xml` file, task localized job configuration. Task localization means that properties have been set that are specific to this particular task within the job.
  - `output/` : A directory for intermediate output files. This contains the temporary MapReduce data generated by the framework such as Map output files etc.
  - `work/` : The current working directory of the task.
  - `work/tmp/` : The temporary directory for the task. This directory will be created if it doesn't exist.

The standard output (`stdout`) and error (`stderr`) streams of the task are read by the TaskTracker and logged to `${HADOOP_LOG_DIR}/userlogs`.

4.2.6 Job Submission and Monitoring

The `JobClient` is the primary interface by which user-job interacts with the `JobTracker`. The `JobClient` provides facilities to submit jobs, track their progress, access component-tasks' reports and logs, get the MapReduce cluster's status information and so on. The job submission process involves:

1. Checking the input and output specifications of the job.
2. Computing the `InputSplit` values for the job.
3. Setting up the requisite accounting information for the `DistributedCache` of the job, if necessary.
4. Copying the job's jar and configuration to the MapReduce system directory on the `FileSystem`.

5. Submitting the job to the JobTracker and optionally monitoring it's status.

Job submission is also possible through the Hadoop FS Shell, with the usage of the `jar` command as follows:

```
bin/hadoop jar <jar file> <main class>  [arguments]
```

Users may need to chain MapReduce jobs to accomplish complex tasks which cannot be done via a single MapReduce job. This is fairly easy since the output of the job typically goes to distributed file-system, and the output, in turn, can be used as the input for the next job.

The `JobClient` interface is also available from the Linux shell. The following examples state how a user can view all the Jobs that are currently running in a MapReduce cluster and how a MapReduce Job can be killed.

```
bin/hadoop job -list
bin/hadoop job -kill <jobId>
```

4.2.7 Job Input

The `InputFormat` describes the input specification for a MapReduce job. The MapReduce framework relies on the `InputFormat` of the job to:

- ⋆ Validate the input-specification of the job.
- ⋆ Split-up the input file(s) into logical `InputSplit` instances, each of which is then assigned to an individual Mapper.
- ⋆ Provide the `RecordReader` implementation used to glean input records from the logical `InputSplit` for processing by the Mapper.

The `InputSplit` represents the data to be processed by an individual Mapper. Typically the `InputSplit` presents a byte-oriented view of the input, and it is the responsibility of `RecordReader` to process and present a record-oriented view. The `RecordReader` reads `<key, value>` pairs from an `InputSplit`. Typically the `Record Reader` converts the byte-oriented view of the input, provided by the `InputSplit`, and presents a record-oriented to the Mapper implementations for processing. The `RecordReader` thus assumes the responsibility of processing record boundaries and presents the tasks with keys and values.

4.2.8 Job Output

The `OutputFormat` describes the output specification for a MapReduce job. The MapReduce framework relies on the `OutputFormat` of the job to:

- ⋆ Validate the output specification of the job; for example, check that the output directory doesn't already exist.

- ⋆ Provide the `RecordWriter` implementation used to write the output `<key, value>` pairs to the output files of the job. The output files of the job are stored in HDFS.

The `OutputCommitter` describes the commit of task output for a MapReduce job. The MapReduce framework relies on the `OutputCommitter` of the job to:

- ⋆ Setup the job during initialization. For example, create the temporary output directory for the job during the initialization of the job. Job setup is done by a separate task when the job is in PREP state and after initializing tasks. Once the setup task completes, the job will be moved to RUNNING state.
- ⋆ Cleanup the job after the job completion. For example, remove the temporary output directory after the job completion. Job cleanup is done by a separate task at the end of the job. Job is declared SUCCEDED/FAILED/KILLED after the cleanup task completes.
- ⋆ Setup the task temporary output. Task setup is done as part of the same task, during task initialization.
- ⋆ Check whether a task needs a commit. This is to avoid the commit procedure if a task does not need to commit.
- ⋆ Commit of the task output. Once task is done, the task will commit it's output if required.
- ⋆ Discard the task commit. If the task has been failed/killed, the output will be cleaned-up. If task could not cleanup (in exception block), a separate task will be launched with same attempt id to do the cleanup.

4.2.9 Configuring the MapReduce Framework

The MapReduce framework is configured by a big set of parameters. A default value is specified for each configuration parameter, which can be overriden. The MapReduce configuration is controlled by two configuration files : `mapred-default.xml` and `mapred-site.xml`. Those files are loaded in the classpath of each MapReduce daemon (JobTracker and TaskTrackers) at runtime. The runtime environment of the MapReduce framework is also set up by the `hadoop-env.sh` configuration script.

mapred-default.xml and **mapred-site.xml** contain information that regards the execution of a MapReduce Job, such as the number of Reducer tasks, the maximum Map and Reducer tasks per node, the task timeout etc. The default values of those parameters are specified in `mapred-default.xml`. Default parameter values overrides should be included in `mapred-site.xml`. Table 4.4 provides a description of configuration parameters that can be defined in `mapred-site.xml` accompanied by example values.

Configuration Parameter	Description	Example Value
mapred.job.tracker	The host and port that the MapReduce JobTracker runs at.	192.168.0.1:54311
mapred.reduce.tasks	The default number of reduce tasks per job.	32
mapred.tasktracker. map.tasks.maximum	The maximum number of map tasks that will be run simultaneously by a task tracker.	1
mapred.tasktracker. reduce.tasks.maximum	The maximum number of reduce tasks that will be run simultaneously by a TaskTracker.	1
mapred.child.java.opts	Java opts for the TaskTracker child processes.	-Xmx160m
mapred.task.timeout	The number of milliseconds before a task will be terminated if it neither reads an input, writes an output, nor updates its status string.	3600000
mapred.map.max.attempts	The maximum number of attempts per map task.	15
mapred.reduce.max.attempts	The maximum number of attempts per reduce task.	10
mapred.jobtracker.taskScheduler	The class responsible for scheduling the tasks.	org.apache.hadoop. mapred.FairScheduler

Table 4.4: Configuration Parameters Defined in mapred-site.xml

Chapter 5

Hadoop Cluster Deployment on the Intel SCC

This chapter provides a detailed description of the tools that have been used and developed so as to deploy and launch Hadoop Clusters on the Intel SCC. The version of Hadoop we have used is 0.20.2. The first section of this chapter analyzes the necessary Runtime Environment setup that has to be performed and the next section explains the deployment process for four Hadoop Cluster topologies on the Intel SCC. The chapter concludes with the installation process of Apache Mahout on the MCPC.

5.1 Hadoop Runtime Environment for the Intel SCC

This section presents the Runtime Environment that is required so as to launch Hadoop Clusters on the Intel SCC. It provides a detailed description of the Gentoo Linux Image that we have used so as to provide all of the software tools and are necessary for a Hadoop Cluster installation that are not provided by the Intel SCC Linux. In addition, several modifications that we have applied regarding the Network Configuration of the Intel SCC and the MCPC are stated. Moreover, the Java installation process and the setup of password-less SSH communication between the Intel SCC cores are described. The section concludes with a script we have developed, which sets up the runtime environment required by HDFS in each Intel SCC core.

5.1.1 Gentoo Linux for the Intel SCC

Since the Intel SCC Linux provides only a restricted application development API that does not cover the requirements of a Hadoop Cluster installation, we have utilized a Gentoo Image which has been developed specifically for the Intel SCC by Sobania and Tröger [26]. This Gentoo Image makes all usual Linux tools available for us, as well as it's software repository, which contains a big variety of software packages for this specific version of Linux.

The Gentoo Linux for the Intel SCC can be downloaded from the link

`http://www.dcl.hpi.uni-potsdam.de/research/scc/scc_gentoo_20101117.tar.bz2.`

The archive that is provided by this link consists of the Gentoo root filesystem under `gentoo/` directory and two bash scripts, `to_gentoo.sh` and `set_nat.sh` which are used to enter the Gentoo Linux interactive shell from an Intel SCC core and to enable the cores of Intel SCC access the public Internet with NAT routing through the MCPC respectively.

to_gentoo.sh is used for entering the Gentoo interactive shell from an Intel SCC core. This script changes the root directory to the Gentoo Linux root directory, which has to be located under `/shared`, so as to be accessible from an Intel SCC core. This is achieved by invoking the `chroot` command. `to_gentoo.sh` contains the following single line of code. The root directory of Gentoo Linux is

```
/shared/ageo/rck00/shared/gentoo
```

in this case. Before this script is executed, `/proc` and `/dev` directories of Intel SCC Linux have to be mounted to the corresponding directories of the Gentoo Linux directory structure.

`to_gentoo.sh`:

```
/usr/sbin/chroot /shared/ageo/rck00/shared/gentoo/ ‘‘$SHELL -i /home/myinit.sh’’
```

gentoo directory contains the Gentoo Linux file system structure:

```
ageo@mitsos:~$ ls /shared/ageo/rck00/shared/gentoo
bin  bonnie  boot  dev  etc  home  lib  mnt  opt  proc  root
sbin  shared  sys  tmp  user  usr  var
```

Since it is required that the Gentoo Linux is mounted on all 48 Intel SCC cores, we have replicated the directory structure under `gentoo/` 48 times, so as to create the Gentoo Linux root directory for each Intel SCC core, as shown below. All of those directories where placed under the `/shared` directory, so that they will be accessible from the Intel SCC cores.

```
/shared/ageo/rck00/shared/gentoo
/shared/ageo/rck01/shared/gentoo
. . . . .
/shared/ageo/rck46/shared/gentoo
/shared/ageo/rck47/shared/gentoo
```

For each core, we have also created one `to_gentoo.sh` script, which contains the corresponding `chroot` directory for this specific core. This way, we are able to enter the Gentoo Linux shell from each Intel SCC core. Each Gentoo Image has access only to the file system structure that is dedicated to this specific core.

```
ageo@mitsos:~$ ssh root@rck00
rck00:/root # /shared/ageo/rck00/shared/to_gentoo.sh
Now in myinit.sh on i586 (SCC)
4 Apr 12:35:48 ntpdate[146]: step time server 192.53.103.108
offset 124325786.277508 sec
rck00 / #
```

set_nat.sh is used to configure the MCPC as a NAT router for the Intel SCC cores, so that they can have access to the Gentoo Linux repositories and download all the software packages that are required so as to launch a Hadoop Cluster on the Intel SCC. The script assumes that `eth0` is the primary (WAN) connection, whereas the Intel SCC cores are connected via `eth1` (LAN), so it enables NAT routing from `eth1` to `eth0`. `set_nat.sh` has to be executed by an administrator with `root` privileges on the MCPC.

`set_nat.sh`:

```
#!/bin/sh
##First we flush our current rules
iptables -F
iptables -t nat -F

##Setup default policies to handle unmatched traffic
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP

##Copy and paste these examples ...
#export LAN=crb0 ## sccKit 1.3.0 and earlier: LAN via crbif
export LAN=eth1 ## sccKit 1.3.1: LAN via Ethernet port
export WAN=eth0

##Then we lock our services so they only work from the LAN
iptables -I INPUT 1 -i ${LAN} -j ACCEPT
iptables -I INPUT 1 -i lo -j ACCEPT
iptables -A INPUT -p UDP --dport bootps ! -i ${LAN} -j REJECT
iptables -A INPUT -p UDP --dport domain ! -i ${LAN} -j REJECT

##(Optional) Allow access to our ssh server from the WAN
##iptables -A INPUT -p TCP --dport ssh -i ${WAN} -j ACCEPT

##Drop TCP / UDP packets to privileged ports
#iptables -A INPUT -p TCP ! -i ${LAN} -d 0/0 --dport 0:1023 -j DROP
#iptables -A INPUT -p UDP ! -i ${LAN} -d 0/0 --dport 0:1023 -j DROP

##Finally we add the rules for NAT
iptables -I FORWARD -i ${LAN} -d 192.168.0.0/255.255.0.0 -j DROP
iptables -A FORWARD -i ${LAN} -s 192.168.0.0/255.255.0.0 -j ACCEPT
iptables -A FORWARD -i ${WAN} -d 192.168.0.0/255.255.0.0 -j ACCEPT
iptables -t nat -A POSTROUTING -o ${WAN} -j MASQUERADE
##Tell the kernel that ip forwarding is OK
echo 1 > /proc/sys/net/ipv4/ip_forward
for f in /proc/sys/net/ipv4/conf/*/rp_filter ; do echo 1 > $f ; done
```

5.1.2 Network Configuration

We have performed several modifications to the network configuration of the Intel SCC and the MCPC. First, we enabled the Intel SCC cores access the public Internet though the MCPC. After running the `set_nat.sh` script on the MCPC, we configured the MCPC as the default gateway and the default DNS server for the Intel SCC cores as follows. This way, the cores can have access to the Gentoo Linux software package repository.

```
rck00:/root # route add default gw 192.168.3.254
rck00:/root # echo "domain rck
> search rck in.rck.net
> nameserver 192.168.3.254
> " > /etc/resolv.conf
rck00:/root # /shared/ageo/rck00/shared/to_gentoo.sh
Now in myinit.sh on i586 (SCC)
4 Apr 12:35:48 ntpdate[146]: step time server 192.53.103.108
offset 124325786.277508 sec
rck00 / # ping -n www.mit.edu
PING e9566.dscb.akamaiedge.net (95.100.78.187) 56(84) bytes of data.
64 bytes from 95.100.78.187: icmp_req=1 ttl=54 time=30.0 ms
64 bytes from 95.100.78.187: icmp_req=2 ttl=54 time=30.0 ms
^C
--- e9566.dscb.akamaiedge.net ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 30.083/30.084/30.086/0.173 ms
```

In addition, we modified the routing table of the MCPC, so that it can access the internal virtual network interfaces of the Intel SCC cores (`mb0`) directly. The purpose of this was to manipulate the way that the file blocks are stored in the HDFS namespace. Since we run Apache Mahout at the MCPC and the communication of the MCPC to the Intel SCC cores takes place via the `emac0` interface, the communication between Mahout, the NameNode and the DataNode takes place through this interface. As a consequence, the mapping of file blocks to DataNodes in the `FsImage` of the NameNode uses the IP addresses that correspond to the `emac0` interfaces (`192.168.3.x, 1 <= x <= 48`). This makes the data transfer between DataNodes and TaskTrackers during the execution of a MapReduce job impossible, since the Intel SCC cores cannot communicate through the `emac0` intefaces.

To overcome this problem, we added static routes to the MCPC routing table, which route the traffic whose destination is the IP address `192.168.0.x` to the IP address `192.168.3.x`. This way, the HDFS file blocks are stored using the IP addresses of the `mb0` interfaces in the HDFS namespace and data transfer between DataNodes and TaskTrackers is made possible. This is achieved by executing the following script. This script has to be executed every time the MCPC is re-booted, by an administrator with `root` privileges on the MCPC.

`add_routes.sh`:

```
for i in {1..48}
```

```
do
    route add -host 192.168.0.$i gw 192.168.3.$i
done

ageo@mitsos:~$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.0.1     192.168.3.1     255.255.255.255 UGH       0 0          0 eth1
192.168.0.2     192.168.3.2     255.255.255.255 UGH       0 0          0 eth1
. . . . . .
192.168.0.47    192.168.3.47    255.255.255.255 UGH       0 0          0 eth1
192.168.0.48    192.168.3.48    255.255.255.255 UGH       0 0          0 eth1
147.102.37.0    0.0.0.0         255.255.255.0   U         0 0          0 eth0
192.168.3.0     0.0.0.0         255.255.255.0   U         0 0          0 eth1
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 crb0
192.168.0.0     192.168.1.1     255.255.255.0   UG        0 0          0 crb0
10.3.16.0       0.0.0.0         255.255.255.0   U         0 0          0 eth1
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 crb0
0.0.0.0         147.102.37.200  0.0.0.0         UG        0 0          0 eth0
```

5.1.3 Java Installation

In order to install Java on the Intel SCC cores, we utilized the Portage package manager which is offered by Gentoo Linux. Gentoo Linux provides several Java packages through its software package repository:

```
rck00 / # emerge --search jdk
Searching...
[ Results for search key : jdk ]
[ Applications found : 12 ]

*  dev-java/apple-jdk-bin [ Masked ]
      Latest version available: 1.6.0
      Latest version installed: [ Not Installed ]
      Size of files: 0 kB
      Homepage:      http://java.sun.com/j2se/1.6.0/
      Description:   Links to Apple's version of Sun's J2SE Development Kit
      License:       as-is

*  dev-java/db4o-jdk11
      Latest version available: 7.4
      Latest version installed: [ Not Installed ]
      Size of files: 312 kB
      Homepage:      http://www.db4o.com
      Description:   Core files for the object database for java
      License:       GPL-2

*  dev-java/db4o-jdk12
```

```
      Latest version available: 7.4
      Latest version installed: [ Not Installed ]
      Size of files: 89 kB
      Homepage:      http://www.db4o.com
      Description:   Core files for the object database for java
      License:       GPL-2

*  dev-java/db4o-jdk5
      Latest version available: 7.4
      Latest version installed: [ Not Installed ]
      Size of files: 63 kB
      Homepage:      http://www.db4o.com
      Description:   Core files for the object database for java
      License:       GPL-2

*  dev-java/diablo-jdk [ Masked ]
      Latest version available: 1.6.0.07.02
      Latest version installed: [ Not Installed ]
      Size of files: 62,591 kB
      Homepage:      http://www.FreeBSDFoundation.org/downloads/java.shtml
      Description:   Java Development Kit
      License:       sun-bcla-java-vm

*  dev-java/gcj-jdk [ Masked ]
      Latest version available: 4.5.1
      Latest version installed: [ Not Installed ]
      Size of files: 0 kB
      Homepage:      http://www.gentoo.org/
      Description:   Java wrappers around GCJ
      License:       GPL-2

*  dev-java/hp-jdk-bin [ Masked ]
      Latest version available: 1.6.0.05
      Latest version installed: [ Not Installed ]
      Size of files: 231,550 kB
      Homepage:      http://www.hp.com/go/java
      Description:   HP JDK/JRE and Plug-In
      License:       HP-JDKJRE6

*  dev-java/ibm-jdk-bin
      Latest version available: 1.6.0.8_p1-r1
      Latest version installed: 1.6.0.8_p1-r1
      Size of files: 374,278 kB
      Homepage:      http://www.ibm.com/developerworks/java/jdk/
      Description:   IBM Java SE Development Kit
      License:       IBM-J1.6
```

```
*  dev-java/jrockit-jdk-bin [ Masked ]
      Latest version available: 1.5.0.14
      Latest version installed: [ Not Installed ]
      Size of files: 241,996 kB
      Homepage:      http://commerce.bea.com/products/weblogicjrockit/jrockit_prod_fam.jsp
      Description:   BEA WebLogic's J2SE Development Kit
      License:       jrockit

*  dev-java/sun-jdk
      Latest version available: 1.6.0.22
      Latest version installed: [ Not Installed ]
      Size of files: 163,745 kB
      Homepage:      http://java.sun.com/javase/6/
      Description:   Sun's Java SE Development Kit
      License:       dlj-1.1

*  java-virtuals/jdk-with-com-sun
      Latest version available: 20100419
      Latest version installed: [ Not Installed ]
      Size of files: 0 kB
      Homepage:      http://www.gentoo.org
      Description:   Virtual ebuilds that require internal com.sun classes from a JDK
      License:       GPL-2

*  virtual/jdk
      Latest version available: 1.6.0
      Latest version installed: 1.6.0
      Size of files: 0 kB
      Homepage:
      Description:   Virtual for JDK
      License:
```

In our setup, we have selected the JDK that is provided by IBM (`dev-java/ibm-jdk-bin`). In order to install this package, we execute the following command. The `Portage` package manager then takes care of downloading all the necessary source files, extracting compiling and installing them to the appropriate directories.

```
rck00 / # emerge dev-java/ibm-jdk-bin
. . . . . .
rck00 / # java -version
java version "1.6.0"
Java(TM) SE Runtime Environment (build pxi3260sr8fp1-20100624_01(SR8 FP1))
IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 Linux x86-32
jvmxi3260sr8ifx-20100609_59383 (JIT enabled, AOT enabled)
J9VM - 20100609_059383
JIT  - r9_20100401_15339ifx2
```

```
GC   - 20100308_AA)
JCL  - 20100624_01
```

Gentoo Linux masks several software packages that are available through its repository, due to license mismatches. The installation of masked packages is not allowed. In order to allow the installation of masked software packages, the following line has to be added to the `/etc/make.conf` file:

```
ACCEPT_LICENSE="*"
```

5.1.4 SSH Communication Between Cluster Nodes

During the start up of a Hadoop Cluster, the master node (where the NameNode and the JobTracker are executed) is responsible for starting the DataNode and TaskTracker daemons in all the remote nodes of the cluster. In order to achieve this, the master node connects using SSH to all the slave nodes. As a consequence, it is essential that the master node can connect to all the slave nodes with SSH using public key authentication. In addition, since in our setup we run the Hadoop daemons on the Gentoo Image and not the Intel SCC Linux, the master node has to be able to connect directly to the Gentoo Image, rather than the Intel SCC Linux Image. In order to meet the above requirements, we have created a second SSH server than runs on each core, listens to the port `1234` and enables the clients to open an interactive Gentoo Linux shell remotely.

We have modified the default SSH server configuration file (`/etc/sshd_config`), which is located in the Intel SCC Linux filesystem, in order to provide the functionality described above. For each cluster node, we configured overrides for the default values of the `Port` and `ChrootDirectory` properties as shown below, for core `rck00`. For different cores, the value for `ChrootDirectory` is set accordingly. The `ChrootDirectory` of each Gentoo Linux instance should be owned by `root`.

```
Port 1234
ChrootDirectory /shared/ageo/rck00/shared/gentoo
```

In order to enable SSH communication using public key authentication, we created a public and private key pair in node `rck00`, using the command

```
ssh-keygen -t rsa
```

This command generates a public key which is stored in `/root/.ssh/id_rsa.pub` and a private key which is stored in `/root/.ssh/id_rsa`. Both of the paths refer to the Gentoo Linux filesystem. Since the SSH communication takes place between the Gentoo Linux shells of the Intel SCC cores, the private key has to be copied in the `/root/.ssh` directory of the Gentoo Linux filesystem of each core and the public key should be added to the `/root/.ssh/authorized_keys` file of the Intel SCC Linux filesystem of each core. The configuration of our SSH server is placed in `/etc/sshd_config1` file, in the Intel SCC Linux of each core. The SSH server can be started by executing the `sshd` command. The Gentoo Linux shell of the cores is then accessible through both the MCPC and the other cores, without a password being requested.

```
rck00:/root # /usr/sbin/sshd -f /etc/sshd_config1

rck00 / # ssh -p 1234 root@rck27
rck27 ~ #

ageo@mitsos:~$ ssh -p 1234 root@rck00
rck00 ~ #
```

5.1.5 Hadoop Runtime Environment Setup

In order to setup the runtime environment for each Intel SCC core, we have created the following script for each core, which is called `start.sh`. This code presented below regards nodes `rck00` and `rck22`. This script copies the public key we have generated to the `/root/.ssh/authorized_keys` directory of each core, starts the SSH server that was described in the previous section, mounts the `/proc` directory of the Intel SCC Linux to the corresponding directory of the Gentoo Linux directory structure and copies the files under `/dev` of Intel SCC Linux to the corresponding directory of the Gentoo Linux directory structure. This script is placed in `/shared/ageo/rck00` directory so that it can be accessed by the core. Similar scripts are used for the other cores, which mount the `/proc` directory into the corresponding directory of the Gentoo Linux filesystem and are placed in the directory that is dedicated for the specific core. For example, the `start.sh` script for `rck22` is placed in the `/shared/ageo/rck22` directory and mounts the `/proc` directory into `/shared/ageo/rck22/shared/gentoo/proc`.

`start.sh` for `rck00` :

```
cat /shared/ageo/rck00/shared/gentoo/root/.ssh/id_rsa.pub
    >> /root/.ssh/authorized_keys
cp /shared/ageo/rck00/sshd_config1 /etc
/usr/sbin/sshd -f /etc/sshd_config1
mount -t proc proc /shared/ageo/rck00/shared/gentoo/proc
cp -r /dev/* /shared/ageo/rck00/shared/gentoo/dev
```

`start.sh` for `rck22` :

```
cat /shared/ageo/rck00/shared/gentoo/root/.ssh/id_rsa.pub
    >> /root/.ssh/authorized_keys
cp /shared/ageo/rck22/sshd_config1 /etc
/usr/sbin/sshd -f /etc/sshd_config1
mount -t proc proc /shared/ageo/rck22/shared/gentoo/proc
cp -r /dev/* /shared/ageo/rck22/shared/gentoo/dev
```

5.2 Hadoop Cluster Topologies on the Intel SCC

This section describes the process that has to be followed so that a Hadoop Cluster is launched on the Intel SCC. Initially, the principal design choices we have made are presented and explained. Subsequently, the list of configuration parameters we

have defined, in order to deploy four Hadoop Cluster topologies on the Intel SCC, is presented. Those topologies consist of 16, 24, 32 and 48 nodes each. The section concludes with the description of a failover mechanism we have developed, which is crucial for the stability of the HDFS clusters that are deployed on the Intel SCC, during the execution of MapReduce jobs.

5.2.1 Design Choices and Platform Limitations

This section states the basic design choices that we have made and the failover mechanisms we have developed, regarding the deployment of Hadoop Cluster topologies on the Intel SCC, with respect to the architectural characteristics of the Intel SCC and the physical limitations that are imposed by this platform.

The most devastating limitation that our setup suffers from, is the lack of sufficient main memory space for each core. Since 32 GB of memory are connected to the Intel SCC die through the memory controllers, only 640 MB of main memory is available for each core, thus Hadoop Cluster node. After testing several MapReduce jobs for various values of maximum Java Heap Size, we decided that a value of `-Xmx128m` is appropriate for the Hadoop Dameons and the Child JVMs, so that the Java processes will neither be killed by the OS (or make the core freeze), nor be terminated with a `java.lang.OutOfMemoryError` exception.

As a consequence of the above, it is essential that the typical Hadoop Cluster deployment strategy, which supposes that DataNodes and TaskTrackers run on the same cluster nodes, is dropped. In our setup, we have configured DataNodes and TaskTrackers to run on different cores and have explicitly divided the on-die cluster to Hadoop Racks, so that the rack locality-aware scheduling mechanism of MapReduce is not thrown away. In addition, the typical MapReduce framework configuration determines that the Reduce phase of a MapReduce job is triggered after only the 5% of the Map phase has completed. That is, reduce tasks are scheduled for cluster nodes where child JVMs are already executing Map tasks. With so little main memory available, this is evidently impossible. As a result, we have configured the Reduce phase to start after the 100% of the Map phase has completed successfully. For the same reason, the TaskTracker nodes are configured to run only one Map or Reduce task at a time. Moreover, we significantly reduced the file block size from the default value of 64 MB to 4 MB. Since a Map task is scheduled for each `InputSplit` we made that decision so that to reduce the I/O load for the DataNode cores.

Another design choice we made regards the placement of the cluster nodes on the Intel SCC die. We decided to locate the DataNodes at the edge of the Intel SCC die, that is in the cores of the tiles with either `x=0` or `x=5` in the Intel SCC core layout. This decision was driven by the fact that since DataNodes are responsible for storing file blocks in their local filesystem, they are expected to have heavier I/O load during the execution of a MapReduce job. As a consequence, placing the DataNodes closer to the memory controllers of the Intel SCC die, reduces the latency of their frequent

accesses to the NFS.

Finally, we noticed that the presence of high I/O load and very low free main memory space causes some cores to freeze and become unreachable rather frequently. That is, at least one core freezing during the execution of a MapReduce job is the norm and not the exception on the Intel SCC. Configuring a replication factor greater than 1 looked a reasonable solution at first, since this is the out-of-the-box mechanism Hadoop provides so as to ensure data is always available. However, a replication factor greater than 1, forces the DataNodes replicate under-replicated file blocks, thus significantly increasing their I/O load, which is the reason that causes them to fail in the first place. As a consequence, we observed that configuring a replication factor greater than 1 causes the DataNode cores to freeze one after the other, rendering the completion of the MapReduce job impossible.

In order to tackle this problem, we followed a completely different approach. We implemented a node-failover watchdog script, which pings the Intel SCC cores periodically so as to ensure that all of them are up and running. In case a core is observed to be unreachable, that is `ping` receives no response packet, the SCC Linux is booted on the core immediately, the Hadoop Runtime Environment is set up and the corresponding Hadoop Daemon is started. This sequence of actions is also triggered if a core is reachable by ping, but the Hadoop Daemon it is supposed to run has been killed by the OS. This way, we have overcome the complication of cores freezing frequently and have ensured the forward progress of MapReduce jobs despite the presence of this situation. Map or Reduce tasks may be terminated with exceptions during a core (especially a DataNode because of data being unavailable) is rebooted, but the retry mechanism of MapReduce guarantees that those tasks will be completed successfully once they are re-executed and the Hadoop Cluster is in a stable state.

5.2.2 The hadoop-env.sh Configuration Script

This section describes the `hadoop-env.sh` configuration script that we have used in our setup.

`hadoop-env.sh :`

```
# Set Hadoop-specific environment variables here.

# The only required environment variable is JAVA_HOME.  All others are
# optional.  When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use.  Required.
export JAVA_HOME=/opt/ibm-jdk-bin-1.6.0.8_p1

# Extra Java CLASSPATH elements.  Optional.
```

```
# export HADOOP_CLASSPATH=

# The maximum amount of heap to use, in MB. Default is 1000.
export HADOOP_HEAPSIZE=128

# Extra Java runtime options.  Empty by default.
# export HADOOP_OPTS=-server

# Command specific options appended to HADOOP_OPTS when specified
export HADOOP_NAMENODE_OPTS=
            "-Dcom.sun.management.jmxremote $HADOOP_NAMENODE_OPTS"
export HADOOP_SECONDARYNAMENODE_OPTS=
            "-Dcom.sun.management.jmxremote $HADOOP_SECONDARYNAMENODE_OPTS"
export HADOOP_DATANODE_OPTS=
            "-Dcom.sun.management.jmxremote  "
export HADOOP_BALANCER_OPTS=
            "-Dcom.sun.management.jmxremote $HADOOP_BALANCER_OPTS"
export HADOOP_JOBTRACKER_OPTS=
            "-Dcom.sun.management.jmxremote $HADOOP_JOBTRACKER_OPTS"
# export HADOOP_TASKTRACKER_OPTS=
# The following applies to multiple commands (fs, dfs, fsck, distcp etc)
# export HADOOP_CLIENT_OPTS

# Extra ssh options.  Empty by default.
export HADOOP_SSH_OPTS="-p 1234 -l root "

# Where log files are stored.  $HADOOP_HOME/logs by default.
# export HADOOP_LOG_DIR=${HADOOP_HOME}/logs

# File naming remote slave hosts.  $HADOOP_HOME/conf/slaves by default.
# export HADOOP_SLAVES=${HADOOP_HOME}/conf/slaves

# host:path where hadoop code should be rsync'd from.  Unset by default.
# export HADOOP_MASTER=master:/home/$USER/src/hadoop

# Seconds to sleep between slave commands.  Unset by default.  This
# can be useful in large clusters, where, e.g., slave rsyncs can
# otherwise arrive faster than the master can service them.
# export HADOOP_SLAVE_SLEEP=0.1

# The directory where pid files are stored. /tmp by default.
# export HADOOP_PID_DIR=/var/hadoop/pids

# A string representing this instance of hadoop. $USER by default.
# export HADOOP_IDENT_STRING=$USER

# The scheduling priority for daemon processes.  See 'man nice'.
```

```
# export HADOOP_NICENESS=10
export HADOOP_ROOT_LOGGER="ERROR,console"
```

In this configuration script we have set the `JAVA_HOME` environment variable to the directory path where Java is installed in each Gentoo Linux image in the Intel SCC cores. In the `HADOOP_SSH_OPTS` variable we have determined that ssh connections should be attempted at port `1234` as the `root` user, so as to utilize the SSH server we described in a previous section of this chapter. Finally, we have disabled Hadoop Logging, for log messages which are marked with severity lower than `ERROR` by `log4j`, by setting the `HADOOP_ROOT_LOGGER` variable, so as to prevent CPU cycles and I/O bandwidth being wasted during the execution of a MapReduce job. This version of `hadoop-env.sh` is used by all of the cluster topologies we describe in this thesis.

5.2.3 The core-site.xml Configuration File

This section describes the `core-site.xml` configuration file that we have used in our setup.

`core-site.xml :`

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://192.168.0.1:54310</value>
    </property>
    <property>
        <name>topology.script.file.name</name>
        <value>/home/ageo/hadoop-topology.sh</value>
    </property>
</configuration>
```

In this configuration file, we have defined the URL of the NameNode in the `fs.default.name` property. In addition, we have determined the `topology.script.file.name`, which assigns cluster nodes to Hadoop Racks. This script receives the IP address of a cluster node as an input and provides the rack name it is assigned to as the output. This version of `core-site.xml` and `hadoop-topology.sh` is used by all of the cluster topologies we describe in this thesis. The complete code of `hadoop-topology.sh` is available in Appendix A.

`hadoop-topology.sh :`

```
if [ "$1" = "192.168.0.1" ]
then
    echo "/rack00";
fi
```

```
if [ "$1" = "192.168.0.2" ]
then
    echo "/rack01";
fi
. . . . . .
if [ "$1" = "192.168.0.47" ]
then
    echo "/rack14";
fi
if [ "$1" = "192.168.0.48" ]
then
    echo "/rack15";
fi
```

5.2.4 The hdfs-site.xml Configuration File

This section describes the `hdfs-site.xml` configuration file that we have used in our setup.

`hdfs-site.xml` for 16-node and 24-node cluster topology:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.block.size</name>
        <value>4194304</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/home/ageo/tmp_dir-topo16-24</value>
    </property>
    <property>
        <name>dfs.name.dir</name>
        <value>/home/ageo/hdfsnames-topo16-24</value>
    </property>
    <property>
        <name>dfs.data.dir</name>
        <value>/home/ageo/hdfsdata-topo16-24</value>
    </property>
</configuration>
```

`hdfs-site.xml` for 32-node and 48-node cluster topology:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.block.size</name>
        <value>4194304</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/home/ageo/tmp_dir-topo32-48</value>
    </property>
    <property>
        <name>dfs.name.dir</name>
        <value>/home/ageo/hdfsnames-topo32-48</value>
    </property>
    <property>
        <name>dfs.data.dir</name>
        <value>/home/ageo/hdfsdata-topo32-48</value>
    </property>
</configuration>
```

The file block replication factor is defined by the `dfs.replication` property and the file block size by the `dfs.block.size` property. The `dfs.name.dir` defines the local file system directory, where the NameNode should store the name table (`FsImage`) and the `dfs.data.dir` defines the directory where the DataNodes should store file data blocks, in their local file systems. The reason why different values have been set for the last two properties, depending on the Hadoop Cluster topology, is explained in a later section.

5.2.5 The mapred-site.xml Configuration File

This section describes the `mapred-site.xml` configuration file that we have used in our setup.

`mapred-site.xml` for 16-node topology:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>192.168.0.1:54311</value>
```

```
    </property>
    <property>
        <name>mapred.reduce.tasks</name>
        <value>8</value>
    </property>
    <property>
        <name>mapred.tasktracker.map.tasks.maximum</name>
        <value>1</value>
    </property>
    <property>
        <name>mapred.tasktracker.reduce.tasks.maximum</name>
        <value>1</value>
    </property>
    <property>
        <name>mapred.reduce.slowstart.completed.maps</name>
        <value>1.00</value>
    </property>
    <property>
        <name>mapred.child.java.opts</name>
        <value>-Xmx128m</value>
    </property>
    <property>
        <name>mapred.task.timeout</name>
        <value>3600000</value>
    </property>
</configuration>
```

`mapred-site.xml` for 24-node and 32-node topology:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>192.168.0.1:54311</value>
    </property>
    <property>
        <name>mapred.reduce.tasks</name>
        <value>16</value>
    </property>
    <property>
        <name>mapred.tasktracker.map.tasks.maximum</name>
        <value>1</value>
    </property>
    <property>
        <name>mapred.tasktracker.reduce.tasks.maximum</name>
        <value>1</value>
```

```
    </property>
    <property>
        <name>mapred.reduce.slowstart.completed.maps</name>
        <value>1.00</value>
    </property>
    <property>
        <name>mapred.child.java.opts</name>
        <value>-Xmx128m</value>
    </property>
    <property>
        <name>mapred.task.timeout</name>
        <value>3600000</value>
    </property>
</configuration>
```

`mapred-site.xml` for 48-node topology:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>192.168.0.1:54311</value>
    </property>
    <property>
        <name>mapred.reduce.tasks</name>
        <value>32</value>
    </property>
    <property>
        <name>mapred.tasktracker.map.tasks.maximum</name>
        <value>1</value>
    </property>
    <property>
        <name>mapred.tasktracker.reduce.tasks.maximum</name>
        <value>1</value>
    </property>
    <property>
        <name>mapred.reduce.slowstart.completed.maps</name>
        <value>1.00</value>
    </property>
    <property>
        <name>mapred.child.java.opts</name>
        <value>-Xmx128m</value>
    </property>
    <property>
        <name>mapred.task.timeout</name>
        <value>3600000</value>
```

```
    </property>
</configuration>
```

The IP address and the TCP port the JobTracker listens to is defined in `mapred.job.tracker` property. The maximum number of Map and Reduce tasks that can be run by a TaskTracker at a time are defined in `mapred.tasktracker.map.tasks.maximum` and `mapred.tasktracker.reduce.tasks.maximum` properties respectively. The percentage of Map tasks that have to be completed successfully before the Reduce phase starts is defined in the `mapred.reduce.slowstart.completed.maps` property. The Java Heap Size used by the Child JVMs that execute the Map and Reduce tasks is defined in `mapred.child.java.opts` property. Other JVM command line arguments can be defined in this property as well. The `mapred.task.timeout` determines the time interval in milliseconds that has to pass, for the JobTracker to kill a Map or Reduce task, if this specific task has not reported its status during that time. Finally, the `mapred.reduce.tasks` determines the number of reduce tasks that have to be executed by a MapReduce job. The value of this property is set equal to the number of TaskTracker nodes, for each Hadoop Cluster topology we have deployed.

5.2.6 The masters Configuration file

The masters configuration file contains the IP address of the master node, which is `192.168.0.1`. This version of `masters` is used by all of the cluster topologies we describe in this thesis.

```
masters :

192.168.0.1
```

5.2.7 16-Node Cluster Topology

This section describes the 16-Node HDFS Cluster Topology we have deployed on the Intel SCC. It contains one master node, where the NameNode and the JobTracker are executed and 15 slave nodes, which break down to 7 DataNodes and 8 TaskTrackers. Figure 5.1 illustrates the layout of the Hadoop Cluster nodes on the Intel SCC die.

We have created two directories under the Hadoop installation root directory, `conf-hdfs-topo16` and `conf-mapred-topo16`, where we have placed the configuration files that contain the properties that will be loaded when the DataNodes and the TaskTrackers are started, respectively. The content of those two directories is identical, except for the `slaves` file, which contains the IP addresses of the nodes where the DataNodes and the TaskTrackers will be executed in each case, as shown below. Since in the 16-Node and the 24-Node cluster topologies the DataNodes run on the same Intel SCC cores, those two cluster topologies share the same `dfs.name.dir` and `dfs.data.dir` directories, where the `FsImage` is stored by the NameNode and the file data blocks are stored by the DataNodes, respectively. That is, those cluster topologies share the same HDFS namespace and differ only in the number of TaskTracker nodes they employ, which execute the Map and Reduce tasks of MapReduce jobs.

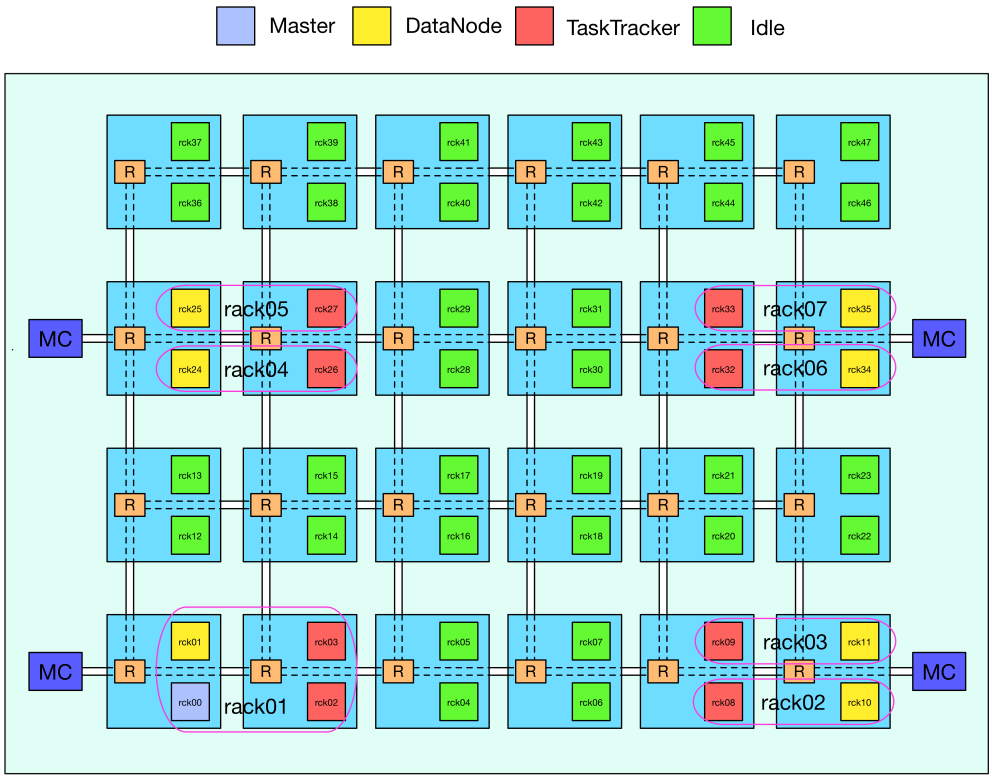


Figure 5.1: 16-Node Hadoop Cluster on the Intel SCC

`conf-hdfs-topo16/slaves :`

```
192.168.0.2
192.168.0.11
192.168.0.12
192.168.0.25
192.168.0.26
192.168.0.35
192.168.0.36
```

`conf-mapred-topo16/slaves :`

```
192.168.0.3
192.168.0.4
192.168.0.9
192.168.0.10
192.168.0.27
192.168.0.28
192.168.0.33
192.168.0.34
```

In order to launch the 16-Node Hadoop Cluster on the Intel SCC, two scripts have to be executed. Firstly, `start_cluster_topo16.sh` has to be invoked from the MCPC so as

to setup the runtime environment for HDFS. It is located in our home directory in the MCPC. Afterwards, `bin/start-all-topo16.sh` has to be executed from the master node (`rck00`) so as to start the Hadoop Daemons on the Intel SCC cores and launch the HDFS cluster. The above path is relative to the Hadoop installation root directory. The Hadoop Cluster can be shut down by invoking the `bin/stop-all-topo16.sh` script from the master node. The above path is relative to the Hadoop installation root directory.

`start_cluster_topo16.sh` :

```
RESOLV_CONF="domain rck
             search rck in.rck.net
             nameserver  192.168.3.254"
for i in {0,1,2,3,8,9}
do
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
for i in {10,11,24,25,26,27,32,33,34,35}
do
        ssh root@rck$i  "shared/ageo/rck$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
```

`bin/start-all-topo16.sh` :

```
bin=‘dirname "$0"‘
bin=‘cd "$bin"; pwd‘

. "$bin"/hadoop-config.sh

# start dfs daemons
"$bin"/start-dfs.sh --config /home/ageo/hadoop-0.20.2/conf-hdfs-topo16

# start mapred daemons
"$bin"/start-mapred.sh --config /home/ageo/hadoop-0.20.2/conf-mapred-topo16
```

`bin/stop-all-topo16.sh` :

```
bin=‘dirname "$0"‘
bin=‘cd "$bin"; pwd‘

. "$bin"/hadoop-config.sh

"$bin"/stop-mapred.sh --config /home/ageo/hadoop-0.20.2/conf-mapred-topo16
"$bin"/stop-dfs.sh --config /home/ageo/hadoop-0.20.2/conf-hdfs-topo16
```

5.2.8 24-Node Cluster Topology

This section describes the 24-Node HDFS Cluster Topology we have deployed on the Intel SCC. It contains one master node, where the NameNode and the JobTracker are executed and 23 slave nodes, which break down to 7 DataNodes and 16 TaskTrackers. Figure 5.2 illustrates the layout of the Hadoop Cluster nodes on the Intel SCC die.

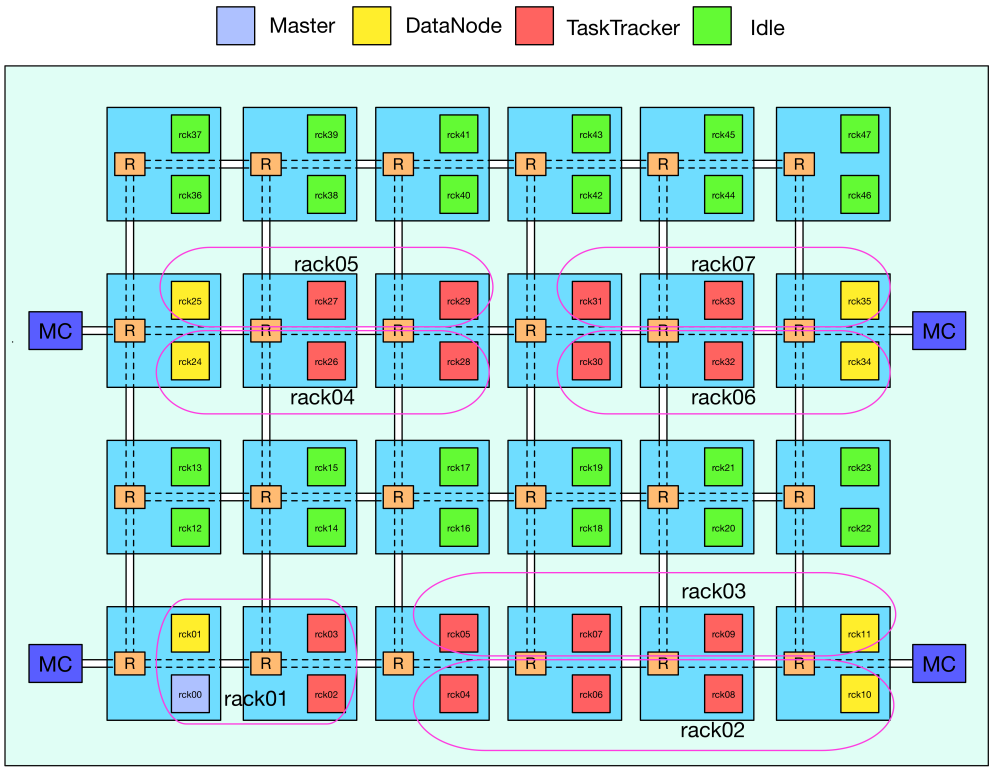


Figure 5.2: 24-Node Hadoop Cluster on the Intel SCC

We have created two directories under the Hadoop installation root directory, `conf-hdfs-topo24` and `conf-mapred-topo24`, where we have placed the configuration files that contain the properties that will be loaded when the DataNodes and the TaskTrackers are started, respectively. The content of those two directories is identical, except for the `slaves` file, which contains the IP addresses of the nodes where the DataNodes and the TaskTrackers will be executed in each case, as shown below. Since in the 16-Node and the 24-Node cluster topologies the DataNodes run on the same Intel SCC cores, those two cluster topologies share the same `dfs.name.dir` and `dfs.data.dir` directories, where the `FsImage` is stored by the NameNode and the file data blocks are stored by the DataNodes, respectively. That is, those cluster topologies share the same HDFS namespace and differ only in the number of TaskTracker nodes they employ, which execute the Map and Reduce tasks of MapReduce jobs.

`conf-hdfs-topo24/slaves :`

```
192.168.0.2
```

```
192.168.0.11
192.168.0.12
192.168.0.25
192.168.0.26
192.168.0.35
192.168.0.36
```

`conf-mapred-topo24/slaves` :

```
192.168.0.3
192.168.0.4
192.168.0.5
192.168.0.6
192.168.0.8
192.168.0.7
192.168.0.9
192.168.0.10
192.168.0.27
192.168.0.28
192.168.0.29
192.168.0.30
192.168.0.31
192.168.0.32
192.168.0.33
192.168.0.34
```

In order to launch the 24-Node Hadoop Cluster on the Intel SCC, two scripts have to be executed. Firstly, `start_cluster_topo24.sh` has to be invoked from the MCPC so as to setup the runtime environment for HDFS. It is located in our home directory in the MCPC. Afterwards, `bin/start-all-topo24.sh` has to be executed from the master node (`rck00`) so as to start the Hadoop Daemons on the Intel SCC cores and launch the HDFS cluster. The above path is relative to the Hadoop installation root directory. The Hadoop Cluster can be shut down by invoking the `bin/stop-all-topo24.sh` script from the master node. The above path is relative to the Hadoop installation root directory.

`start_cluster_topo24.sh` :

```
RESOLV_CONF="domain rck
             search rck in.rck.net
             nameserver  192.168.3.254"
for i in {0..9}
do
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
for i in {10,11}
do
```

```
        ssh root@rck$i  "shared/ageo/rck$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
for i in {24..35}
do
        ssh root@rck$i  "shared/ageo/rck$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
```

bin/start-all-topo24.sh :

```
bin=‘dirname "$0"‘
bin=‘cd "$bin"; pwd‘

. "$bin"/hadoop-config.sh

# start dfs daemons
"$bin"/start-dfs.sh --config /home/ageo/hadoop-0.20.2/conf-hdfs-topo24

# start mapred daemons
"$bin"/start-mapred.sh --config /home/ageo/hadoop-0.20.2/conf-mapred-topo24
```

bin/stop-all-topo24.sh :

```
bin=‘dirname "$0"‘
bin=‘cd "$bin"; pwd‘

. "$bin"/hadoop-config.sh

"$bin"/stop-mapred.sh --config /home/ageo/hadoop-0.20.2/conf-mapred-topo24
"$bin"/stop-dfs.sh --config /home/ageo/hadoop-0.20.2/conf-hdfs-topo24
```

5.2.9 32-Node Cluster Topology

This section describes the 32-Node HDFS Cluster Topology we have deployed on the Intel SCC. It contains one master node, where the NameNode and the JobTracker are executed and 31 slave nodes, which break down to 15 DataNodes and 16 TaskTrackers. Figure 5.3 illustrates the layout of the Hadoop Cluster nodes on the Intel SCC die.

We have created two directories under the Hadoop installation root directory, `conf-hdfs-topo32` and `conf-mapred-topo32`, where we have placed the configuration files that contain the properties that will be loaded when the DataNodes and the TaskTrackers are started, respectively. The content of those two directories is identical, except for the `slaves` file, which contains the IP addresses of the nodes where the DataNodes and the TaskTrackers will be executed in each case, as shown below. Since in the 32-Node and the 48-Node cluster topologies the DataNodes run on the same Intel SCC cores, those two cluster topologies share the same `dfs.name.dir` and `dfs.data.dir`

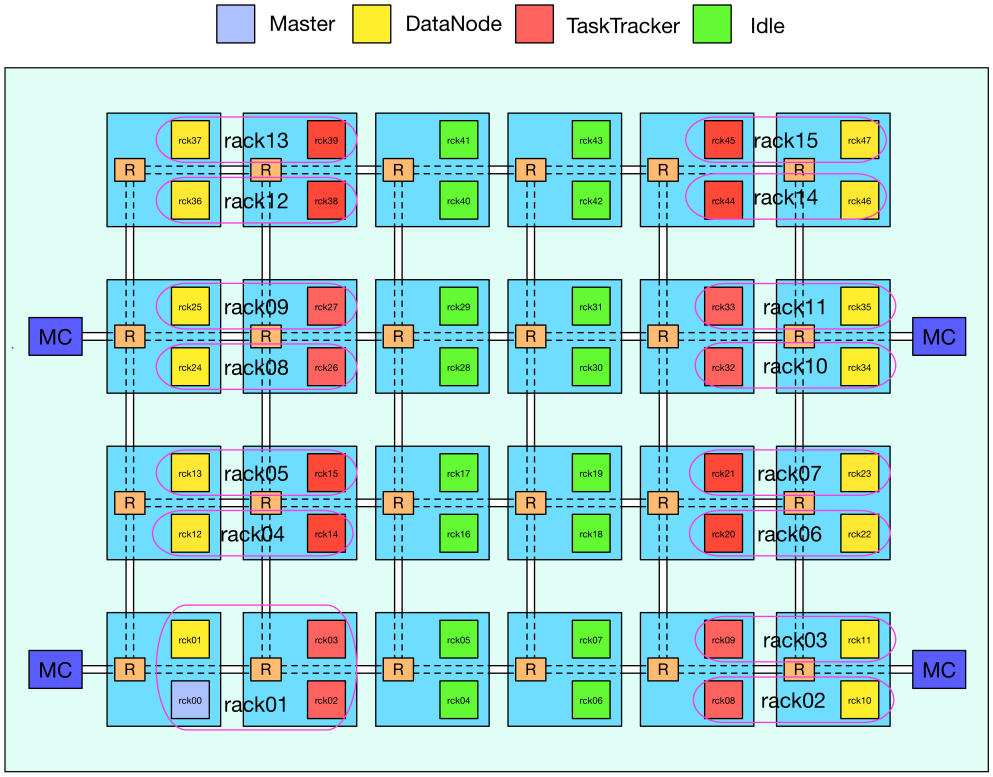


Figure 5.3: 32-Node Hadoop Cluster on the Intel SCC

directories, where the `FsImage` is stored by the NameNode and the file data blocks are stored by the DataNodes, respectively. That is, those cluster topologies share the same HDFS namespace and differ only in the number of TaskTracker nodes they employ, which execute the Map and Reduce tasks of MapReduce jobs.

`conf-hdfs-topo32/slaves :`

```
192.168.0.2
192.168.0.11
192.168.0.12
192.168.0.13
192.168.0.14
192.168.0.22
192.168.0.23
192.168.0.25
192.168.0.26
192.168.0.35
192.168.0.36
192.168.0.37
192.168.0.38
```

`conf-mapred-topo32/slaves :`

```
192.168.0.3
```

```
192.168.0.4
192.168.0.9
192.168.0.10
192.168.0.15
192.168.0.16
192.168.0.21
192.168.0.22
192.168.0.27
192.168.0.28
192.168.0.33
192.168.0.34
192.168.0.39
192.168.0.40
192.168.0.45
192.168.0.46
```

In order to launch the 32-Node Hadoop Cluster on the Intel SCC, two scripts have to be executed. Firstly, `start_cluster_topo32.sh` has to be invoked from the MCPC so as to setup the runtime environment for HDFS. It is located in our home directory in the MCPC. Afterwards, `bin/start-all-topo32.sh` has to be executed from the master node (`rck00`) so as to start the Hadoop Daemons on the Intel SCC cores and launch the HDFS cluster. The above path is relative to the Hadoop installation root directory. The Hadoop Cluster can be shut down by invoking the `bin/stop-all-topo32.sh` script from the master node. The above path is relative to the Hadoop installation root directory.

`start_cluster_topo32.sh` :

```
RESOLV_CONF="domain rck
             search rck in.rck.net
             nameserver  192.168.3.254"
for i in {0..3}
do
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
for i in {8..9}
do
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
for i in {10..15}
do
        ssh root@rck$i  "shared/ageo/rck$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
```

```
done
for i in {20..27}
do
        ssh root@rck$i  "shared/ageo/rck$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
for i in {32..39}
do
        ssh root@rck$i  "shared/ageo/rck$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
for i in {44..47}
do
        ssh root@rck$i  "shared/ageo/rck$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
```

bin/start-all-topo32.sh :

```
bin=‘dirname "$0"‘
bin=‘cd "$bin"; pwd‘

. "$bin"/hadoop-config.sh

# start dfs daemons
"$bin"/start-dfs.sh --config /home/ageo/hadoop-0.20.2/conf-hdfs-topo32

# start mapred daemons
"$bin"/start-mapred.sh --config /home/ageo/hadoop-0.20.2/conf-mapred-topo32
```

bin/stop-all-topo32.sh :

```
bin=‘dirname "$0"‘
bin=‘cd "$bin"; pwd‘

. "$bin"/hadoop-config.sh

"$bin"/stop-mapred.sh --config /home/ageo/hadoop-0.20.2/conf-mapred-topo32
"$bin"/stop-dfs.sh --config /home/ageo/hadoop-0.20.2/conf-hdfs-topo32
```

5.2.10 48-Node Cluster Topology

This section describes the 48-Node HDFS Cluster Topology we have deployed on the Intel SCC. It contains one master node, where the NameNode and the JobTracker are executed and 47 slave nodes, which break down to 15 DataNodes and 32 TaskTrackers. Figure 5.4 illustrates the layout of the Hadoop Cluster nodes on the Intel SCC die.

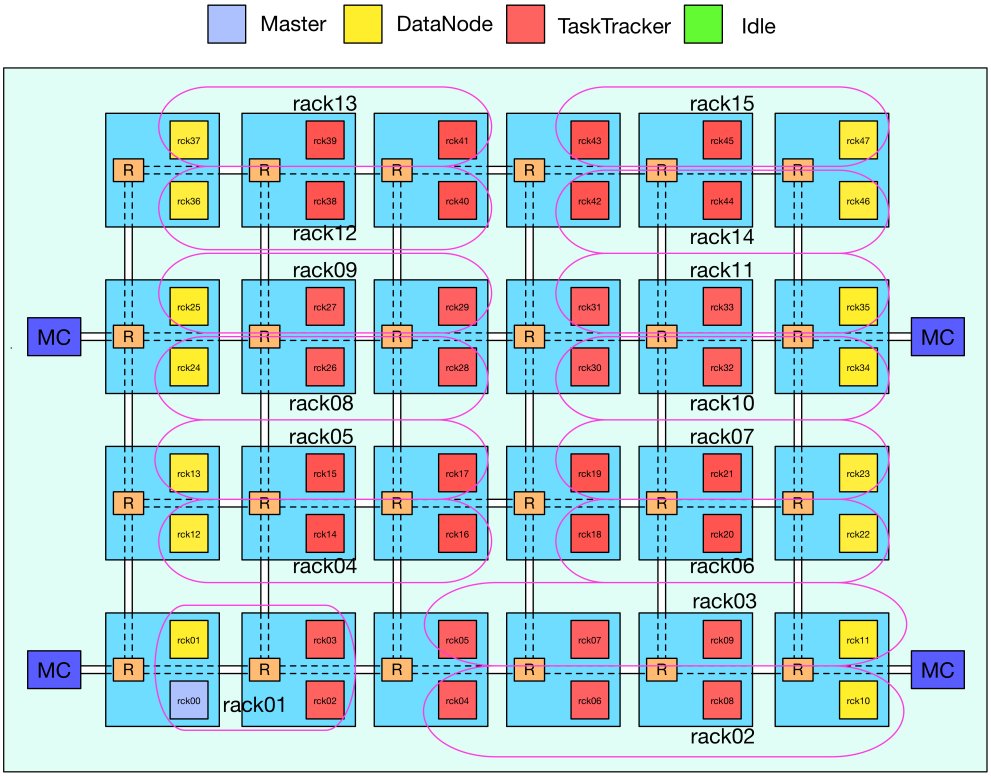


Figure 5.4: 48-Node Hadoop Cluster on the Intel SCC

We have created two directories under the Hadoop installation root directory, `conf-hdfs-topo48` and `conf-mapred-topo48`, where we have placed the configuration files that contain the properties that will be loaded when the DataNodes and the TaskTrackers are started, respectively. The content of those two directories is identical, except for the `slaves` file, which contains the IP addresses of the nodes where the DataNodes and the TaskTrackers will be executed in each case, as shown below. Since in the 32-Node and the 48-Node cluster topologies the DataNodes run on the same Intel SCC cores, those two cluster topologies share the same `dfs.name.dir` and `dfs.data.dir` directories, where the `FsImage` is stored by the NameNode and the file data blocks are stored by the DataNodes, respectively. That is, those cluster topologies share the same HDFS namespace and differ only in the number of TaskTracker nodes they employ, which execute the Map and Reduce tasks of MapReduce jobs.

`conf-hdfs-topo48/slaves :`

```
192.168.0.2
192.168.0.11
192.168.0.12
192.168.0.13
192.168.0.14
192.168.0.22
192.168.0.23
```

```
192.168.0.25
192.168.0.26
192.168.0.35
192.168.0.36
192.168.0.37
192.168.0.38
```

`conf-mapred-topo48/slaves :`

```
192.168.0.3
192.168.0.4
192.168.0.5
192.168.0.6
192.168.0.7
192.168.0.8
192.168.0.9
192.168.0.10
192.168.0.15
192.168.0.16
192.168.0.17
192.168.0.18
192.168.0.19
192.168.0.20
192.168.0.21
192.168.0.22
192.168.0.27
192.168.0.28
192.168.0.29
192.168.0.30
192.168.0.31
192.168.0.32
192.168.0.33
192.168.0.34
192.168.0.39
192.168.0.40
192.168.0.41
192.168.0.42
192.168.0.43
192.168.0.44
192.168.0.45
192.168.0.46
```

In order to launch the 48-Node Hadoop Cluster on the Intel SCC, two scripts have to be executed. Firstly, `start_cluster_topo48.sh` has to be invoked from the MCPC so as to setup the runtime environment for HDFS. It is located in our home directory in the MCPC. Afterwards, `bin/start-all-topo48.sh` has to be executed from the master node (`rck00`) so as to start the Hadoop Daemons on the Intel SCC cores and launch the HDFS cluster. The above path is relative to the Hadoop installation root directory. The Hadoop Cluster can be shut down by invoking the `bin/stop-all-topo48.sh`

script from the master node. The above path is relative to the Hadoop installation root directory.

start_cluster_topo48.sh :

```
RESOLV_CONF="domain rck
             search rck in.rck.net
             nameserver  192.168.3.254"
for i in {0..9}
do
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
for i in {10..47}
do
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh ;
                         route add default gw 192.168.3.254 ;
                         echo "${RESOLV_CONF}" > /etc/resolv.conf"
done
```

bin/start-all-topo48.sh :

```
bin=‘dirname "$0"‘
bin=‘cd "$bin"; pwd‘

. "$bin"/hadoop-config.sh

# start dfs daemons
"$bin"/start-dfs.sh --config /home/ageo/hadoop-0.20.2/conf-hdfs-topo48

# start mapred daemons
"$bin"/start-mapred.sh --config /home/ageo/hadoop-0.20.2/conf-mapred-topo48
```

bin/stop-all-topo48.sh :

```
bin=‘dirname "$0"‘
bin=‘cd "$bin"; pwd‘

. "$bin"/hadoop-config.sh

"$bin"/stop-mapred.sh --config /home/ageo/hadoop-0.20.2/conf-mapred-topo48
"$bin"/stop-dfs.sh --config /home/ageo/hadoop-0.20.2/conf-hdfs-topo48
```

5.2.11 Node Failover Watchdog

This section presents the watchdog mechanism that we have developed and was described earlier in this chapter. We have developed one watchdog script per node type, i.e. DataNode or TaskTracker and per operating frequency. Cluster nodes operate at

200 MHz, 533 MHz or 800 MHz in our setup. Code samples of the watchdog scripts are available in Appendix A.

Those six scripts operate in a similar fashion. Initially, they check if an Intel SCC core is reachable by ping periodically every 30 seconds. If not, they immediately boot Linux on that specific core (`sccBoot -l`) and wait for 200, 75 or 50 seconds, depending on the core frequency, so that core will be reachable by TCP/IP. The reason we have used different wait intervals is that the lower the frequency of the tile clock is, the more time Linux needs to be booted and a core to be accessible. After Linux has been booted, the script `start.sh` is called so as to setup the Hadoop Runtime Environment, `gmond` monitoring daemon is started (more details in the next chapter) and the corresponding Hadoop Daemon is launched on the core. If the core corresponds to a DataNode, `start-dfs.sh` is executed. `start-mapred.sh` is executed if the core corresponds to a TaskTracker. The `conf-local` directory's contents are identical to the ones described in the previous section. The only file that is different is the slaves file, which contains only the IP address of the specific core, so that the Hadoop Daemon is started only on that core. For instance, core `rck31` contains only the IP address `192.168.0.32` in the `conf-local/slaves` file. Specifically for DataNodes, the same sequence of actions is triggered if the specific core can been reached by `ping`, but the DataNode process has been killed by the OS. This situation is detected when `netstat -na | grep 50010` returns no lines. `50010` is the default port for data transfer between DataNodes and TaskTrackers or other HDFS clients.

The scripts described above are combined in a parent script, for each cluster topology and frequency setting for DataNodes and TaskTrackers. Since we perform frequency perturbations only at the 48-Node cluster, we need $9 + 3 = 12$ parent scripts. The following script is invoked so as to support the HDFS cluster when the 48-node topology has been launched, the DataNodes operate at 200MHz and the TaskTrackers at 800MHz. Similar scripts have been developed for the rest of the cases.

`watchdog-topo48-dn200-tt800.sh :`

```
for i in 1
do
    ./watchdog-datanode-200.sh $i > watchdog-logs/rck0$i.out &
done
for i in {10,11,12,13,22,23,24,25,34,35,36,37,46,47}
do
    ./watchdog-datanode-200.sh $i > watchdog-logs/rck$i.out &
done
for i in {2..9}
do
    ./watchdog-tasktracker-800.sh $i > watchdog-logs/rck0$i.out &
done
for i in {14,15,16,17,18,19,20,21,26,27,28,29,30,31,32,33,38,39,40,41,42,43,44,45}
do
    ./watchdog-tasktracker-800.sh $i > watchdog-logs/rck$i.out &
done
```

5.3 Apache Mahout Installation on the MCPC

In out setup, we use Apache Mahout 0.6, which is provided by Cloudsuite. Apache Mahout can be installed using Maven as shown below.

```
mvn install -DskipTests
. . . . . .
ageo@mitsos:~$ $MAHOUT_HOME/bin/mahout
MAHOUT_LOCAL is not set; adding HADOOP_CONF_DIR to classpath.
Running on hadoop, using HADOOP_HOME=/home/ageo/hadoop-0.20.2
No HADOOP_CONF_DIR set, using /home/ageo/hadoop-0.20.2/conf
MAHOUT-JOB: /home/ageo/analytics-release/mahout-distribution-0.6/
            examples/target/mahout-examples-0.6-job.jar
An example program must be given as the first argument.
Valid program names are:
. . . . . .
  fpg: : Frequent Pattern Growth
. . . . . .
  kmeans: : K-means clustering
. . . . . .
  trainclassifier: : Train the text based Bayes Classifier
. . . . . .
```

We also have copied the Hadoop root directory on the MCPC, so that Mahout can find the NameNode and DataNode URLs at the `core-site.xml` and `mapred-site.xml` configuration files and access the HDFS cluster that is deployed on the Intel SCC.

Chapter 6

Runtime Monitoring Framework for the Intel SCC

This chapter presents and analyzes the Runtime Monitoring Framework we have developed for the Intel SCC. The first section of the chapter explains the process of configuring a Ganglia Cluster that consists of the Intel SCC cores and the MCPC, as well as the way this Ganglia Cluster operates, i.e. how per-core runtime metrics are collected and reported. The second section of the chapter provides a detailed description of the scripts we have developed, which query the Ganglia Cluster and the BMC so as to capture the runtime metrics, store those metrics in the Monitoring Database we have designed and visualize the data that has been collected.

6.1 Ganglia Monitoring Infrastructure for the Intel SCC

This section describes the process of setting up a Ganglia Monitoring Cluster that consists of the Intel SCC cores and the MCPC. In our implementation we have utilized the `gmond` daemon of Ganglia, which is responsible for collecting and transmitting the per-core runtime metrics we are interested in capturing. This section initially analyzes the way that `gmond` operates and subsequently states the process of configuring a Ganglia Cluster on the Intel SCC. Finally, the way that the Ganglia Cluster state is reported, is presented.

6.1.1 The gmond Monitoring Daemon

`gmond` stands for Ganglia Monitoring Daemon. It is a lightweight service that must be installed on each node from which metrics should be collected. It interacts with the host operating system to obtain metrics and shares the metrics it collects with other hosts in the same cluster. Every `gmond` instance in the cluster knows the value of every metric collected by every host in the same cluster and provides and XML-formatted dump of the entire cluster state to any client that connects to `gmond`'s port.

`gmond`'s default topology is a multicast mode, meaning that all nodes in the cluster both send and receive metrics and every node maintains an in-memory database containing the metrics of all nodes in the cluster. This topology is illustrated in Figure 6.1. Internally, `gmond`'s sending and receiving halves are not linked. `gmond` does not talk to itself, it only talks to the network. Any local data captured by the metric modules are transmitted directly to the network by the sender and the receiver's internal database contains only metric data gleaned from the network.

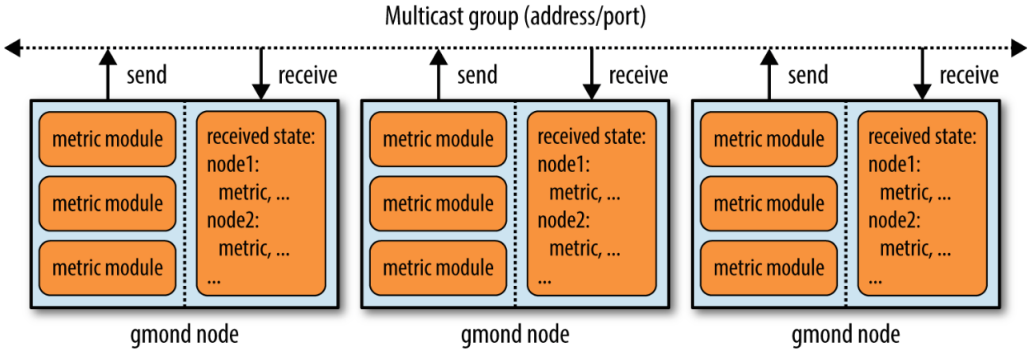


Figure 6.1: `gmond` Multicast Topology

This topology is adequate for most environments, but in some cases it is desirable to specify a few specific listeners rather than allowing every node to receive metrics from each other node. The use of `deaf` nodes, as illustrated in Figure 6.2 eliminates the processing overhead associated with large clusters. The `deaf` and `mute` parameters exist to allow some `gmond` nodes to act as special-purpose aggregators and relays for other `gmond` nodes. `mute` means that the node does not transmit; it will not even collect information about itself but will aggregate the metric data from other `gmond` daemons in the cluster. `deaf` means that the node does not receive any metrics from the network; it will not listen to state information from multicast peers, but if it is not muted it will continue sending out its own metrics for any other node that does listen.

The use of multicast in not required in any topology. The `deaf`/`mute` topology can be implemented using UDP unicast, which may be desirable when multicast is not practical or preferred, as depicted in Figure 6.3.

6.1.2 Ganglia Cluster Topology on the Intel SCC

In order to collect per-core metrics, such as CPU utilization and Network Traffic, we have configured Ganglia with a UDP unicast topology. The Intel SCC cores are configured as `deaf` nodes, that is they do not listen to any unicast or multicast channel for cluster state information. The MCPC is configured as a `mute` node, so that it

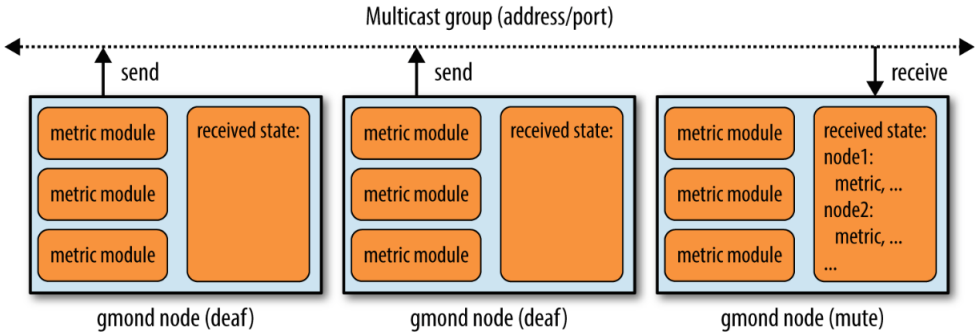


Figure 6.2: `gmond` Deaf/Mute Multicast Topology

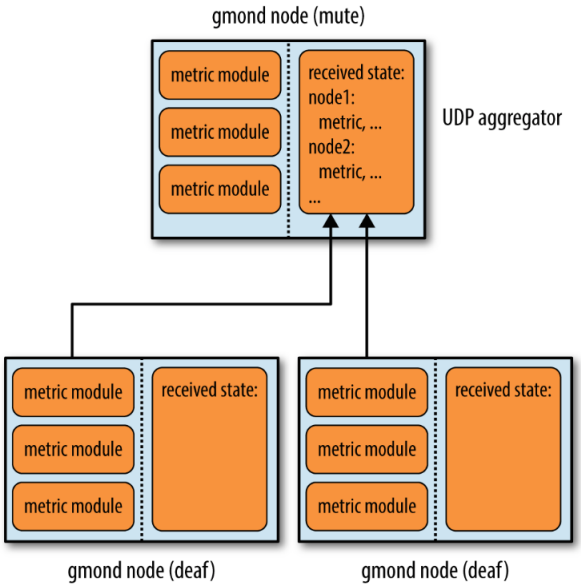


Figure 6.3: `gmond` UDP Unicast Topology

aggregates all the metrics collected from the Intel SCC cores and can provide an XML-formatted dump from a `telnet` interface. The Ganglia Topology we have implemented is illustrated in Figure 6.4.

6.1.3 The gmond.conf Configuration File

Each `gmond` instance that runs on an Intel SCC core, is configured by the `gmond.conf` configuration file. This file is located in the `/etc/ganglia` directory of the MCPC and the Gentoo Image of the Intel SCC cores. Ganglia is pre-installed in the Gentoo Image that we use. This section states the configuration properties that have been set for the gmond instances that run on the Intel SCC cores and the MCPC, so as that the Ganglia Cluster we have developed on the Intel SCC operates as described before. The whole `gmond.conf` configuration file, for both the MCPC and the Intel SCC cores is included in Appendix A.

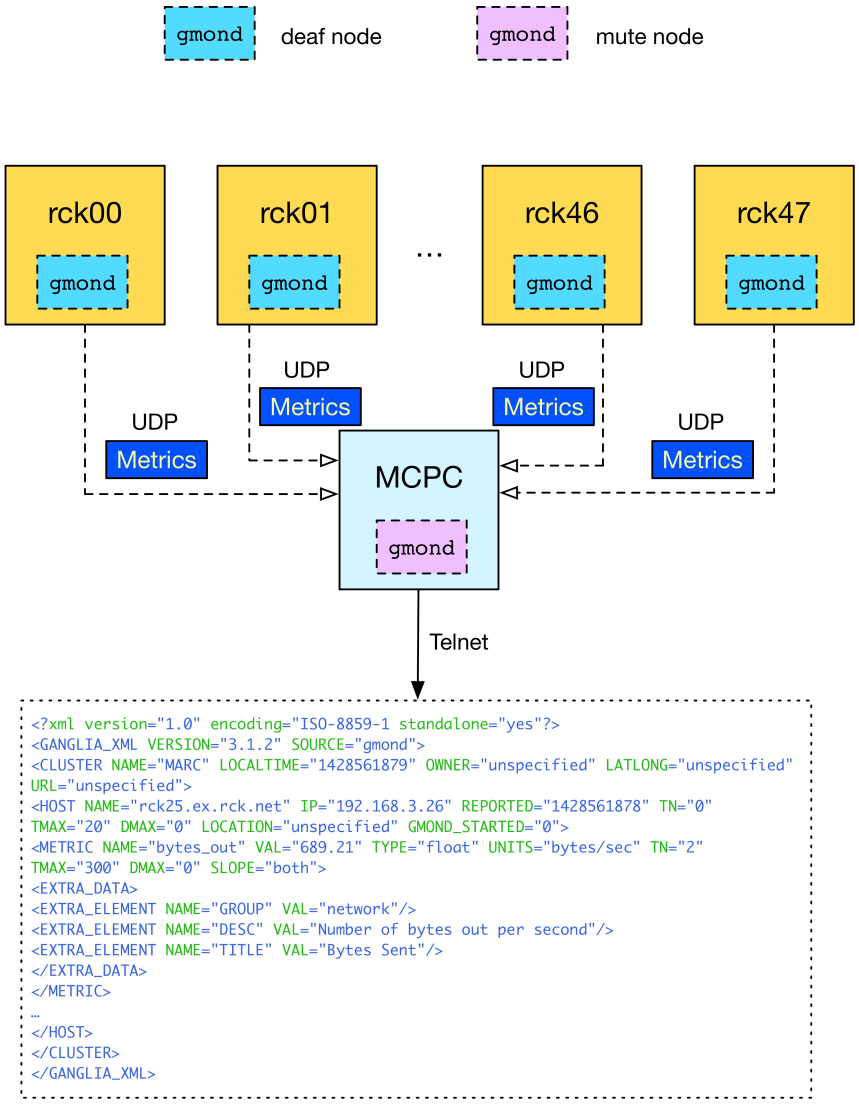


Figure 6.4: Ganglia Cluster Topology on the Intel SCC

The `gmond.conf` file which configures the `gmond` instance that runs on the MCPC defines that this instance is `mute`, that is `gmond` does not collect runtime metrics that regard the MCPC. Those properties are included in the `globals` section of `gmond.conf`. In the `cluster` section, the Ganglia Cluster name is set to MARC. The `udp_recv_channel` section configures the `gmond` instance that runs on the MCPC to listen to one UDP unicast channel, at port 8649. The runtime metrics which are reported by the Intel SCC cores are received through this channel. The `tcp_accept_channel` section configures `gmond` to accept TCP connections at port 8649. External pollers can query and receive an XML dump of the cluster state through this channel. Code samples that regard the configuration parameters of the `gmond` instance that runs on the MCPC, are listed below.

```
globals {
. . . . . .
  mute = yes
  deaf = no
. . . . . .
}
cluster {
  name = "MARC"
}
udp_recv_channel {
  port = 8649
}
tcp_accept_channel {
  port = 8649
}
```

The `gmond.conf` which configures each `gmond` instance that runs on the Intel SCC cores defines that this instance is `deaf`, that is it does not receive any metrics data from other peers. Those properties are included in the `globals` section of `gmond.conf`. The Ganglia Cluster name is set to MARC in the `cluster` section. The `udp_send_channel` section configures `gmond` to send the core metrics it collects to the MCPC, at port 8649.

The `collection_group` sections configure which metrics are to be collected and reported by `gmond`. We have configured two collection groups. The first collection group concerns CPU-related runtime metrics. This collection group contains four metrics, which are `cpu_user`, `cpu_system`, `cpu_wio` and `cpu_idle`. `cpu_user` contains the percentage of CPU utilization that occurred while executing at the user level. `cpu_system` reports the percentage of CPU utilization that occured while executing at the system level. `cpu_wio` regards the percentage of time that the CPU was idle during which the system had an outstanding I/O request. `cpu_idle` concerns the percentage of time that the CPU was idle during which the system did not have any outstanding I/O request. The second collection group concerns runtime metrics that capture the Network Traffic from and to an Intel SCC core. This collection group contains two metrics, which are `bytes_in` and `bytes_out` and capture the traffic in bytes/second that was received and sent by the core respectively. Code samples that regard the configuration parameters of the `gmond` instance that runs on an Intel SCC core are listed below.

```
globals {
. . . . . .
  mute = no
  deaf = yes
. . . . . .
}
cluster {
  name = "MARC"
}
udp_send_channel {
  host = 192.168.3.254
```

```
  port = 8649
  ttl = 1
}
/* CPU status */
collection_group {
  collect_every = 1
  time_threshold = 1
  metric {
    name = "cpu_user"
    value_threshold = 0.1
    title = "CPU User"
  }
  metric {
    name = "cpu_system"
    value_threshold = 0.1
    title = "CPU System"
  }
  metric {
    name = "cpu_wio"
    value_threshold = 0.1
    title = "CPU WIO"
  }
  metric {
    name = "cpu_idle"
    value_threshold = 0.1
    title = "CPU Idle"
  }
}
/* network traffic */
collection_group {
  collect_every = 1
  time_threshold = 1
  metric {
    name = "bytes_in"
    value_threshold = 0.01
    title = "Bytes Received"
  }
  metric {
    name = "bytes_out"
    value_threshold = 0.01
    title = "Bytes Sent"
  }
}
```

The `collect_every` attribute of each `collection_group` section specifies the polling interval for each metric in this collection group. The `time_threshold` determines the maximum amount of time that can pass before `gmond` sends all metrics specified in the `collection_group` to all configured `udp_send_channels`. The `value_threshold`

attribute of each `metric` section defines the least difference that the current value of the metric should have compared to the previous value of this metric, so that the `collection_group` is sent to the `udp_send_channels` defined. It has to be noted that the `collect_every` and `time_threshold` attributes have to be set with respect to the core frequency. We have noticed that the Intel SCC cores always assume that they operate at 800 MHz. That is, if a core operates at 200 MHz and we have set `collect_every` to 2 seconds, the metrics of this specific collection group will be collected every 8 seconds instead of 2.

6.1.4 Ganglia Cluster State Reporting

We have developed the following simple script, that starts the `gmond` daemon on all the Intel SCC cores, that participate in the current active Hadoop Cluster Topology. This script assumes that the SSH server that listens to port `1234` has been started on those cores. When this script attempts to start the `gmond` daemon on cores which do not participate in the current active Hadoop topology, it gets a `Connection refused` error, because the SSH server has not been started on that core. This way, the `gmond` daemon is started only in the cores that participate in the current active Hadoop topology.

`start_gmond.sh` :

```
ssh -p 1234 root@rck00 "gmond"
for i in {1..9}
do
        ssh -p 1234 root@rck0$i "gmond"
done
for i in {10..47}
do
        ssh -p 1234 root@rck$i "gmond"
done
```

Once `gmond` has been started on the Intel SCC cores, an XML dump of the Ganglia Cluster state is available from `gmond` UDP aggregator that runs on the MCPC. The cluster state can be obtained by opening a `telnet` connection to the MCPC, at port `8649`. The next section describes how this XML can be mined, so as to extract the runtime metrics of each Intel SCC core.

```
ageo@mitsos:~$ telnet localhost 8649
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE GANGLIA_XML [
   <!ELEMENT GANGLIA_XML (GRID|CLUSTER|HOST)*>
      <!ATTLIST GANGLIA_XML VERSION CDATA #REQUIRED>
      <!ATTLIST GANGLIA_XML SOURCE CDATA #REQUIRED>
. . . . . .
```

```
]>
<GANGLIA_XML VERSION="3.1.2" SOURCE="gmond">
<CLUSTER NAME="MARC" LOCALTIME="1428561879"
          OWNER="unspecified" LATLONG="unspecified" URL="unspecified">
<HOST NAME="rck25.ex.rck.net" IP="192.168.3.26"
          REPORTED="1428561878" TN="0" TMAX="20" DMAX="0"
          LOCATION="unspecified" GMOND_STARTED="0">
<METRIC NAME="cpu_wio" VAL="47.5" TYPE="float"
          UNITS="%" TN="2" TMAX="90" DMAX="0" SLOPE="both">
<EXTRA_DATA>
<EXTRA_ELEMENT NAME="GROUP" VAL="cpu"/>
<EXTRA_ELEMENT NAME="DESC" VAL="Percentage of time
          that the CPU or CPUs were idle during which
          the system had an outstanding disk I/O request"/>
<EXTRA_ELEMENT NAME="TITLE" VAL="CPU wio"/>
</EXTRA_DATA>
</METRIC>
. . . . . .
</HOST>
<HOST NAME="rck43.ex.rck.net" IP="192.168.3.44"
          REPORTED="1428561876" TN="2" TMAX="20" DMAX="0"
          LOCATION="unspecified" GMOND_STARTED="0">
<METRIC NAME="bytes_in" VAL="1313026.13" TYPE="float"
          UNITS="bytes/sec" TN="2" TMAX="300" DMAX="0" SLOPE="both">
<EXTRA_DATA>
<EXTRA_ELEMENT NAME="GROUP" VAL="network"/>
<EXTRA_ELEMENT NAME="DESC" VAL="Number of bytes in per second"/>
<EXTRA_ELEMENT NAME="TITLE" VAL="Bytes Received"/>
</EXTRA_DATA>
</METRIC>
. . . . . .
</HOST>
</CLUSTER>
</GANGLIA_XML>
```

6.2 Runtime Metrics Extraction and Visualization

This section presents and explains the `Python` and `gnuplot` scripts we have developed so as to collect, store and visualize the runtime metrics of the Intel SCC cores and the Intel SCC board. Initially, the structure of the Monitoring Database we have designed is described. Subsequently the set `Python` scripts that extract the runtime metrics from Ganglia and the BMC are introduced. After that, the set of `Python` scripts that query the monitoring database so as to create CSV files are analyzed. Finally the `gnuplot` scripts that visualize the data contained in the CSV files mentioned above are explained.

6.2.1 Monitoring Database Structure

We utilize a relational MySQL database to store the runtime metrics that are collected from each core. Platform aggregate metrics are also collected. The platform aggregate metrics that we capture is the Power Consumption, the Board Temperature and the Fan Speed of the Intel SCC die. Those metrics are mined from the output of the sccBmc command. We have designed two database table prototypes to store the runtime metrics. The first prototype is called CPU_NETWORK and is used to store the CPU and Network related metrics that we extract from the Ganglia XML. The second prototype is called POWER_THERMAL and is used to store the overall Energy Consumption of the chip for a specific time interval, the Board Temperature and the Fan Speed. The DDLs of CPU_NETWORK and POWER_THERMAL are shown below.

```
mysql> SHOW CREATE TABLE CPU_NETWORK;
| Table        | Create Table
|  CPU_NETWORK | CREATE TABLE ‘CPU_NETWORK‘ (
  ‘ID‘ int(11) NOT NULL AUTO_INCREMENT,
  ‘TIMESTAMP‘ datetime NOT NULL,
  ‘CORE‘ varchar(5) DEFAULT NULL,
  ‘CPU_USER‘ varchar(5) DEFAULT NULL,
  ‘CPU_SYSTEM‘ varchar(5) DEFAULT NULL,
  ‘CPU_WIO‘ varchar(5) DEFAULT NULL,
  ‘CPU_IDLE‘ varchar(5) DEFAULT NULL,
  ‘BYTES_IN‘ varchar(15) DEFAULT NULL,
  ‘BYTES_OUT‘ varchar(15) DEFAULT NULL,
  PRIMARY KEY (‘ID‘)
) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=latin1 |

mysql> SHOW CREATE TABLE POWER_THERMAL;
|  Table         | Create Table
|  POWER_THERMAL | CREATE TABLE ‘POWER_THERMAL‘ (
  ‘ID‘ int(11) NOT NULL AUTO_INCREMENT,
  ‘TIMESTAMP‘ datetime DEFAULT NULL,
  ‘FAN_SPEED‘ varchar(8) DEFAULT NULL,
  ‘TEMPERATURE‘ varchar(8) DEFAULT NULL,
  ‘POWER_FILE‘ varchar(40) DEFAULT NULL,
  ‘ENERGY_CONSUMPTION‘ varchar(30) DEFAULT NULL,
  PRIMARY KEY (‘ID‘)
) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=latin1 |
```

In order to calculate the instant Power Consumption and the overall Energy Consumption of the chip, we use shorter time intervals, because of the big variations that characterize the intensity of the electric current that is drawn by the Intel SCC board. Metrics that concern the Voltage supply of the Intel SCC board as well as the Intensity of the Electric Current it draws are stored in a seperate POWER_FILE and concern the time interval from the TIMESTAMP value of the previous database entry to the TIMESTAMP value of the current entry of the POWER_THERMAL table. For instance, supposing we have two entries in the POWER_THERMAL table with ID=372

and ID=373, the POWER_FILE that appears in the row with ID=373 contains voltage, current and power consumption measurements that cover the time interval between the TIMESTAMP of the row with ID=372 and the TIMESTAMP of the row with ID=373. The power file is a CSV file which contains the above mentioned metrics in tab delimited columns as <Timestamp>\t<Voltage>\t<Current>\t<Power Consumption> \t<Energy Consumption>. The Power Consumption refers to the instant value and the Energy Consumption regards the time interval between the current measurement and the previous. The value that is stored in the ENERGY_CONSUMPTION column results from the summation of all the individual energy consumption measurements of the corresponding power file. This way, we manage to accurately capture the overall energy consumption of the chip. Sample measurements that are stored in a power file are presented below.

```
2015-03-07 10:30:13.341655      3.300   21.485  70.9005 0.0162447406054
2015-03-07 10:30:13.593029      3.300   21.485  70.9005 0.0109199817181
2015-03-07 10:30:13.863168      3.300   21.584  71.2272 0.0098155311584
2015-03-07 10:30:14.143957      3.300   21.584  71.2272 0.0099683681488
2015-03-07 10:30:14.429008      3.300   21.386  70.5738 0.0110715771675
2015-03-07 10:30:14.704333      3.300   21.287  70.2471 0.0098981943369
2015-03-07 10:30:14.974797      3.300   21.287  7 0.2471 0.00941249613762
. . . . . .
```

6.2.2 Extracting and Storing Runtime Metrics

This section presents two Python scripts we have developed so as to extract the runtime metrics we mentioned above in the monitoring database and the power files. The code of both scripts is included in Appendix A. The first script, which is called store-power.py repeatedly executes the sccBmc command, it parses the output it provides and stores the voltage, current, power and energy consumption in a power file. The power file and the time interval that this script is executed are provided as command line arguments.

The second script, which is called store-metrics.py initially invokes the store-power .py script. After store-power.py is completed, store-metrics.py mines the power file that was populated by store-power.py and calculates the total energy consumption for the specific time interval. Subsequently, it queries the BMC with the sccBmc command so as to obtain the Board Temperature and the Fan Speed of the platform. Finally, store-metrics.py queries the Ganglia Cluster so as to obtain the CPU Utilization and the Network Traffic of each core, parses the XML that is returned by Ganglia and stores the extracted metrics in the monitoring database. The extracted and calculated metrics are stored in database tables which have been created from the CPU_NETWORK and POWER_THERMAL prototypes. The names of those tables are provided as command line arguments. The time interval that is passed to the store-power.py script is also provided as a command line argument, so as to determine the time interval that should separate subsequent monitoring entries.

6.2.3 Runtime Metrics Mining

This section presents three `Python` scripts that we have developed so as to mine the monitoring database and the power files and create CSV files, which be given as inputs to the `gnuplot` scripts that will be presented in the subsequent section, which visualize the runtime metrics which have been collected. The code of the `Python` scripts is available in Appendix A.

The first script, which is called `prepare-metrics-cpu-network.py` queries a database table that has been created based on the `CPU_NETWORK` prototype, so as to create a CSV file which contains the following metrics in tab delimited columns as `<Timestamp>\t <CPU User>\t<CPU System>\t<CPU WIO>\t<CPU Idle>\t<Bytes In>\t<Bytes Out>`. Those metrics concern a specific Intel SCC core, whose name is provided as a command line argument. The name of the database table to be queried, as well as the name of the output CSV file are also provided as command line arguments. Sample metrics stored in this file are presented below.

```
ageo@mitsos:~$ cat ganglia-monitoring/
    plot_files/cpu_network/bayes-dn200-tt533.dat
0:00:00 1.1     2.2     0.0     96.7    630.94  1342.14
0:00:21 0.7     3.9     0.0     95.4    835.37  1447.84
0:00:42 0.6     2.3     0.0     97.1    695.06  1378.13
0:01:03 0.7     4.7     0.0     94.6    1384.27 2123.21
0:01:24 0.2     2.2     0.0     97.6    606.78  1186.35
0:01:44 2.6     3.8     0.0     93.5    620.15  1281.80
0:02:05 0.1     2.3     0.0     97.6    489.62  1038.12
0:02:26 0.2     2.1     0.0     97.7    614.23  1274.73
. . . . . .
```

The second script, which is called `prepare-metrics-thermal.py` queries a database table that has been created based on the `POWER_THERMAL` prototype, so as to create a CSV file which contains the following metrics in tab delimited columns as `<Timestamp>\t<Board Temperature>\t<Fan Speed>`. Those metrics concern the entire Intel SCC Board. The name of the database table to be queried, as well as the name of the output CSV file are provided as command line arguments. Sample metrics stored in this file are presented below.

```
ageo@mitsos:~$ cat ganglia-monitoring/
    plot_files/thermal/wordcount-topo16.dat
0:00:00 34      213
0:00:05 34      223
0:00:10 34      234
0:00:15 34      245
0:00:20 34      255
0:00:25 34      10
0:00:30 33      20
0:00:34 34      31
. . . . . .
```

The third script, which is called `prepare-metrics-power.py` queries the power files that are located at a specific file system directory, so as to create a CSV file which contains the following metrics in tab delimited columns as `<Timestamp>\t<Voltage>\t <Current>\t<Power Consumption>`. Those metrics concern the entire Intel SCC Board. Because of the fact that those metrics have been collected with a shorter polling interval than the ones mentioned earlier in this section, they are sampled at a sample rate that is provided as a command line argument, so that the Power Consumption plot will contain relatively smooth curves. The name of the file system directory to be queried, as well as the name of the output CSV file are also provided as command line arguments. Sample metrics stored in this file are presented below.

```
ageo@mitsos:~$ cat ganglia-monitoring/
    plot_files/power/kmeans-low.dat
0:00:06.006023  3.304   17.327  57.248408
0:00:12.053117  3.304   17.129  56.594216
0:00:18.079477  3.304   17.129  56.594216
0:00:24.725223  3.304   16.931  55.940024
0:00:30.957759  3.304   17.030  56.267120
0:00:36.985528  3.304   17.228  56.921312
0:00:43.079844  3.304   18.218  60.192272
0:00:49.640558  3.304   18.119  59.865176
. . . . . .
```

6.2.4 Runtime Metrics Visualization

This section presents five `gnuplot` scripts we have developed so as to visualize the metrics that we have collected. The first two scripts, called `plot-cpu.gp` and `plot-network.gp` query the CSV file that was created by `prepare-cpu-network.py`. `plot-temperature.gp` and `plot-fan-speed.gp` query the CSV files that were created by `prepare-thermal.py`. Finally, `plot-power.gp` queries the CSV file that has been created by `prepare-power.py`. Sample plots that are generated by those files are presented below, in Figures 6.5 - 6.8. All the above mentioned scripts receive the step of the x-axis of the plot, the time interval that the x-axis spans and the input CSV file as command line arguments, like in the following example.

```
ageo@mitsos:~/ganglia-monitoring$ gnuplot -e "xtics='0:04:0'"
    -e "time='1:54:54'" -e "datafile='fpg-dn200-tt200.dat'" plot_cpu.gp
```

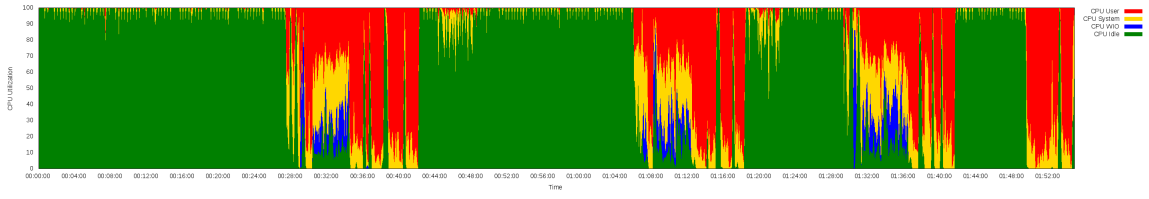


Figure 6.5: CPU Utilization Plot

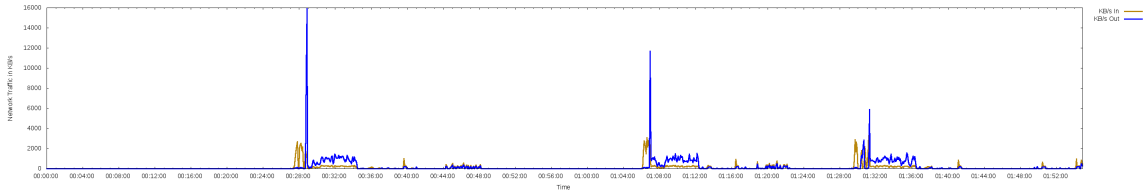


Figure 6.6: Network Traffic Plot

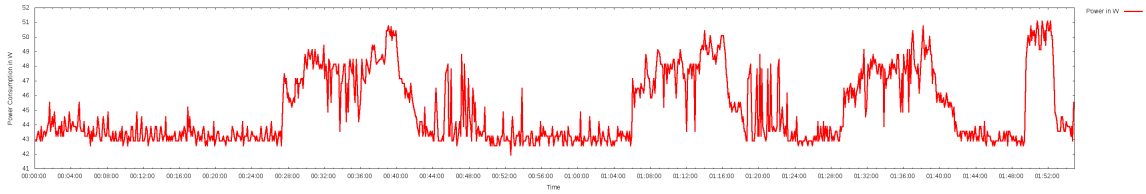


Figure 6.7: Power Consumption Plot

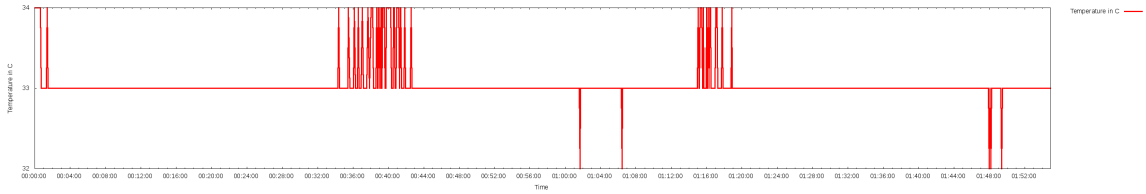


Figure 6.8: Board Temperature Plot

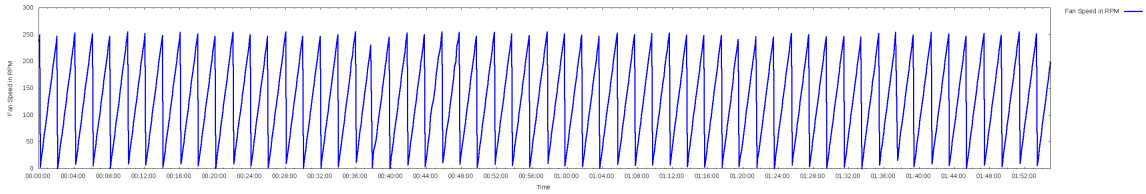


Figure 6.9: Fan Speed Plot

Chapter 7

Workload Characterization of Big Data Applications on the Intel SCC

This chapter states the experimental analysis we have conducted for four MapReduce applications that run on top of the HDFS cluster topologies we have deployed on the Intel SCC. Each section of this chapter is dedicated to one application. Initially pseudocode that describes the algorithm that is implemented by each application is provided and the input file generation and application execution process is presented. In the remainder of each section, the experimental results we have collected are explained and analyzed in detail in order to investigate the behavior of these applications when they run on the Intel SCC, on top of different HDFS cluster topologies, with different frequency settings for the cluster nodes and for different input file sizes.

7.1 The Wordcount Application

This section presents our analysis regarding the execution of the `Wordcoun` application on the Intel SCC. We initially describe the MapReduce implementation of the `Wordcount` algorithm. In addition, the input file generation and application execution process are stated. We have utilized resources which are provided by DCBench for that purpose. Subsequently, the experimental results we have received are analyzed, in order to draw conclusions regarding the behavior of the `Wordcount` application on the Intel SCC for different input sizes, cluster topologies and frequency settings.

7.1.1 Algorithm Description

The Wordcount application counts the number of words of an input text. The MapReduce implementation of this application consists of a `Mapper` and a `Reducer` function, whose pseudocode is presented below.

```
map(String key, String value):
    words = tokenize(value)
    for each word in words:
        output(word,'1')
```

```
reduce(String key, List<String> values):
    sum = 0
    for each value in values:
        sum += 1
    output(key,sum)
```

The Mapper function receives `<key,value>` pairs for each line in the input text file. The key of each pair represents the character offset of the specific line and the value is the character `String` of this line. The `Mapper` function tokenizes each line so as to extract the words it contains, and outputs intermediate `<key>,<value>` pairs. The key of each intermediate pair contains one word of the input text and the value is always `1`.

The `Reducer` function receives `<key>,<list(value)>` pairs. That is, each `Reducer` receives a list that contains all the `1`s that where generated by the `Mappers`. The `Reducer` sums all the `1`s that are contained in the list and outputs the final `<key>,<value>` pairs. The key of each output pair contains one word that was included in the input text file and the value contains the number of occurences of this specific word in the input text file.

7.1.2 Application Execution and Input Files

We use four different input files for the `Wordcount` application, whose size is 256 MB, 512 MB, 1 GB and 2 GB. Those files are generated randomly by the `RandomTextWriter` class which is included in `hadoop-0.20.2-examples.jar`. DC Bench provides a script called `prepare-wordcount.sh`, which receives the desired input size as an argument, generates a random text file of this specific size and uploads this file to HDFS. We have modified this script to also receive the number of TaskTracker nodes as a command line argument. The code of this script is included in Appendix A.

```
ageo@mitsos:~/HVCBench-hadoop/workloads/base-operations/wordcount$
        ./prepare-wordcount.sh 256m 32
BYTES_PER_MAP 8388608
MAPS_PER_HOST 1
HOSTS 32
generating rtw-wordcount-256M data
Running 32 maps.
Job started: Fri Apr 10 15:13:42 EEST 2015
15/04/10 15:13:51 INFO mapred.JobClient: Running job: job_201504101105_0004
15/04/10 15:13:52 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
15/04/10 15:18:10 INFO mapred.JobClient:  map 100% reduce 0%
15/04/10 15:18:40 INFO mapred.JobClient: Job complete: job_201504101105_0004
. . . . . .
The job took 298 seconds.
```

In order to run the `Wordcount` benchmark, the `run-wordcount.sh` script, which is provided by DCBench has to be executed. This script receives the input size of the text file as a command line argument and searches in HDFS for the input file with this specific size that was created by `prepare-wordcount.sh`. The code of this script is included in Appendix A.

```
ageo@mitsos:~/HVCBench-hadoop/workloads/base-operations/wordcount$
    ./run-wordcount.sh 256m
rmr: cannot remove /cloudrank-out/rtw-wordcount-256M-out: No such file or directory.
15/03/15 15:13:25 INFO input.FileInputFormat: Total input paths to process : 32
15/03/15 15:13:30 INFO mapred.JobClient: Running job: job_201503151246_0001
15/03/15 15:13:31 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
15/03/15 15:31:22 INFO mapred.JobClient:  map 100% reduce 100%
15/03/15 15:32:01 INFO mapred.JobClient: Job complete: job_201503151246_0001
. . . . . .
```

7.1.3 Scalability Analysis Per Input Size

This section presents the analysis we have conducted regarding the scalability of the `Wordcount` application, in terms of input size, when it runs on the Intel SCC. We have executed the application with four different input files, whose size is 256 MB, 512 MB, 1 GB and 2 GB. We have utilized the 48-Node Cluster Topology for this analysis and have configured both the DataNodes and the TaskTrackers to operate at the maximum frequency of 800 MHz. The experimental results we have received regarding the execution time and the energy consumption of the `Wordcount` application are presented below. Detailed plots that illustrate the CPU utilization and the network traffic for one DataNode and one TaskTracker as well as the overall power consumption and board temperature of the Intel SCC, for each run, are included in Appendix B1.

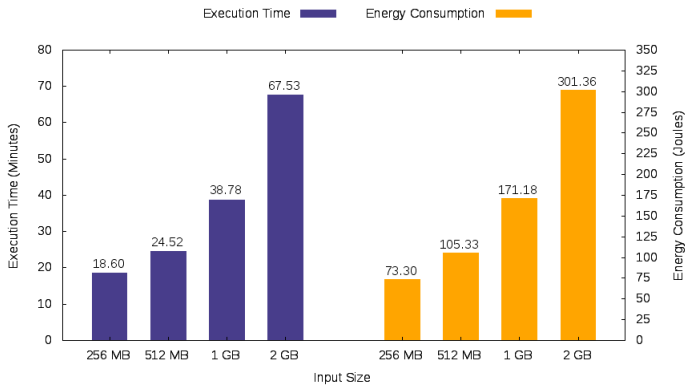


Figure 7.1: Wordcount Input Size Scalability Analysis (1/2)

Our analysis indicates clearly that both the execution time and the energy consumption of the `Wordcount` application scale linearly as the size of the input text file increases.

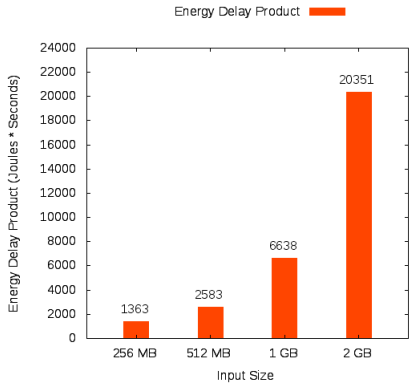


Figure 7.2: Wordcount Input Size Scalability Analysis (2/2)

The CPU utilization plots of the TaskTracker nodes that are presented in the Appendix denote that the Map phase of the `Wordcount` MapReduce job is expanded when the input size increases, since the number of the `InputSplits` and thus the number of issued Map tasks rises. The intermediate `<key,value>` pairs are evenly distributed among the reducers by the `HashPartitioner` and as consequence, a slight increase in the duration of the Reduce phase is also observed when the size of the input text file increases.

The idle period in the beginning of the execution accounts for the Job initialization phase that is performed by the JobTracker. The idle period between the Map and the Reduce phases indicates that the Map task of this specific TaskTracker has finished, but the JobTracker waits for the completion of Map tasks that run on other TaskTrackers, so that the Reduce phase can be started for the Job. The idle period after the Reduce phase depicts that the Reduce task of this specific TaskTracker has completed its execution, but the JobTracker waits for Reduce tasks that run on other TaskTrackers to finish as well.

7.1.4 Cluster Topology Analysis

This section presents our analysis concerning the behavior of the `Wordcount` application when it is executed on top of different HDFS cluster topologies on the Intel SCC. We have created one input file for this study, with a size of 256 MB. We have configured both the DataNodes and the TaskTrackers of each cluster topology to operate at the maximum frequency of 800 MHz. The idle nodes of each topology (if any) operate at the minimum frequency of 100 MHz. `gmond` is not active on those nodes as well. The experimental results we have received regarding the execution time and the energy consumption of the `Wordcount` application are presented below. Detailed plots are included in Appendix B1, as in the previous case.

Our results evidently suggest that the `Wordcount` application benefits when the number of TaskTracker nodes increases, both in execution time and in energy consumption. However, the energy consumption gain is not proportional to the decrease of the exe-

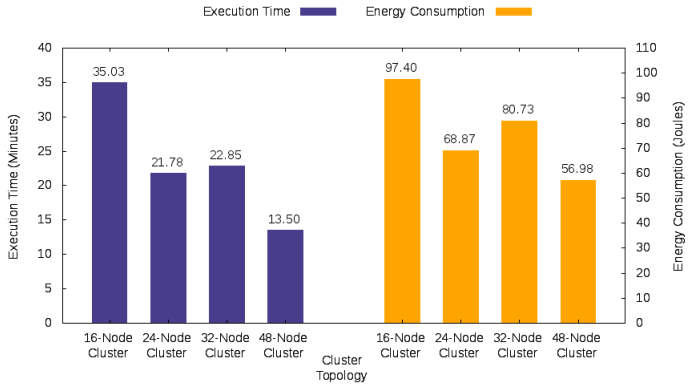


Figure 7.3: Wordcount Cluster Topology Analysis (1/2)

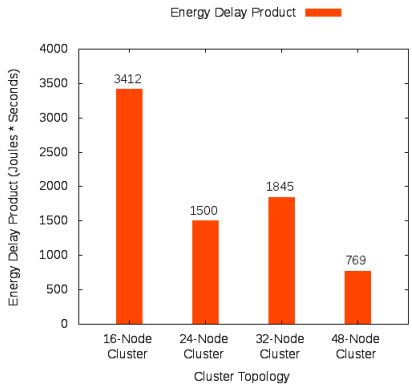


Figure 7.4: Wordcount Cluster Topology Analysis (2/2)

cution time, since the power that is drawn by the Intel SCC is higher as the number of nodes of the Hadoop cluster increases. The CPU utilization plots of the TaskTracker nodes illustrate that the increase in execution time can be interpreted by the fact that more Map tasks have to be issued for each TaskTracker node, as the TaskTracker node count decreases. The intermediate `<key,value>` pairs are evenly distributed among the reducers by the `HashPartitioner` and as consequence, a slight increase in the duration of the Reduce phase is also observed when the number of TaskTracker nodes of the HDFS cluster decreases.

Another conclusion that can be drawn is that for a given name of TaskTracker nodes, both the execution time and the energy consumption of the application deteriorate when the number of DataNodes is increased from 7 to 15. (24-Node Topology versus 32-Node Topology). This observation can be attributed to the fact that the DataNodes do not execute any computation regarding the MapReduce job and are only responsible for providing the TaskTrackers with data from HDFS. As a consequence, the application is charged with higher power consumption, without yielding any benefit in execution time, which results to an increase in the overall energy consumption.

7.1.5 Frequency Scaling Analysis

This section analyzes the impacts of frequency scaling on the execution time and the energy consumption of the `Wordcount` application on the Intel SCC. We have tested the input text file with the size of 256 MB in the 48-Node Cluster topology for nine frequency settings. We have configured the DataNodes and Master Node and the TaskTrackers to run at either 200 MHz, 533 MHz or 800 MHz and each frequency setting represents one combination of those values. The experimental results we have received regarding the execution time and the energy consumption of the `Wordcount` application are presented below. Detailed plots are included in Appendix B1, as in the previous case.

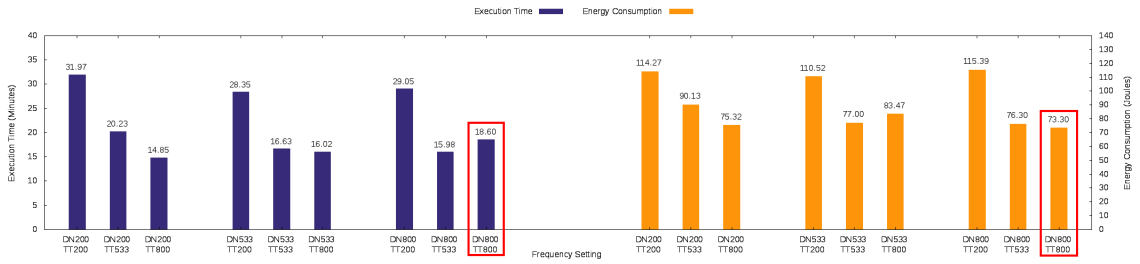


Figure 7.5: Wordcount Frequency Scaling Analysis (1/2)

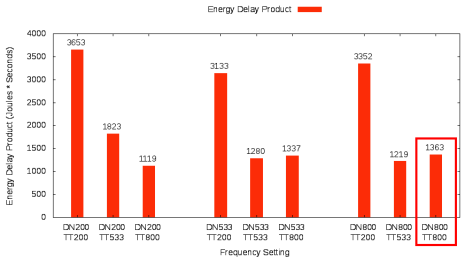


Figure 7.6: Wordcount Frequency Scaling Analysis (2/2)

The experimental results presented above point out that the `Wordcount` application benefits from the TaskTrackers running at the maximum frequency of 800 MHz, both in terms of execution time and energy consumption. The frequency of the DataNodes and the Master node appears to have a minor impact the execution time and the energy consumption of the application. This assumption could suggest that scaling down the frequency of the DataNodes to 200 MHz, while the TaskTrackers operate at 800 MHz, could result in lower energy consumption because the slightly higher execution time would be mitigated by the lower power consumption resulting in lower overall energy consumption. Such a conclusion cannot be drawn by the energy consumption observations mentioned above. However, it has to be mentioned that the energy consumption of the DN200-TT800 and DN800-TT800 setting differ by less than 3%, indicating that we cannot draw a safe conclusion regarding that matter.

In addition, it has to be mentioned that the execution time of the DN800-TT800 setting is misleading, because of the fact that Map tasks were re-run because of errors in this specific execution, during the idle period that is depicted in the CPU utilization plot of TaskTracker `rck45` in the Appendix, between the 10th and the 16th minute. This fact is illustrated in the following CPU utilization plots of TaskTrackers `rck04`, `rck29` and `rck32`. To corroborate this hypothesis, we re-executed the application for this specific setting and it was completed in 13.50 minutes. However, the power consumption that was recorded was on average 20 W less than the power consumption that we observed in the first run. This fact can be attributed to the lower board temperature of the Intel SCC, because of lower platform utilization at the time. As a consequence, the results of this re-execution cannot be used so as to yield a more accurate measurement for the energy consumption of the DN800-TT800 setting. The safest conclusion that can be drawn is that it is expected to be less than the value of 73.30 Joules that observed in the first run.

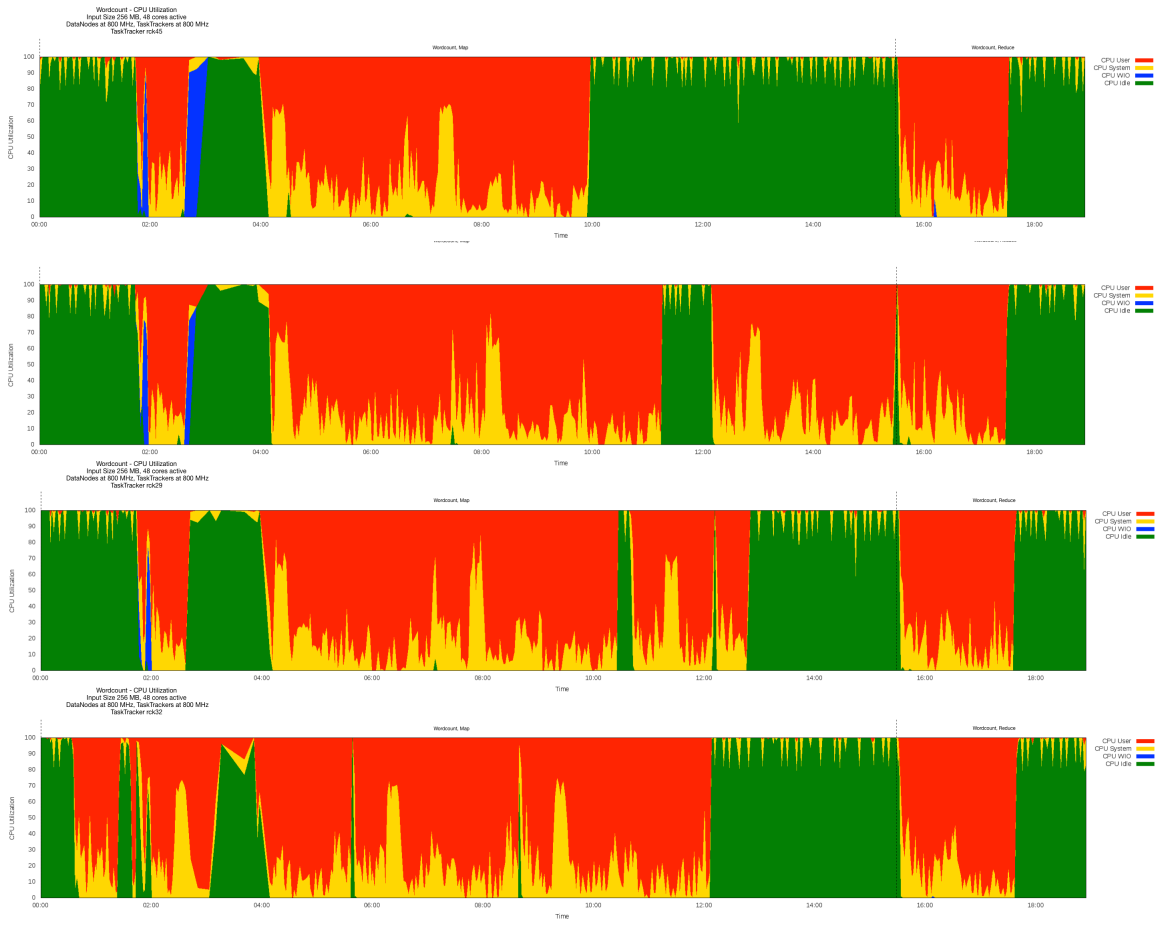


Figure 7.7: CPU utilization plots for the Wordcount application

Moreover, it has to be mentioned that the energy consumption saving would have been more significant if the Intel SCC architecture allowed Voltage Scaling at the tile

level. Because of the fact that in our setting all voltage domains contain DataNodes and TaskTrackers as well, we were not able to perform Voltage Scaling. Since power consumption is proportional to the product of the core frequency and the square of the voltage, as the following equation denotes, scaling down the voltage would decrease energy consumption even more, without impairing performance at all.

$$P \propto CV^2$$

7.1.6 Cluster Utilization Overview

This section provides an overview figure (Figures 7.11-7.16) for the CPU utilization of all cluster nodes, when the `Wordcount` application is executed with the input file of 512 MB, on the 48-Node HDFS cluster topology and with the DataNodes and the TaskTrackers configured to operate at 800 MHz. The very low utilization of all the DataNodes which explains the minimal performance impairment that we observe when their operating frequency is scaled down to 200 MHz. The CPU utilization diagrams of the `TaskTracker` nodes depict the execution on Map and Reduce tasks on the cores that they are hosted.

Those figures clearly point out that `Wordcount` is a CPU-intensive application and as a consequence it benefits greatly from a cluster topology with many TaskTracker nodes, which operate at the maximum frequency of 800 MHz.

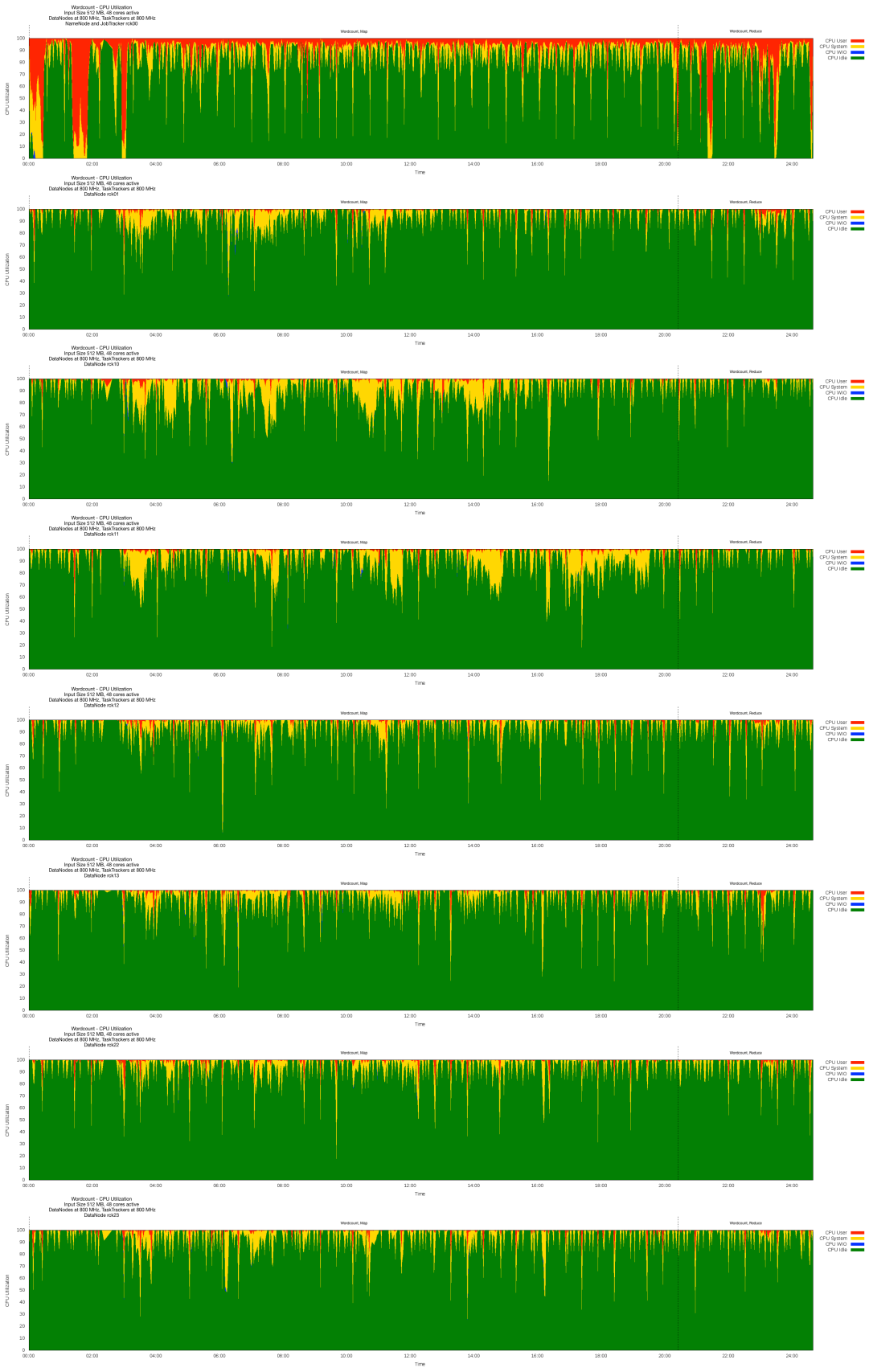


Figure 7.8: Wordcount Overall Cluster Utilization (1/6)

Figure 7.9: Wordcount Overall Cluster Utilization (2/6)

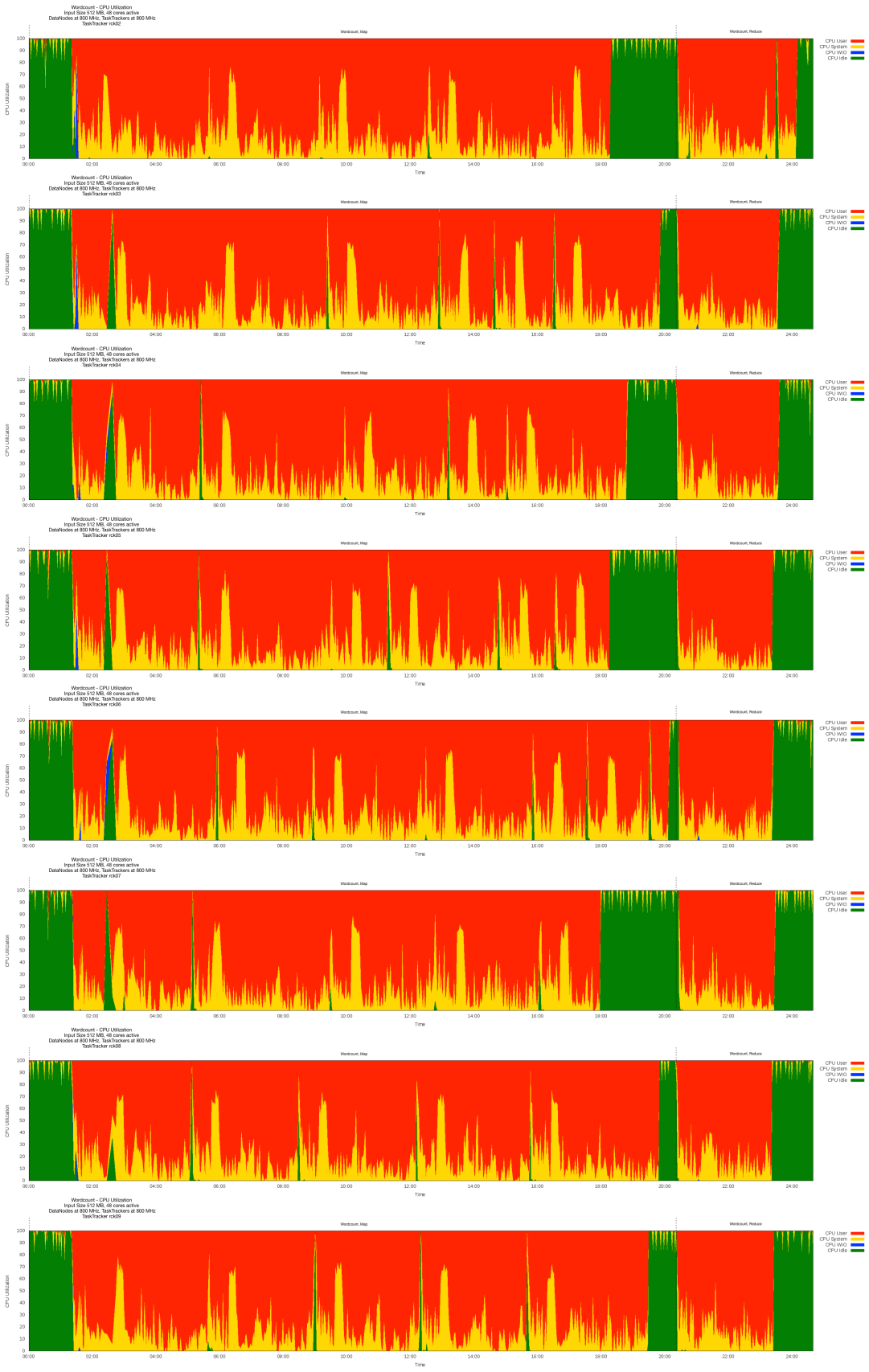


Figure 7.10: Wordcount Overall Cluster Utilization (3/6)

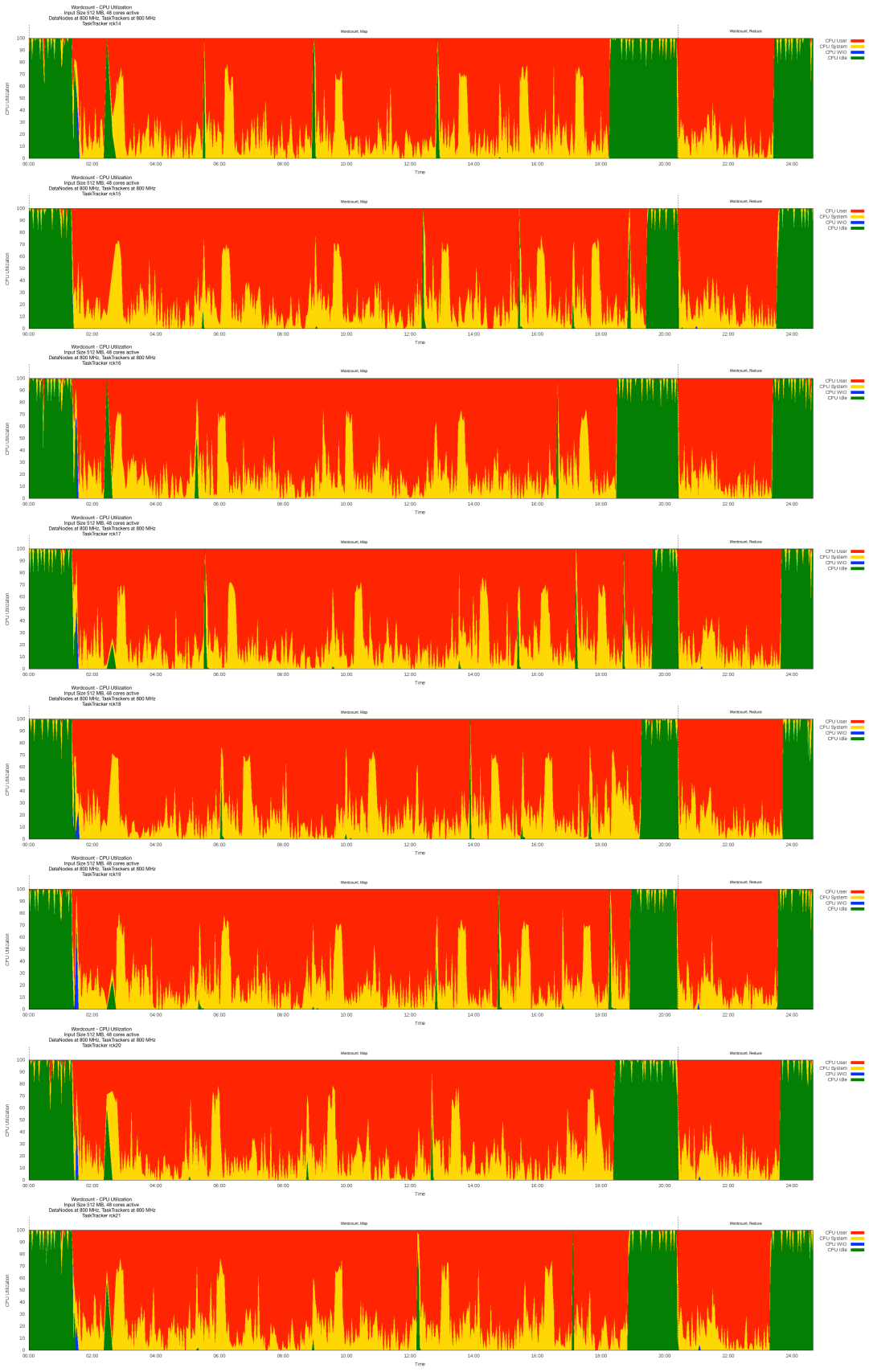


Figure 7.11: Wordcount Overall Cluster Utilization (4/6)

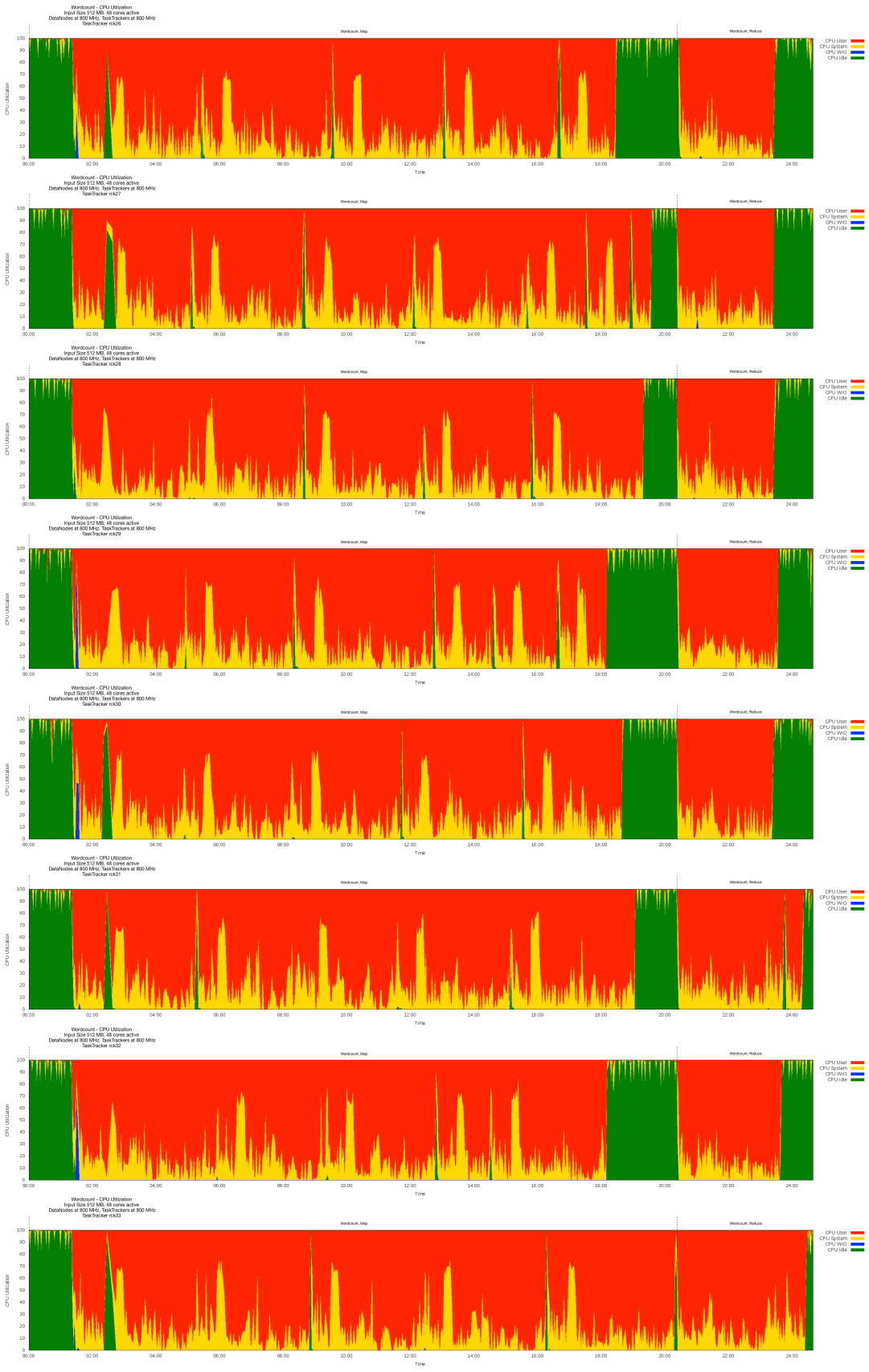


Figure 7.12: Wordcount Overall Cluster Utilization (5/6)

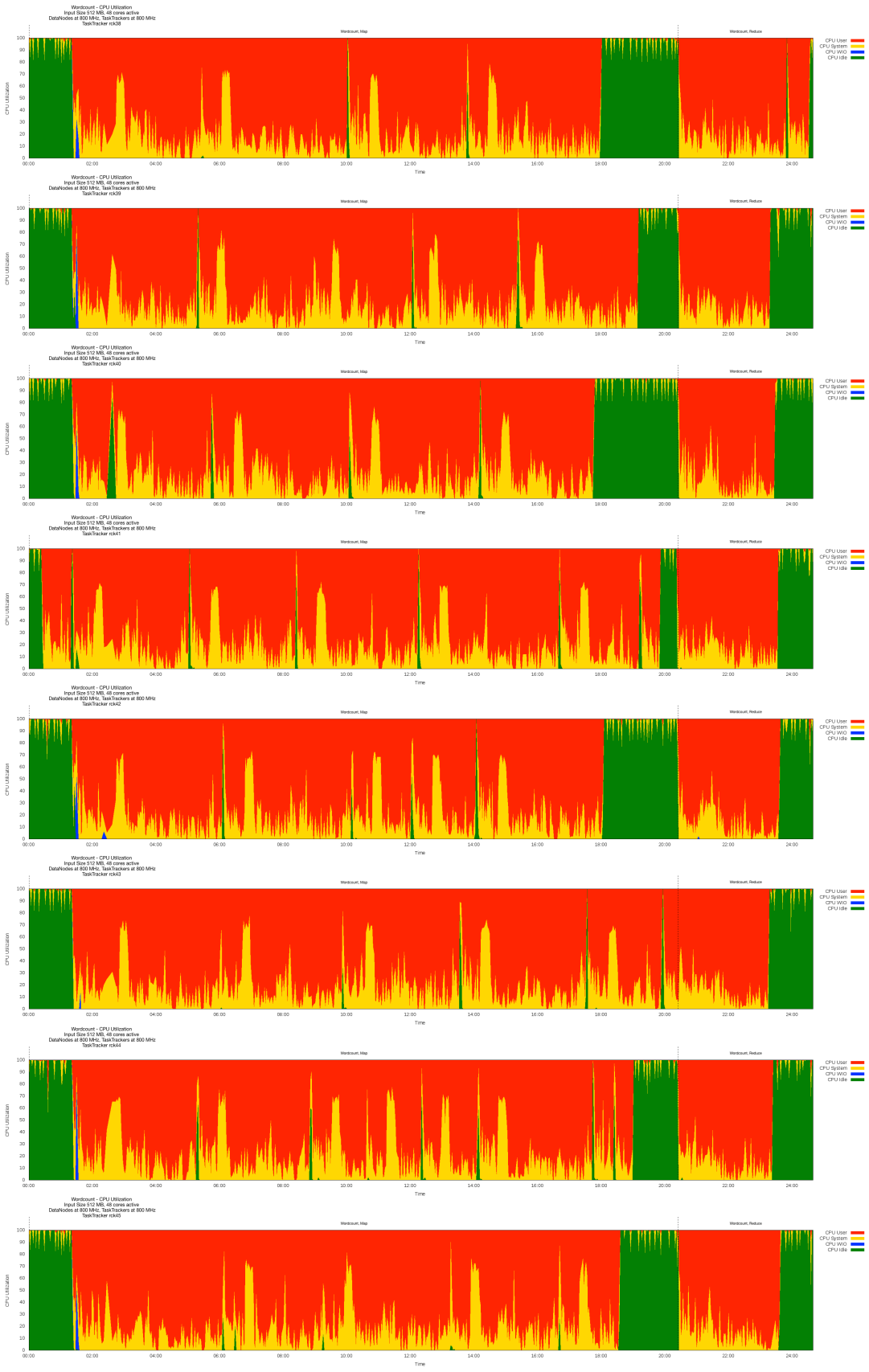


Figure 7.13: Wordcount Overall Cluster Utilization (6/6)

7.2 The Bayes Classification Application

This section presents our analysis regarding the execution of the `Bayes Classification` application on the Intel SCC. We initially describe the MapReduce implementation of the `Bayes Classification` algorithm. In addition, the input file generation and application execution process are stated. We have utilized resources which are provided by Cloudsuite for that purpose. Subsequently, the experimental results we have received are analyzed, in order to draw conclusions regarding the behavior of the `Bayes Classification` application on the Intel SCC for different input sizes, cluster topologies and frequency settings.

7.2.1 Algorithm Description

The purpose of the Bayes Classification Application is to train a text classification model, based on given training set of classified documents. This classification model is based on the frequency of the words that appear in the training documents of a specific class. The classification model enables us to calculate the probability of an unclassified document being a member of a specific class. The document is assigned to the class that yields the highest probability depending on the classification model.

The Bayes Classification implementation that is provided by Mahout splits the training of the classification model in four MapReduce jobs. The first two jobs calculate the normalized Term Frequency - Inverse Document Frequency (Tf-Idf) for each `<class,term>` pair. Supposed that $\vec{d} = (d_1, d_2, ..., d_n)$ is the vector of the training documents, d_{ij} is the number of occurrencies of term j in document i, $\vec{t} = (t_1, t_2, ..., t_m)$ is the vector of the term vocabulary and $\vec{y} = (y_1, y_2, ..., y_n)$ is the vector of the document classes, then the normalized Tf-Idf of term j for document class k is calculated by the formula

$$TfIdf(k,j) = ln(\frac{|\{d_i : d_i \in y_k\}|}{\sum_{i:d_i \in y_k} d_{ij}}) \sum_{i:d_i \in y_k} \frac{ln(d_{ij}+1)}{\sqrt{\sum_{d_{ij} \in d_i} d_{ij}^2}}$$

The third MapReduce job calculates the $Sigma_j$ for each term, the $Sigma_k$ for each class and the $Sigma_j Sigma_k$, based on the following formulas.

$$Sigma_j(j) = \sum_k TfIdf(k,j)$$

$$Sigma_k(k) = \sum_j TfIdf(k,j)$$

$$Sigma_j Sigma_k = \sum_{j,k} TfIdf(k,j)$$

The fourth MapReduce job calculates the weight normalization factor for each class, which is denoted by $Theta(k)$, based on the following formula. M represents the total number of terms in the document vocabulary.

$$Theta(k) = \sum_j \frac{TfIdf(k,j)+1}{Sigma_k(k)+M}$$

The calculation of the probability of a document being a member of a specific class breaks down to the calculation of the contribution of each of its terms. The contribution of each term j for class k is equal to the expression

$$p(k,j) = -ln(\frac{TfIdf(k,j)+1}{Sigma_k(k)+M})$$

As a consequence, the probability of document d being a member of class k is

$$p(d,k) = \sum_{j:t_j \in d} f(j) * p(k,j)$$

where $f(j)$ denotes the number of occurences of term t_j in the document to be classified. The document is assigned to the class that yields the maximum value for $p(d,c)$.

The following pseudocode points out the master flow of the Mahout implementation of the Bayes Classification algorithm, which consists of four MapReduce jobs.

`BayesDriver :`

```
BayesDriver.main(input,output,params):
    BayesFeatureDriver.runJob(input,output,params)
    BayesTfIdfDriver.runJob(input,output,params)
    BayesWeightSummerDriver.runJob(input,output,params)
    BayesThetaNormalizerDriver.runJob(input,output,params)
```

The `BayesFeatureDriver` job processes the input training documents, which are stored as `<key,value>` pairs in HDFS. The `key` of each file (and `InputSplit`) represents the class to which this document has been assigned to and the value of each pair contains the terms of each document separated by spaces. `BayesFeatureDriver` outputs `<key,value>` pairs of four types : `LABEL_COUNT`, `DOCUMENT_FREQUENCY`, `FEATURE_COUNT` and `WEIGHT`, as presented below.

`LABEL_COUNT` pairs denote the number of documents that belong to each class:

$$< (\text{"}LABEL\_COUNT\text{"}, class(y_k)), |\{d_i : d_i \in y_k\}| >$$

`DOCUMENT_FREQUENCY` pairs represent the number of occurences of a term in the documents that belong to a specific class:

$$< (\text{"}DOCUMENT\_FREQUENCY\text{"}, class(y_k), term(t_j)), \sum_{i:d_i \in y_k} d_{ij} >$$

`FEATURE_COUNT` pairs include the number of occurences of a specific term in all the training documents:

$$< (\text{"}FEATURE\_COUNT\text{"}, term(t_j)), \sum_{j,i \in \{1,..,n\}} d_{ij} >$$

`WEIGHT_COUNT` pairs contain the length normalized and TF transformed frequency of a term for a specific class:

$$< (\text{"}WEIGHT\text{"}, class(y_k), term(t_j)), \sum_{i:d_i \in y_k} \frac{ln(d_{ij}+1)}{\sqrt{\sum_{d_{ij} \in d_i} d_{ij}^2}} >$$

The above purpose is achieved by the following `Map` and `Reduce` functions. Please note that the terms `label` and `class` and the terms `term` and `feature` are used interchangeably.

```
BayesFeatureMapper.map(String key, String value):
    class = key
    terms = tokenize(value)
    termCountMap = []
    for (term : terms):
        if (!termCountMap.contains(term)):
            termCountMap.put(term,1)
        else
            termCountMap.put(term,termCountMap.get(term)+1)
    lengthNormalization = 0.0
    for (term : termCountMap)
        lengthNormalization += termCountMap.get(term) * termCountMap.get(term)
    lengthNormalization = sqrt(lengthNormalization)
    for (term : termCountMap):
        output(("WEIGHT",class,term), ln(termCountMap.get(term)) / lengthNormalization)
        output(("DOCUMENT_FREQUENCY",class,term),1)
        output(("FEATURE_COUNT",term),1)
        output(("FEATURE_TF",term),termCountMap.get(term))
    output(('LABEL_COUNT",class),1)

BayesFeatureReducer.reduce(StringTuple key, List<Double> values)
    sum = 0
    for (value : values)
        sum += value
    if (key.get(0).equals("WEIGHT") or
        key.get(0).equals("DOCUMENT_FREQUENCY") or
        key.get(0).equals("FEATURE_COUNT") or
        key.get(0).equals("LABEL_COUNT")):
        output(key,sum)
```

The `BayesTfIdfDriver` job processes the intermediate results which were generated by `BayesFeatureDriver` so as to calculate the normalized TfIdf for each `<class,term>` pair. `BayesTfIdfDriver` outputs pairs of two types : `WEIGHT` and `FEATURE_SET_SIZE`.

`WEIGHT` pairs include the normalized TfIdf for each `<class,term>` pair:

$$< (\text{"}WEIGHT\text{"}, class(y_k), term(t_j)), ln(\frac{|\{d_i : d_i \in y_k\}|}{\sum_{i:d_i \in y_k} d_{ij}}) \sum_{i:d_i \in y_k} \frac{ln(d_{ij}+1)}{\sqrt{\sum_{d_{ij} \in d_i} d_{ij}^2}} >$$

Only one FEATURE_SET_SIZE pair is generated in the output and it contains the number of all terms in the document vocabulary:

$$< (\text{"}FEATURE\_SET\_SIZE\text{"}), \sum_{j,i \in \{1,..,n\}} d_{ij} >$$

The above purpose is achieved by the following Map and Reduce functions. The getClassDocumentCount() method retrieves the number of documents that belong to a specific class from HDFS, that was calculated by BayesFeatureDriver.

```
BayesTfIdfMapper.map(StringTuple key, Double value):
    if (key.get(0).equals("WEIGHT"):
        output(key,value)
    else if (key.get(0).equals("DOCUMENT_FREQUENCY"):
        class = key.get(1)
        classDocumentCount = getClassDocumentCount(class)
        output(("WEIGHT",class,term), ln(classDocumentCount / value))
    else:
        output(("FEATURE_SET_SIZE"),1)

BayesTfIdfReducer.reduce(StringTuple key, Double value):
    if (key.get(0).equals("FEATURE_SET_SIZE"):
        vocabCount = 0.0
        for (value : values):
            vocabCount += value
        output(key,vocabCount)
    else if (key.get(0).equals("WEIGHT"):
        tfIdf = 1.0
        for (value : values)
            tfIdf *= value
        output(key,tfIdf)
```

The BayesWeightSummerDriver job processes the results that were produced by Bayes TfIdf driver so as to calculate the weight sums for each term and each class. Bayes WeightSummerDriver outputs pairs of three types : FEATURE_SUM, LABEL_SUM and TOTAL_SUM.

FEATURE_SUM pairs hold the weight sum values for each term:

$$< (\text{"}FEATURE\_SUM\text{"}, term(t_j)), \sum_k TfIdf(k,j) >$$

LABEL_SUM pairs hold the weight sum values for each class:

$$< (\text{"}LABEL\_SUM\text{"}, class(y_k)), \sum_j TfIdf(k,j) >$$

The TOTAL_SUM pair holds the total sum value:

$$< (\text{"}TOTAL\_SUM\text{"}), \sum_{k,j} TfIdf(k,j) >$$

The above purpose is achieved by the following Map and Reduce functions.

```
BayesWeightSummerMapper.map(StringTuple key, Double value):
    class = key.get(1)
    term = key.get(2)
    output(("FEATURE_SUM",term), value)
    output(("LABEL_SUM",class), value)
    output(("TOTAL_SUM"), value)

BayesWeightSummerReducer.reduce(StringTuple key, List<Double> values)
    sum = 0.0
    for value : values
        sum += value
    output(key,sum)
```

The `BayesThetaNormalizerDriver` job calculates the weight normalization factor for each class, based on the results that were generated by `BayesTfIdfDriver` and `BayesWeightSummerDriver`. `BayesThetaNormalizerDriver` outputs pairs of `LABEL_THETA_NORMALIZER` type, one for each class. Each of these pairs contains the weight normalization factor for this class:

$$< (("LABEL\_THETA\_NORMALIZER", class(y_k)), \sum_j \frac{TfIdf(k,j)+1}{Sigma_k(k)+M}) >$$

The above purpose is achieved by the following `Map` and `Reduce` functions. `getSigma_k()` and `getVocabCount()` methods retrieve the `Sigma_k` value of class k that was calculated by `BayesWeightSummerDriver` and the number of all terms that are included in the document vocabulary that was calculated by `BayesTfIdfDriver` respectively. The `Map` function operates on the output that was created by `BayesTfIdfDriver`.

```
BayesThetaNormalizerMapper.map(StringTuple key, Double value):
    class = key.get(1)
    output(("LABEL_THETA_NORMALIZER", label),
        ln((value + 1.0) / (getSigma_k(class) + getVocabCount())))

BayesThetaNormalizerReducer.reduce(StringTuple key, List<Double> value):
    sum = 0.0
    for (value : values):
        sum += value
    output(key,sum)
```

7.2.2 Application Execution and Input Files

In order to train the Bayes Text Classifier, we use the training set provided by Cloudsuite, which includes classified Wikipedia texts in an XML format. The training set file, whose size is 5.4 GB, is uploaded to HDFS using the `wikipediaXMLSplitter` command of Mahout as follows. The name of the input Wikipedia xml is `wikipedia-training-input.xml` and is placed in the `$MAHOUT_HOME/examples/temp` directory. The size of each XML chunk is determined by the `-c` switch. In this case, we have used a chunk size of 16 MB.

```
$MAHOUT_HOME/bin/mahout wikipediaXMLSplitter
    -d $MAHOUT_HOME/examples/temp/wikipeida-training-input.xml
    -o wikipedia/chunks -c 16

ageo@mitsos:~$ $HADOOP_HOME/bin/hadoop dfs
    -lsr wikipedia-training
drwxr-xr-x   - root supergroup          0 2015-04-14 13:11
    /user/root/wikipedia-training
drwxr-xr-x   - root supergroup          0 2015-04-14 13:13
    /user/root/wikipedia-training/chunks
-rw-r--r--   1 root supergroup   16934212 2015-04-14 13:11
    /user/root/wikipedia-training/chunks/chunk-0001.xml
-rw-r--r--   1 root supergroup   16912840 2015-04-14 13:12
    /user/root/wikipedia-training/chunks/chunk-0002.xml
-rw-r--r--   1 root supergroup   16921155 2015-04-14 13:12
    /user/root/wikipedia-training/chunks/chunk-0003.xml
-rw-r--r--   1 root supergroup   16891897 2015-04-14 13:12
    /user/root/wikipedia-training/chunks/chunk-0004.xml
. . . . . .
```

Subsequently, we split the Wikipedia XML chunks, so as to create four datasets, whose size is 256 MB, 512 MB, 1 GB and 2 GB.

```
ageo@mitsos:~$ $HADOOP_HOME/bin/hadoop dfs
    -lsr wikipedia-training/dataset-512
drwxr-xr-x   - root supergroup          0 2015-04-14 13:11
    /user/root/wikipedia-training
drwxr-xr-x   - root supergroup          0 2015-04-14 13:13
    /user/root/wikipedia-training/chunks
-rw-r--r--   1 root supergroup   16934212 2015-04-14 13:11
    /user/root/wikipedia-training/chunks/chunk-0009.xml
-rw-r--r--   1 root supergroup   16912840 2015-04-14 13:12
    /user/root/wikipedia-training/chunks/chunk-0010.xml
-rw-r--r--   1 root supergroup   16921155 2015-04-14 13:12
    /user/root/wikipedia-training/chunks/chunk-0011.xml
-rw-r--r--   1 root supergroup   16891897 2015-04-14 13:12
    /user/root/wikipedia-training/chunks/chunk-0012.xml
```

Before the Bayes Classifier can be trained, the category based splits of the Wikipedia training dataset have to be created, by `wikipediaDataSetCreator`. The `categories.txt` file contains the possible categories (i.e. classes) that an input document can be assigned to. In this setup, we have 25 classes. The following example demonstrates how the 256 MB input dataset is transformed into category based splits, which are subsequently used by Mahout so as to train the classifier model. Each file that is created by `wikipediaDataSetCreator` contains `<key,value>` pairs, where each key represents the class of the document and the value contains the terms of this document seperated by spaces.

```
ageo@mitsos:~$ $MAHOUT_HOME/bin/mahout wikipediaDataSetCreator
```

```
    -i wikipedia-training/chunks/dataset-256
    -o traininginput-256
    -c $MAHOUT_HOME/examples/temp/categories.txt

bin/hadoop dfs -lsr traininginput/dataset-256
. . . . . .
-rw-r--r--   1 ageo supergroup          0 2015-02-04 22:59
    /user/ageo/traininginput/dataset-256/part-r-00000
-rw-r--r--   1 ageo supergroup   13907858 2015-02-04 23:00
    /user/ageo/traininginput/dataset-256/part-r-00001
. . . . . .
-rw-r--r--   1 ageo supergroup   36979226 2015-02-04 23:01
    /user/ageo/traininginput/dataset-256/part-r-00023
-rw-r--r--   1 ageo supergroup          0 2015-02-04 22:59
    /user/ageo/traininginput/dataset-256/part-r-00024

ageo@mitsos:~/hadoop-0.20.2$ bin/hadoop dfs -cat
    /user/ageo/traininginput/dataset-256/part-r-00018
religion        monty python's life brian 17920 382763986 2010-09-03t22 32
49z polisher cobwebs 12812034 infobox film name monty python s life brian
image lifeofbrianfilmposter jpg writer unbulleted list graham chapman john
. . . . . .
```

In order to train the Bayes Classifier and create the document classification model, the `trainclassifier` command of Mahout has to be executed. In the following example, after the training of the model has completed, the model parameters, i.e. $TfIdf(k, j)$, $Sigma_j(j)$, $Sigma_k(k)$, $Sigma_j Sigma_k$ and $Theta(k)$ will be stored in `trainer-tfIdf/trainer-tfIdf`, `trainer-weights/Sigma_j`, `trainer-weights/Sigma_k`, `trainer-weights/Sigma_jSigma_k` and `trainer-thetaNormalizer` HDFS directories respectively. All of those directories are located under `wikipediamodel-256` directory.

```
ageo@mitsos:~$ $MAHOUT_HOME/bin/mahout trainclassifier
    -i traininginput/dataset-256
    -o wikipediamodel-256
    -mf 4 -ms 4
. . . . . .
15/04/13 12:20:39 INFO bayes.TrainClassifier: Training Bayes Classifier
15/04/13 12:20:40 INFO bayes.BayesDriver: Reading features...
. . . . . .
15/04/13 12:21:49 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
15/04/13 13:22:56 INFO mapred.JobClient:  map 100% reduce 100%
. . . . . .
15/04/13 13:23:26 INFO bayes.BayesDriver: Calculating Tf-Idf...
. . . . . .
15/04/13 13:24:53 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
```

```
15/04/13 13:49:35 INFO mapred.JobClient:  map 100% reduce 100%
. . . . . .
15/04/13 13:50:10 INFO bayes.BayesDriver:
    Calculating weight sums for labels and features...
. . . . . .
15/04/13 13:51:33 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
15/04/13 14:11:06 INFO mapred.JobClient:  map 100% reduce 100%
. . . . . .
15/04/13 14:11:37 INFO bayes.BayesDriver:
    Calculating the weight Normalisation factor for each class...
. . . . . .
15/04/13 14:13:11 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
15/04/13 14:29:03 INFO mapred.JobClient:  map 100% reduce 100%
. . . . . .
15/04/13 14:29:36 INFO driver.MahoutDriver:
    Program took 7736347 ms (Minutes: 128.93911666666668)

ageo@mitsos:~/hadoop-0.20.2$ bin/hadoop dfs -lsr wikipediamodel-256
. . . . . .
drwxr-xr-x   - ageo supergroup          0 2015-04-13 13:48
    /user/ageo/wikipediamodel/trainer-tfIdf/trainer-tfIdf
-rw-r--r--   1 ageo supergroup     788697 2015-04-13 13:46
    /user/ageo/wikipediamodel/trainer-tfIdf/trainer-tfIdf/part-00000
    -rw-r--r--   1 ageo supergroup     806391 2015-04-13 13:46
    /user/ageo/wikipediamodel/trainer-tfIdf/trainer-tfIdf/part-00001
. . . . . .
-rw-r--r--   1 ageo supergroup     797885 2015-04-13 13:46
    /user/ageo/wikipediamodel/trainer-tfIdf/trainer-tfIdf/part-00030
-rw-r--r--   1 ageo supergroup     794212 2015-04-13 13:47
    /user/ageo/wikipediamodel/trainer-tfIdf/trainer-tfIdf/part-00031
. . . . . .
drwxrwxrwx   - ageo supergroup          0 2015-04-13 14:28
    /user/ageo/wikipediamodel/trainer-thetaNormalizer
-rw-r--r--   1 ageo supergroup         99 2015-04-13 14:26
    /user/ageo/wikipediamodel/trainer-thetaNormalizer/part-00000
-rw-r--r--   1 ageo supergroup        131 2015-04-13 14:26
    /user/ageo/wikipediamodel/trainer-thetaNormalizer/part-00001
. . . . . .
-rw-r--r--   1 ageo supergroup        163 2015-04-13 14:26
    /user/ageo/wikipediamodel/trainer-thetaNormalizer/part-00030
-rw-r--r--   1 ageo supergroup         99 2015-04-13 14:27
    /user/ageo/wikipediamodel/trainer-thetaNormalizer/part-00031
drwxrwxrwx   - ageo supergroup          0 2015-04-13 14:10
    /user/ageo/wikipediamodel/trainer-weights
drwxr-xr-x   - ageo supergroup          0 2015-04-13 14:09
```

```
    /user/ageo/wikipediamodel/trainer-weights/Sigma_j
-rw-r--r--   1 ageo supergroup      82475 2015-04-13 14:08
    /user/ageo/wikipediamodel/trainer-weights/Sigma_j/part-00000
-rw-r--r--   1 ageo supergroup      83082 2015-04-13 14:08
    /user/ageo/wikipediamodel/trainer-weights/Sigma_j/part-00001
. . . . . .
-rw-r--r--   1 ageo supergroup      81464 2015-04-13 14:09
    /user/ageo/wikipediamodel/trainer-weights/Sigma_j/part-00030
-rw-r--r--   1 ageo supergroup      82977 2015-04-13 14:09
    /user/ageo/wikipediamodel/trainer-weights/Sigma_j/part-00031
drwxr-xr-x   - ageo supergroup          0 2015-04-13 14:09
    /user/ageo/wikipediamodel/trainer-weights/Sigma_k
-rw-r--r--   1 ageo supergroup        133 2015-04-13 14:08
    /user/ageo/wikipediamodel/trainer-weights/Sigma_k/part-00001
-rw-r--r--   1 ageo supergroup        130 2015-04-13 14:08
    /user/ageo/wikipediamodel/trainer-weights/Sigma_k/part-00003
. . . . . .
-rw-r--r--   1 ageo supergroup        169 2015-04-13 14:08
    /user/ageo/wikipediamodel/trainer-weights/Sigma_k/part-00026
-rw-r--r--   1 ageo supergroup        167 2015-04-13 14:09
    /user/ageo/wikipediamodel/trainer-weights/Sigma_k/part-00029
drwxr-xr-x   - ageo supergroup          0 2015-04-13 14:09
    /user/ageo/wikipediamodel/trainer-weights/Sigma_kSigma_j
-rw-r--r--   1 ageo supergroup        125 2015-04-13 14:09
    /user/ageo/wikipediamodel/trainer-weights/Sigma_kSigma_j/part-00013
. . . . . .
```

7.2.3 Scalability Analysis Per Input Size

This section presents the analysis we have conducted regarding the scalability of the `Bayes Classifier` application, in terms of input size, when it runs on the Intel SCC. We have executed the application with four different input files, whose size is 256 MB, 512 MB, 1 GB and 2 GB. Those files were transformed to category based splits before they were used in order to train the text classification model. We have utilized the 48-Node Cluster Topology for this analysis and have configured both the DataNodes and the TaskTrackers to operate at the maximum frequency of 800 MHz. The experimental results we have received regarding the execution time and the energy consumption of the `Bayes Classifier` application are presented below. Detailed plots that illustrate the CPU utilization and the network traffic for one DataNode and one TaskTracker as well as the overall power consumption and board temperature of the Intel SCC, for each run, are included in Appendix B2.

Our analysis indicates evidently that both the execution time and the energy consumption of the `Bayes Classifier` application scale linearly as the size of the input documents increases. The detailed plots of the CPU utilization of the TaskTracker nodes indicate that this increase is primarily attributed to the expansion of the ex-

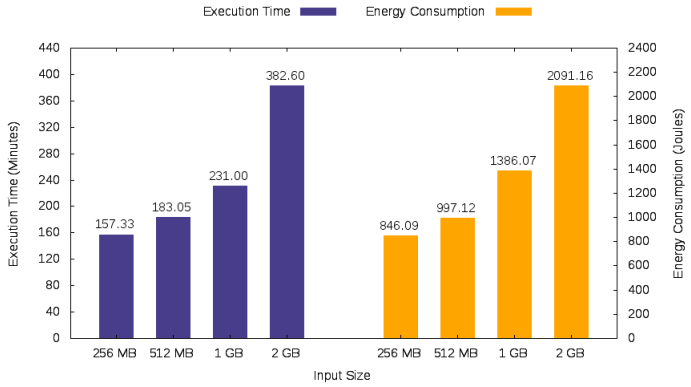


Figure 7.14: Bayes Classifier Input Size Scalability Analysis (1/2)

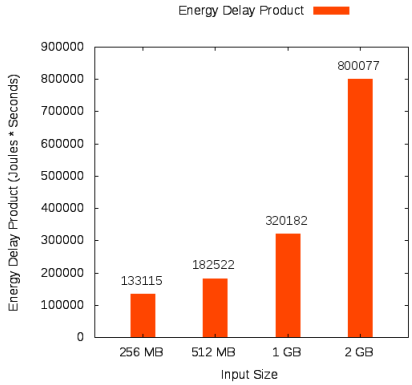


Figure 7.15: Bayes Classifier Input Size Scalability Analysis (2/2)

ecution time of the Map phase of the `BayesFeatureDriver` job and secondarily to a smaller increase of the Reduce phase of this job. The increased execution time of the Map phase can be explained by the fact that the category based splits that are processed by `BayesFeatureDriver` include more `<class,document>` pairs as the input size increases and as a result are stored in more `InpuSplits` in HDFS, resulting in an increased amount of issued Map tasks. The increase in the Reduce phase can be explained by the fact that the Map phase of the MapReduce job generates more intermediate `<key,value>` pairs, increasing the processing load of the Reduce phase and thus the execution time.

In the beginning of each MapReduce job a period which is dominated by idle CPU cycles because of outstanding I/O, for all input sizes, is noticed. This behavior can be attributed to the fact that the `mahout-examples-0.6-job.jar` is distributed to each TaskTracker during the initialization of a MapReduce job and is expanded when the first task (Map or Reduce) is executed on this node. This jar file has a size of 23 MB and its expanded contents have a total size of 85 MB, adding up to a total of 108 MB outgoing I/O per TaskTracker. The presence of a high percentage of idle CPU cycles due to outstanding I/O indicates that the I/O bandwidth between the Intel SCC and the NFS that is mounted on `/shared` is saturated, causing the cores to stall until

the jar is expanded and its contents are stored in the physical storage of the MCPC. The increased I/O of this period is also depicted in the Network Traffic plots of the TaskTracker nodes, since each core accesses the NFS through the `emac0` virtual network interface and as a consequence, disk I/O is recorded as network traffic by `gmond`. The reason that such a behavior was not evident in the `Wordcount` application is that the corresponding jar was `hadoop-0.20.2-examples.jar`, whose size is 140 KB, and whose expanded contents in each TaskTracker are 484 KB, adding up to a total of only 624 KB outgoing I/O per TaskTracker, which did not cause the I/O bandwidth saturation we observe at the `Bayes Classification` application.

7.2.4 Cluster Topology Analysis

This section presents our analysis concerning the behavior of the `Bayes Classifier` application when it is executed on top of different HDFS cluster topologies on the Intel SCC. For this study, we have used the category based splits which were generated by the 256 MB dataset. We have configured both the DataNodes and the TaskTrackers of each cluster topology to operate at the maximum frequency of 800 MHz. The idle nodes of each topology (if any) operate at the minimum frequency of 100 MHz. `gmond` is not active on those nodes as well. The experimental results we have received regarding the execution time and the energy consumption of the `Bayes Classifier` application are presented below. Detailed plots are included in Appendix B2, as in the previous case.

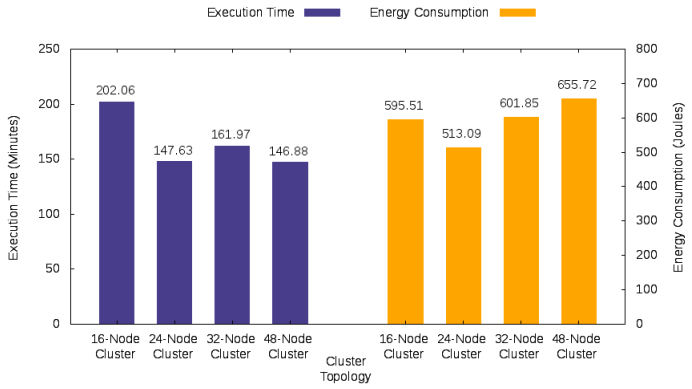


Figure 7.16: Bayes Classifier Cluster Topology Analysis (1/2)

Our experimental results clearly indicate that the 48-Node cluster topology is non-optimal for the `Bayes Classifier` application, if the energy consumption is taken into account apart from the execution time, in contrast to our conclusion for the `Wordcount` application. The application completes at approximately the same time when it is executed on the 24-Node and the 48-Node cluster, but because of the lower power consumption of the 24-Node cluster, it consumes 22% less energy. This observation can be explained by the fact that the period that was characterized by a high percentage of idle CPU cycles because of outstanding I/O is reduced significantly

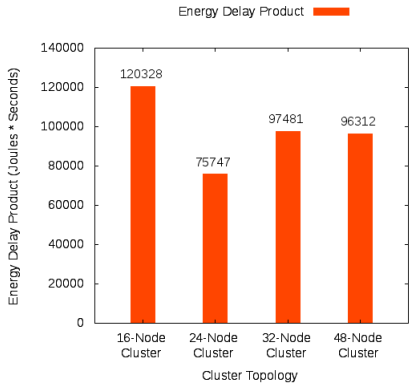


Figure 7.17: Bayes Classifier Cluster Topology Analysis (2/2)

as the number of cores that participate in the cluster drops, since the requested I/O bandwidth and thus the I/O saturation diminish. As a consequence, the reduced parallelism that is imposed by the smaller number of TaskTracker nodes is mitigated by the fact that less CPU cycles are wasted for outstanding I/O requests, resulting in the same execution time and reduced energy consumption for the 24-Node cluster topology compared to the 48-Node cluster topology.

It has to be noted however, that this conclusion would be most probably overturned if the total size of the input category based splits was increased, because of the fact that the CPU-intensive part of the application would be expanded and the impact of the reduced parallelism would be more intense. This fact would result in higher execution time for the 24-Node topology and probably higher energy consumption if the execution time overhead is significant.

The 24-Node topology outperforms the 32-Node topology in the `Bayes Classifier` application, similar to our conclusion regarding the `Wordcount` application. That is, the application does not benefit from the increased number of DataNodes, yielding higher execution time than the 24-Node cluster topology and even higher energy consumption because of the higher power consumption it is charged with.

7.2.5 Frequency Scaling Analysis

This section analyzes the impacts of frequency scaling on the execution time and the energy consumption of the `Bayes Classifier` application on the Intel SCC. We have tested the category based splits that were generated by the 256 MB dataset in the 48-Node Cluster topology for nine frequency settings. We have configured the DataNodes and Master Node and the TaskTrackers to run at either 200 MHz, 533 MHz or 800 MHz and each frequency setting represents one combination of those values. The experimental results we have received regarding the execution time and the energy consumption of the `Bayes Classifier` application are presented below. Detailed plots are included in Appendix B2, as in the previous case.

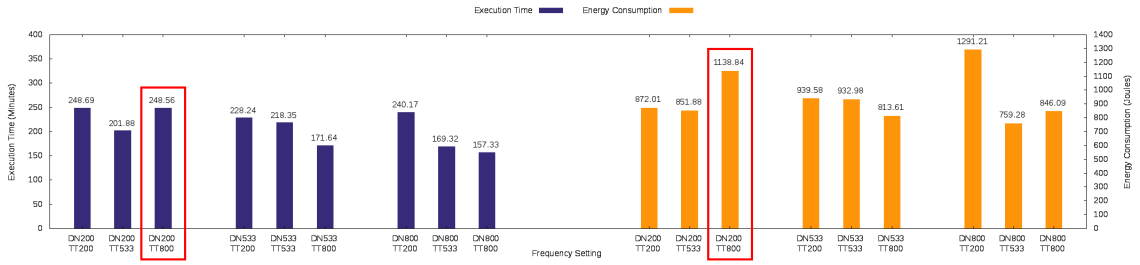


Figure 7.18: Bayes Classifier Frequency Scaling Analysis (1/2)

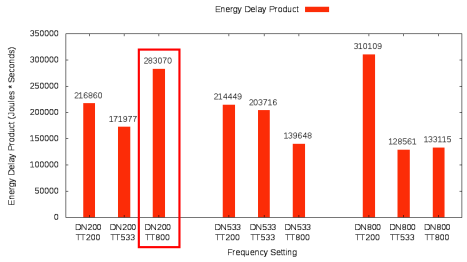


Figure 7.19: Bayes Classifier Frequency Scaling Analysis (2/2)

Similar to the `Wordcount` application, the conclusion that both the execution time and the energy consumption are driven by the frequency of the `TaskTrackers` can be drawn. Increasing the frequency of the DataNodes, does not appear to yield any significant benefit in terms of the execution time, while charging the application with higher energy consumption. The experimental results for the DN200-TT800 setting seem to contradict the above conclusions.

In the DN200-TT800 case, the failure of DataNode `rck13` between the 30th and the 40th minute and the unusually long time it took to be rebooted and rejoin the cluster by the node failover watchdog caused a series of Map tasks to fail and be re-executed. This fact prolonged the Map phase of `BayesFeatureDriver` for more than an hour, leading to a misleading execution time and energy consumption outcome. The CPU utilization plots that are included in the Figure 7.20 illustrate that situation. In the CPU utilization plot of `rck13`, the period that is distinguished by persistent CPU utilization metrics corresponds to the time when the core was unreachable, thus `gmond` did not report any updated metrics.

In order to corroborate the claim that the execution time we observed is misleading, we re-executed the `Bayes Classification`, using the DN200-TT800 setting and the execution time we recorded was 159.27 minutes. This execution was not characterized by any unusual node failures and the time it took to complete is close to the DN533-TT800 and DN800-TT800 settings, indicating that scaling down the frequency of DataNodes does not impair performance, while yielding energy consumption savings.

However, we could not obtain an accurate estimation of the energy consumption of this re-execution, because the power consumption we recorded was on average 20 W lower than the power consumption we observed in the first run. This fact is attributed to lower platform temperature, which is a result of lower platform utilization during the period the second execution was performed. As a consequence, comparing the energy consumption measurement we observed in the second run with the one we observed in the first run would be also misleading.

In order to provide a fair comparison in terms of energy consumption as well, we also re-executed the application using the DN800-TT800 setting. Our results regarding execution time and energy consumption are included in the following table. This comparison indicates that scaling down the frequency of the DataNodes to 200 MHz, despite increasing the execution time by 8.4% manages to reduce the energy consumption of the application by 3.7%, because of the reduced power budget of the cluster.

Frequency Setting	Execution Time	Energy Consumption	Energy Delay Product
DN200-TT800	159.27	631.29	100546
DN800-TT800	146.89	655.72	96319

This behavior is expected to be maintained for bigger input sizes, because of the fact that increasing the input size prolongs the CPU-intensive parts of the application which are executed by the TaskTrackers, which operate at the maximum frequency of 800 MHz. Moreover, it has to be mentioned that, as in the previous applications, the energy consumption saving would have been more notable if the Intel SCC architecture allowed Voltage Scaling at the tile level, enabling us to scale down the voltage of the DataNodes which operate at the frequency of 200 MHz.

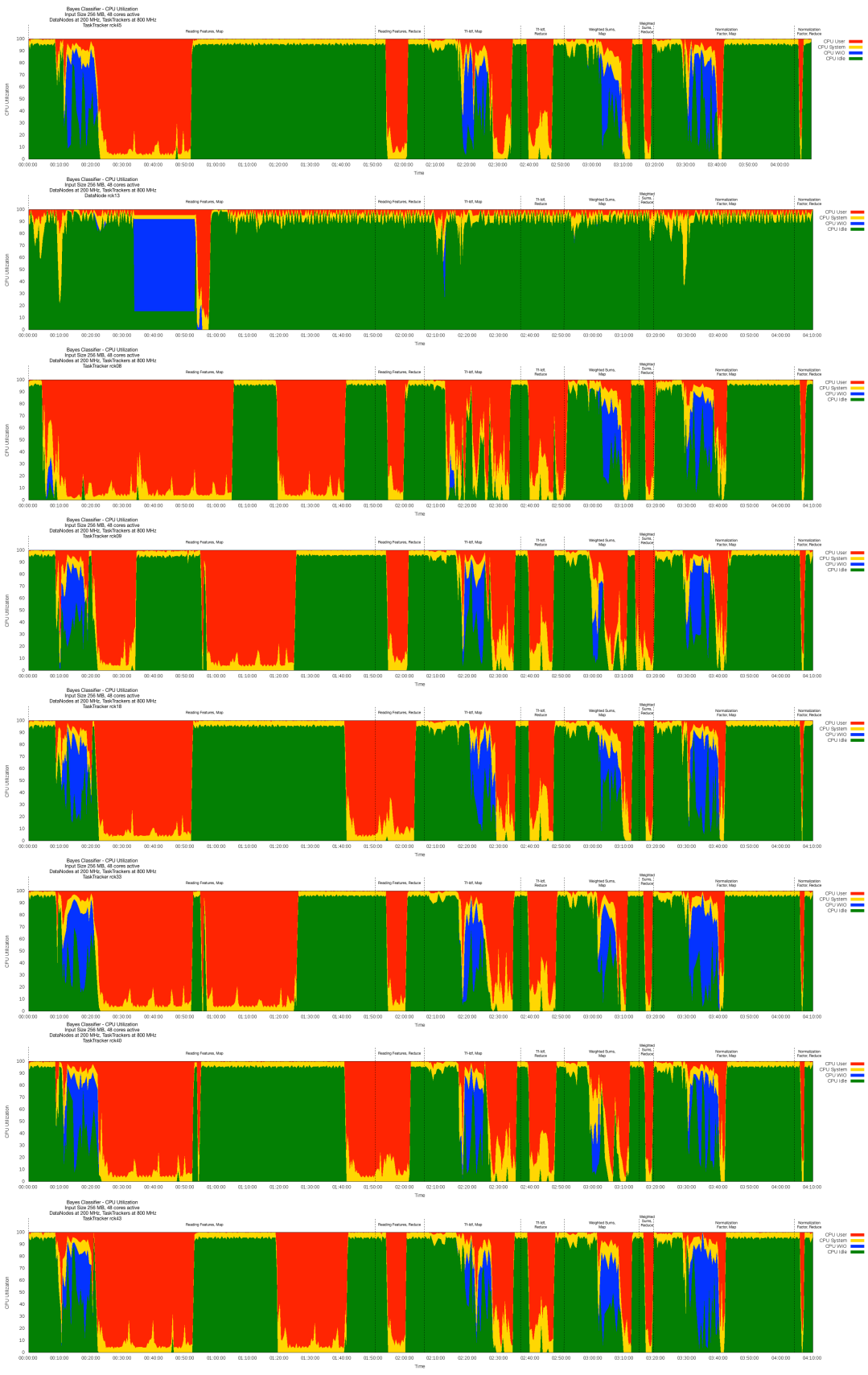


Figure 7.20: CPU utilization plots for the Bayes Classifier application

7.2.6 Cluster Utilization Overview

This section provides an overview figure (Figures 7.21-7.26) for the CPU utilization of all cluster nodes, when the `Bayes Classification` application is executed with category-based splits that has been generated from the input file of 512 MB, on the 48-Node HDFS cluster topology and with the DataNodes and the TaskTrackers configured to operate at 800 MHz. The very low utilization of all the DataNodes which explains the minimal performance impairment that we observe when their operating frequency is scaled down to 200 MHz. The CPU utilization diagrams of the `TaskTracker` nodes depict the execution on Map and Reduce tasks on the cores that they are hosted.

The CPU utilization plots of the TaskTracker nodes also denote the period at the beginning of each MapReduce job, when the `mahout-examples-0.6-job.jar` is distributed and expanded at those cores. This period is characterized by a high percentage of CPU cycles due to outstanding I/O. The overview utilization figure also depicts that the CPU idle period that is observed in the beginning of each MapReduce job is attributed to the execution of the setup Map task in one TaskTracker node. Each MapReduce job executes one setup Map task before the beginning of computation, which performs the job initialization.

Another observation that we can make from that figure is that the execution time of each Map or Reduce task varies per node. In addition a different number of Map and Reduce tasks are assigned to each TaskTracker by the JobTracker, depending on the execution status of the MapReduce jobs. Finally, we can spot the cores that froze and were rebooted during the execution of the application. The time period during which the cores where unreachable is characterized by persistent measurements of the CPU utilization. This behavior is explained by the fact that since the `gmond` instance that runs on the MCPC did not receive any updated values for the CPU utilization, it reported the last value it received from the core again and again, until the core was rebooted and new metrics were received through UDP datagrams.

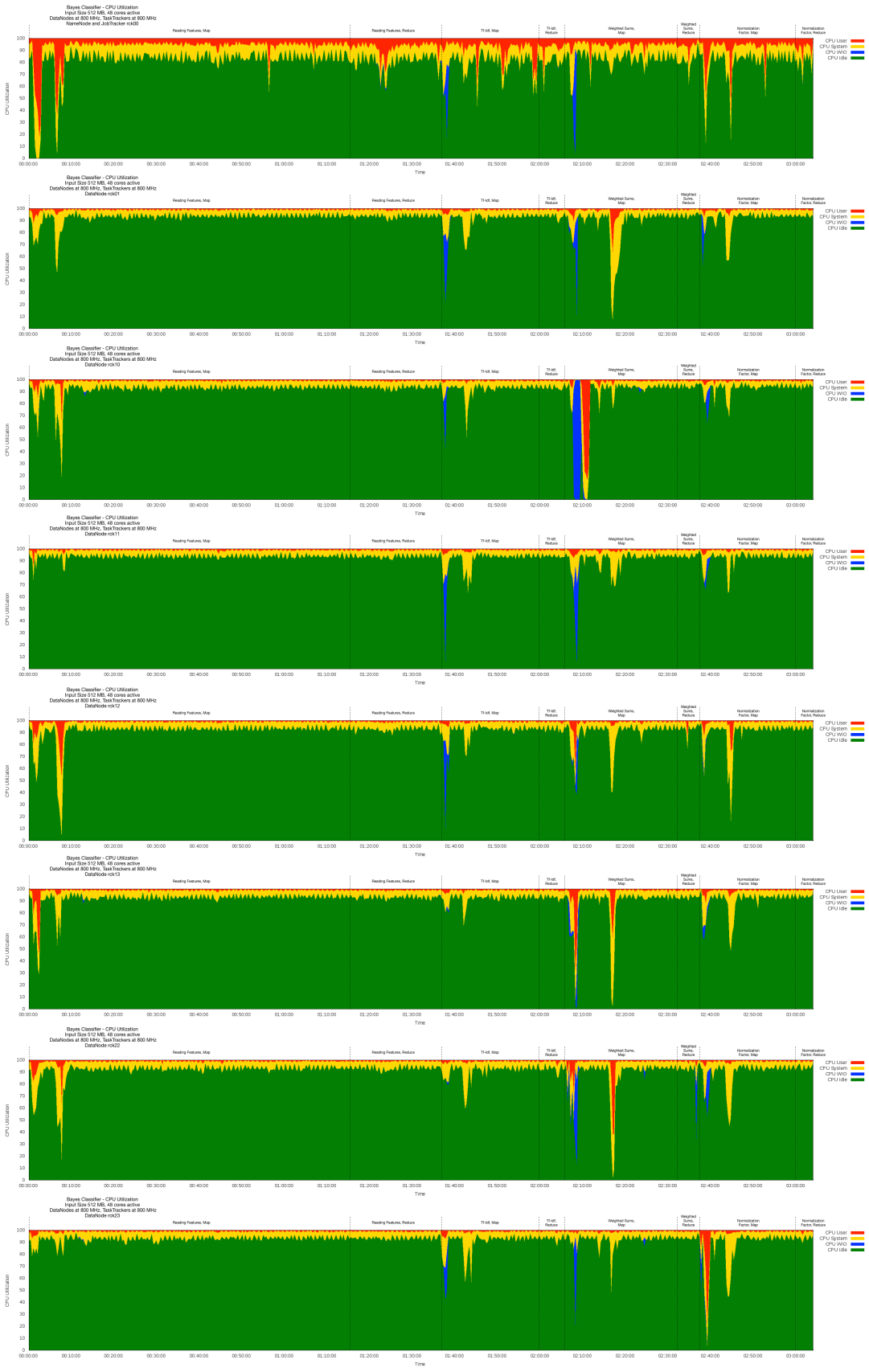


Figure 7.21: Bayes Classification Overall Cluster Utilization (1/6)

Figure 7.22: Bayes Classification Overall Cluster Utilization (2/6)

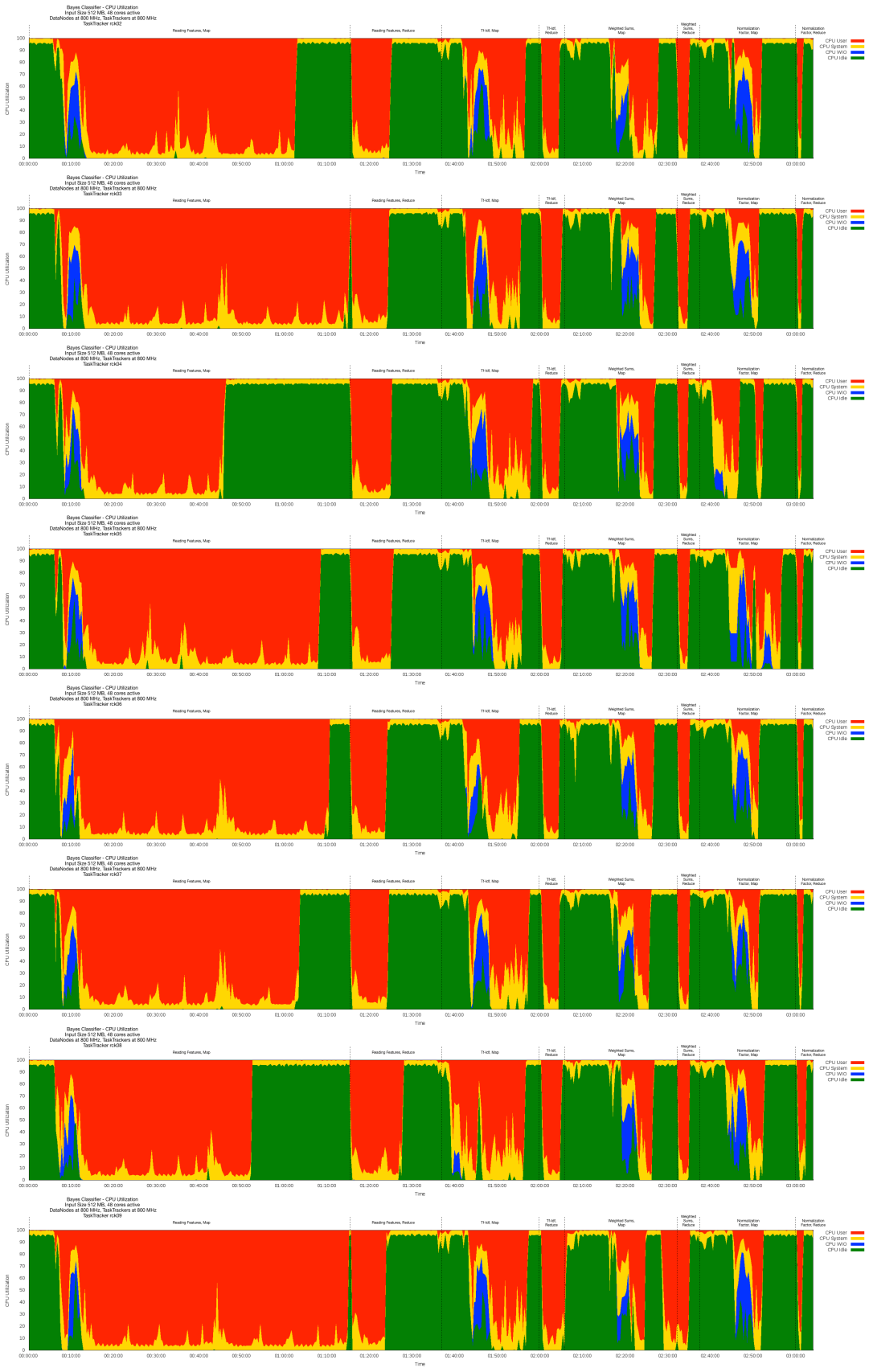


Figure 7.23: Bayes Classification Overall Cluster Utilization (3/6)

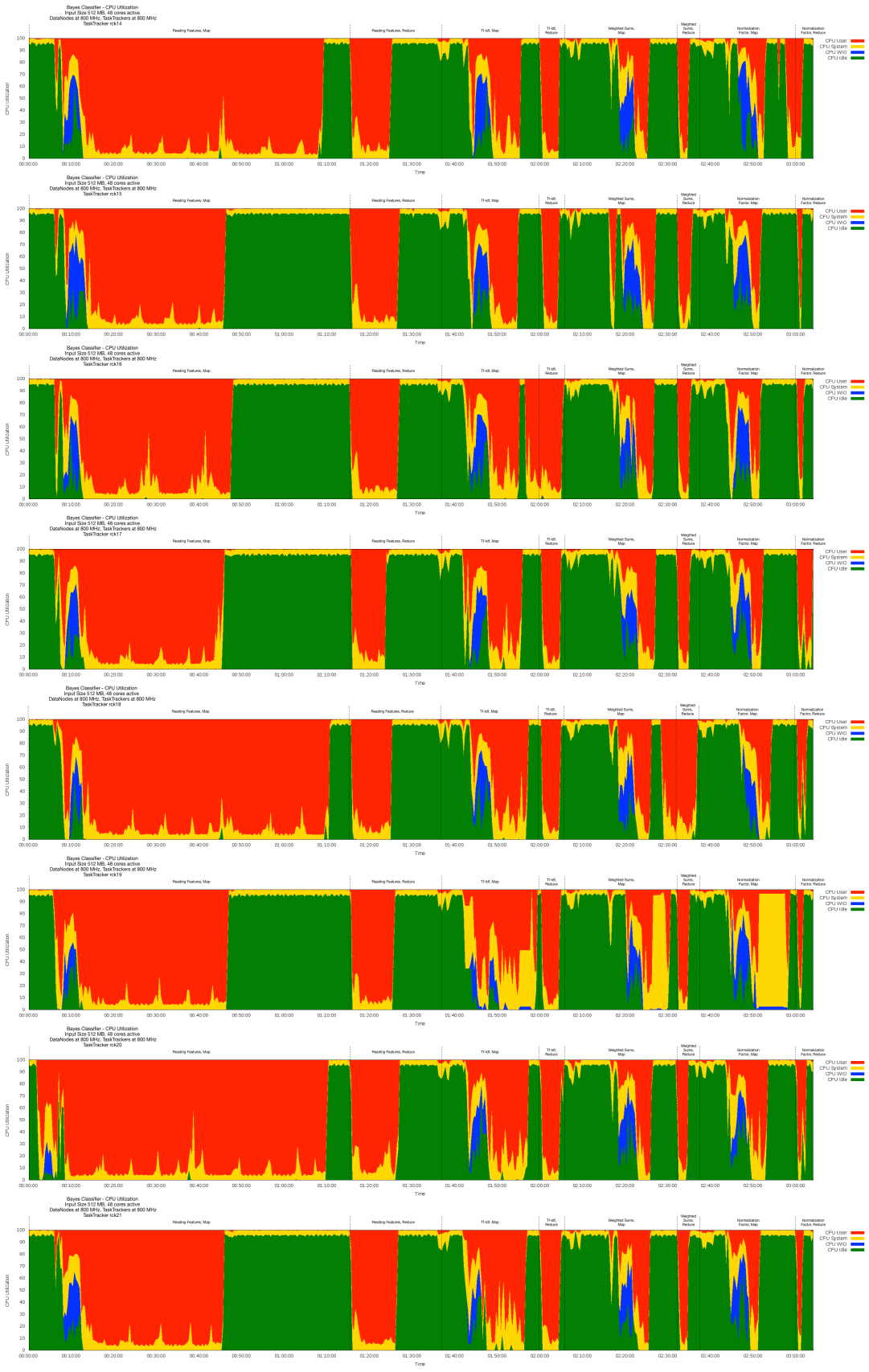


Figure 7.24: Bayes Classification Overall Cluster Utilization (4/6)

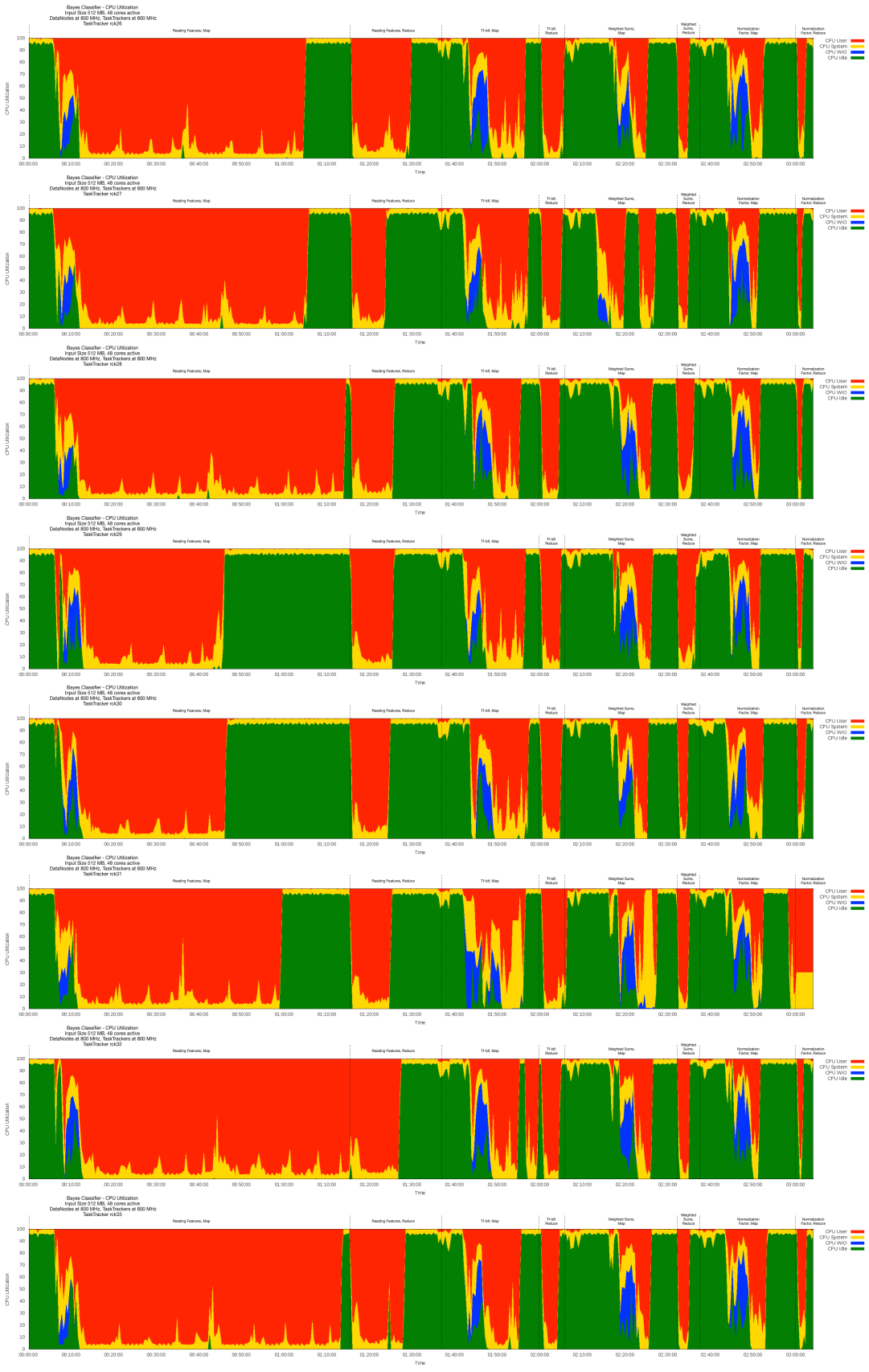


Figure 7.25: Bayes Classification Overall Cluster Utilization (5/6)

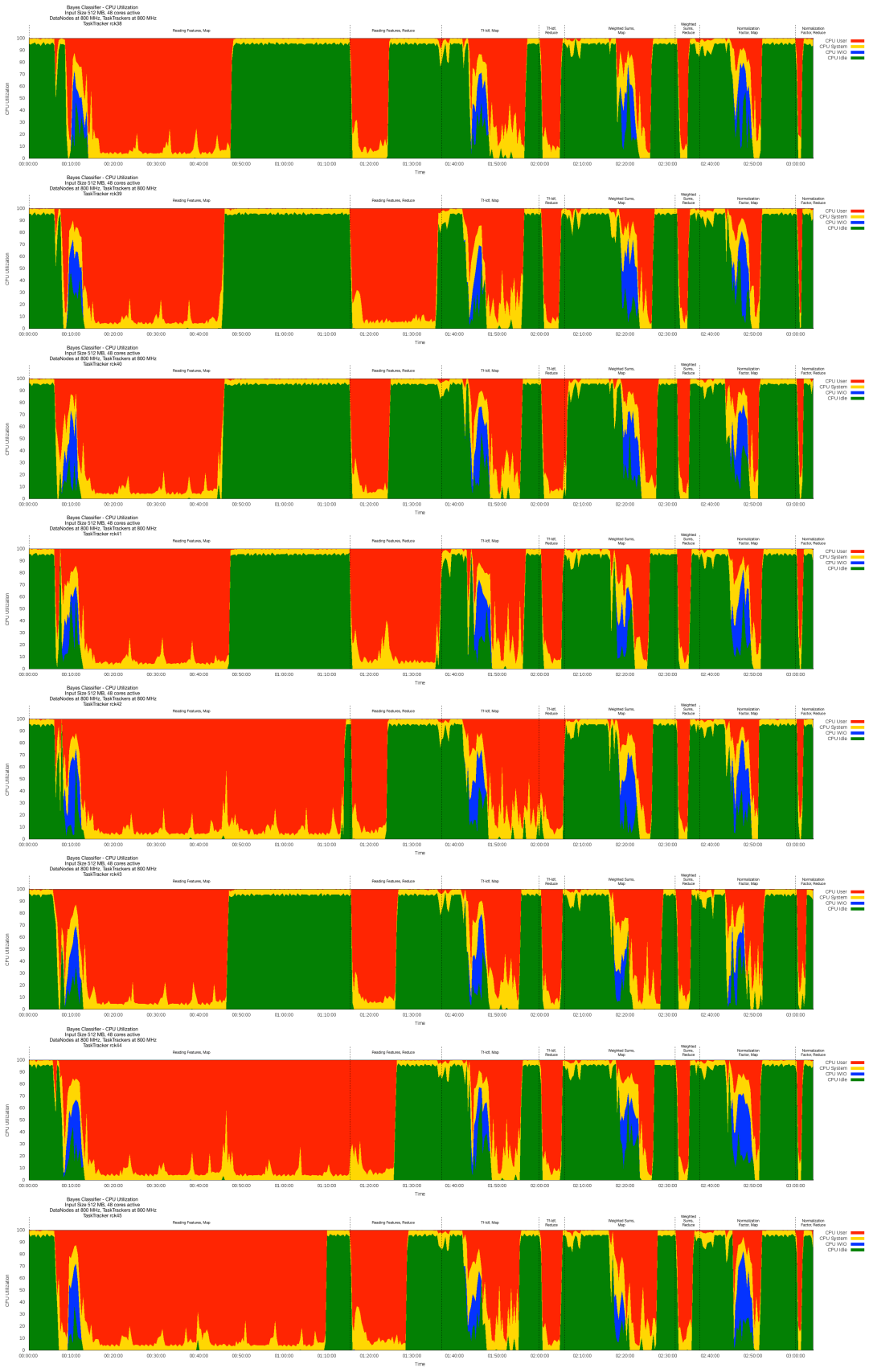


Figure 7.26: Bayes Classification Overall Cluster Utilization (6/6)

7.3 The K-Means Clustering Application

This section presents our analysis regarding the execution of the `K-Means Clustering` application on the Intel SCC. We initially describe the MapReduce implementation of the `K-Means Clustering` algorithm. In addition, the input file generation and application execution process are stated. We have utilized resources which are provided by DCBench for that purpose. Subsequently, the experimental results we have received are analyzed, in order to draw conclusions regarding the behavior of the `K-Means Clustering` application on the Intel SCC for different input sizes, cluster topologies and frequency settings.

7.3.1 Algorithm Description

K-Means algorithm is the most well-known and commonly used clustering method. It takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intra-cluster similarity is high whereas the inter-cluster similarity is low. Cluster similarity is measured according to the mean value of the objects in the cluster, which can be regarded as the cluster's center of gravity.

The algorithm proceeds as follows : Firstly, it randomly selects k objects from the whole objects, which represent initial cluster centers. Each remaining object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster center. The new mean for each cluster is then calculated. This process iterates until the criterion function converges.

The MapReduce implementation of K-Means clustering executes repeatedly a MapReduce job, which implements a parallel version of the K-Means algorithm. The input objects that have to be clustered are point vectors. The execution stops if the convergence criterion is met or if the job has been executed for the maximum number of iterations has been reached. The K-Means MapReduce job consists of a Mapper and a Reducer function, and a Combiner function, which combines the intermediate `<key,value>` pairs that are generated by the Mappers locally before they are processed by the Reducers, for performance optimization.

`KMeansMapper` iterates over the point vectors of the input file and searches for the cluster that yields the minimum distance for each specific point. `KMeansMapper` outputs intermediate `<key,value>` pairs, whose key is the `clusterId` of the nearest cluster and the value is a tuple that consists of the number `1`, the point vector and the point vector which consists the squared values of the dimensions of the original vector. `KMeansCombiner` combines all the intermediate pairs that were generated by a specific Mapper and share the same `clusterId` by summing the elements that are included in the value tuple.

`KMeansReducer` processes all `<key,value>` pairs that share the same `clusterId` and computes the new center of the specific cluster, as the mean value of the point vectors that were assigned to it. `KMeansReducer` also checks if the cluster it processes has converged and if so it marks the corresponding flag of the cluster as true. The output

<key,value> pairs of each reducer contain the clusterId as the key and the Cluster object instance as the value.

The list of Clusters that is provided as input both in the Map and the Reduce functions contains the clusters that were computed by the Reduce phase of the previous iteration of the K-Means algorithm. In the first iteration of the K-Means algorithm, this lists contains a set of clusters with randomly selected center vector points.

```
KMeansMapper.map(String key, Point value, List<Cluster> clusters):
    nearestDistance = Double.MAX_VALUE
    nearestCluster = null
    for (cluster : clusters):
        distance = computeDistance(value,cluster)
        if (distance < minDistance):
            nearestDistance = distance
            nearestCluster = cluster
    nearestClusterId = nearestCluster.getId()
    output(nearestClusterId, (1,value,squareElements(value)))

KMeansMapper.computeDistance(point,cluster):
    clusterCenter = cluster.getCenter()
    return dotProduct((clusterCenter-point), (clusterCenter-point))

KMeansCombiner.combine(String key, List<ClusterObservation> values):
    sum1 = 0
    sum2 = new Point()
    sum3 = new Point()
    for (value : values):
        sum1 += value.get(1)
        sum2 += value.get(2)
        sum3 += value.get(3)
    output(key,(sum1,sum2,sum3))

KMeansReducer.reduce(String key, List<ClusterObservation> values,
    List<Cluster> clusters):
    sum1 = 0
    sum2 = new Point()
    sum3 = new Point()
    for (value : values):
        sum1 += value.get(1)
        sum2 += value.get(2)
        sum3 += value.get(3)
    cluster = clusters.get(key)
    clusterCenter = cluster.getCenter
    clusterCentroid = sum2.divide(sum1)
    vectorSumSquared = sum3
    converged = checkConvergence(clusterCenter,clusterCentroid)
    if (converged):
```

```
        cluster.setConverged(true)
    cluster.setCenter(clusterCentroid)
    output(key,cluster)

KMeansReducer.checkConvergence(clusterCenter, clusterCentroid, convergenceDelta):
    return dotProduct(clusterCentroid-clusterCenter,
                      clusterCentroid-ClusterCenter) <= convergenceDelta
```

7.3.2 Application Execution and Input Files

We use three different input files for the `K-Means Clustering` application, whose size is 121 KB, 4 MB and 16 MB. Those input files are provided by DC Bench. DC Bench also provides a script called `prepare-kmeans.sh`, which receives the desired input size as an input and uploads the corresponding file to HDFS. The code of this script is included in Appendix A.

```
ageo@mitsos:~/HVCBench-hadoop/workloads/cluster/kmeans$
    ./prepare-kmeans.sh low

ageo@mitsos:~/hadoop-0.20.2$
    bin/hadoop dfs -ls /cloudrank-data/sougou*
Found 2 items
drwxr-xr-x   - root supergroup          0 2015-04-15 12:08
    /cloudrank-data/sougou-low-tfidf-vec/_logs
-rw-r--r--   1 root supergroup     124357 2015-04-15 12:08
    /cloudrank-data/sougou-low-tfidf-vec/part-r-00000
```

In order to run the `K-Means Clustering` benchmark, the `run-kmeans.sh` script, which is provided by DCBench has to be executed. This script receives the size of the input file that contains the point vectors as a command line argument and searches in HDFS for the input file with this specific size that was uploaded by `prepare-kmeans.sh`. The code of this script is included in Appendix A.

```
ageo@mitsos:~/HVCBench-hadoop/workloads/cluster/kmeans$
    ./run-kmeans.sh low
. . . . . .
15/03/22 19:37:16 INFO kmeans.KMeansDriver: K-Means Iteration 1
. . . . . .
15/03/22 19:38:24 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
15/03/22 19:56:58 INFO mapred.JobClient:  map 100% reduce 100%
. . . . . .
15/03/22 19:57:34 INFO kmeans.KMeansDriver: K-Means Iteration 2
. . . . . .
15/03/22 20:01:14 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
15/03/22 20:23:12 INFO mapred.JobClient:  map 100% reduce 100%
. . . . . .
15/03/22 20:23:55 INFO driver.MahoutDriver:
```

```
Program took 2800226 ms (Minutes: 46.670433333333335)
```

7.3.3 Scalability Analysis Per Input Size

This section presents the analysis we have conducted regarding the scalability of the `K-Means Clustering` application, in terms of input size, when it runs on the Intel SCC. We have executed the application with three different input files, whose size is 121 KB, 4 MB and 16 MB as mentioned above. We have utilized the 48-Node Cluster Topology for this analysis and have configured both the DataNodes and the TaskTrackers to operate at the maximum frequency of 800 MHz. The experimental results we have received regarding the execution time and the energy consumption of the `K-Means Clustering` application are presented below. Both the execution time and the energy consumption are divided by the number of iterations the K-Means Clustering algorithm was executed, so as to provide a basis for fair comparison. Detailed plots that illustrate the CPU utilization and the network traffic for one DataNode and one TaskTracker as well as the overall power consumption and board temperature of the Intel SCC, for each run, are included in Appendix B3.

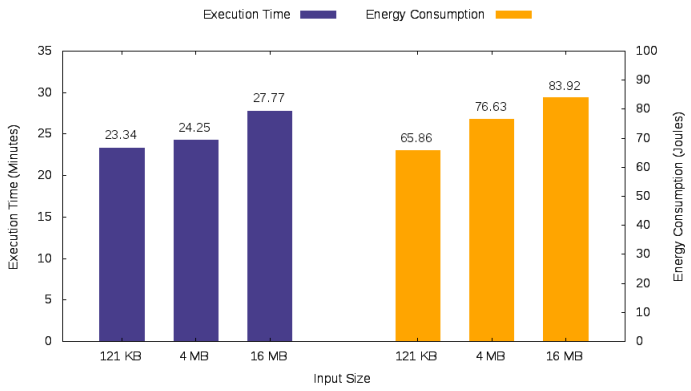


Figure 7.27: K-Means Clustering Input Size Scalability Analysis (1/2)

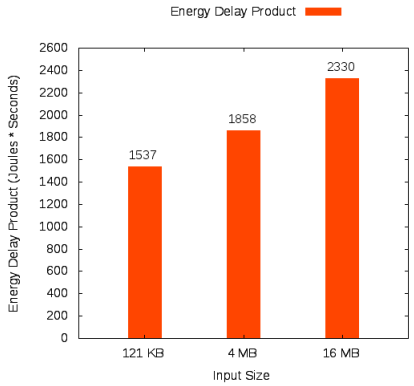


Figure 7.28: K-Means Clustering Input Size Scalability Analysis (2/2)

Our analysis points out that execution time and energy consumption do not scale significantly when the size of the input file increases. This behavior can be explained by the fact that the input files we have provided are stored in 1, 3 and 5 input splits in HDFS respectively. That is, the Map tasks that are issued are much less than the available TaskTracker nodes, meaning that the application cannot leverage the level of parallelism that is provided by the platform. The same conclusion can be drawn regarding the Reduce tasks, since the application attempts to create 5 clusters of point vectors, meaning that the rest 27 reduce tasks will not receive and process any intermediate `<key,value>` pairs. The slight increase in energy consumption for the 4 MB and 16 MB input files can be attributed to the fact that more Map tasks were issued in these cases, causing more Intel SCC cores operate at high CPU utilization and as a result increasing the power consumption of the platform.

In the 121 KB input file case, only 1 Map task is issued for both iterations, which is not evident in the diagram we have provided in the Appendix. Figure 7.29 depicts the execution of the setup Map tasks on `rck26` and `rck41` and the execution of the only Map task for Iterations 1 and 2 on `rck09` and `rck18` respectively.

It also has to be mentioned, that the period of high percentage of CPU idle cycles because of outstanding I/O and increased outgoing network traffic is also present in the `K-Means Clustering` application, since the `mahout-examples-0.6-job.jar` is distributed and expanded by all TaskTracker nodes in this case as well. In this case, this period is observed at the beginning of the Map phase, for nodes where Map tasks were executed and in the beginning of the Reduce phase, for nodes that did not execute Map tasks. In addition, it is evident in Figure 7.29 that this behavior is much more intense during the Reduce phase, because more nodes are attempting to expand their jar file, leading in higher I/O bandwidth contention and a higher percentage of wasted CPU cycles.

7.3.4 Cluster Topology Analysis

This section presents our analysis concerning the behavior of the `K-Means Clustering` application when it is executed on top of different HDFS cluster topologies on the Intel SCC. For this study, we have used the 121 KB input file which is provided by DCBench. We have configured both the DataNodes and the TaskTrackers of each cluster topology to operate at the maximum frequency of 800 MHz. The idle nodes of each topology (if any) operate at the minimum frequency of 100 MHz. `gmond` is not active on those nodes as well. The experimental results we have received regarding the execution time and the energy consumption of the `K-Means Clustering` application are presented below. Both the execution time and the energy consumption are divided by the number of iterations the K-Means Clustering algorithm was executed, so as to provide a basis for fair comparison. Detailed plots are included in Appendix B3, as in the previous case.

Our experimental results point out that increasing the number of DataNodes or TaskTrackers that participate in the cluster deteriorates both the execution time and the

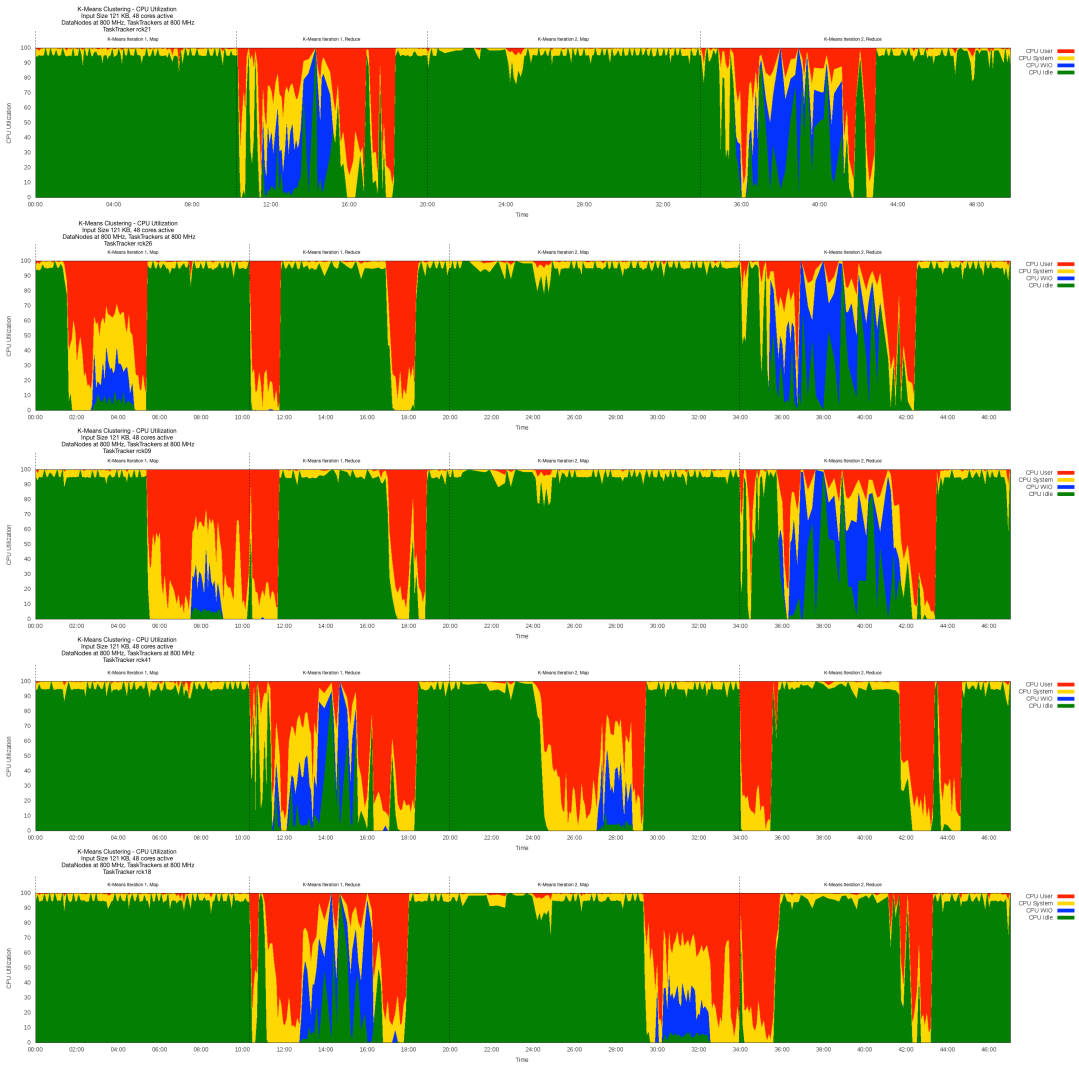


Figure 7.29: CPU Utilization Plots for the K-Means Clustering Application

energy consumption of the application. This behavior can be explained by the analysis we presented in the previous section. Since the input file we use is stored in one `InputSplit` in HDFS and only 5 reducer tasks will process intermediate key value pairs, since we attempt to group the input vector points to 5 clusters, the application cannot leverage the increased parallelism that is offered by the 24-Node, 32-Node and 48-Node topologies. In fact, it cannot fully take advantage of the parallelism that is offered by the 16-Node topology as well. As a consequence, increasing the number of nodes that participate in the cluster does not reduce the execution time of the application, but increases it, since the idle CPU cycles period because of outstanding I/O is prolonged because of the fact that the increased number of nodes increases the I/O bandwidth congestion, leading in a higher percentage of wasted CPU cycles. Moreover, since cluster topologies with more nodes charge the application with higher power consumption, the devastating impact of increased execution time, increases the energy consumption of the application even more.

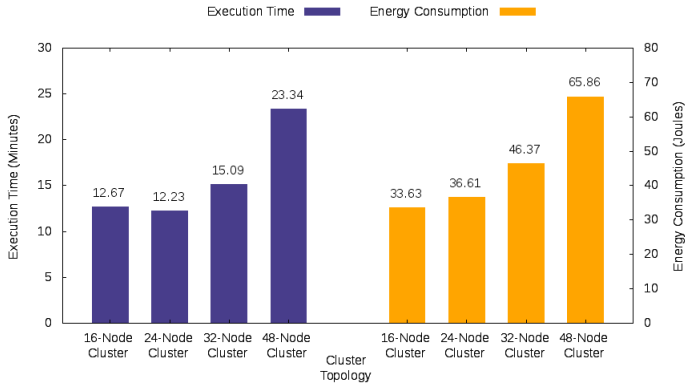


Figure 7.30: K-Means Clustering Cluster Topology Analysis (1/2)

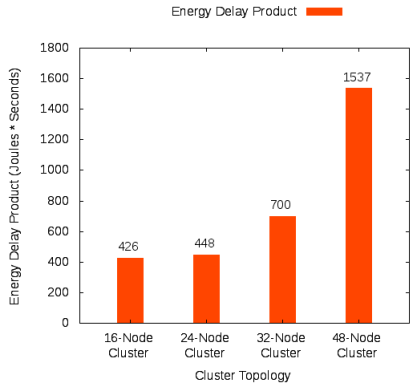


Figure 7.31: K-Means Clustering Cluster Topology Analysis (2/2)

7.3.5 Frequency Scaling Analysis

This section analyzes the impacts of frequency scaling on the execution time and the energy consumption of the `K-Means Clustering` application on the Intel SCC. We have tested the input file which has a size of 121 KB in the 48-Node Cluster topology for nine frequency settings. We have configured the DataNodes and Master Node and the TaskTrackers to run at either 200 MHz, 533 MHz or 800 MHz and each frequency setting represents one combination of those values. The experimental results we have received regarding the execution time and the energy consumption of the `K-Means Clustering` application are presented below. Both the execution time and the energy consumption are divided by the number of iterations the K-Means Clustering algorithm was executed, so as to provide a basis for fair comparison. Detailed plots are included in Appendix B3, as in the previous case.

As in the previous applications, the execution time appears to be driven primarily by the frequency of the TaskTracker nodes, since the CPU-intensive parts of the `K-Means Clustering` algorithm are executed on those nodes. For a given frequency for the DataNodes, increasing the TaskTracker nodes frequency significantly reduces the execution time and the energy consumption of the application. On the contrary, increasing

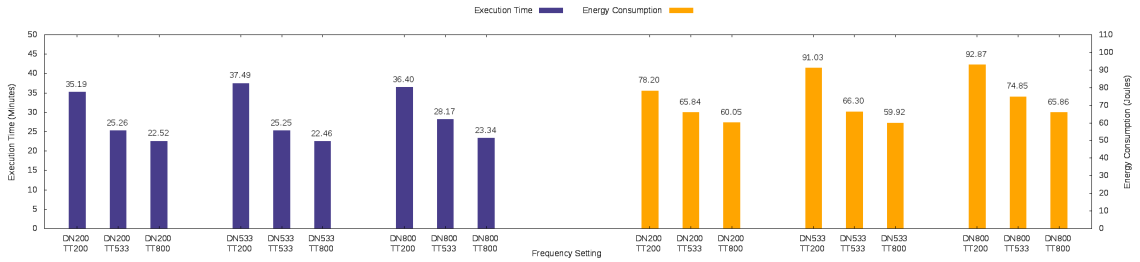


Figure 7.32: K-Means Clustering Frequency Scaling Analysis (1/2)

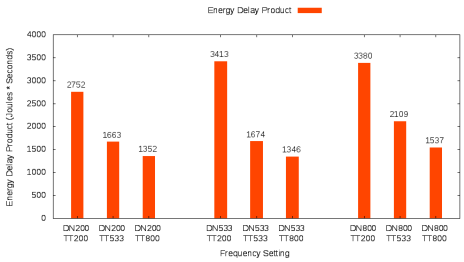


Figure 7.33: K-Means Clustering Frequency Scaling Analysis (2/2)

the frequency of the DataNodes for a fixed frequency for the TaskTrackers, does not reduce the execution time and since the application is charged with higher power consumption, the energy consumption is increased. As a consequence, we can deduce that scaling down the frequency of the DataNodes does not impair the performance of the application, while yielding benefits in terms of energy consumption. For example, while the execution time of the DN200-TT800 and DN800-TT800 settings is almost the same, the DN200-TT800 setting consumes 10% less energy.

7.3.6 Cluster Utilization Overview

This section provides an overview figure (Figures 7.34-7.39) for the CPU utilization of all cluster nodes, when the `K-Means Clustering` application is executed with the input file of 121 KB, on the 48-Node HDFS cluster topology and with the DataNodes and the TaskTrackers configured to operate at 800 MHz. The conclusions that were presented in the previous sections are corroborated by the following plots.

The cluster utilization figures clearly point out that only one Map task was executed for each iteration of the `K-Means Clustering` application, plus one set up Map task during the initialization phase of each MapReduce job. In addition, it is also evident that the percentage of idle CPU cycles due to outstanding I/O for a specific is significantly higher when it takes place during the Reduce phase of each job, because of the fact that more cluster nodes attempt to expand the `mahout-examples-0.6-job.jar` at that time resulting to higher I/O bandwidth saturation.

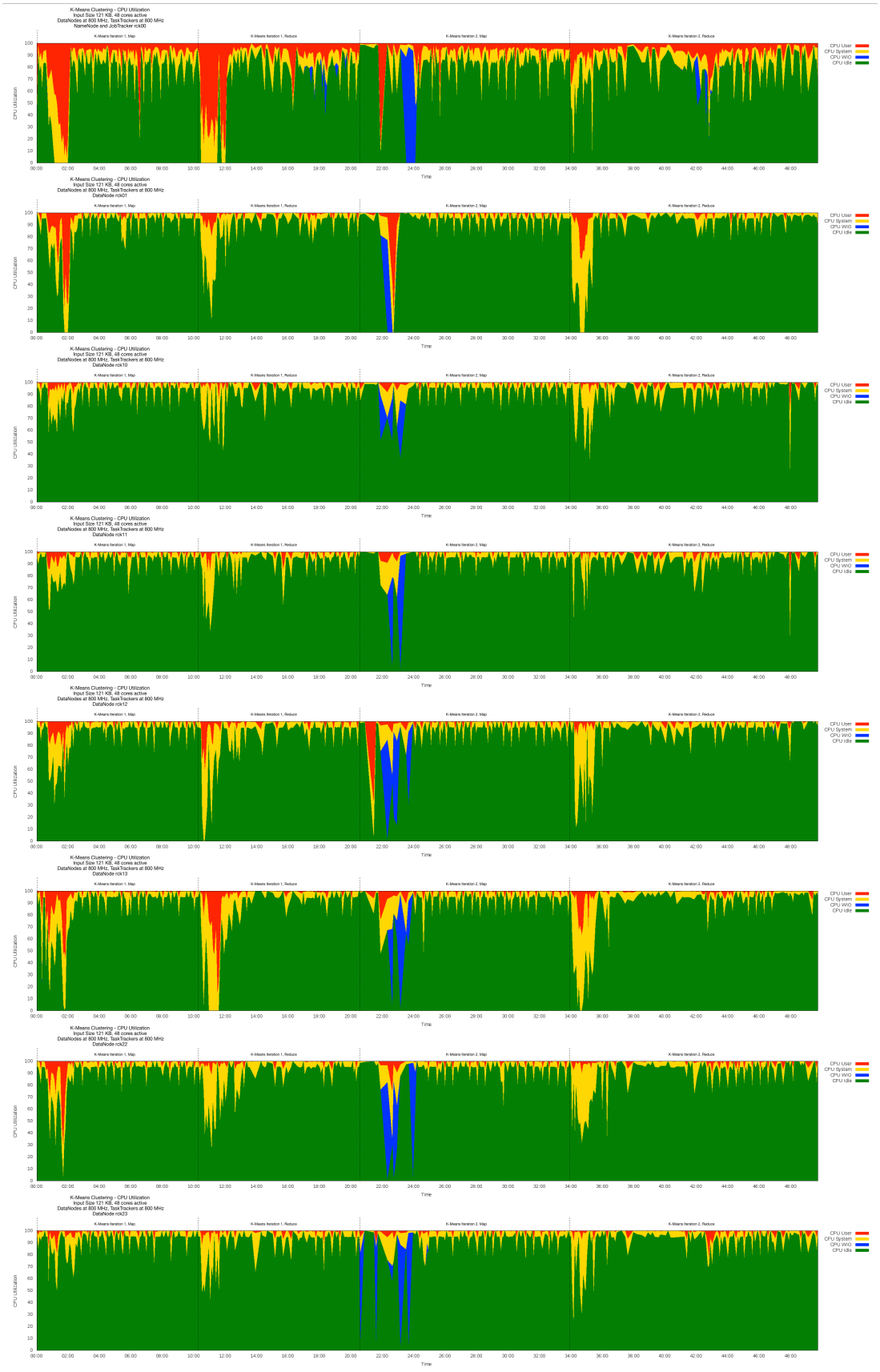


Figure 7.34: K-Means Clustering Overall Cluster Utilization (1/6)

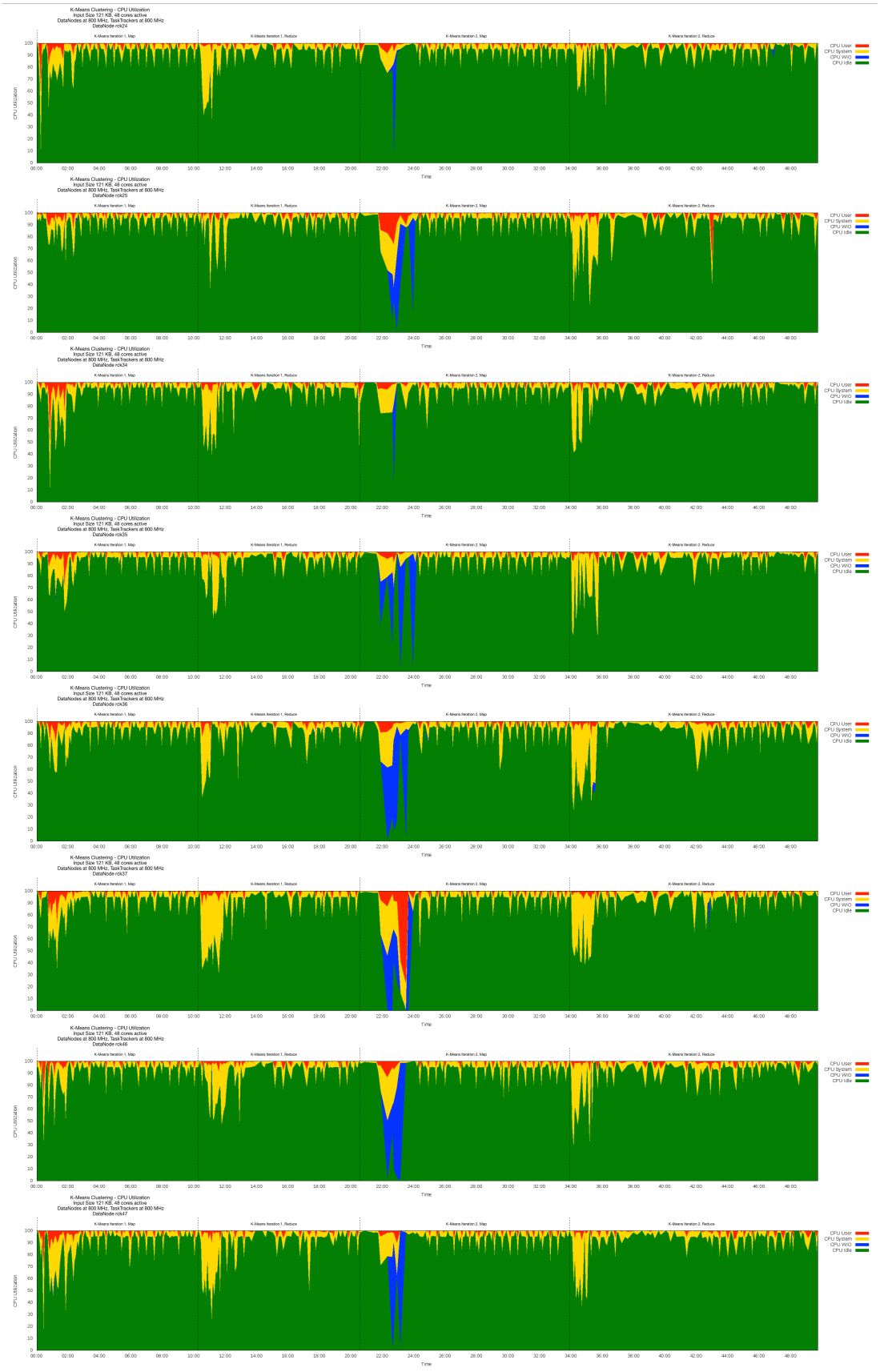


Figure 7.35: K-Means Clustering Overall Cluster Utilization (2/6)

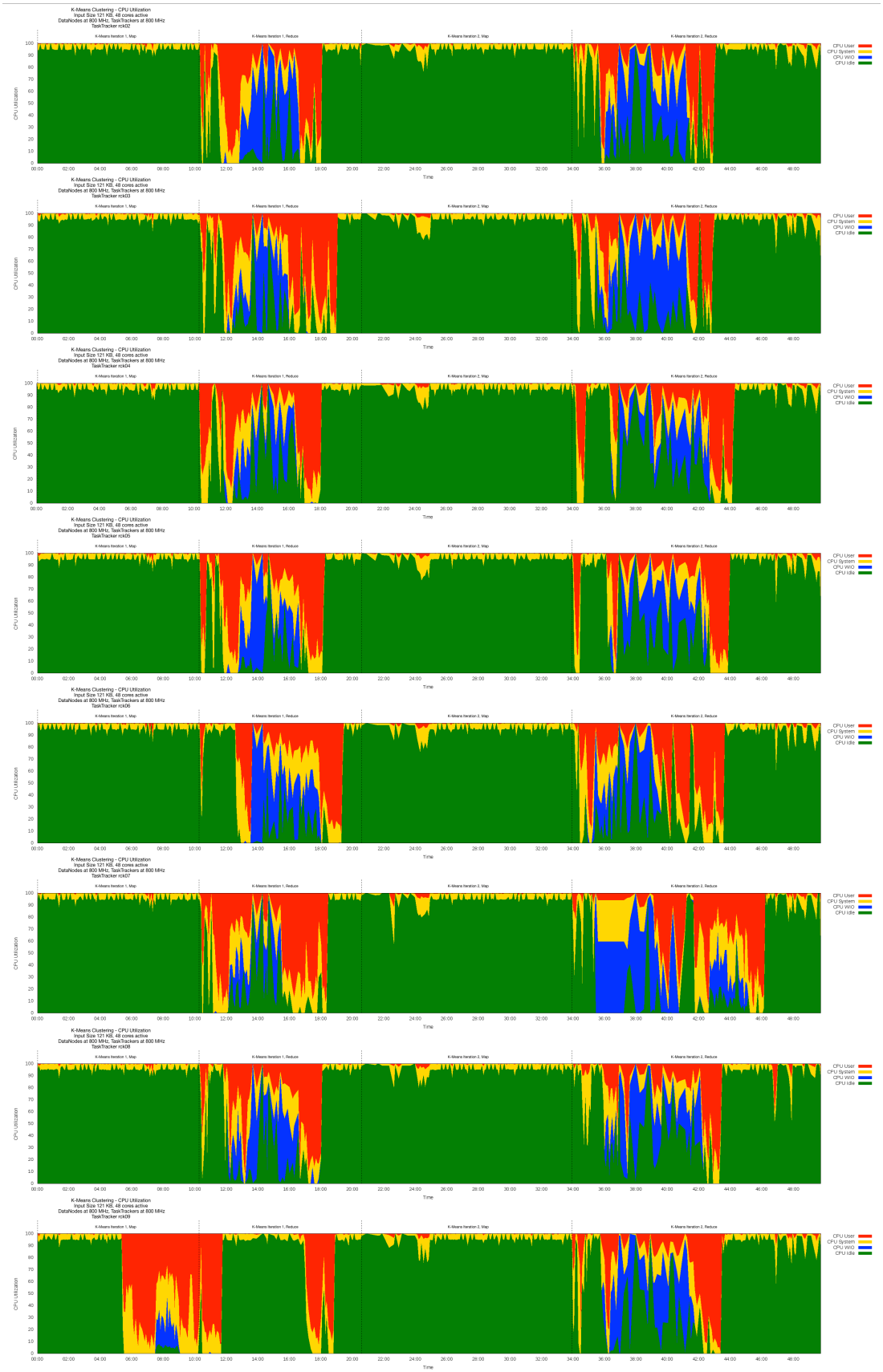


Figure 7.36: K-Means Clustering Overall Cluster Utilization (3/6)

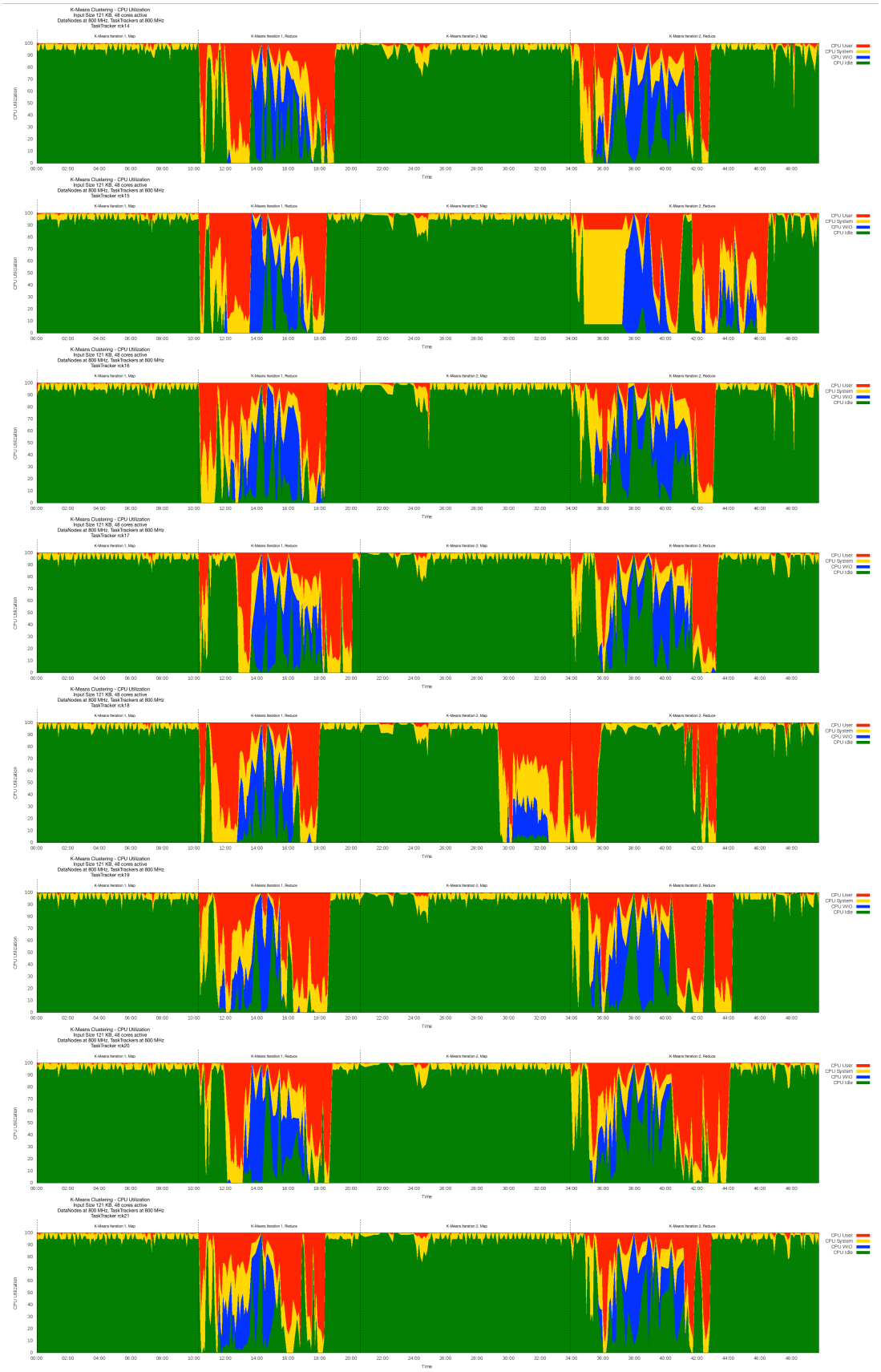


Figure 7.37: K-Means Clustering Overall Cluster Utilization (4/6)

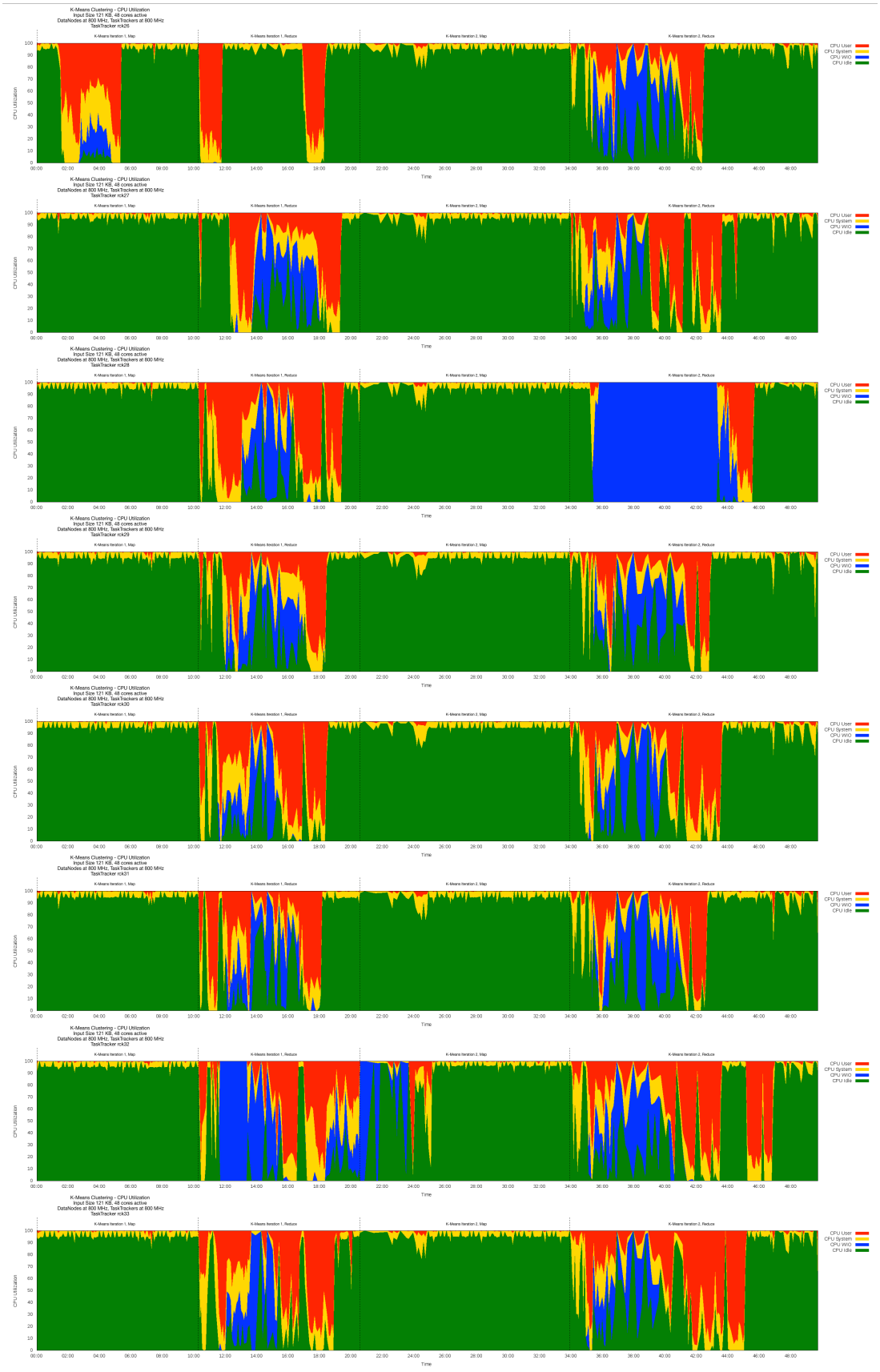


Figure 7.38: K-Means Clustering Overall Cluster Utilization (5/6)

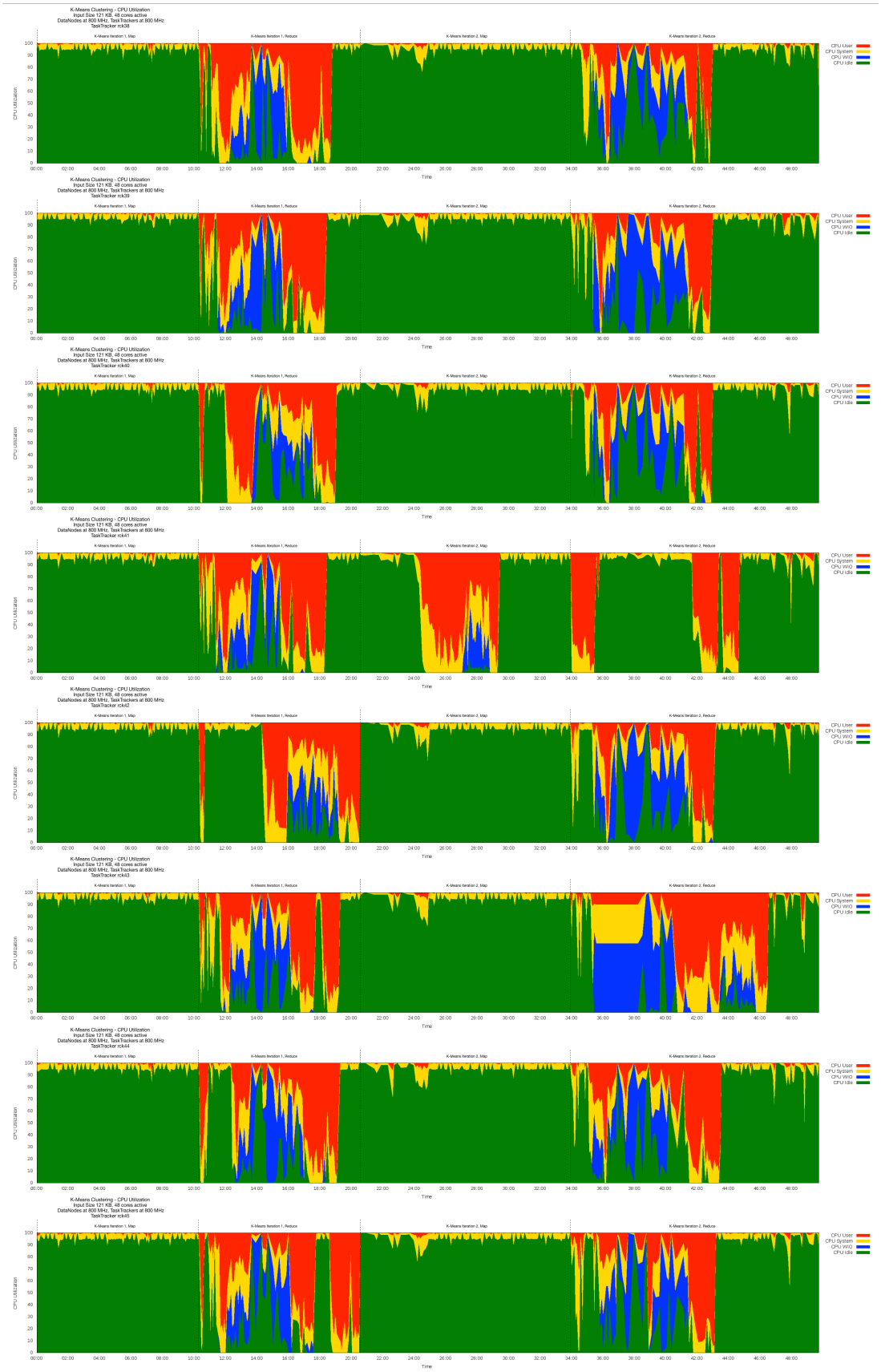


Figure 7.39: K-Means Clustering Overall Cluster Utilization (6/6)

7.4 The Frequent Pattern Growth Application

This section presents our analysis regarding the execution of the `Frequent Pattern Growth` application on the Intel SCC. We initially describe the MapReduce implementation of the `Frequent Pattern Growth` algorithm. In addition, the input file generation and application execution process are stated. We have utilized resources which are provided by DCBench for that purpose. Subsequently, the experimental results we have received are analyzed, in order to draw conclusions regarding the behavior of the `Frequent Pattern Growth` application on the Intel SCC for different input sizes, cluster topologies and frequency settings.

7.4.1 Algorithm Description

The Frequent Pattern Growth Algorithm is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree). Let $I = \{a_1, a_2, ..., a_m\}$ be a set of items, and a transaction database DB is a set of subsets of I, denoted by $DB = \{T_1, T_2, ..., T_n\}$, where each $T_i \subset I (1 \leq i \leq n)$ is said a transaction. The support of a pattern $A \subset I$, denoted by $supp(A)$, is the number of transactions containing A in DB. A is a frequent pattern if and only if $supp(A) \geq \xi$, where ξ is a predefined minimum support threshold. Given DB and ξ, the problem of finding the complete set of frequent patterns is called the frequent itemset mining problem.

Frequent Pattern Growth works in a divide and conquer way. It requires two scans on the database. Frequent Pattern Growth first computes a list of frequent items sorted by frequency in descending order (F-List) during its first database scan. In its second scan, the database is compressed into an FP-tree. Then Frequent Pattern Growth starts to mine the FP-tree for each item whose support is larger than by recursively building its conditional FP-tree. The algorithm performs mining recursively on FP-tree. The problem of finding frequent itemsets is converted to searching and constructing trees recursively.

The Frequent Pattern Growth MapReduce implementation that is provided by Apache Mahout consists of three MapReduce jobs, which are called `ParallelCounting`, `Parallel FPGrowth` and `Aggregator`. Those jobs are orchestrated by the `FrequentPatternGrowth Driver` master flow. Pseudocode for the `FrequentPatternGrowthDriver` flow is presented below.

```
FrequentPatternGrowthDriver.runJob(input,output,params):
    ParallelCounting.runJob(input,output,params)
    createFList()
    createGList()
    ParallelFPGrowth.runJob(input,output,params)
    Aggregator.runJob(input,output,params)
```

The `ParallelCounting` job counts the number of occurences of each item in the transaction database, similar to the `Wordcount` application. `ParallelCounting` outputs

<key,value> pairs, whose key is an item and whose value is its number of occurences in the transaction database.

$$< item, N >$$

The above purpose is achieved by the following Map and Reduce functions.

```
ParallelCountingMapper.map(String key, Transaction value):
    List<Item> itemList = tokenizeTransaction(value)
    for (item : itemList):
        output(item,1)

ParallelCountingReducer.reduce(String key, List<Integer> values):
    sum = 0
    for (value : values):
        sum += value
    output(key,sum)
```

The output of this job is used by FrequentPatternGrowthDriver so as to create FList, which is a sorted list of the number of occurences (i.e. the support value) of each item, in descending order. Items with a number of occurences less than the minimum support value are eliminated from FList. Subsequently, the items present in fList are assigned to groups, that is each item is assigned to a specific groupId. The total number of groups is provided as an input parameter of FrequentPatternGrowthDriver. gList contains the mapping of items to groupIds. fList and gList can be considered as global invariants for the ParallelFPGrowth job, meaning that they can accessed by all Map and Reduce tasks.

The ParallelFPGrowth job performs a second scan of the transaction database, in order to convert the transactions of the database into group dependent transactions and build independent FP-trees in parallel. ParallelFPGrowth outputs <key,value> pairs whose key is an item $a_i \in I$ and whose value is a frequent pattern A (with $supp(A) \geq \xi$) that contains that item.

$$< item, pattern >$$

The above purpose is achieved by the following Map and Reduce functions. growth() mines the FP-tree that has been created by the group-dependent transaction database, so as to discover frequent patterns. The frequent patterns that are generated by ParallelFPGrowthReducer are stored in a MaxHeap data structure. k denotes the maximum number of frequent patterns that are included in each MaxHeap and is provided as an input parameter for the FrequentPatternGrowth algorithm.

```
ParallelFPGrowthMapper.map(String key, Transaction value):
    gMap = new HashMap(gList)
    List<Item> itemList = tokenizeTransaction(value)
    itemList.sortBySupport(fList)
    for (i=itemList.size(); i > 0 ; i--):
        item = itemList(i)
```

```
        if (gMap.containsKey(item)):
            groupId = gMap(item)
            gMap.deleteAllValues(groupId)
            transaction = new Transaction(sublist(itemList,1,i))
            output(groupId,transaction)

ParallelFPGrowthReducer.reduce(String key, List<Transaction> values):
    localFList = new FList()
    itemSet = new Set<String>
    for (value : values):
        itemSet = tokenizeTransaction(value)
        for (item : itemSet):
            localFList.updateFList(item)
            itemSet.add(item)
     values.deleteInfrequentItems(fList)
     localFPTree = new FPTree()
     for (value : values):
         localFPTree.addPattern(value)
     k = getPatternMaxHeapSize()
     for (item : itemSet):
         topKPatternHeap = growth(localFPTree,item,minSupport,k)
         output(item,topKPatternHeap)
```

The `Aggregator` job processes the output that has been provided by `ParallelFPGrowth` and merges them in order to output `<key,value>` pairs which contain all the frequent patterns that contain a specific item. The frequent patterns that are generated by `AggregatorReducer` are stored in a `MaxHeap` data structure. k denotes the maximum number of frequent patterns that are included in each `MaxHeap` and is provided as an input parameter for the `FrequentPatternGrowth` algorithm. The above purpose is achieved by the following `Map` and `Reduce` functions.

```
Aggregator.map(String key, MaxHeap<Item> value):
    output(key,value)

Aggregator.reduce(String key, List<MaxHeap<Item>> values):
    topKPatternMaxHeap = new MaxHeap<Item>()
    k = getPatternMaxHeapSize()
    for (value : values):
        topKPatternMaxHeap.merge(value,k)
    output(key,topKPatternMaxHeap)
```

7.4.2 Application Execution and Input Files

We use three different input files for the `Frequent Pattern Growth` application, whose size is 4 MB, 34 MB and 377 MB. Those input files are provided by DC Bench. Each line of these files contains a transaction present in the transaction database as follows.

```
ageo@mitsos:~/HVCBench-hadoop/basedata$ cat fpg-accidents.dat
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
```

```
30 31 32
33 34 35
36 37 38 39 40 41 42 43 44 45 46
38 39 47 48
38 39 48 49 50 51 52 53 54 55 56 57 58
32 41 59 60 61 62
3 39 48
63 64 65 66 67 68
32 69
. . . . . .
```

DC Bench also provides a script called `prepare-fpg.sh`, which receives the desired input size as an input and uploads the corresponding file to HDFS. The code of this script is included in Appendix A.

```
ageo@mitsos:~/HVCBench-hadoop/workloads/associationrulemining/
    kmeans$ ./prepare-fpg.sh low

ageo@mitsos:~/hadoop-0.20.2$
    bin/hadoop dfs -lsr /cloudrank-data/fpg*
-rw-r--r--   1 ageo supergroup    4167490 2015-03-30 10:46
    /cloudrank-data/fpg-accidents.dat
```

In order to run the `Frequent Pattern Growth` benchmark, the `run-fpg.sh` script, which is provided by DCBench has to be executed. This script receives the size of the input file that contains the transaction database as a command line argument and searches in HDFS for the input file with this specific size that was uploaded by `prepare-fpg.sh`. The code of this script is included in Appendix A.

```
ageo@mitsos:~/HVCBench-hadoop/workloads/associationrulemining/fpg$
    ./run-fpg.sh high
15/04/01 01:14:37 INFO mapred.JobClient:
    Running job: job_201503312203_0001
15/04/01 01:14:38 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
15/04/01 01:42:37 INFO mapred.JobClient:  map 100% reduce 100%
15/04/01 01:43:21 INFO mapred.JobClient: Job complete:
    job_201503312203_0001
. . . . . .
15/04/01 01:47:14 INFO mapred.JobClient: Running job:
    job_201503312203_0002
15/04/01 01:47:15 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
15/04/01 02:05:52 INFO mapred.JobClient:  map 100% reduce 100%
15/04/01 02:07:02 INFO mapred.JobClient: Job complete:
    job_201503312203_0002
. . . . . .
15/04/01 02:09:28 INFO mapred.JobClient:Running job:
    job_201503312203_0003
```

```
15/04/01 02:09:29 INFO mapred.JobClient:  map 0% reduce 0%
. . . . . .
15/04/01 02:25:23 INFO mapred.JobClient:  map 100% reduce 100%
15/04/01 02:26:02 INFO mapred.JobClient: Job complete:
    job_201503312203_0003
. . . . . .
15/04/01 02:26:02 INFO driver.MahoutDriver:
    Program took 4353692 ms (Minutes: 72.56153333333333)
```

7.4.3 Scalability Analysis Per Input Size

This section presents the analysis we have conducted regarding the scalability of the `Frequent Pattern Growth` application, in terms of input size, when it runs on the Intel SCC. We have executed the application with three different input files, whose size is 4 MB, 34 MB and 377 MB as mentioned above. We have utilized the 48-Node Cluster Topology for this analysis and have configured both the DataNodes and the TaskTrackers to operate at the maximum frequency of 800 MHz. The experimental results we have received regarding the execution time and the energy consumption of the `Frequent Pattern Growth` application are presented below. Detailed plots that illustrate the CPU utilization and the network traffic for one DataNode and one TaskTracker as well as the overall power consumption and board temperature of the Intel SCC, for each run, are included in Appendix B4.

Figure 7.40: Frequent Pattern Growth Input Size Scalability Analysis (1/2)

Our experimental results point out that the execution time of the `Frequent Pattern Growth` application scales up only when the 377 MB input file is executed, while the execution time of the 4 MB and 34 MB input files is almost the same. The reason for that is that the 4 MB file is stored in one `InputSplit` in HDFS and as a consequence 1 Map task is issued for the `ParallelCounting` and `ParallelFPGrowth` jobs, while the 34 MB input file is stored in 12 `InputSplits` and 12 Map tasks are issued for the first two jobs. As a result, since the 4 MB and 34 MB input files do not fully leverage the parallelism that is provided by the underlying HDFS cluster, the execution time of the application is almost the same for these two cases. The 377 MB file is stored

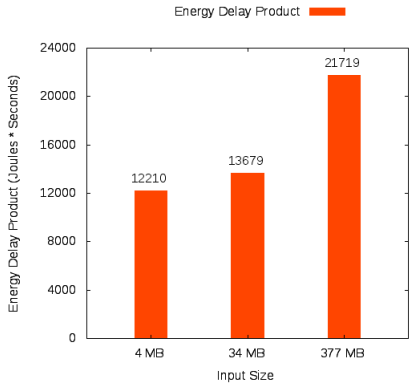


Figure 7.41: Frequent Pattern Growth Input Size Scalability Analysis (2/2)

in 95 `InputSplits`, causing at least 95 Map tasks to be issued, which are more than the number of the `TaskTracker` nodes, thus leading to an increase in execution time compared to the first two cases.

In addition, it has to be mentioned that the energy consumption of the application execution with the 34 MB input file is higher compared to the one of the execution with the 4 MB input file. The reason for that is that the higher number of issued Map tasks result in higher overall CPU utilization of the platform, thus increasing the power consumption that the application is charged with, resulting in a higher overall energy consumption for this case.

In the 4 MB input file case, only 1 Map task is issued in `ParallelCounting` and `ParallelFPGrowth` jobs, which is not evident in the diagram we have provided in the Appendix. Figure 7.42 depicts the execution of the setup Map tasks for `ParallelCounting`, `ParallelFPGrowth` and `Aggregaton` on the Intel SCC cores `rck17`, `rck27` and `rck45` respectively and the execution of the only Map task for `ParallelCounting` and `Parallel FPGrowth` on Intel SCC cores `rck08` and `rck07` respectively.

It also has to be mentioned, that the period of high percentage of CPU idle cycles because of outstanding I/O and increased outgoing network traffic is also present in the `Frequent Pattern Growth` application, since the `mahout-examples-0.6-job.jar` is distributed and expanded by all TaskTracker nodes in this case as well. In this case, this period is observed at the beginning of the Map phase, for nodes where Map tasks were executed and in the beginning of the Reduce phase, for nodes that did not execute Map tasks. In addition, it is evident in Figure 7.42 that this behavior is much more intense during the Reduce phase of `ParallelCounting` and `ParallelFPGrowth` and during the Map phase of `Aggregation`, because more nodes are attempting to expand their jar file, leading in higher I/O bandwidth contention and a higher percentage of wasted CPU cycles.

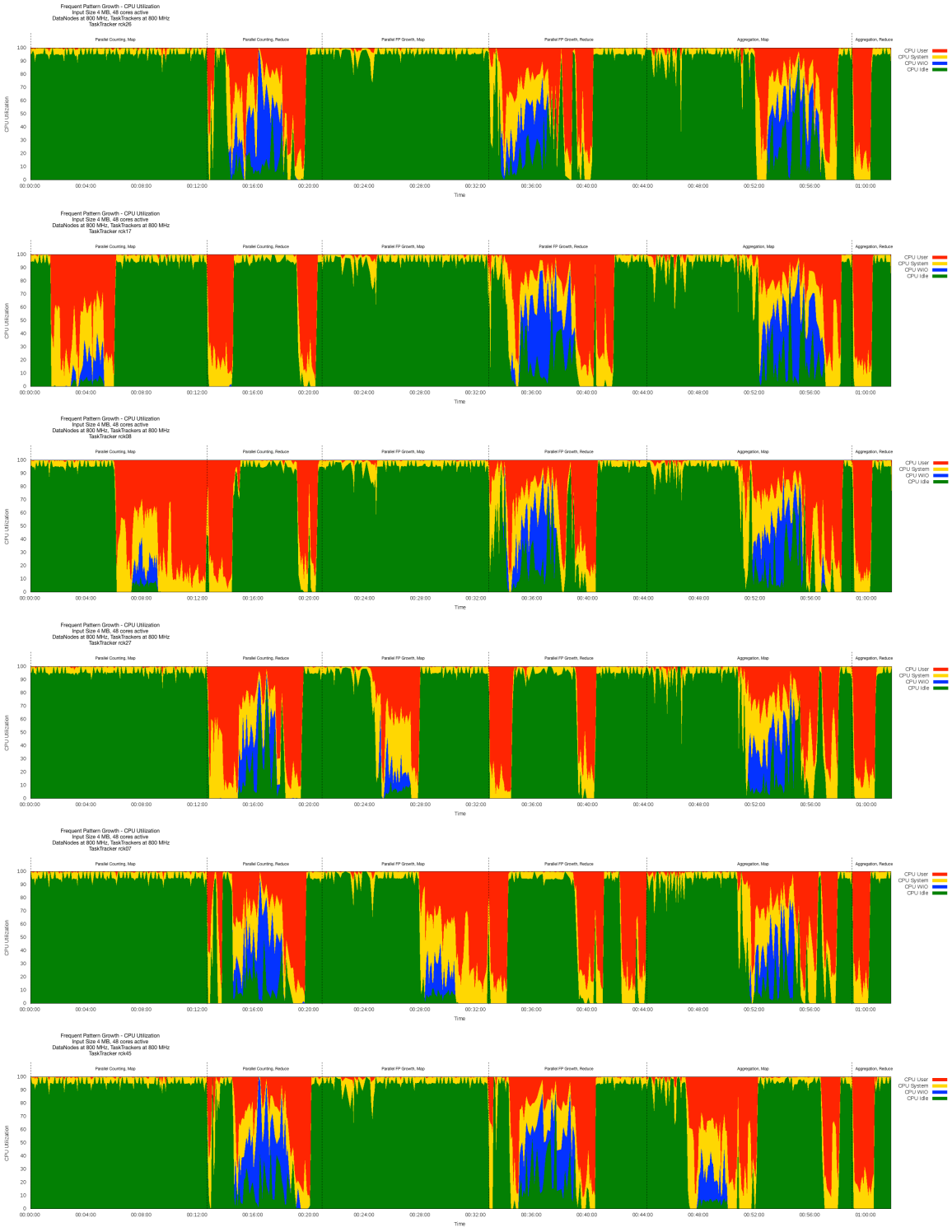


Figure 7.42: CPU Utilization Plots for the Frequent Pattern Growth Application

7.4.4 Cluster Topology Analysis

This section presents our analysis concerning the behavior of the `Frequent Pattern Growth` application when it is executed on top of different HDFS cluster topologies on the Intel SCC. For this study, we have used the 4 MB input file which is provided by DCBench. We have configured both the DataNodes and the TaskTrackers of each cluster topology to operate at the maximum frequency of 800 MHz. The idle nodes of each topology (if any) operate at the minimum frequency of 100 MHz. `gmond` is not active on those nodes as well. The experimental results we have received regarding the execution time and the energy consumption of the `Frequent Pattern Growth` application are presented below. Detailed plots are included in Appendix B4, as in the previous case.

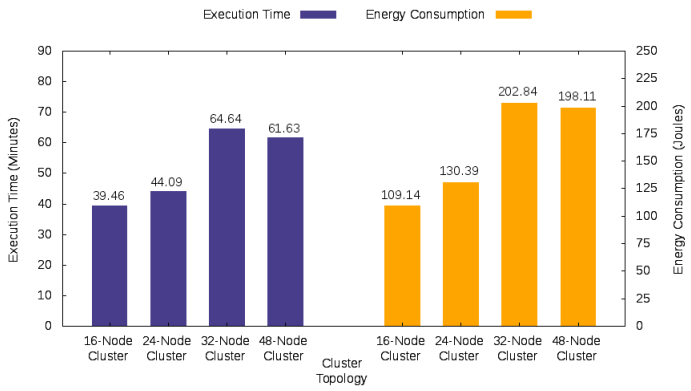


Figure 7.43: Frequent Pattern Growth Cluster Topology Analysis (1/2)

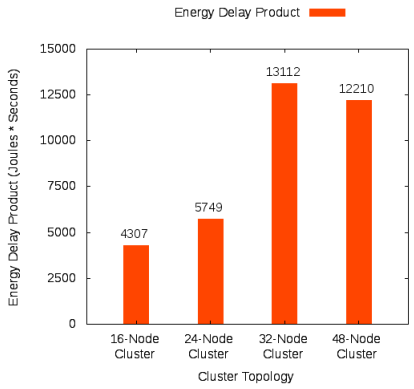


Figure 7.44: Frequent Pattern Growth Cluster Topology Analysis (2/2)

Our experimental results illustrate that similar to the `K-Means Clustering` application, since the application for the given input file does not fully utilize the parallelism that is offered by the underlying HDFS cluster completes faster and with a lower energy consumption when it is executed on top of the 16-Node cluster topology.

One significant difference between these two applications, regards the Reduce phase of all three MapReduce jobs of the `Frequent Pattern Growth` application. Since the `numGroups` parameter of the application is set to `1000`, all Reduce tasks of the three MapReduce jobs receive intermediate `<key,value>` pairs. As a consequence, the load of the Reduce tasks increases for cluster topologies with less `TaskTracker` nodes. However, this increased load is mitigated by the absence of a high percentage of idle CPU cycles because of outstanding I/O during the period when the `mahout-examples-0.6-job.jar` is distributed and expanded in all `TaskTracker` nodes of the cluster, resulting in lower execution time and energy consumption for cluster topologies that employ less Intel SCC cores.

7.4.5 Frequency Scaling Analysis

This section analyzes the impacts of frequency scaling on the execution time and the energy consumption of the `Frequent Pattern Growth` application on the Intel SCC. We have tested the input file which has a size of 4 MB in the 48-Node Cluster topology for nine frequency settings. We have configured the DataNodes and Master Node and the TaskTrackers to run at either 200 MHz, 533 MHz or 800 MHz and each frequency setting represents one combination of those values. The experimental results we have received regarding the execution time and the energy consumption of the `Frequent Pattern Growth` application are presented below. Detailed plots are included in Appendix B4, as in the previous case.

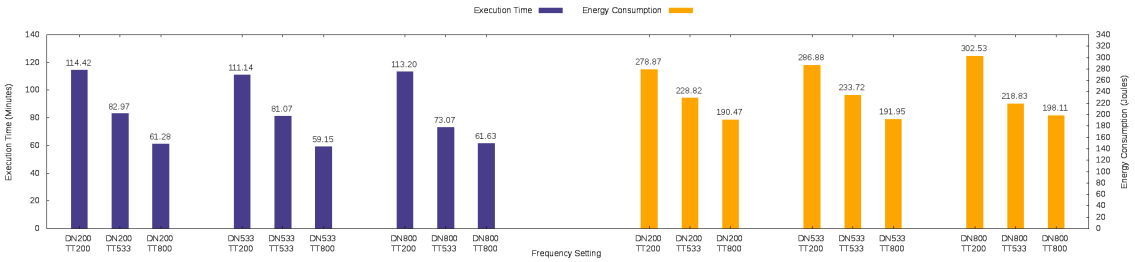


Figure 7.45: Frequent Pattern Growth Frequency Scaling Analysis (1/2)

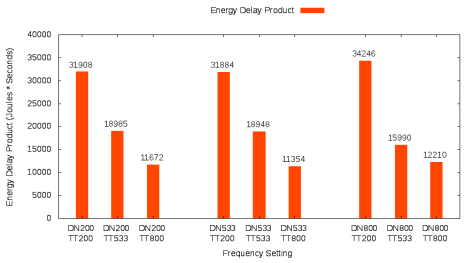


Figure 7.46: Frequent Pattern Growth Frequency Scaling Analysis (2/2)

Similar to the previous applications, the execution time and the energy consumption

of the `Frequent Pattern Growth` application appears to be driven primarily by the frequency of the `TaskTracker` nodes. The frequency of the `DataNodes` seems to have a negligible impact in some cases, since increasing the `DataNodes` frequency seems to slightly reduce the execution time and the energy consumption of the application for a given frequency for the `TaskTracker` nodes.

As a consequence, scaling down the frequency of the DataNodes does not impair the performance of the application by increasing its execution time and additionally yields energy consumption saving benefits. Like in all other application the DN200-TT800 setting is optimal if both performance and energy consumption are taken into account. Energy consumption savings would have been more significant in this case as well if the Intel SCC architecture allowed us to perform voltage scaling at a finer granularity, enabling us to scale down the voltage of the DataNodes that operate at 200 MHz.

7.4.6 Cluster Utilization Overview

This section provides an overview figure (Figures 7.47-7.52) for the CPU utilization of all cluster nodes, when the `Frequent Pattern Growth` application is executed with the input file of 34 MB, on the 48-Node HDFS cluster topology and with the DataNodes and the TaskTrackers configured to operate at 800 MHz. The conclusions that were presented in the previous sections are corroborated by the following plots.

The cluster utilization figures clearly point out that 12 Map tasks was executed for each `ParallelCounting` and `ParallelFPGrowth` MapReduce jobs, plus one set up Map task during their initialization phase. In addition, it is also evident for the first two MapReduce jobs that the percentage of idle CPU cycles due to outstanding I/O for a specific is higher when it takes place during the Reduce phase of each job, because of the fact that more cluster nodes attempt to expand the `mahout-examples-0.6-job.jar` at that time resulting to higher I/O bandwidth saturation. On the contrary, during the `Aggregation` phase of the `Frequent Pattern Growth` application, this behavior is observed only during the Map phase, since more than 32 Map tasks are issued for this MapReduce job, since it processes the intermediate results that were generated by `ParallelFPGrowth`.

Figure 7.47: Frequent Pattern Growth Overall Cluster Utilization (1/6)

Figure 7.48: Frequent Pattern Growth Overall Cluster Utilization (2/6)

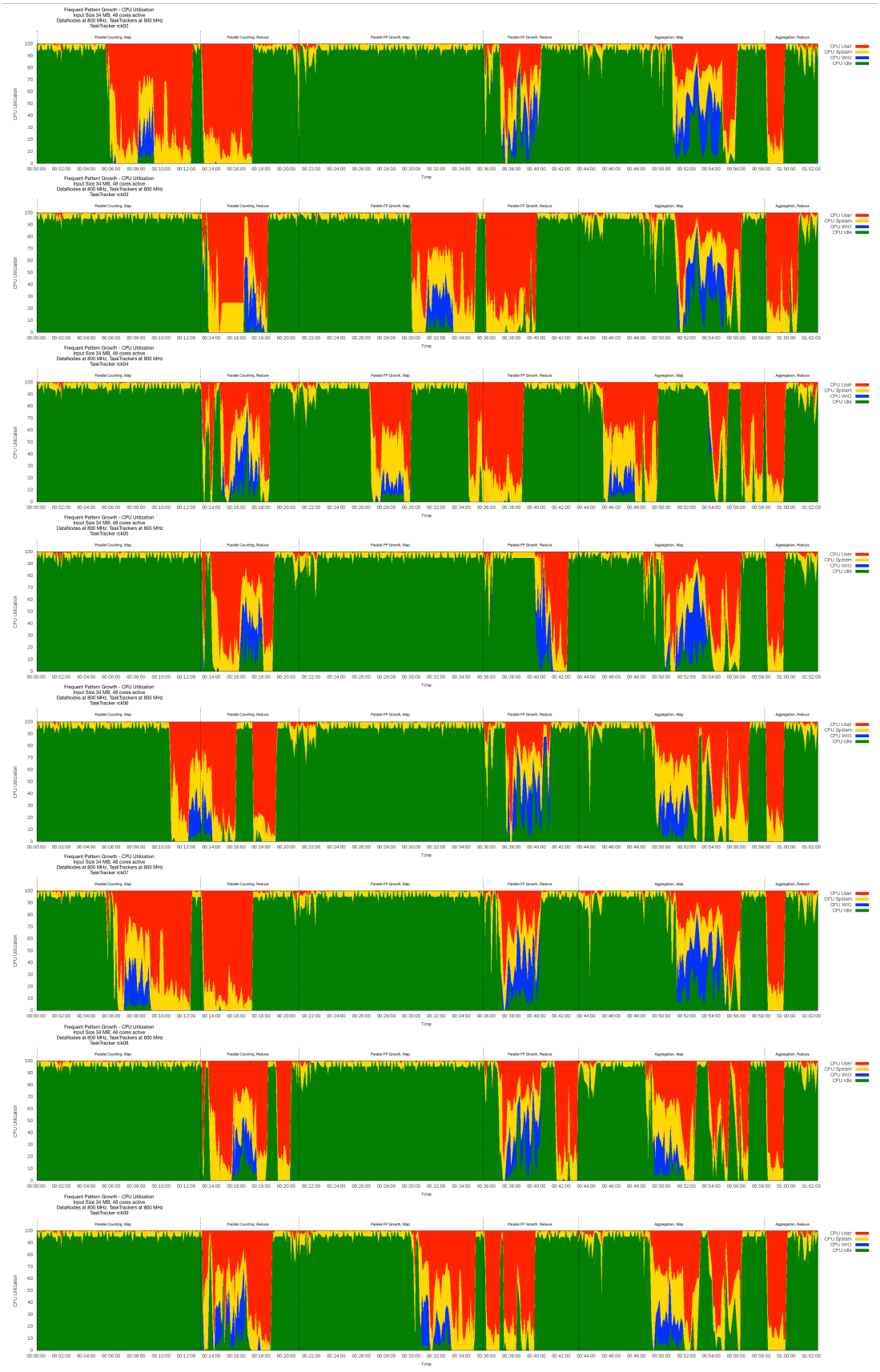


Figure 7.49: Frequent Pattern Growth Overall Cluster Utilization (3/6)

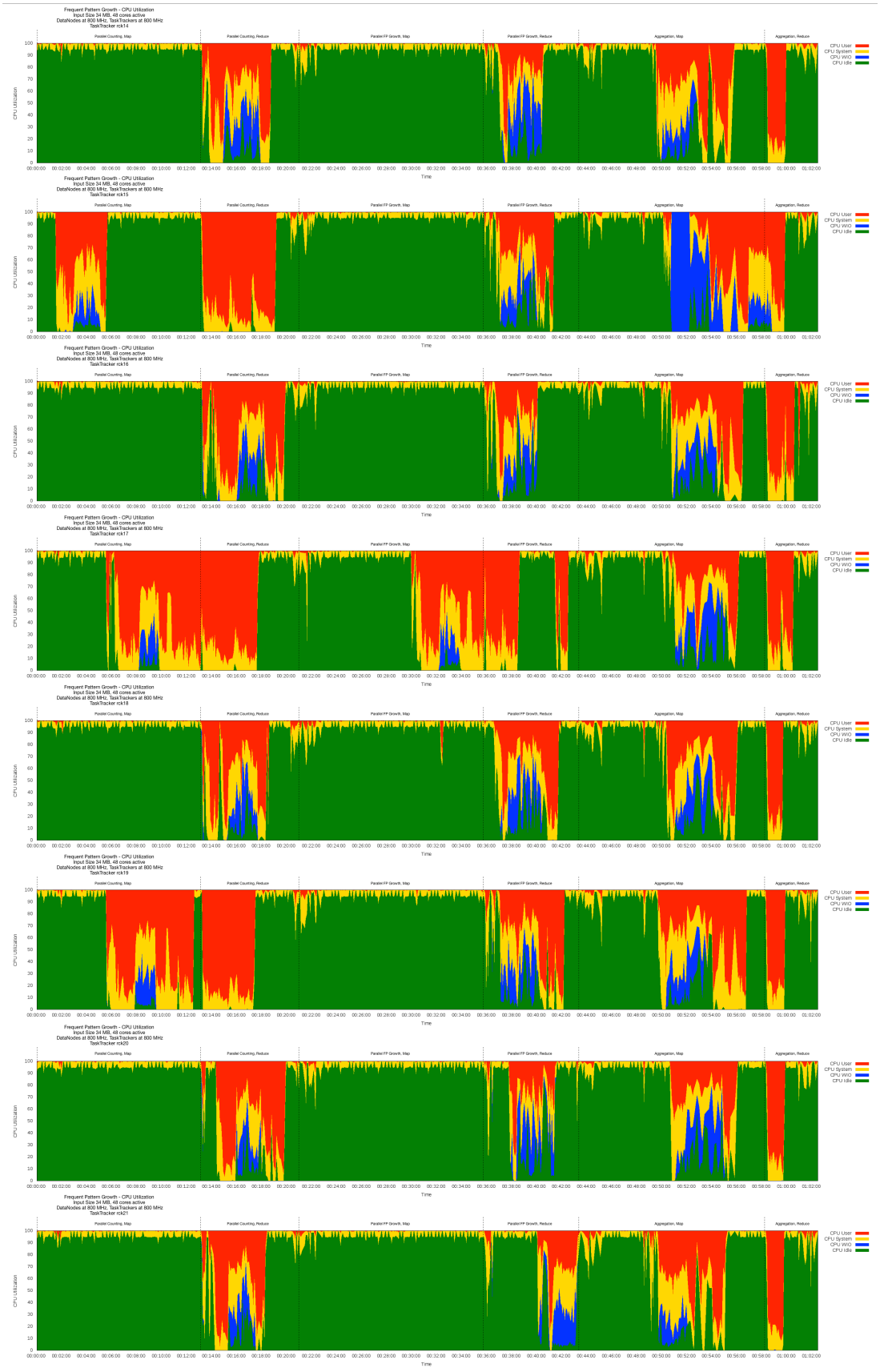


Figure 7.50: Frequent Pattern Growth Overall Cluster Utilization (4/6)

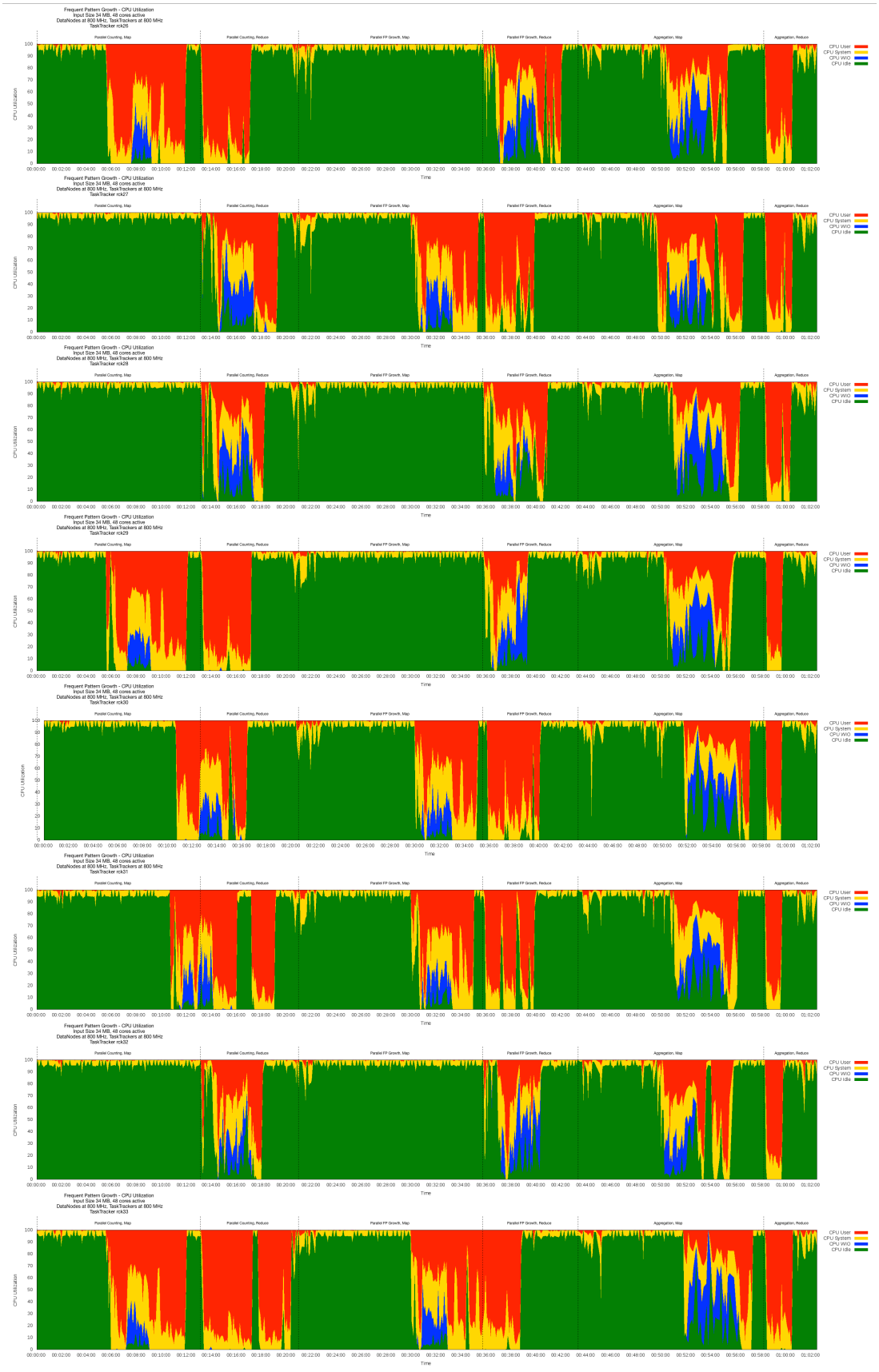


Figure 7.51: Frequent Pattern Growth Overall Cluster Utilization (5/6)

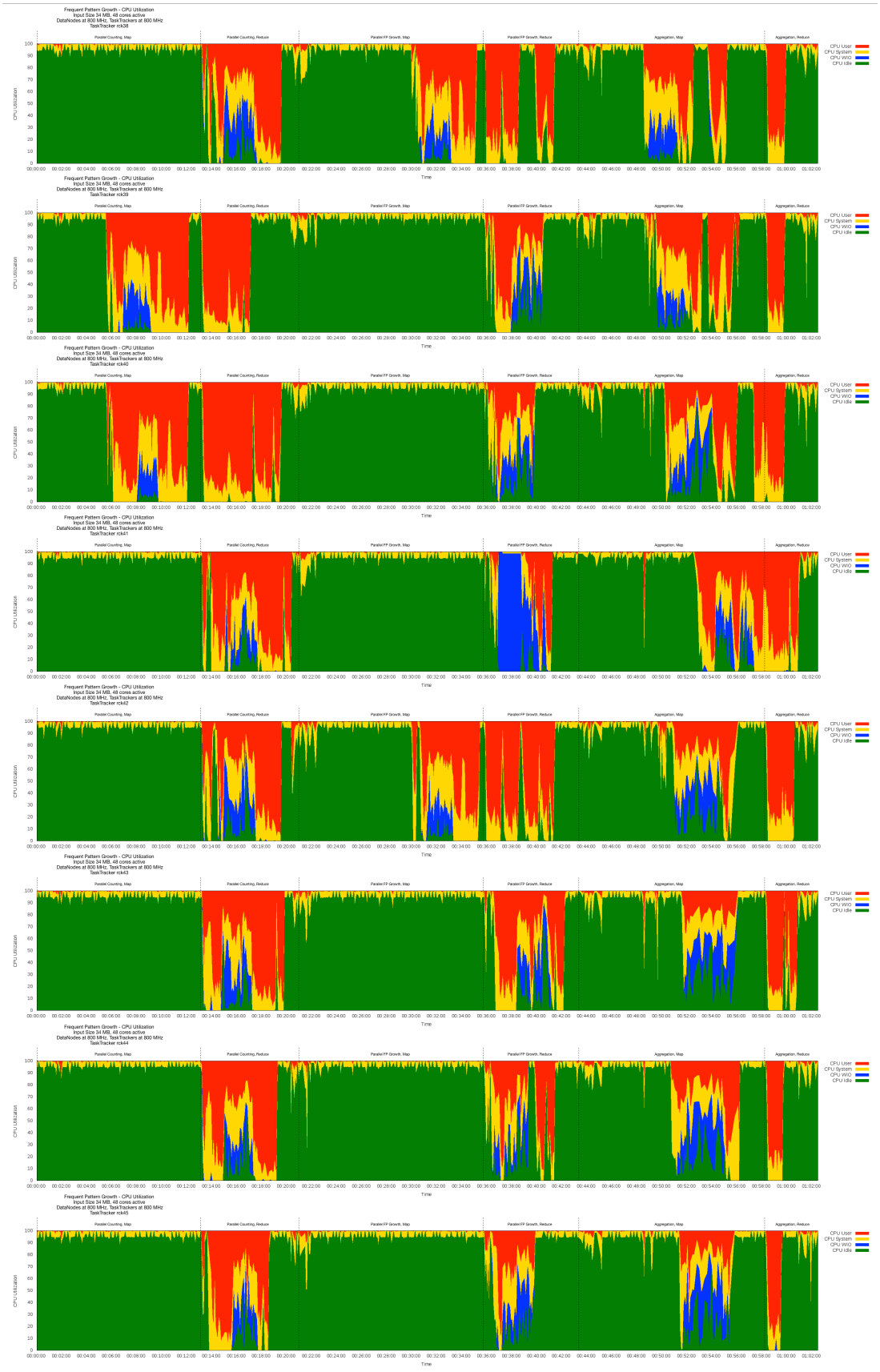


Figure 7.52: Frequent Pattern Growth Overall Cluster Utilization (6/6)

Chapter 8

Thesis Conclusion

This chapter concludes the findings that are provided by this diploma thesis and presents suggestions for future work.

8.1 General Remarks

This thesis studied the runtime behavior of Big Data applications that have been implemented with the MapReduce framework on top of HDFS clusters that were deployed on the Intel SCC cores. The experimental results that we collected by executing those applications helped us draw meaningful conclusions regarding the execution of these scale-out workloads on Intel SCC hardware. Our analysis focused on the scalability of those applications in terms of input size, their ability to leverage the parallelism offered by different HDFS cluster topology organizations and the impact of frequency perturbations on the application performance and the energy consumption.

Our analysis illustrated that the execution time and the energy consumption of the Big Data workloads we studied scaled linearly with the input size increase. An exception to that rule is the case when the size of application input files is too small to fully leverage the parallelism offered by the platform, such as the input files of the `K-Means Clustering` application for instance. In addition, our investigation regarding different HDFS cluster topology organizations poses the trade-off between the number of available TaskTracker nodes and the per-node available I/O bandwidth. This trade-off is the key of understanding the optimal cluster topology for each application and input file size. In addition, our results suggest that increasing the number of DataNodes of a cluster topology while keeping the number of TaskTrackers constant does not improve application performance and aggravates energy consumption because the cluster is charged with higher power consumption. Finally, our experimental results clearly point out that scaling down the frequency of the Intel SCC cores which host the DataNode HDFS daemons to 200 MHz does not impair application performance and can yield performance saving benefits, due the lower platform power consumption.

Finally, it has to be stated that the Intel SCC hardware and platform architecture

introduces grave limitations regarding the deployment of HDFS clusters and the execution of MapReduce jobs. Firstly, the very low amount of private memory per Intel SCC core ($\sim$ 640 MB) allowed us to configure a maximum of 128 MB for the Java Heap Space of the Child JVMs which execute the Map and Reduce tasks. As a consequence, a significant percentage of CPU cycles was wasted for garbage collection, so that the application can respect that constraint. In addition, the available I/O bandwidth offered by the Memory Controllers cannot satisfy concurrent I/O requests from all Intel SCC cores, causing them to stall thus deteriorating application performance. Specifically, we have found out that increasing the number of requesting cores from 16 to 32 (24-Node and 32-Node cluster topologies vs 48-Node cluster topology) increases the I/O bandwidth saturation significantly resulting in a higher percentage of idle cycles due to outstanding I/O. Moreover, the frequent failure of cores during the presence of high I/O and low free memory made it necessary for us to implement a node failover mechanism.

8.2 Future Work

This section presents propositions for future work that can be inspired by the experimental results and the conclusions that have been provided by this thesis. Future investigation and research could be focused on two diverse areas, tackling the Intel SCC platform limitations and inefficiencies so as to meet the hardware requirements of scale-out workloads and the development of a power-aware MapReduce framework, based on our findings.

In order to achieve more efficient and performant execution of MapReduce applications on the Intel SCC, the issue of low per-core private memory has to be addressed. We expect that Intel SCC boards configured with 64 GB of main memory, instead of the 32 GB configuration in our case would yield significant improvements in terms of application performance, since this would enable us to configure the MapReduce framework with a higher amount of Java Heap Space for the Child JVMs. The whole extra 640 MB of private memory for each core would be available for the application user space, enabling as to configure a maximum heap size of 512 MB or even 768 MB, which could result in tremendous performance improvements. If the expansion of the Intel SCC memory is not possible an alternative would be to change the system memory map, assigning the whole memory to 24 cores for instance. This approach would double the per-core available private memory but would also yield half of the Intel SCC cores unusable. Another research proposition could be the development of a hypervisor which would enable the Intel SCC Linux to be booted on top of 2 or 4 Intel SCC cores. The purpose of this approach would be to reduce the percentage of the platform memory being used by the OS, in order to make more memory available for the application user space while being able to leverage all the cores of the Intel SCC board.

Finally, our conclusions regarding the frequency scaling analysis of MapReduce applications that run on top of HDFS clusters clearly point out that energy consumption can

be reduced significantly if the operating frequency of cluster nodes with low CPU utilization is scaled down. This conclusion could lead to the development of a power-aware version of MapReduce, which would dynamically scale down the operating frequency (and voltage if possible) of slave nodes that do not execute any Map or Reduce task at the time, based on the information held by the JobTracker regarding the application execution. Our proposition is that when a Map or Reduce task is issued for a slave node by the JobTracker, then this node should transition to a high-power state, with a higher operating frequency and when all Map and Reduce tasks that are executed on this node are completed, then the node should transition to a power-saving state with a lower or minimum operating frequency (and voltage if possible).

Bibliography

[1] Dhruba Borthakur. *The Hadoop Distributed File System : Architecture and Design*. The Apache Software Foundation, 2007

[2] Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung. *The Google File System*. Google, 2003

[3] *MapReduce Tutorial*. The Apache Software Foundation, 2003

[4] Jeffrey Dean and Sanjay Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. Google, 2004

[5] Jacob Leverich, Christos Kozyrakis. *On the Energy (In)efficiency of Hadoop Clusters*. Computer Systems Laboratory, Stanford University

[6] *The SCC Platform Overview*. Intel Labs, 2012

[7] *The SCC Programmer's Guide*. Intel Labs, 2012

[8] *SCC External Architecture Specification (EAS)*. Intel Labs, 2010

[9] Matt Massie, Bernard Li, Brad Nichols and Vladimir Vuksan. *Monitoring with Ganglia*. O'Reilly, 2013

[10] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki and Babak Falsafi. *Clearing the Clouds. A Study of Emerging Scale-out Workloads on Modern Hardware*. CALCM, Carnegie Melon University. Eco Cloud, Ecole Polytechnique Federale de Lausanne, 2012

[11] Pejman lotfi-Kamran, Boris Grot, Michael Ferdman, Stavros Volos, Onur KocBerber, Javier Picorel, Almutaz Adileh, Djordje Jevdjic, Sachin Idgunji, Emre Ozer and Babak Falsafi. *Scale-Out Processors*. EcoCloud, EPFL. CALCM, Carnegie Mellon. ARM, 2012

[12] Nikos Hardavellas, Michael Ferdman, Anastasia Ailamaki, and Babak Falsafi. *Toward Dark Silicon in Servers*. IEEE Micro, 2011

[13] Mark Horowitz, Elad Alon, Dinesh Patil, Samuel Naffziger, Rajesh Kumar, and Kerry Bernstein. *Scaling, power, and the future of CMOS*. Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International, December 2005

[14] Google Data Centers. http://www.google.com/intl/en/corporate/datacenter/

[15] Open Compute Project. http://opencompute.org/

[16] B. Wheeler. *Tilera sees opening in clouds*. Microprocessor Report, 2011

[17] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. *Understanding and designing new server architectures for emerging warehouse-computing environments*. International Symposium on Computer Architecture, 2008

[18] N. Hardavellas, I. Pandis, R. Johnson, N. Mancheril, A. Ailamaki, and B. Falsafi. *Database servers on chip multiprocessors: limitations and opportunities*. Conference on Innovative Data Systems Research, 2007

[19] Philip Gschwandtner, Thomas Fahringer, Radu Prodan. *Performance Analysis and Benchmarking of the Intel SCC*. University of Innsbruck, 2011

[20] John-Nicholas Furst, Ayse K. Coskun. *Performance and Power Analysis of RCCE Message Passing on the Intel Single-Chip Cloud Computer*. Boston University.

[21] John Matienzo, Natalie Enright Jerger. *Performance Analysis of Broadcasting Algorithms on the Intel Single-Chip Cloud Computer*. University of Toronto.

[22] Andrea Bartolini, Mohammad Sadegh Sadri, John-Nicholas Furst, Ayse Kivilcim Coskun and Luca Benini. *Quantifying the Impact of Frequency Scaling on the Energy Efficiency of the Single-Chip Cloud Computer*. Boston University, 2012

[23] R. Van Der Wijngaart and H. Jin. *NAS Parallel Benchmarks, MultiZone Versions*. NASA Ames Research Center, 2003

[24] V. Springel. *The cosmological simulation code gadget-2*. MonthlyNotices of the Royal Astronomical Society, vol. 364, no. 4, pp. 11051134, 2005.

[25] J. D. McCalpin. *Memory Bandwidth and Machine Balance in Current High Performance Computers*. IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, pp. 1925, Dec. 1995.

[26] Jan-Arne Sobania, Peter Tröger. *Gentoo Linux on Intel SCC*. Operating Systems and Middleware Group, IT Systems Engineering, University Potsdam, 2010 https://www.dcl.hpi.uni-potsdam.de/research/scc/gentoo.htm

[27] *Cloudsuite, a benchmark suite for scale-out applications*. Parallel Systems Architecture Lab, EPFL. http://parsa.epfl.ch/cloudsuite/cloudsuite.html

[28] *DC Bench, a Benchmark Suite for Data Center Workloads*. ICT, Chinese Academy of Sciences. http://prof.ict.ac.cn/DCBench/

[29] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008

[30] Jason D.M. Rennie, Lawrence Shih, Jaime Teevan, David R. Karger. *Tackling the Poor Assumptions of Naive Bayes Text Classifiers*. Artificial Intelligence Laboratory, MIT, 2003

[31] Weizhong Zhao, Huifang Ma, Qing He. *Parallel K-Means Clustering Based on MapReduce*. The Key Laboratory of Intelligent Information Processing, Institute of Compution Technology, Chinese Academy of Sciences. Graduate University of Chinese Academy of Sciences, 2009

[32] Florian Verhein. *Frequent Pattern Growth (FP-Growth) Algorithm. An Introduction*. School of Information Technologies, The University of Sydney, Australia, 2008. http://www.florian.verhein.com/teaching/2008-01-09/fp-growth-presentation_v1%20(handout).pdf

[33] *Data Mining Algorithms In R/Frequent Pattern Mining/The FP-Growth Algorithm*. http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm

[34] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, Edward Chang. *PFP: Parallel FP-Growth for Query Recommendation*. Google Beijing Research. Dept. Computer Science, Peking University. Google Research, Mountain View

Appendix A

Code Samples

A.1 hadoop-topology.sh

```
if [ "$1" = "192.168.0.1" ]
then
    echo "/rack01";
fi
if [ "$1" = "192.168.0.2" ]
then
    echo "/rack01";
fi
if [ "$1" = "192.168.0.3" ]
then
    echo "/rack01";
fi
if [ "$1" = "192.168.0.4" ]
then
    echo "/rack01";
fi
if [ "$1" = "192.168.0.5" ]
then
    echo "/rack02";
fi
if [ "$1" = "192.168.0.6" ]
then
    echo "/rack03";
fi
if [ "$1" = "192.168.0.7" ]
then
    echo "/rack02";
fi
if [ "$1" = "192.168.0.8" ]
then
    echo "/rack03";
```

```
fi
if [ "$1" = "192.168.0.9" ]
then
    echo "/rack02";
fi
if [ "$1" = "192.168.0.10" ]
then
    echo "/rack03";
fi
if [ "$1" = "192.168.0.11" ]
then
    echo "/rack02";
fi
if [ "$1" = "192.168.0.12" ]
then
    echo "/rack03";
fi
if [ "$1" = "192.168.0.13" ]
then
    echo "/rack04";
fi
if [ "$1" = "192.168.0.14" ]
then
    echo "/rack05";
fi
if [ "$1" = "192.168.0.15" ]
then
    echo "/rack04";
fi
if [ "$1" = "192.168.0.16" ]
then
    echo "/rack05";
fi
if [ "$1" = "192.168.0.17" ]
then
    echo "/rack04";
fi
if [ "$1" = "192.168.0.18" ]
then
    echo "/rack05";
fi
if [ "$1" = "192.168.0.19" ]
then
    echo "/rack06";
fi
if [ "$1" = "192.168.0.20" ]
then
```

```
    echo "/rack07";
fi
if [ "$1" = "192.168.0.21" ]
then
    echo "/rack06";
fi
if [ "$1" = "192.168.0.22" ]
then
    echo "/rack07";
fi
if [ "$1" = "192.168.0.23" ]
then
    echo "/rack06";
fi
if [ "$1" = "192.168.0.24" ]
then
    echo "/rack07";
fi
if [ "$1" = "192.168.0.25" ]
then
    echo "/rack08";
fi
if [ "$1" = "192.168.0.26" ]
then
    echo "/rack09";
fi
if [ "$1" = "192.168.0.27" ]
then
    echo "/rack08";
fi
if [ "$1" = "192.168.0.28" ]
then
    echo "/rack09";
fi
if [ "$1" = "192.168.0.29" ]
then
    echo "/rack08";
fi
if [ "$1" = "192.168.0.30" ]
then
    echo "/rack09";
fi
if [ "$1" = "192.168.0.31" ]
then
    echo "/rack10";
fi
if [ "$1" = "192.168.0.32" ]
```

```
then
    echo "/rack11";
fi
if [ "$1" = "192.168.0.33" ]
then
    echo "/rack10";
fi
if [ "$1" = "192.168.0.34" ]
then
    echo "/rack11";
fi
if [ "$1" = "192.168.0.35" ]
then
    echo "/rack10";
fi
if [ "$1" = "192.168.0.36" ]
then
    echo "/rack11";
fi
if [ "$1" = "192.168.0.37" ]
then
    echo "/rack12";
fi
if [ "$1" = "192.168.0.38" ]
then
    echo "/rack13";
fi
if [ "$1" = "192.168.0.39" ]
then
    echo "/rack12";
fi
if [ "$1" = "192.168.0.40" ]
then
    echo "/rack13";
fi
if [ "$1" = "192.168.0.41" ]
then
    echo "/rack12";
fi
if [ "$1" = "192.168.0.42" ]
then
    echo "/rack13";
fi
if [ "$1" = "192.168.0.43" ]
then
    echo "/rack14";
fi
```

```
if [ "$1" = "192.168.0.44" ]
then
    echo "/rack15";
fi
if [ "$1" = "192.168.0.45" ]
then
    echo "/rack14";
fi
if [ "$1" = "192.168.0.46" ]
then
    echo "/rack15";
fi
if [ "$1" = "192.168.0.47" ]
then
    echo "/rack14";
fi
if [ "$1" = "192.168.0.48" ]
then
    echo "/rack15";
fi
```

A.2 watchdog-datanode-200.sh

```
while true
do
i=$1
if [ "$i" -lt 10 ]
    then
    PING="$(ping -c 1 rck0$i)";
    IFS=',';
    TOKENS=( $PING );
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING DATANODE rck0"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
            TOKENS=( $BOOT );
        done
        sleep 200;
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh";
        ssh -p 1234 root@rck0$i "gmond";
        ssh -p 1234 root@rck0$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                  --config /home/ageo/hadoop-0.20.2/conf-local"
        sleep 600;
    else
        CHECK_SOCKET="$(ssh root@rck0$i 'netstat -na | grep 50010')";
        if [ "${CHECK_SOCKET}" = "" ]
            then
                echo "WATCHDOG : REBOOTING DATANODE rck0"$i;
                BOOT="$(sccBoot -l $i)";
                echo "$BOOT";
                IFS=' ';
                TOKENS=( $BOOT );
                while [ "${TOKENS[0]}" = "ERROR:" ]
                do
                        sleep 10;
                        BOOT=$(sccBoot -l $i);
                        echo "$BOOT";
                        IFS=' ';
                        TOKENS=( $BOOT );
                done
```

```
                sleep 200;
                ssh root@rck0$i "/shared/ageo/rck0$i/start.sh";
                ssh -p 1234 root@rck0$i "gmond";
                ssh -p 1234 root@rck0$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                          --config /home/ageo/hadoop-0.20.2/conf-local"
                sleep 600;
        fi
    fi
else
    PING="$(ping -c 1 rck$i)";
    IFS=',';
    TOKENS=( $PING );
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING DATANODE rck"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
            TOKENS=( $BOOT );
        done
        sleep 200;
        ssh root@rck$i "/shared/ageo/rck$i/start.sh";
        ssh -p 1234 root@rck$i "gmond";
        ssh -p 1234 root@rck$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                  --config /home/ageo/hadoop-0.20.2/conf-local"
        sleep 600;
    else
        CHECK_SOCKET="$(ssh root@rck$i 'netstat -na | grep 50010')";
        if [ "${CHECK_SOCKET}" = "" ]
            then
                echo "WATCHDOG : REBOOTING DATANODE rck"$i;
                BOOT="$(sccBoot -l $i)";
                echo "$BOOT";
                IFS=' ';
                TOKENS=( $BOOT );
                while [ "${TOKENS[0]}" = "ERROR:" ]
                do
                        sleep 10;
                        BOOT=$(sccBoot -l $i);
                        echo "$BOOT";
```

```
                        IFS=' ';
                        TOKENS=( $BOOT );
                done
                sleep 200;
                ssh root@rck$i "/shared/ageo/rck0$i/start.sh";
                ssh -p 1234 root@rck$i "gmond";
                ssh -p 1234 root@rck$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                          --config /home/ageo/hadoop-0.20.2/conf-local"
                sleep 600;
        fi
    fi
fi
sleep 30
done
```

A.3 watchdog-datanode-533.sh

```
while true
do
i=$1
if [ "$i" -lt 10 ]
    then
    PING="$(ping -c 1 rck0$i)";
    IFS=',';
    TOKENS=( $PING );
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING DATANODE rck0"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
            TOKENS=( $BOOT );
        done
        sleep 75;
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh";
        ssh -p 1234 root@rck0$i "gmond";
        ssh -p 1234 root@rck0$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                  --config /home/ageo/hadoop-0.20.2/conf-local"
        sleep 225;
    else
        CHECK_SOCKET="$(ssh root@rck0$i 'netstat -na | grep 50010')";
        if [ "${CHECK_SOCKET}" = "" ]
            then
                echo "WATCHDOG : REBOOTING DATANODE rck0"$i;
                BOOT="$(sccBoot -l $i)";
                echo "$BOOT";
                IFS=' ';
                TOKENS=( $BOOT );
                while [ "${TOKENS[0]}" = "ERROR:" ]
                do
                        sleep 10;
                        BOOT=$(sccBoot -l $i);
                        echo "$BOOT";
                        IFS=' ';
                        TOKENS=( $BOOT );
                done
```

```
                sleep 75;
                ssh root@rck0$i "/shared/ageo/rck0$i/start.sh";
                ssh -p 1234 root@rck0$i "gmond";
                ssh -p 1234 root@rck0$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                          --config /home/ageo/hadoop-0.20.2/conf-local"
                sleep 225;
        fi
    fi
else
    PING="$(ping -c 1 rck$i)";
    IFS=',';
    TOKENS=( $PING );
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING DATANODE rck"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
            TOKENS=( $BOOT );
        done
        sleep 75;
        ssh root@rck$i "/shared/ageo/rck$i/start.sh";
        ssh -p 1234 root@rck$i "gmond";
        ssh -p 1234 root@rck$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                  --config /home/ageo/hadoop-0.20.2/conf-local"
        sleep 225;
    else
        CHECK_SOCKET="$(ssh root@rck$i 'netstat -na | grep 50010')";
        if [ "${CHECK_SOCKET}" = "" ]
            then
                echo "WATCHDOG : REBOOTING DATANODE rck"$i;
                BOOT="$(sccBoot -l $i)";
                echo "$BOOT";
                IFS=' ';
                TOKENS=( $BOOT );
                while [ "${TOKENS[0]}" = "ERROR:" ]
                do
                        sleep 10;
                        BOOT=$(sccBoot -l $i);
                        echo "$BOOT";
```

```
                        IFS=' ';
                        TOKENS=( $BOOT );
                done
                sleep 75;
                ssh root@rck$i "/shared/ageo/rck0$i/start.sh";
                ssh -p 1234 root@rck$i "gmond";
                ssh -p 1234 root@rck$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                          --config /home/ageo/hadoop-0.20.2/conf-local"
                sleep 225;
        fi
    fi
fi
sleep 30
done
```

A.4 watchdog-datanode-800.sh

```
while true
do
i=$1
if [ "$i" -lt 10 ]
    then
    PING="$(ping -c 1 rck0$i)";
    IFS=',';
    TOKENS=( $PING );
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING DATANODE rck0"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
            TOKENS=( $BOOT );
        done
        sleep 50;
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh";
        ssh -p 1234 root@rck0$i "gmond";
        ssh -p 1234 root@rck0$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                  --config /home/ageo/hadoop-0.20.2/conf-local"
        sleep 150;
    else
        CHECK_SOCKET="$(ssh root@rck0$i 'netstat -na | grep 50010')";
        if [ "${CHECK_SOCKET}" = "" ]
            then
                echo "WATCHDOG : REBOOTING DATANODE rck0"$i;
                BOOT="$(sccBoot -l $i)";
                echo "$BOOT";
                IFS=' ';
                TOKENS=( $BOOT );
                while [ "${TOKENS[0]}" = "ERROR:" ]
                do
                        sleep 10;
                        BOOT=$(sccBoot -l $i);
                        echo "$BOOT";
                        IFS=' ';
                        TOKENS=( $BOOT );
                done
```

```
                sleep 50;
                ssh root@rck0$i "/shared/ageo/rck0$i/start.sh";
                ssh -p 1234 root@rck0$i "gmond";
                ssh -p 1234 root@rck0$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                          --config /home/ageo/hadoop-0.20.2/conf-local"
                sleep 150;
        fi
    fi
else
    PING="$(ping -c 1 rck$i)";
    IFS=',';
    TOKENS=( $PING );
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING DATANODE rck"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
            TOKENS=( $BOOT );
        done
        sleep 50;
        ssh root@rck$i "/shared/ageo/rck$i/start.sh";
        ssh -p 1234 root@rck$i "gmond";
        ssh -p 1234 root@rck$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                  --config /home/ageo/hadoop-0.20.2/conf-local"
        sleep 150;
    else
        CHECK_SOCKET="$(ssh root@rck$i 'netstat -na | grep 50010')";
        if [ "${CHECK_SOCKET}" = "" ]
            then
                echo "WATCHDOG : REBOOTING DATANODE rck"$i;
                BOOT="$(sccBoot -l $i)";
                echo "$BOOT";
                IFS=' ';
                TOKENS=( $BOOT );
                while [ "${TOKENS[0]}" = "ERROR:" ]
                do
                        sleep 10;
                        BOOT=$(sccBoot -l $i);
                        echo "$BOOT";
```

```
                        IFS=' ';
                        TOKENS=( $BOOT );
                done
                sleep 50;
                ssh root@rck$i "/shared/ageo/rck0$i/start.sh";
                ssh -p 1234 root@rck$i "gmond";
                ssh -p 1234 root@rck$i "/home/ageo/hadoop-0.20.2/bin/start-dfs.sh
                          --config /home/ageo/hadoop-0.20.2/conf-local"
                sleep 150;
        fi
    fi
fi
sleep 30
done
```

A.5 watchdog-tasktracker-200.sh

```
while true
do
i=$1
if [ "$i" -lt 10 ]
then
    PING="$(ping -c 1 rck0$i)";
    IFS=',';
    TOKENS=( $PING );
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING TASKTRACKER rck0"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
            TOKENS=( $BOOT );
        done
        sleep 200;
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh";
        ssh -p 1234 root@rck0$i "gmond";
        ssh -p 1234 root@rck0$i "/home/ageo/hadoop-0.20.2/bin/start-mapred.sh
                  --config /home/ageo/hadoop-0.20.2/conf-local"
    fi
else
    PING="$(ping -c 1 rck$i)"
    IFS=','
    TOKENS=( $PING )
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING TASKTRACKER rck"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
```

```
            TOKENS=( $BOOT );
        done
        sleep 200;
        ssh root@rck$i "/shared/ageo/rck$i/start.sh";
        ssh -p 1234 root@rck$i "gmond";
        ssh -p 1234 root@rck$i "/home/ageo/hadoop-0.20.2/bin/start-mapred.sh
                --config /home/ageo/hadoop-0.20.2/conf-local"
    fi
fi
sleep 30
done
```

A.6 watchdog-tasktracker-533.sh

```
while true
do
i=$1
if [ "$i" -lt 10 ]
then
    PING="$(ping -c 1 rck0$i)";
    IFS=',';
    TOKENS=( $PING );
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING TASKTRACKER rck0"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
            TOKENS=( $BOOT );
        done
        sleep 75;
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh";
        ssh -p 1234 root@rck0$i "gmond";
        ssh -p 1234 root@rck0$i "/home/ageo/hadoop-0.20.2/bin/start-mapred.sh
                  --config /home/ageo/hadoop-0.20.2/conf-local"
    fi
else
    PING="$(ping -c 1 rck$i)"
    IFS=','
    TOKENS=( $PING )
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING TASKTRACKER rck"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
```

```
            TOKENS=( $BOOT );
        done
        sleep 75;
        ssh root@rck$i "/shared/ageo/rck$i/start.sh";
        ssh -p 1234 root@rck$i "gmond";
        ssh -p 1234 root@rck$i "/home/ageo/hadoop-0.20.2/bin/start-mapred.sh
                --config /home/ageo/hadoop-0.20.2/conf-local"
    fi
fi
sleep 30
done
```

A.7 watchdog-tasktracker-800.sh

```
while true
do
i=$1
if [ "$i" -lt 10 ]
then
    PING="$(ping -c 1 rck0$i)";
    IFS=',';
    TOKENS=( $PING );
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING TASKTRACKER rck0"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
            TOKENS=( $BOOT );
        done
        sleep 50;
        ssh root@rck0$i "/shared/ageo/rck0$i/start.sh";
        ssh -p 1234 root@rck0$i "gmond";
        ssh -p 1234 root@rck0$i "/home/ageo/hadoop-0.20.2/bin/start-mapred.sh
                  --config /home/ageo/hadoop-0.20.2/conf-local"
    fi
else
    PING="$(ping -c 1 rck$i)"
    IFS=','
    TOKENS=( $PING )
    if [ ${TOKENS[1]} != " 1 received" ]
        then
        echo "WATCHDOG : REBOOTING TASKTRACKER rck"$i;
        BOOT="$(sccBoot -l $i)";
        echo "$BOOT";
        IFS=' ';
        TOKENS=( $BOOT );
        while [ "${TOKENS[0]}" = "ERROR:" ]
        do
            sleep 10;
            BOOT=$(sccBoot -l $i);
            echo "$BOOT";
            IFS=' ';
```

```
            TOKENS=( $BOOT );
        done
        sleep 50;
        ssh root@rck$i "/shared/ageo/rck$i/start.sh";
        ssh -p 1234 root@rck$i "gmond";
        ssh -p 1234 root@rck$i "/home/ageo/hadoop-0.20.2/bin/start-mapred.sh
                --config /home/ageo/hadoop-0.20.2/conf-local"
    fi
fi
sleep 30
done
```

A.8 gmond.conf for the MCPC

```
/* This configuration is as close to 2.5.x default behavior as possible
   The values closely match ./gmond/metric.h definitions in 2.5.x */
globals {
  daemonize = yes
  setuid = yes
  user = ganglia
  debug_level = 0
  max_udp_msg_len = 1472
  mute = yes
  deaf = no
  host_dmax = 0 /*secs */
  cleanup_threshold = 300 /*secs */
  gexec = no
  send_metadata_interval = 0
}

/* If a cluster attribute is specified, then all gmond hosts are wrapped inside
 * of a <CLUSTER> tag.  If you do not specify a cluster tag, then all <HOSTS> will
 * NOT be wrapped inside of a <CLUSTER> tag. */
cluster {
  name = "MARC"
  owner = "unspecified"
  latlong = "unspecified"
  url = "unspecified"
}

/* The host section describes attributes of the host, like the location */
host {
  location = "unspecified"
}

/* You can specify as many udp_recv_channels as you like as well. */
udp_recv_channel {
  port = 8649
}

/* You can specify as many tcp_accept_channels as you like to share
   an xml description of the state of the cluster */
tcp_accept_channel {
  port = 8649
}

/* Each metrics module that is referenced by gmond must be specified and
   loaded. If the module has been statically linked with gmond, it does not
   require a load path. However all dynamically loadable modules must include
   a load path. */
```

```
modules {
  module {
    name = "core_metrics"
  }
  module {
    name = "cpu_module"
    path = "/usr/lib/ganglia/modcpu.so"
  }
  module {
    name = "disk_module"
    path = "/usr/lib/ganglia/moddisk.so"
  }
  module {
    name = "load_module"
    path = "/usr/lib/ganglia/modload.so"
  }
  module {
    name = "mem_module"
    path = "/usr/lib/ganglia/modmem.so"
  }
  module {
    name = "net_module"
    path = "/usr/lib/ganglia/modnet.so"
  }
  module {
    name = "proc_module"
    path = "/usr/lib/ganglia/modproc.so"
  }
  module {
    name = "sys_module"
    path = "/usr/lib/ganglia/modsys.so"
  }
}

include ('/etc/ganglia/conf.d/*.conf')
```

A.9 gmond.conf for an Intel SCC Core

```
/* This configuration is as close to 2.5.x default behavior as possible
   The values closely match ./gmond/metric.h definitions in 2.5.x */
globals {
  daemonize = yes
  setuid = yes
  user = nobody
  debug_level = 0
  max_udp_msg_len = 1472
  mute = no
  deaf = yes
  allow_extra_data = yes
  host_dmax = 0 /*secs */
  cleanup_threshold = 300 /*secs */
  gexec = no
  send_metadata_interval = 0
}
/*
 * The cluster attributes specified will be used as part of the <CLUSTER>
 * tag that will wrap all hosts collected by this instance.
 */
cluster {
  name = "MARC"
  owner = "unspecified"
  latlong = "unspecified"
  url = "unspecified"
}

/* The host section describes attributes of the host, like the location */
host {
  location = "unspecified"
}

/* Feel free to specify as many udp_send_channels as you like.  Gmond
   used to only support having a single channel */
udp_send_channel {
  host = 192.168.3.254
  port = 8649
  ttl = 1
}

/* Each metrics module that is referenced by gmond must be specified and
   loaded. If the module has been statically linked with gmond, it does
   not require a load path. However all dynamically loadable modules must
   include a load path. */
modules {
  module {
```

```
    name = "core_metrics"
  }
  module {
    name = "cpu_module"
    path = "modcpu.so"
  }
  module {
    name = "disk_module"
    path = "moddisk.so"
  }
  module {
    name = "load_module"
    path = "modload.so"
  }
  module {
    name = "mem_module"
    path = "modmem.so"
  }
  module {
    name = "net_module"
    path = "modnet.so"
  }
  module {
    name = "proc_module"
    path = "modproc.so"
  }
  module {
    name = "sys_module"
    path = "modsys.so"
  }
}

include ('/etc/ganglia/conf.d/*.conf')

/* CPU status */
collection_group {
  collect_every = 1
  time_threshold = 1
  metric {
    name = "cpu_user"
    value_threshold = 0.1
    title = "CPU User"
  }
  metric {
    name = "cpu_system"
    value_threshold = 0.1
    title = "CPU System"
```

```
  }
  metric {
    name = "cpu_wio"
    value_threshold = 0.1
    title = "CPU WIO"
  }
  metric {
    name = "cpu_idle"
    value_threshold = 0.1
    title = "CPU Idle"
  }
}
/* network traffic */
collection_group {
  collect_every = 1
  time_threshold = 1
  metric {
    name = "bytes_in"
    value_threshold = 0.01
    title = "Bytes Received"
  }
  metric {
    name = "bytes_out"
    value_threshold = 0.01
    title = "Bytes Sent"
  }
}
```

A.10 store-power.py

```
import time
import datetime
import sys
import signal

def signal_handler(signal,frame):
    sys.exit(0)

#register signal handler
signal.signal(signal.SIGINT,signal_handler)

f1 = open(sys.argv[1],'w+');

start_time = time.time()
t1 = time.time()
t2 = time.time()

while (t2 - start_time < sys.argv[2]):
t1 = time.time()
f2 = os.popen('sccBmc -c status | grep 3V3SCC')
t2 = time.time()

status = f2.read().split()

voltage = status[1]
current = status[3]
time_interval = t2 - t1
power = float(status[1]) * float(status[3])
energy_consumption = power * time_interval

f1.write(str(datetime.datetime.now()) + '\t' + voltage + '\t'
    + current + '\t' + str(power) + '\t' + str(energy_consumption) +'\n');
f1.close()
```

A.11 store-metrics.py

```
import xml.etree.ElementTree
import mysql.connector
import datetime
import os
import time
import signal
import sys

def signal_handler(signal,frame):
    #close db connection
    cursor.close()
    connection.close()
    sys.exit(0)

#open db connection
connection = mysql.connector.connect(user='ageo',password='ageo',
    host='127.0.0.1',database='SCC_CLOUDSUITE_METRICS')
cursor = connection.cursor()

#register signal handler
signal.signal(signal.SIGINT,signal_handler)

f = os.popen('sccTherm -initTherm 9556')
f.close()

while (True):

    now = datetime.datetime.now()

    power_file = 'power_metrics/' + sys.argv[1]
        + '/power-' + str(time.time()) + '.txt'
    f = os.popen('python store-power.py ' + power_file + ' ' + sys.argv[4])
    f.close()

    f = open(power_file,'r')

    energy_consumption = 0
    for line in f:
       energy_consumption +=  float(line.split('\t',6)[5])

    f.close()
    f = os.popen('sccBmc -c status ')
    flag = 'other'
```

```
for line in f:
    if (flag == 'temperature'):
        temperature = line.split()[1]
        flag = 'other'
    elif (flag == 'fan_speed'):
        fan_speed = line.split()[1]
        flag = 'other'
    elif (line == 'Temperatures:\n'):
        flag = 'temperature'
    elif (line == 'Fan speed:\n'):
        flag = 'fan_speed'

f.close()
insert_metrics_query = "INSERT INTO " + sys.argv[3] +
    " (TIMESTAMP,FAN_SPEED,TEMPERATURE,ENERGY_CONSUMPTION,POWER_FILE)
    VALUES (%s,%s,%s,%s,%s)"
insert_metrics_data = (now,fan_speed,temperature,
    energy_consumption,power_file.split('/',3)[2])
cursor.execute(insert_metrics_query,insert_metrics_data)
connection.commit()
f = os.popen("telnet localhost 8649 > telnet_output" )
f.close()
f = os.popen("sed -n 5,7885p telnet_output > ganglia.xml")
f.close()

tree = xml.etree.ElementTree.parse("ganglia.xml")
root = tree.getroot()

hosts = root.findall("./CLUSTER/HOST")
for host in hosts:
    hostname_of_core = host.get('NAME').split('.')[0]
    metrics = host.findall('./METRIC')
    for metric in metrics:
        metric_name = metric.get('NAME')
        metric_value = metric.get('VAL')
        if metric_name == 'cpu_user':
            cpu_user = metric_value
        elif metric_name == 'cpu_system':
            cpu_system = metric_value
        elif metric_name == 'cpu_wio':
            cpu_wio = metric_value
        elif metric_name == 'cpu_idle':
            cpu_idle = metric_value
        elif metric_name == 'bytes_in':
            bytes_in = metric_value
```

```
        elif metric_name == 'bytes_out':
            bytes_out = metric_value

    #store metrics in DB
    insert_metrics_query = ("INSERT INTO " + sys.argv[2] +
        "(TIMESTAMP,CORE, CPU_USER,CPU_SYSTEM,
        CPU_WIO,CPU_IDLE,BYTES_IN,BYTES_OUT)
        VALUES (%s,%s,%s,%s,%s,%s,%s,%s)")
    insert_metrics_data = (now,hostname_of_core,cpu_user,
        cpu_system,cpu_wio,cpu_idle,bytes_in,bytes_out)
    cursor.execute(insert_metrics_query,insert_metrics_data)
    connection.commit()
```

A.12 prepare-metrics-cpu-network.py

```
import mysql.connector
import sys

#open db connection
connection = mysql.connector.connect(user='ageo',password='ageo',
    host='127.0.0.1',database='SCC_CLOUDSUITE_METRICS')
cursor = connection.cursor()

f = open('plot_files/cpu_network/' + sys.argv[2],'w+')
sql = "select TIMESTAMP,CPU_USER,CPU_SYSTEM,
    CPU_WIO,CPU_IDLE,BYTES_IN,BYTES_OUT
    from " + sys.argv[1] + " where CORE='" + sys.argv[3] + "'"

try:
    #execute sql query
    cursor.execute(sql)
    #fetch all rows in a list of lists
    i=0
    results = cursor.fetchall()
    for row in results:
        if (i==0):
            start_timestamp = row[0]
            i += 1
        timestamp = row[0] - start_timestamp
        cpu_user = row[1]
        cpu_system = row[2]
        cpu_wio = row[3]
        cpu_idle = row[4]
        bytes_in = row[5]
        bytes_out = row[6]
        f.write(str(timestamp) + '\t' + cpu_user + '\t' +
            cpu_system + '\t' + cpu_wio + '\t' + cpu_idle +
            '\t' + bytes_in + '\t' + bytes_out+ '\n')
except:
    print "sql error"

#close file and db connection
f.close()
cursor.close()
connection.close()
```

A.13 prepare-metrics-thermal.py

```
import mysql.connector
import sys

#open db connection
connection = mysql.connector.connect(user='ageo',password='ageo',
    host='127.0.0.1',database='SCC_CLOUDSUITE_METRICS')
cursor = connection.cursor()

f = open('plot_files/thermal/' + sys.argv[2],'w+')
sql = "select TIMESTAMP,TEMPERATURE,FAN_SPEED from " + sys.argv[1]

try:
    #execute sql query
    cursor.execute(sql)
    #fetch all rows in a list of lists
    i=0
    results = cursor.fetchall()
    for row in results:
    if (i==0):
        start_timestamp = row[0]
        i += 1
    timestamp = row[0] - start_timestamp
    temperature = row[1]
    fan_speed = row[2]
    f.write(str(timestamp) + '\t' + temperature + '\t' + fan_speed + '\n')
except:
    print "sql error"

#close file and db connection
f.close()
cursor.close()
connection.close()
```

A.14 prepare-metrics-power.py

```
import mysql.connector
import sys
import os
import datetime

os.system('rm tmp/tmp.dat')
os.system('cat power_metrics/' + sys.argv[1] + '/* >> tmp/tmp.dat')
f1 = open('tmp/tmp.dat','r+')
f2 = open('plot_files/power/' + sys.argv[2],'w+')

i = 1
for line in f1:
    if (i==1):
        words = line.split()
        start_timestamp = datetime.datetime.strptime(words[0]+
            ' '+words[1],"%Y-%m-%d %H:%M:%S.%f")
    if (i % sys.argv[3]== 0):
        words = line.split()
        timestamp = datetime.datetime.strptime(words[0]+
            ' '+words[1],"%Y-%m-%d %H:%M:%S.%f") - start_timestamp
        voltage = words[2]
        current = words[3]
        power = words[5]
        f2.write(str(timestamp) + '\t' + voltage + '\t' +
            current + '\t' + power + '\n')
    i+=1

#close file connections
f1.close()
f2.close()
```

A.15 plot-cpu.gp

```
set terminal png size 2880, 480
set output "image.png"
set key outside
set key right top
set xdata time
set timefmt  x "%H:%M:%S"
set xtics xtics
set ytics 10
set xr ["0:00:00":time]
set yr [0:100]
set xlabel "Time"
set ylabel "CPU Utilization"
plot "plot_files/cpu_network/".datafile using 1:(100)
           title 'CPU User' with filledcurves x1 lt rgb "#FF0000", \
     "plot_files/cpu_network/".datafile using 1:($3+$4+$5)
           title 'CPU System' with filledcurves x1 lt rgb "#FFD700", \
     "plot_files/cpu_network/".datafile using 1:($4+$5)
           title 'CPU WIO' with filledcurves x1 lt rgb "#0000FF", \
     "plot_files/cpu_network/".datafile using 1:($5)
           title 'CPU Idle' with filledcurves x1 lt rgb "#008000"
```

A.16 plot-network.gp

```
set terminal png size 2880, 480
set output "image.png"
set key outside
set key right top
set xdata time
set timefmt  x "%H:%M:%S"
set xtics xtics
set ytics auto
set xr ["0:00:00":time]
set xlabel "Time"
set ylabel "Network Traffic in KB/s"
plot "plot_files/cpu_network/".datafile using 1:($6/1000)
           title 'KB/s In' with lines linewidth 3 lt rgb "#B8860B", \
     "plot_files/cpu_network/".datafile using 1:($7/1000)
           title 'KB/s Out' with lines linewidth 3 lt rgb "#0000FF"
```

A.17 plot-power.gp

```
set terminal png size 2880, 480
set output "image.png"
set key outside
set key right top
set xdata time
set timefmt  x "%H:%M:%S"
set xtics xtics
set ytics auto
set xr ["0:00:00":time]
set xlabel "Time"
set ylabel "Power Consumption in W"
plot "plot_files/power/".datafile using 1:4
    title 'Power in W' with lines linewidth 3 lt rgb "#FF0000"
```

A.18 plot-temperature.gp

```
set terminal png size 2880, 480
set output "image.png"
set key outside
set key right top
set xdata time
set timefmt  x "%H:%M:%S"
set xtics xtics
set ytics 1
set xr ["0:00:00":time]
set xlabel "Time"
set ylabel "Temperature in C"
plot "plot_files/thermal/".datafile using 1:2
    title 'Temperature in C' with lines linewidth 3 lt rgb "#FF0000"
```

A.19 plot-fan-speed.gp

```
set terminal png size 2800, 480
set output "image.png"
set key outside
set key right top
set xdata time
set timefmt  x "%H:%M:%S"
set xtics xtics
set ytics auto
set xr ["0:00:00":time]
set xlabel "Time"
set ylabel "Fan Speed in RPM"
plot "plot_files/thermal/".datafile using 1:3
    title 'Fan Speed in RPM' with lines linewidth 3 lt rgb "#0000FF"
```

A.20 prepare-wordcount.sh

```
#!/bin/bash

source ../../configuration/config.include

# generate parameters
unit=‘echo $1 | sed ’s/[0-9.]//g’ | tr [a-z] [A-Z]‘
size=‘echo $1 | sed ’s/[A-Za-z]//g’‘
bytes_per_map=0
maps_per_host=0
unit_size=0
#hosts=‘/liuwb/hadoop-1.0.2/bin/hadoop dfsadmin
    -report | grep -Po "Datanodes available: \d+" | grep -Po "\d+"‘
hosts=$2

if   [ "$unit" = "M" ]; then
    unit_size=20
elif [ "$unit" = "G" ]; then
    unit_size=30
elif [ "$unit" = "T" ]; then
    unit_size=40
elif test -z $1; then
    echo "Workload wasnt specified, please specify one(for example:1m/1g/1t)"
    exit
fi

size_per_host=‘echo "scale=2; $size / $hosts" | bc‘

index=0
while [ $(echo "$size_per_host < 0.5 " | bc) -eq 1
    -o $(echo "$size_per_host > 1.5" | bc) -eq 1 ]
do
    if [ $(echo "$size_per_host < 0.5 " | bc) -eq 1 ]
    then
        size_per_host=‘echo "scale=2; $size_per_host * 2" | bc‘
        let "index-=1"
    else
        size_per_host=‘echo "scale=2; $size_per_host / 2" | bc‘
        let "index+=1"
    fi
done
let "unit_size+=$index"

if   [ $unit_size -gt 33 ]
then
    let "maps_per_host=8*2**($unit_size-33)"
    let "bytes_per_host=2**$unit_size";
```

```
    bytes_per_map=‘echo
        "($bytes_per_host*$size_per_host)/$maps_per_host"| bc‘
elif [ $unit_size -gt 29 ]
then
    maps_per_host=8
    let "bytes_per_host=2**$unit_size";
    bytes_per_map=‘echo
        "($bytes_per_host*$size_per_host)/$maps_per_host"| bc‘
else
    maps_per_host=1
    let "bytes_per_host=2**$unit_size";
    bytes_per_map=‘echo "($bytes_per_host*$size_per_host)"| bc‘
fi
bytes_per_map=${bytes_per_map%.*}

echo BYTES_PER_MAP $bytes_per_map
echo MAPS_PER_HOST $maps_per_host
echo HOSTS $hosts


# fix the config file
lineno=‘grep -n "bytes_per_map" config-wordcount.xml‘
lineno=${lineno%:*}
let "lineno+=1"
sed -i "$lineno s/<value>[0-9]*<\/value>/<value>
    $bytes_per_map<\/value>/" config-wordcount.xml

lineno=‘grep -n "maps_per_host" config-wordcount.xml‘
lineno=${lineno%:*}
let "lineno+=1"
sed -i "$lineno s/<value>[0-9]*<\/value>/<value>
    $maps_per_host<\/value>/" config-wordcount.xml


echo "generating rtw-wordcount-$size$unit data"
#${HADOOP_HOME}/bin/hadoop fs -rmr /cloudrank-data/rtw-wordcount-$size$unit
${HADOOP_HOME}/bin/hadoop jar ../../jars/${hadoop_examples_jar}
    randomtextwriter -conf config-wordcount.xml  /cloudrank-data/rtw-wordcount-$size$unit
sed -i "/$size$unit/d" ./file.include
sed -i "/$size$unit/d" ../../configuration/file_all.include
echo "wordcount_file=rtw-wordcount-$size$unit-$1"
    >>./file.include
echo "wordcount_file=rtw-wordcount-$size$unit-$1"
    >>../../configuration/file_all.include
```

A.21 run-wordcount.sh

```
#!/bin/bash
source ../../configuration/config.include

dataset=‘echo $1 | tr [a-z] [A-Z]‘
${HADOOP_HOME}/bin/hadoop fs -rmr /cloudrank-out/rtw-wordcount-$dataset-out
${HADOOP_HOME}/bin/hadoop jar  ${HADOOP_HOME}/hadoop-0.20.2-examples.jar
    wordcount  /cloudrank-data/rtw-wordcount-$dataset
    /cloudrank-out/rtw-wordcount-$dataset-out
```

A.22 prepare-kmeans.sh

```
#!/bin/bash

source ../../configuration/config.include
source file.include

if   [ $1 = low  ]
then
    kmeans_file="sougou-low-tfidf-vec"
    ratio="low"
elif [ $1 = mid  ]
then
    kmeans_file="sougou-mid-tfidf-vec"
    ratio="mid"
elif [ $1 = high ]
then
    kmeans_file="sougou-high-tfidf-vec"
    ratio="high"
elif test -z $1
then
    echo "Workload wasnt specified, run the low workload as default."
    kmeans_file="sougou-low-tfidf-vec"
    ratio="low"
else
    echo "Workload specified doesnot exist, please doublecheck."
    exit
fi
#${HADOOP_HOME}/bin/hadoop fs -rmr "${hdfsdata_dir}/kmeans*"
${HADOOP_HOME}/bin/hadoop fs -copyFromLocal
    "${basedata_dir}/${kmeans_file}" ${hdfsdata_dir}/
sed -i "/$ratio/d" ./file.include
#sed -i "/$ratio/d" ../../configuration/file.include
echo "kmeans_file=$kmeans_file-$ratio" >>  ./file.include
echo "kmeans_file=$kmeans_file-$ratio" >> ../../configuration/file_all.include
```

A.23 run-kmeans.sh

```
#!/bin/bash

source ../../configuration/config.include
source file.include

kmeans_file=
if   [ $1 = low  ]
then
    kmeans_file="sougou-low-tfidf-vec"
elif [ $1 = mid  ]
then
    kmeans_file="sougou-mid-tfidf-vec"
elif [ $1 = high ]
then
    kmeans_file="sougou-high-tfidf-vec"
elif test -z $1
then
    echo "Workload wasnt specified, run the low workload as default."
    kmeans_file="sougou-low-tfidf-vec"
else
    echo "Workload specified doesnot exist, please doublecheck."
    exit
fi

${HADOOP_HOME}/bin/hadoop fs -rmr /cloudrank-out/${kmeans_file}
${MAHOUT_HOME}/bin/mahout kmeans
    -i /cloudrank-data/${kmeans_file}
    -o /cloudrank-out/${kmeans_file}
    -k 5
    -c /cloudrank-out/${kmeans_file}
    -x 5
```

A.24 prepare-fpg.sh

```
#!/bin/bash

source ../../configuration/config.include
source file.include

fpg_file=
if   [ $1 = low  ]
then
    fpg_file="fpg-accidents.dat"
    ratio="low"
elif [ $1 = mid  ]
then
    fpg_file="fpg-retail.dat"
    ratio="mid"
elif [ $1 = high ]
then
    fpg_file="fpg-webdocs.dat"
    ratio="high"
elif test -z $1
then
    echo "Workload wasnt specified, run the low workload as default."
    fpg_file="fpg-accidents.dat"
    ratio="low"
else
    echo "Workload specified doesnot exist, please doublecheck."
    exit
fi

#${HADOOP_HOME}/bin/hadoop fs -rmr "${hdfsdata_dir}/fpg*"
${HADOOP_HOME}/bin/hadoop fs -put "${basedata_dir}/${fpg_file}" ${hdfsdata_dir}/

sed -i "/$ratio/d" ./file.include
sed -i "/$ratio/d" ../../configuration/file_all.include
echo "fpg_file=${fpg_file}-$ratio" >> ./file.include
echo "fpg_file=${fpg_file}-$ratio" >> ../../configuration/file_all.include
```

A.25 run-fpg.sh

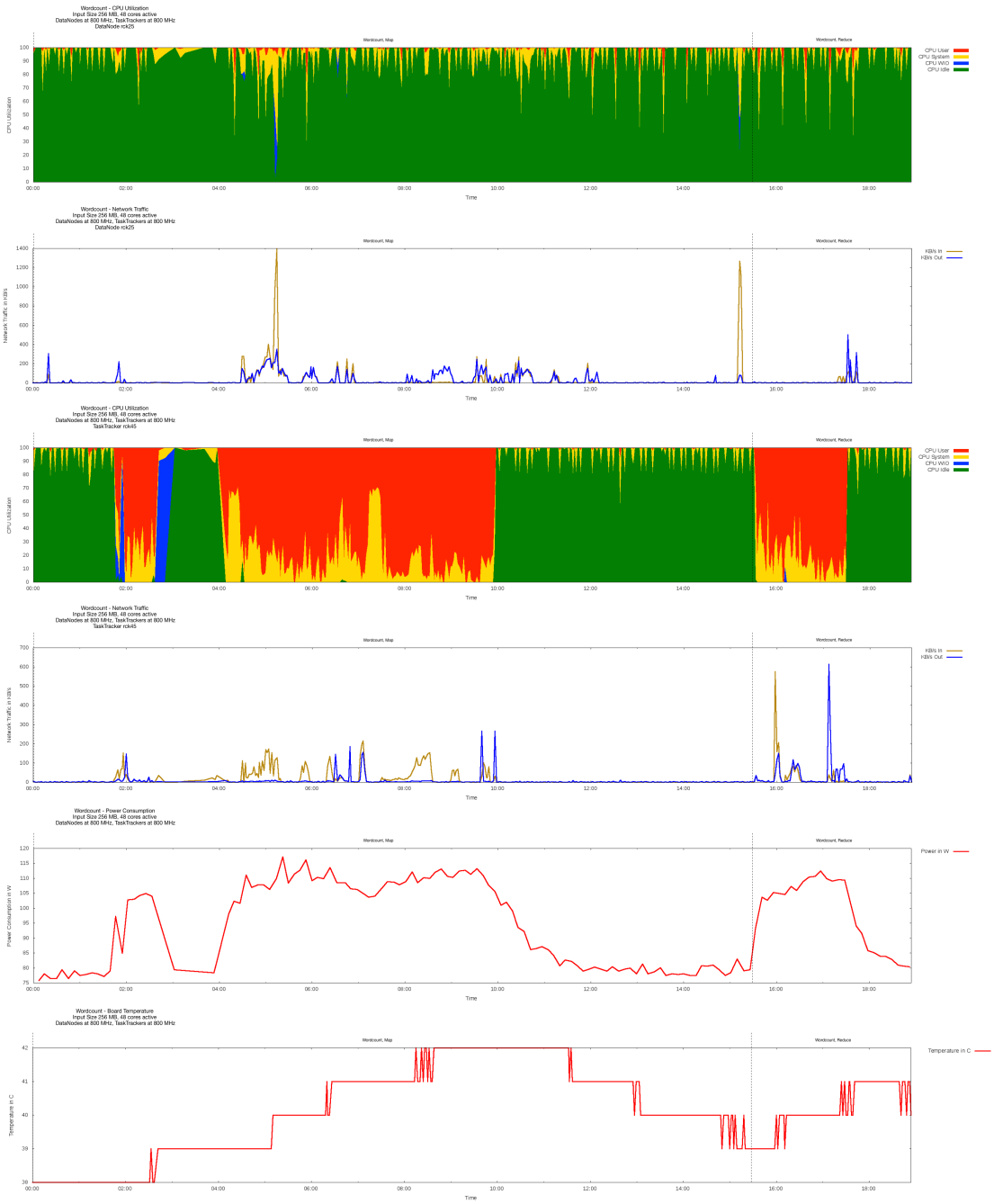
```
#!/bin/bash

source ../../configuration/config.include
source file.include

fpg_file=
if   [ $1 = "low" ]
then
    fpg_file="fpg-accidents.dat"
elif [ $1 = "mid" ]
then
    fpg_file="fpg-retail.dat"
elif [ $1 = "high" ]
then
    fpg_file="fpg-webdocs.dat"
elif test -z $1
then
    echo "Workload wasnt specified, run the low workload as default."
    fpg_file="fpg-accidents.dat"
else
    echo "Workload specified doesnot exist, please doublecheck."
    exit
fi
${HADOOP_HOME}/bin/hadoop fs -rmr /cloudrank-out/${fpg_file}-out
${MAHOUT_HOME}/bin/mahout fpg
    -i /cloudrank-data/${fpg_file}
    -o /cloudrank-out/${fpg_file}-out
    -s 4
    -k 100
    -method mapreduce
```

Appendix B

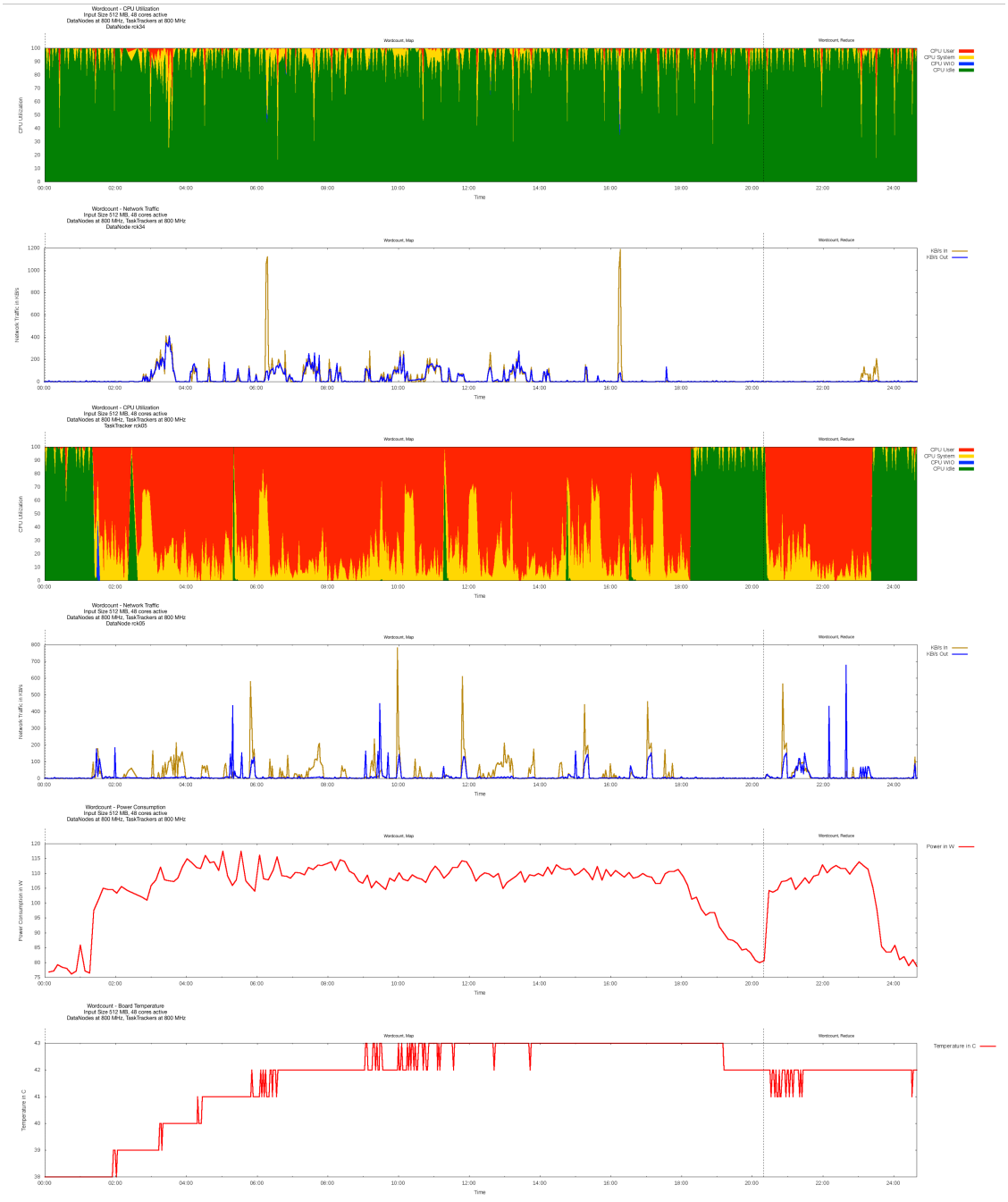
Plots

B.1 Wordcount

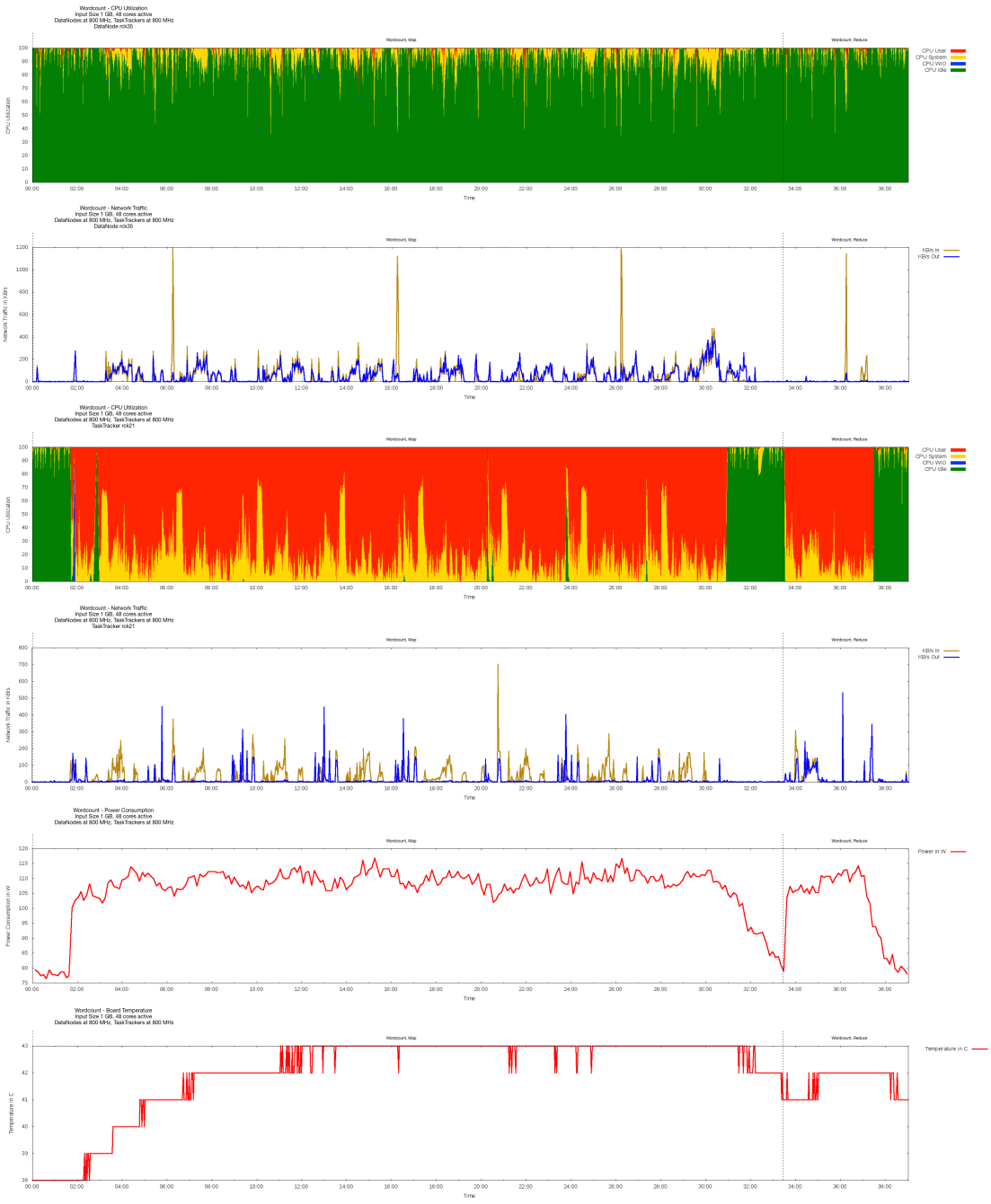
B.1.1 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



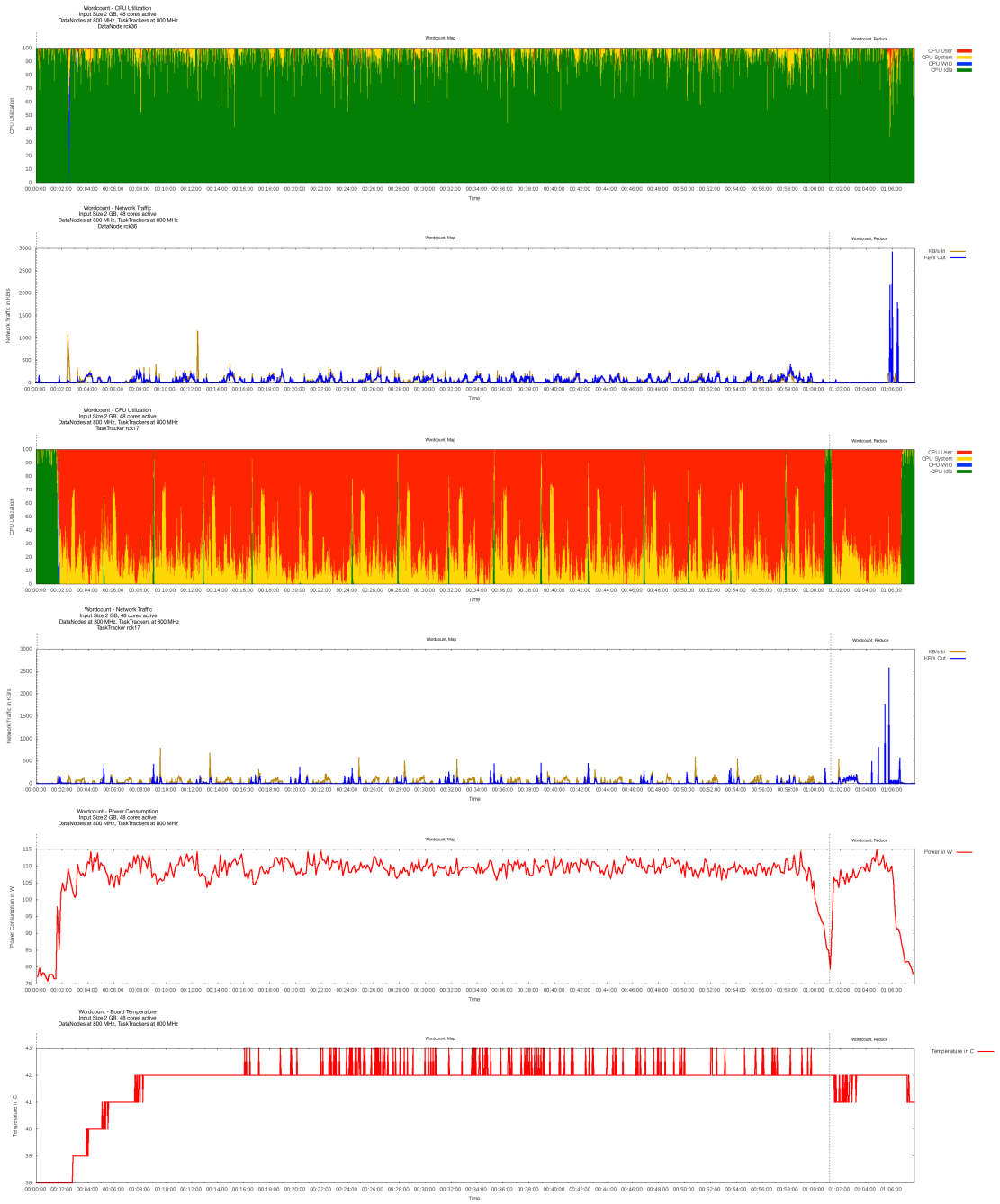
B.1.2 Input Size 512 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



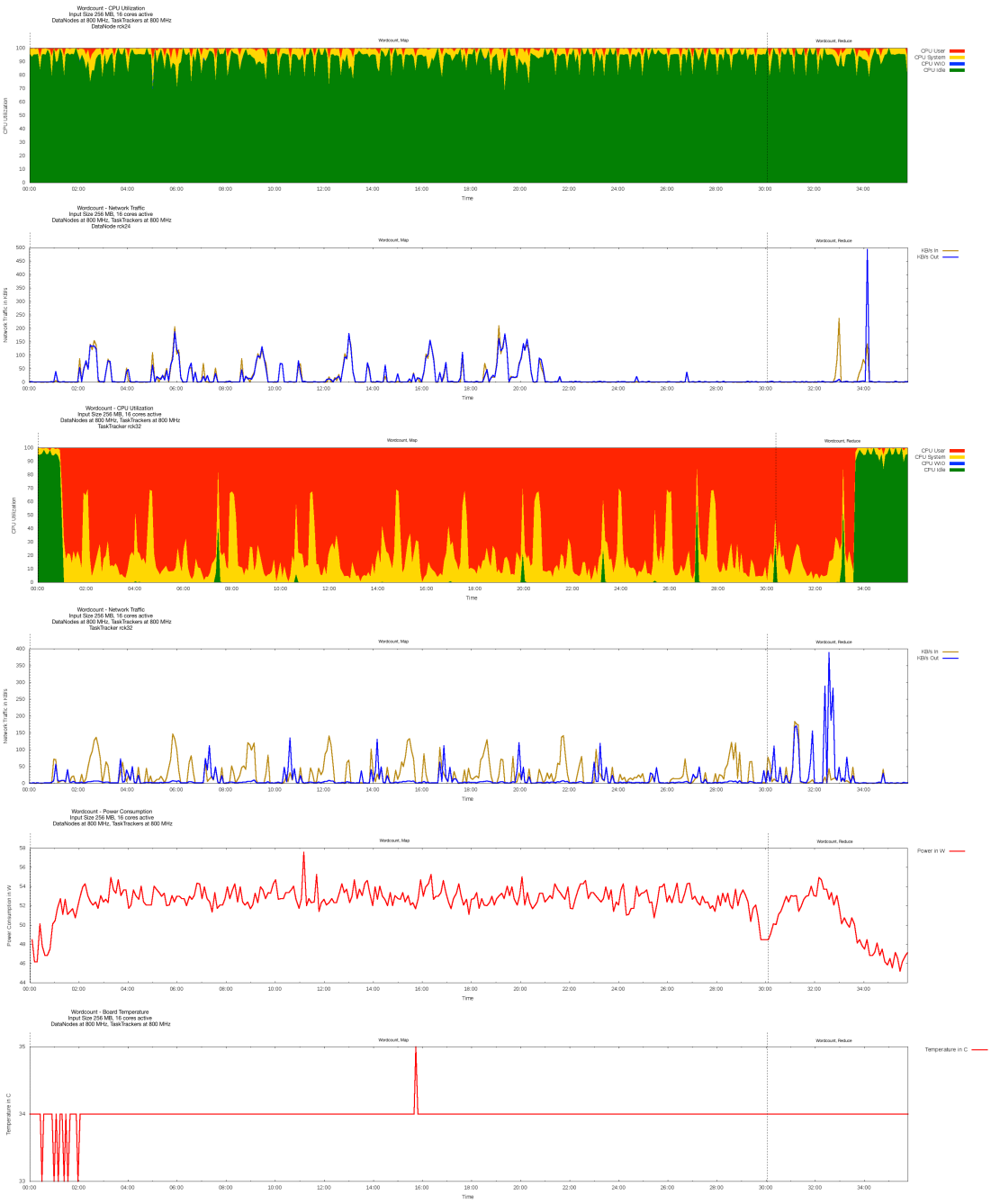
B.1.3 Input Size 1 GB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



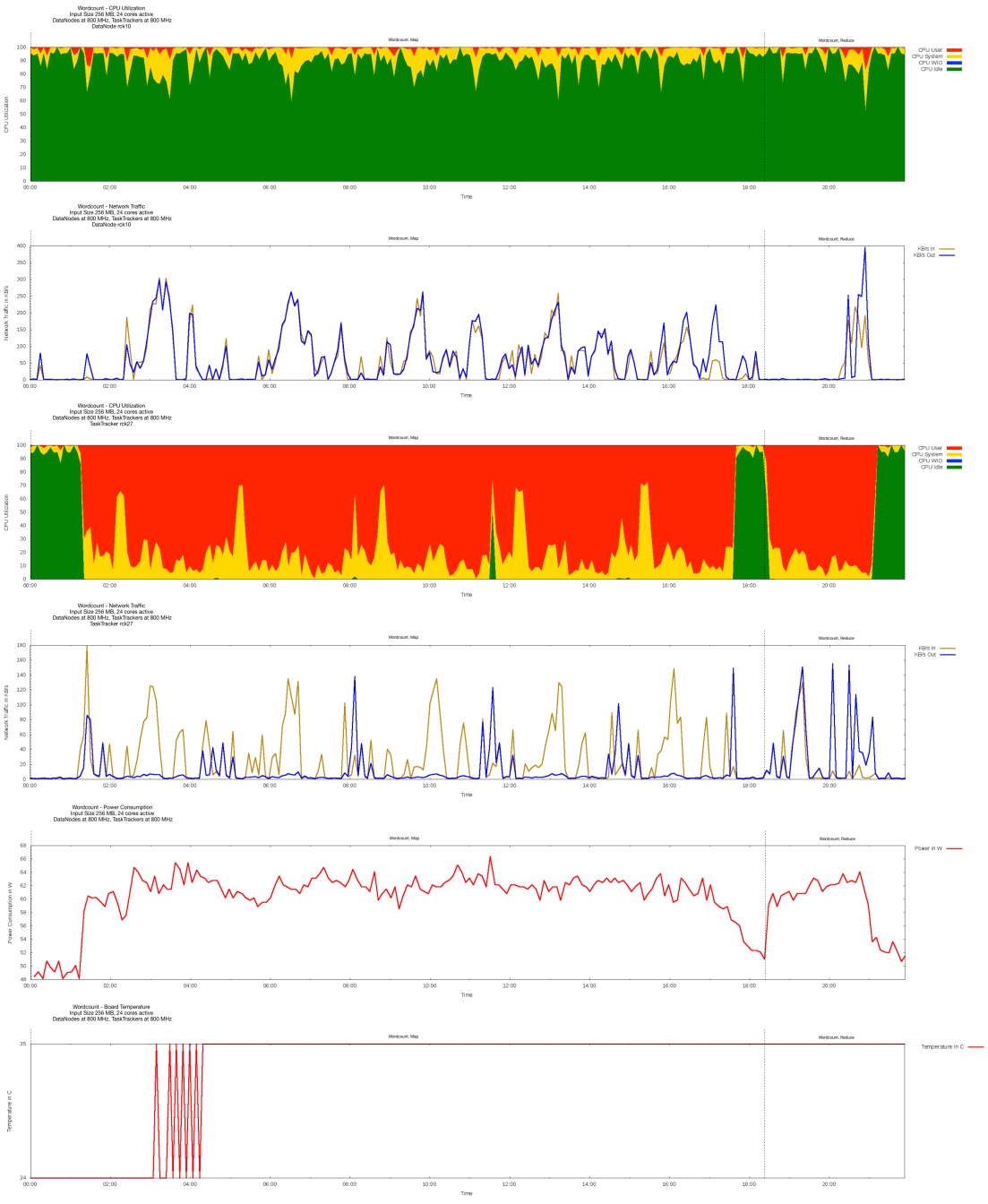
B.1.4 Input Size 2 GB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



B.1.5 Input Size 256 MB, 16-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

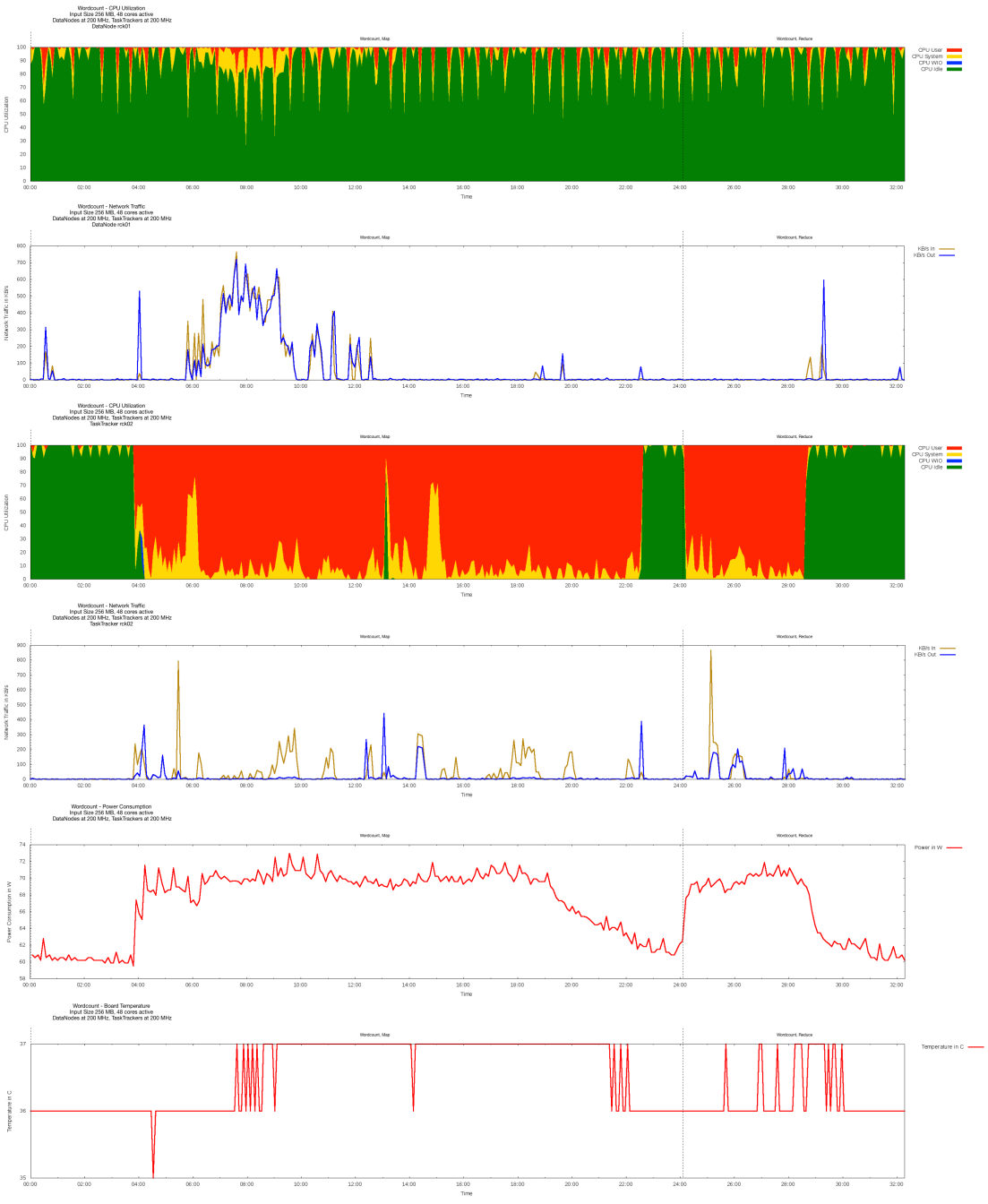


B.1.6 Input Size 256 MB, 24-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

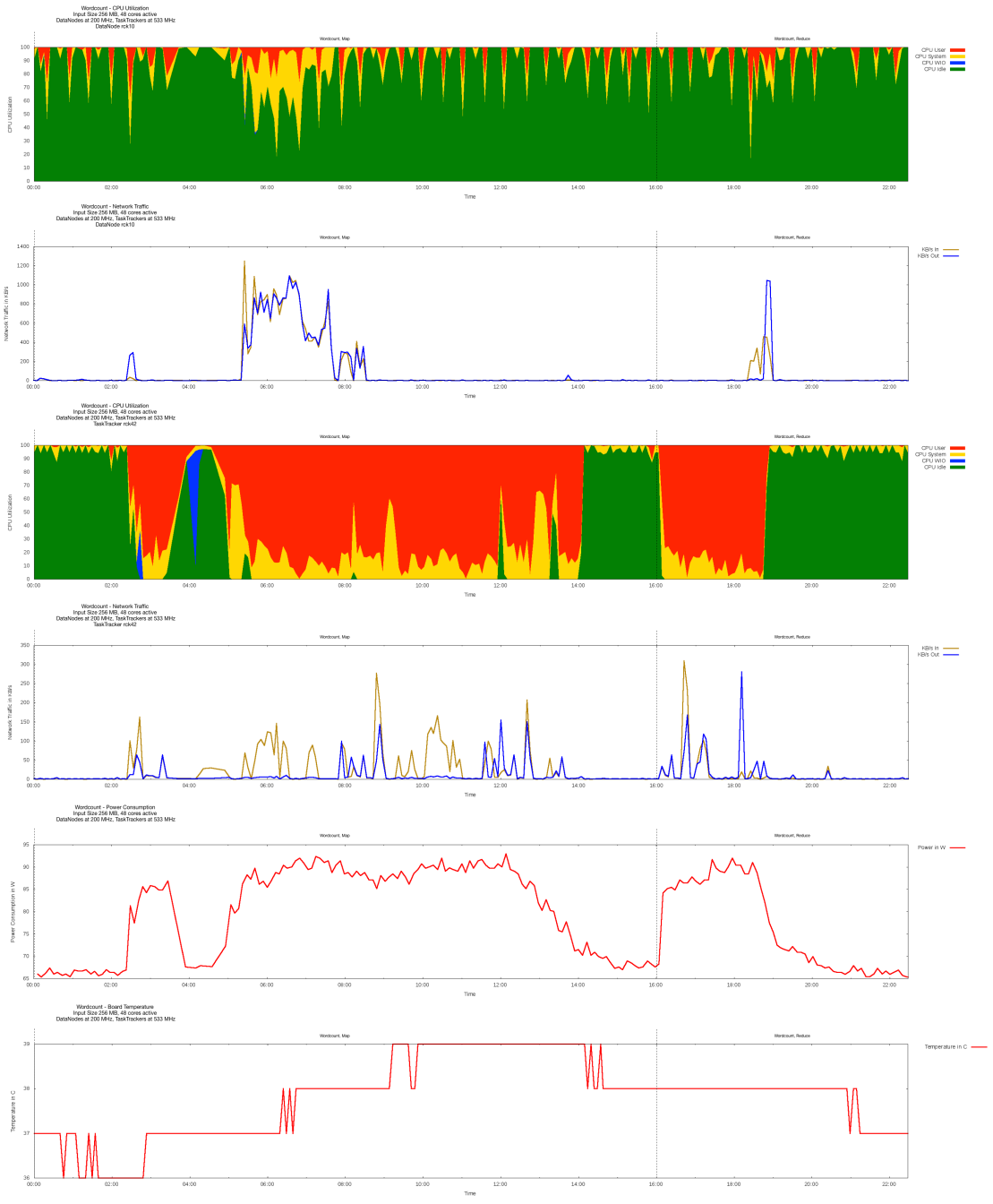


B.1.7 Input Size 256 MB, 32-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

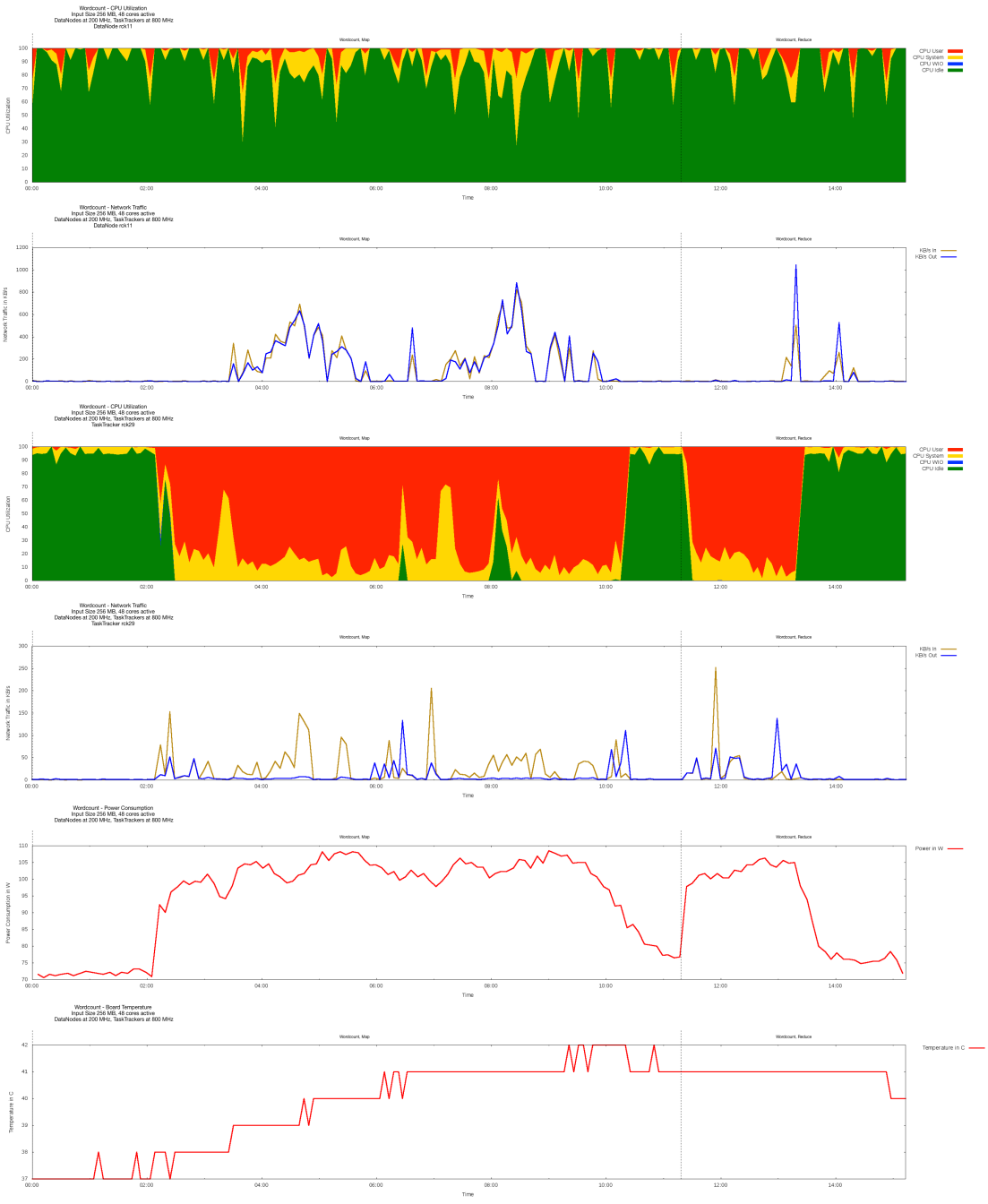
B.1.8 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 200 MHz



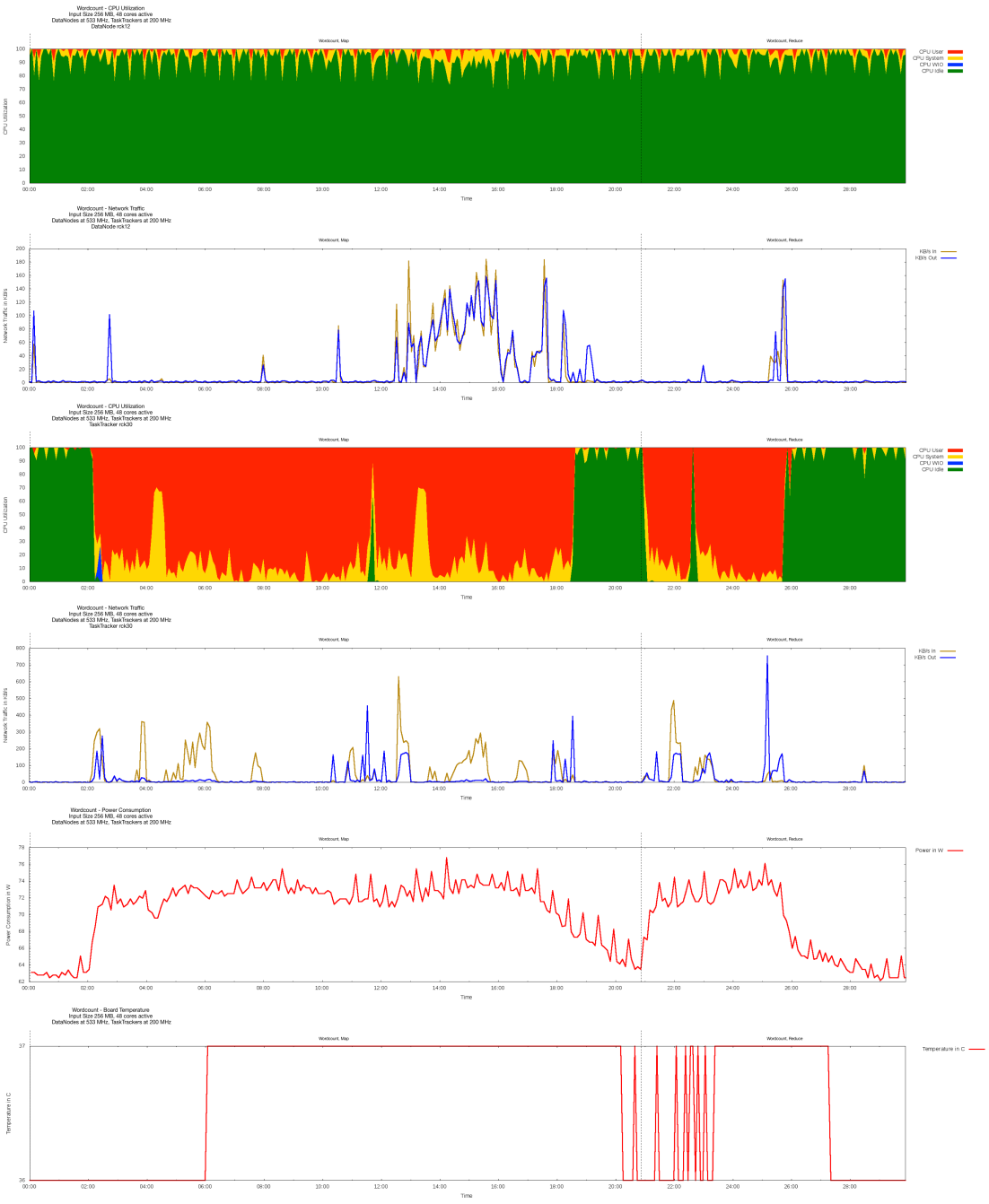
B.1.9 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 533 MHz



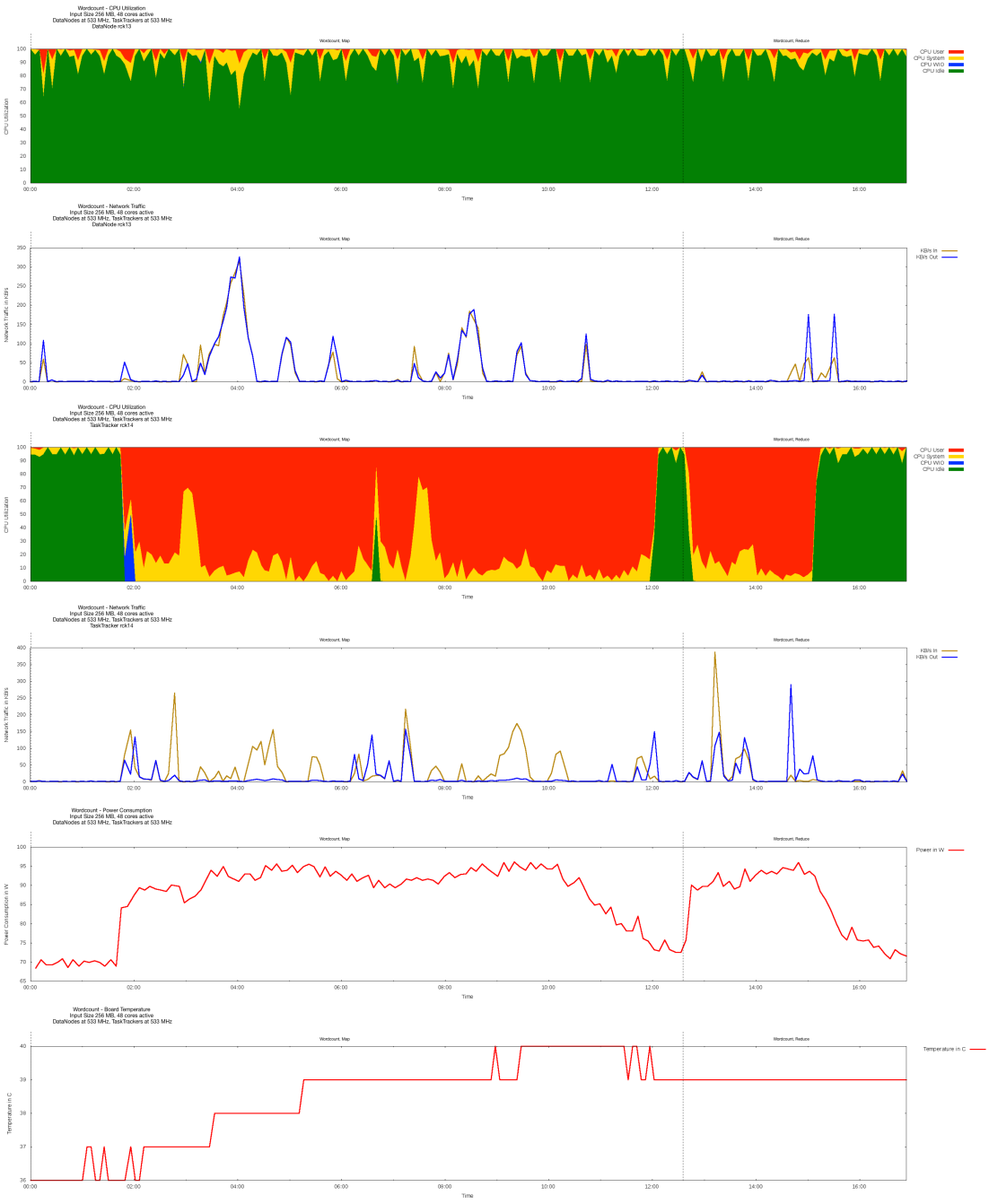
B.1.10 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 800 MHz



B.1.11 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 200 MHz

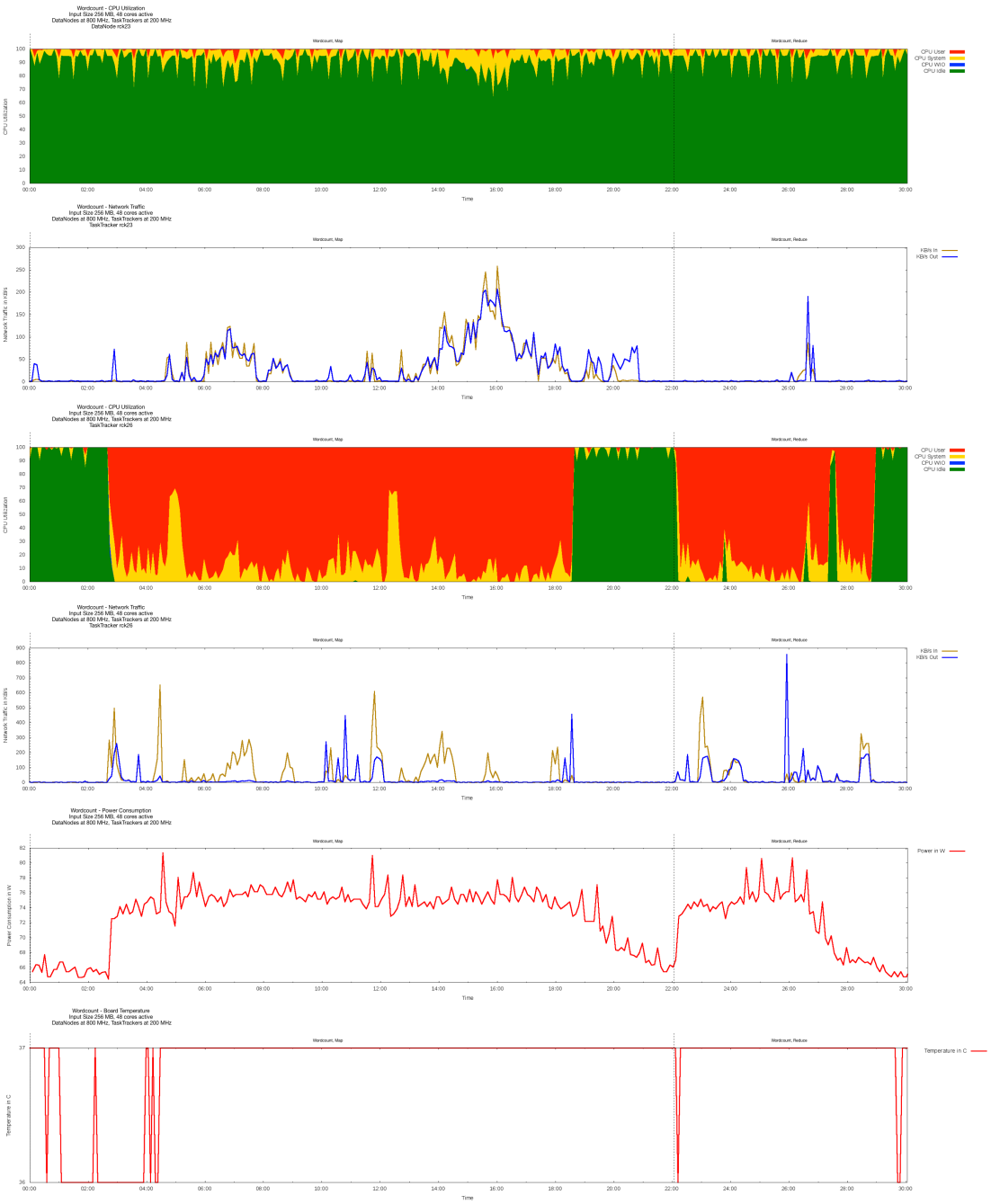


B.1.12 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 533 MHz

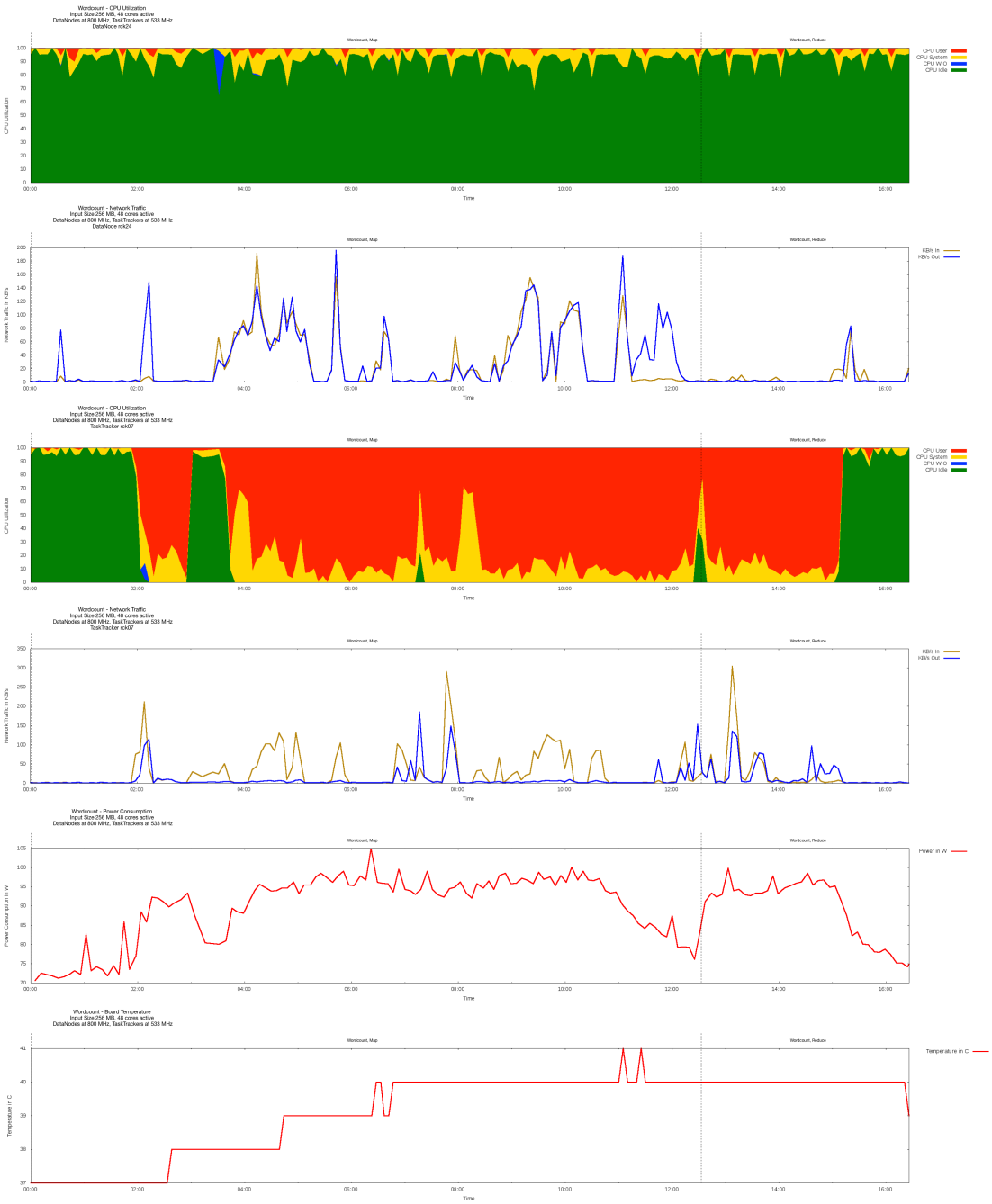


B.1.13 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 800 MHz

B.1.14 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 200 MHz

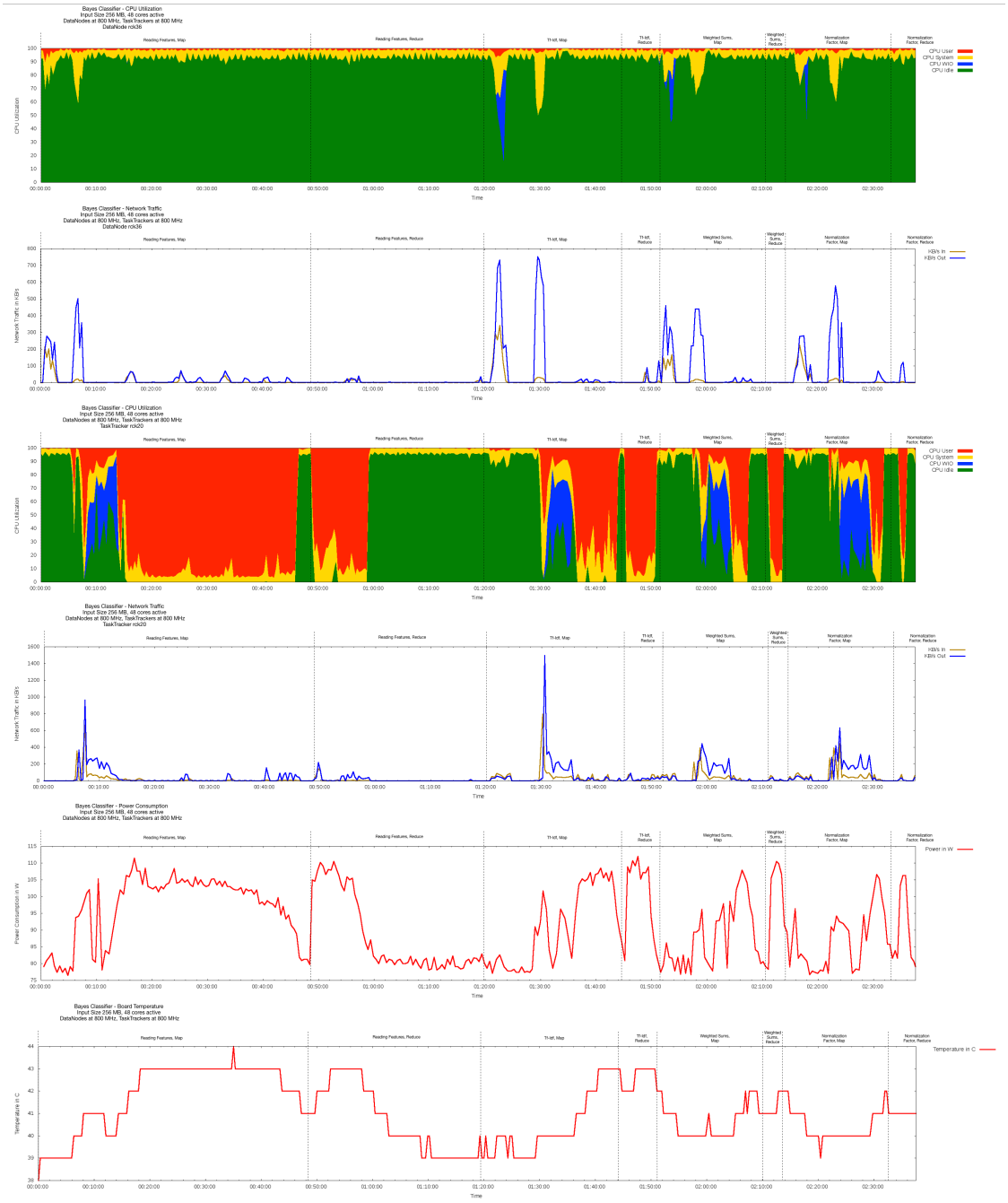


B.1.15 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 533 MHz

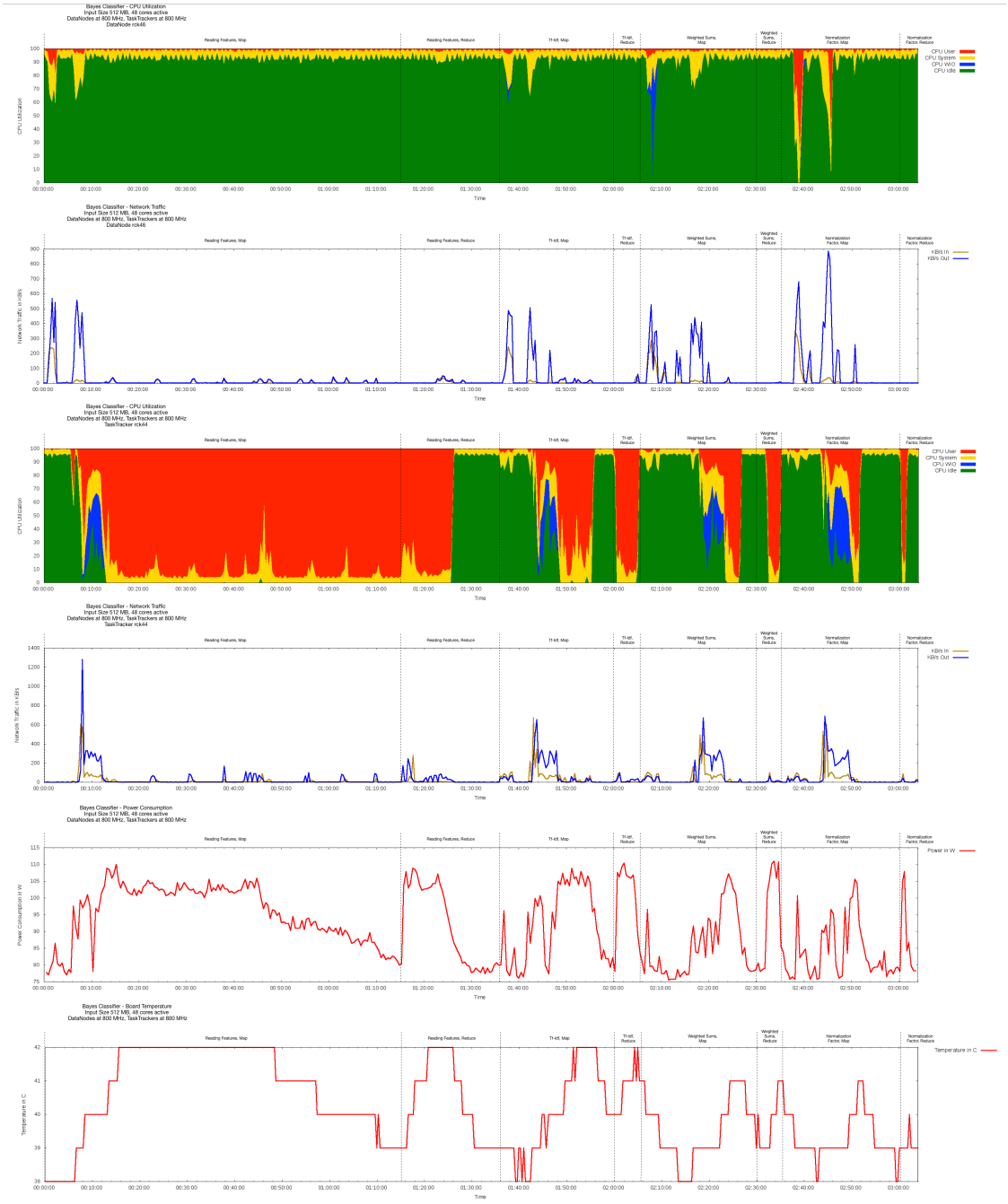


B.2 Bayes Classifier

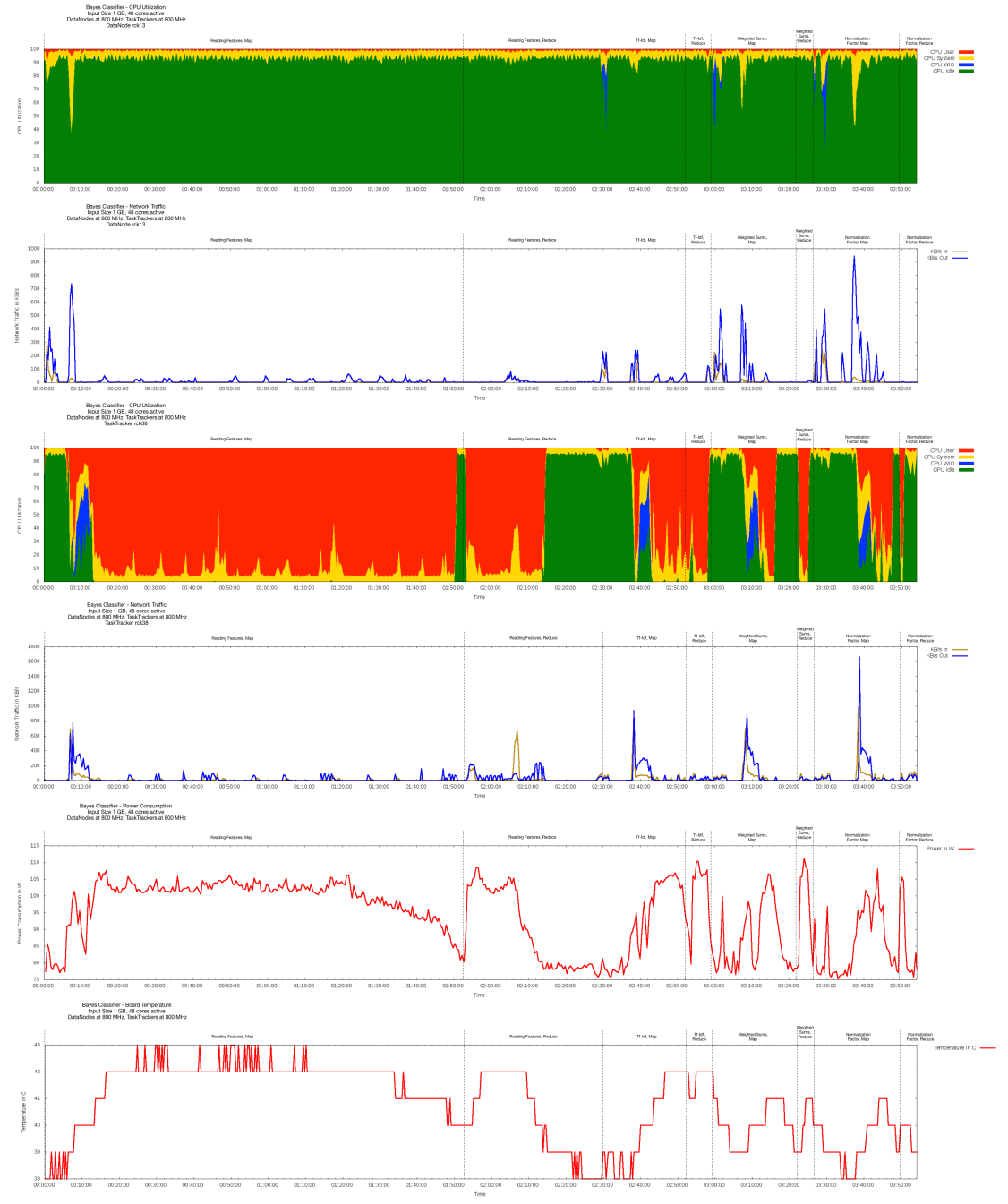
B.2.1 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



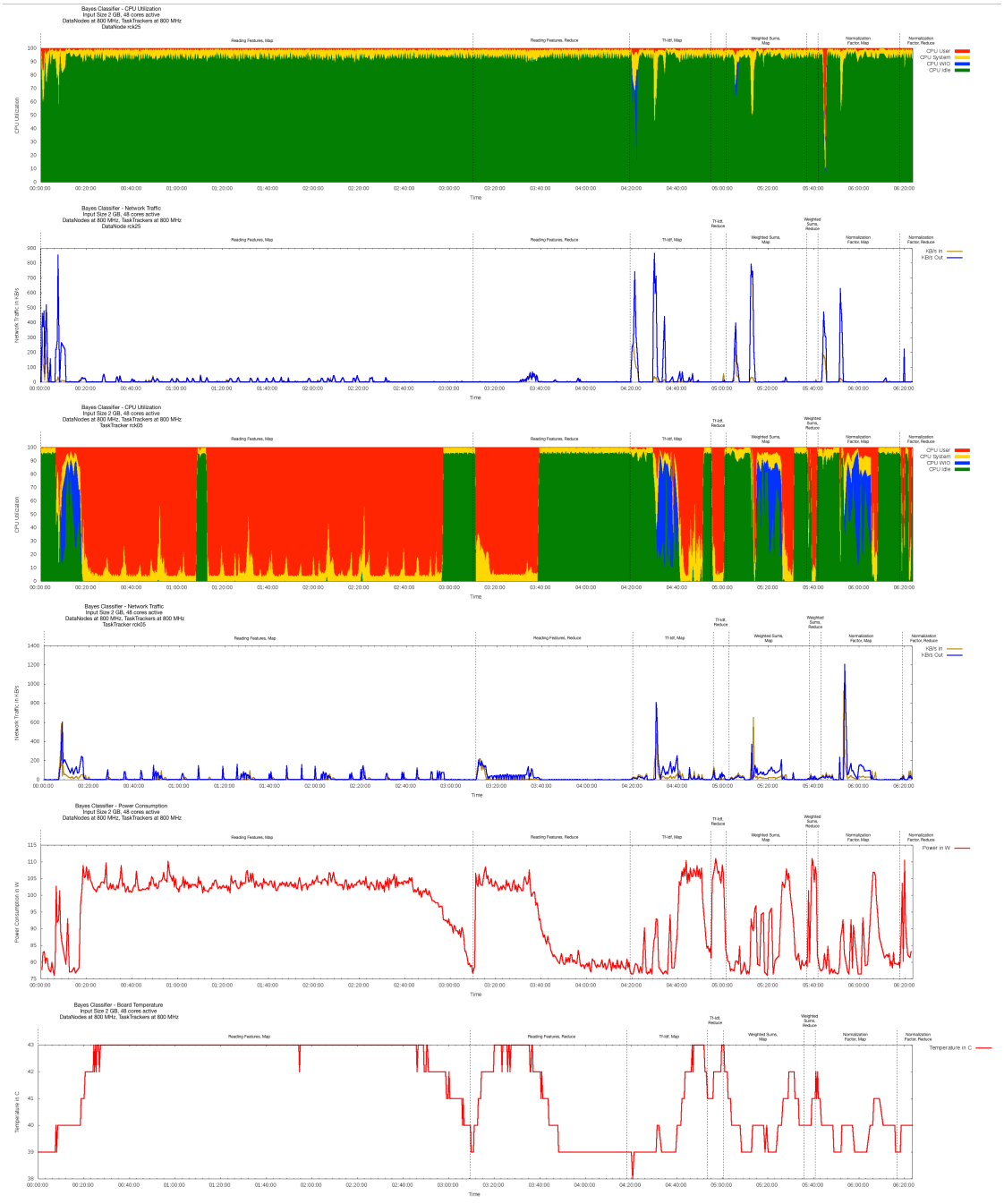
B.2.2 Input Size 512 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



B.2.3 Input Size 1 GB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

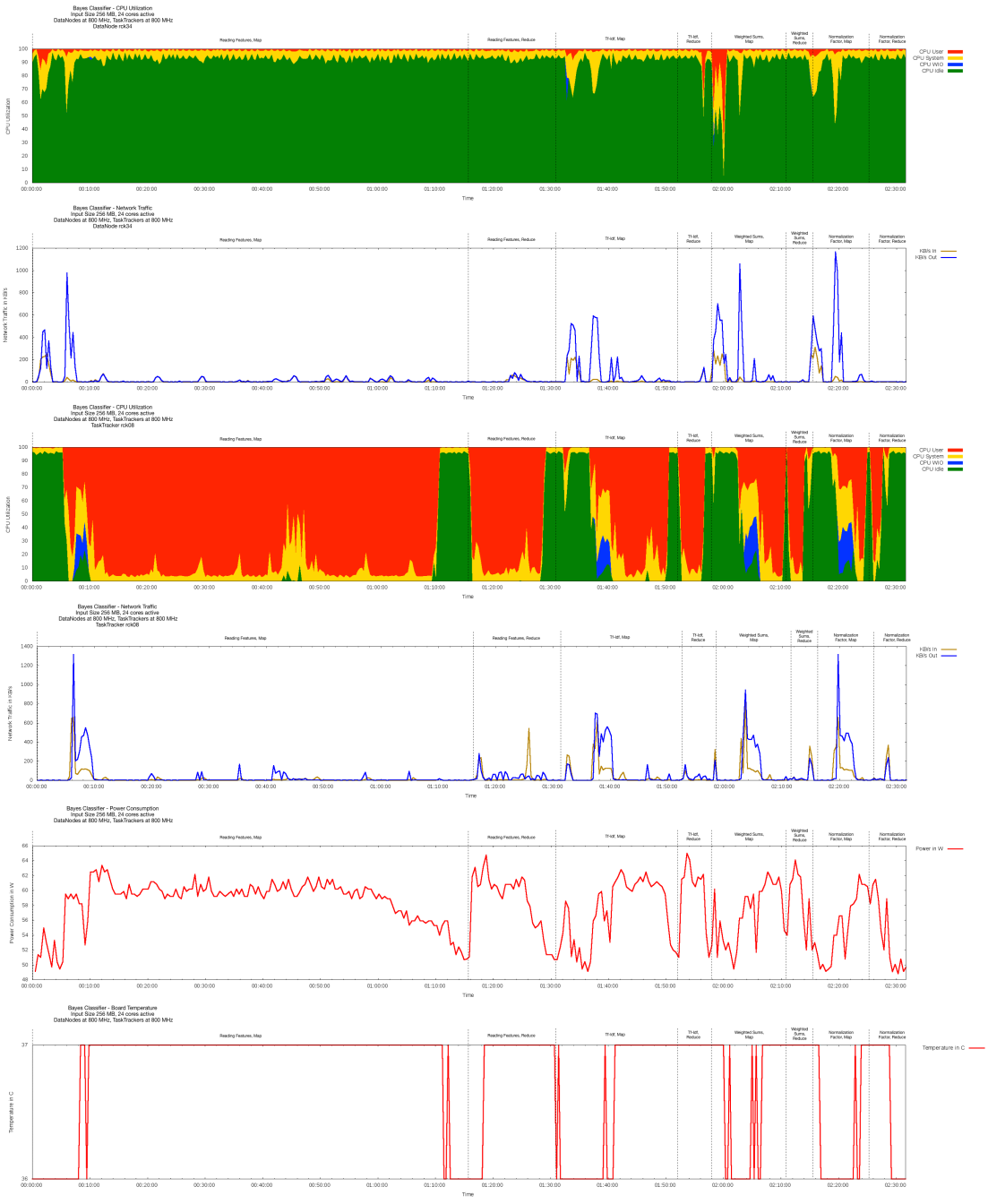


B.2.4 Input Size 2 GB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



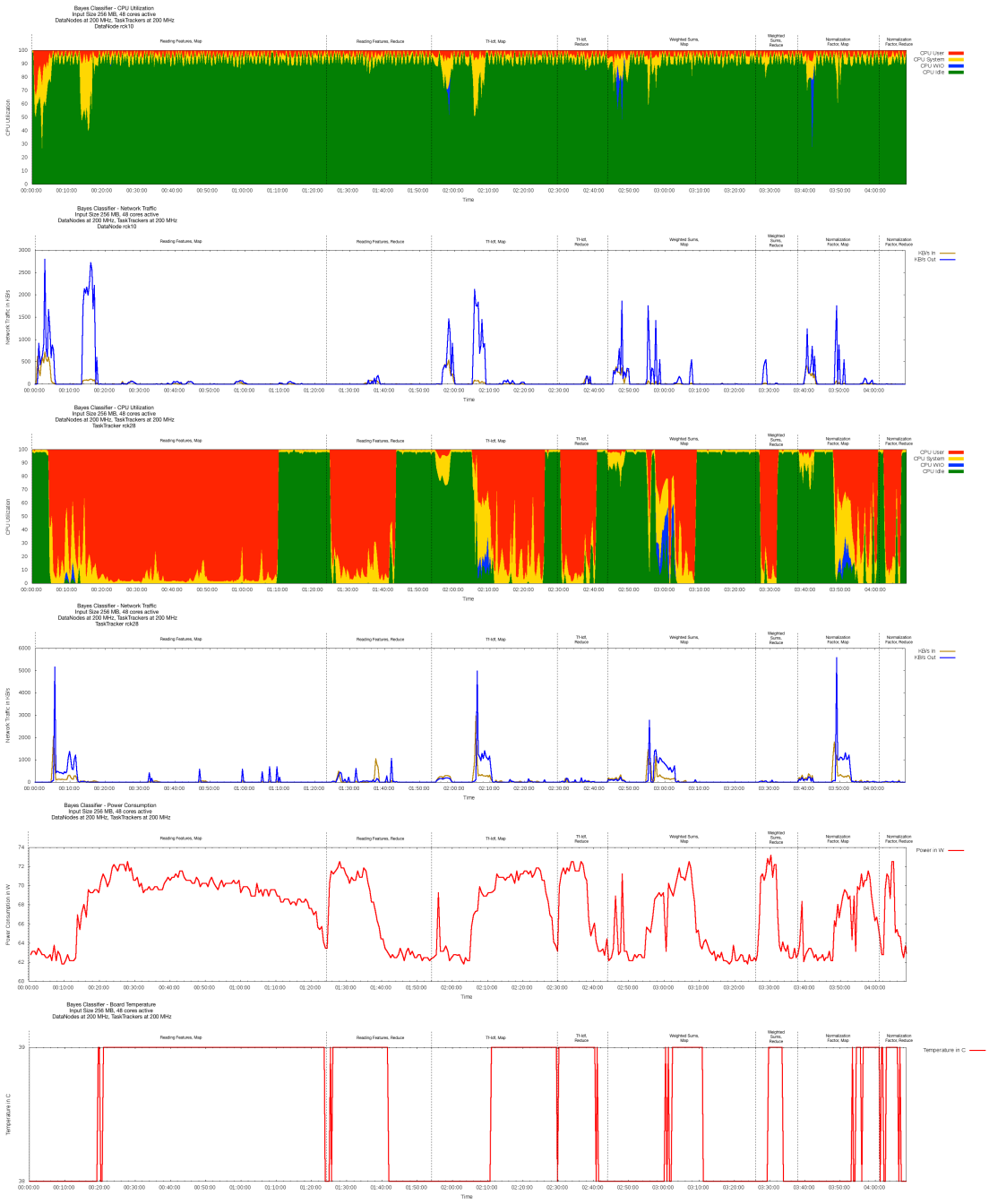
B.2.5 Input Size 256 MB, 16-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

B.2.6 Input Size 256 MB, 24-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

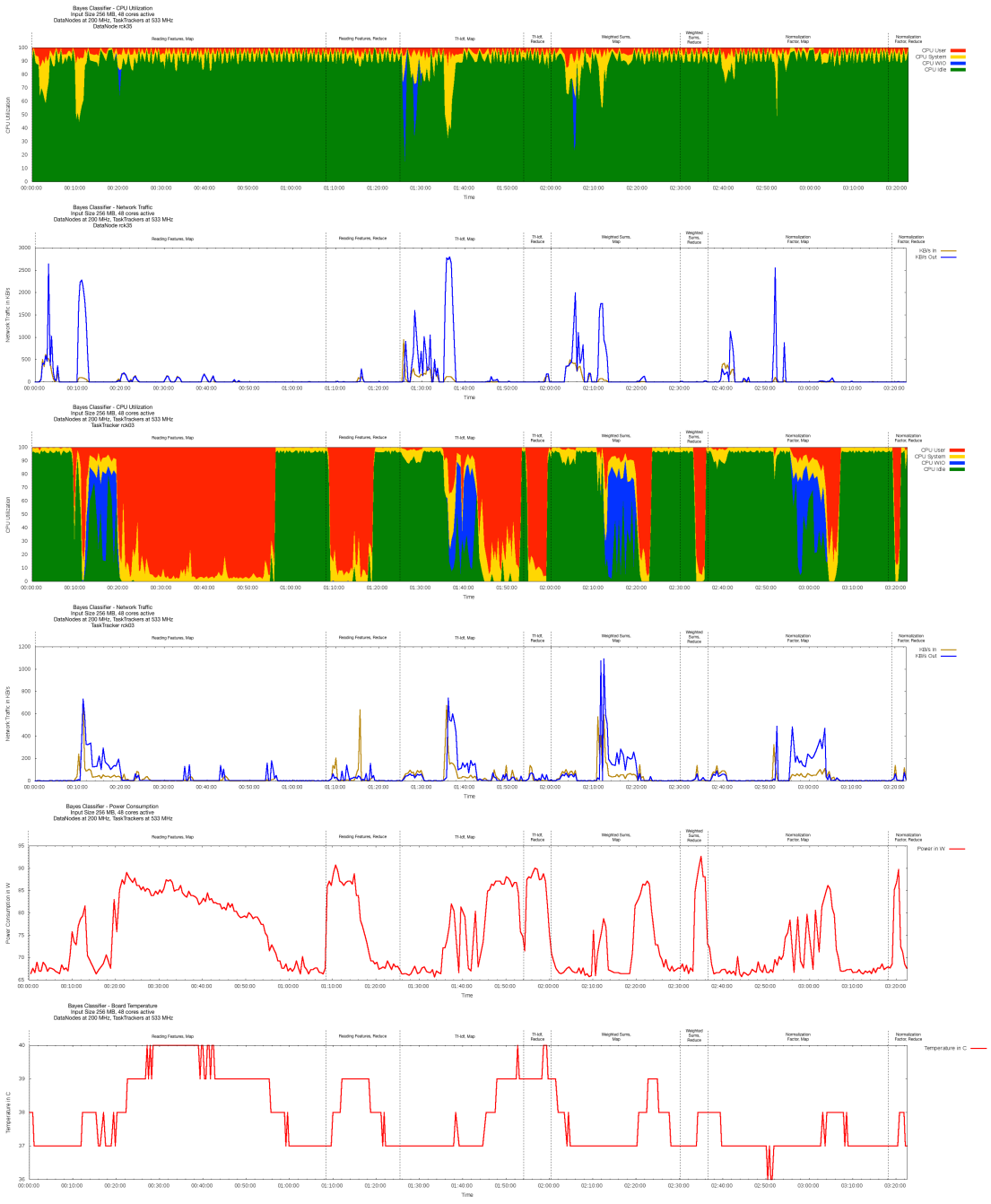


B.2.7 Input Size 256 MB, 32-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

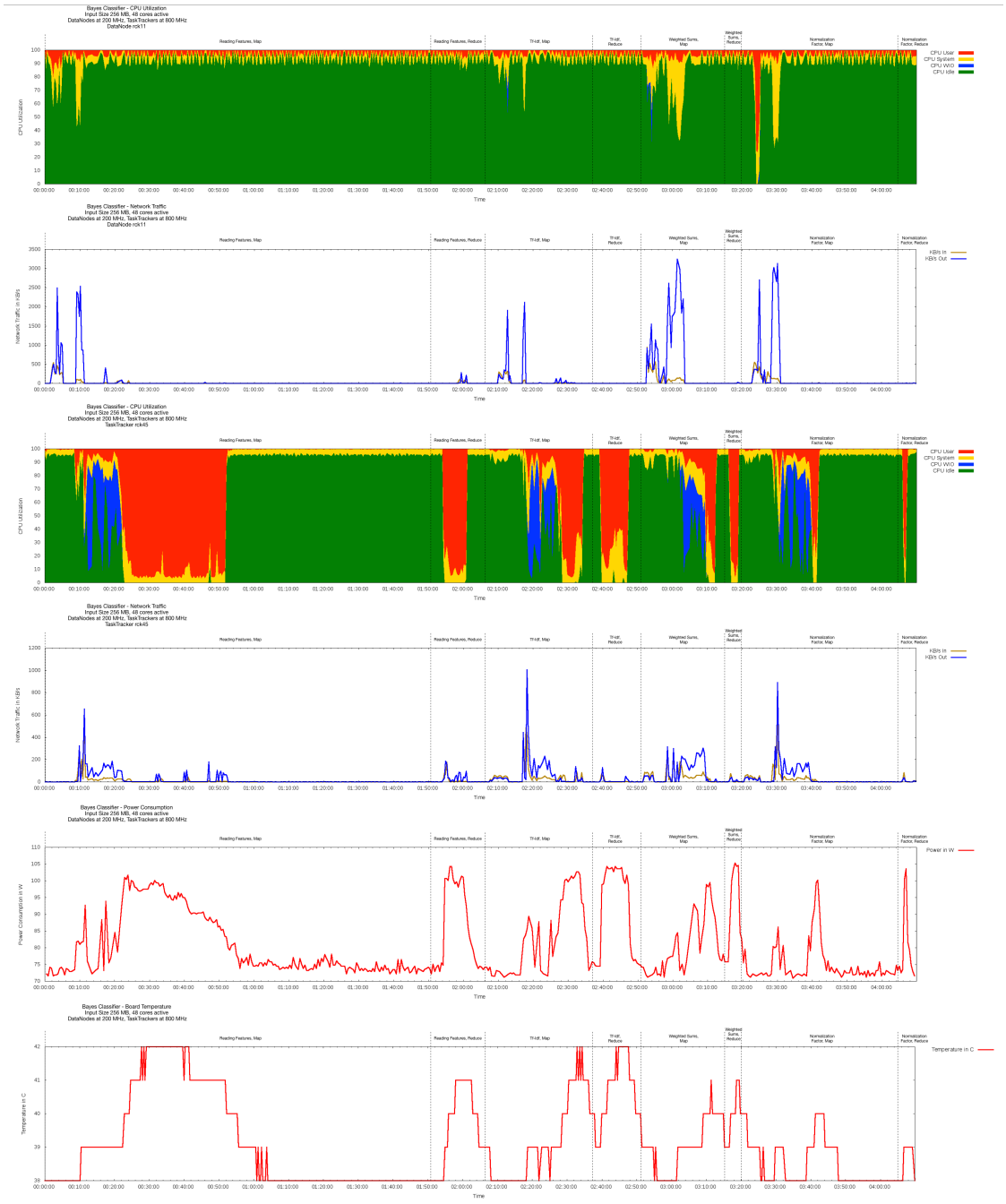
B.2.8 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 200 MHz



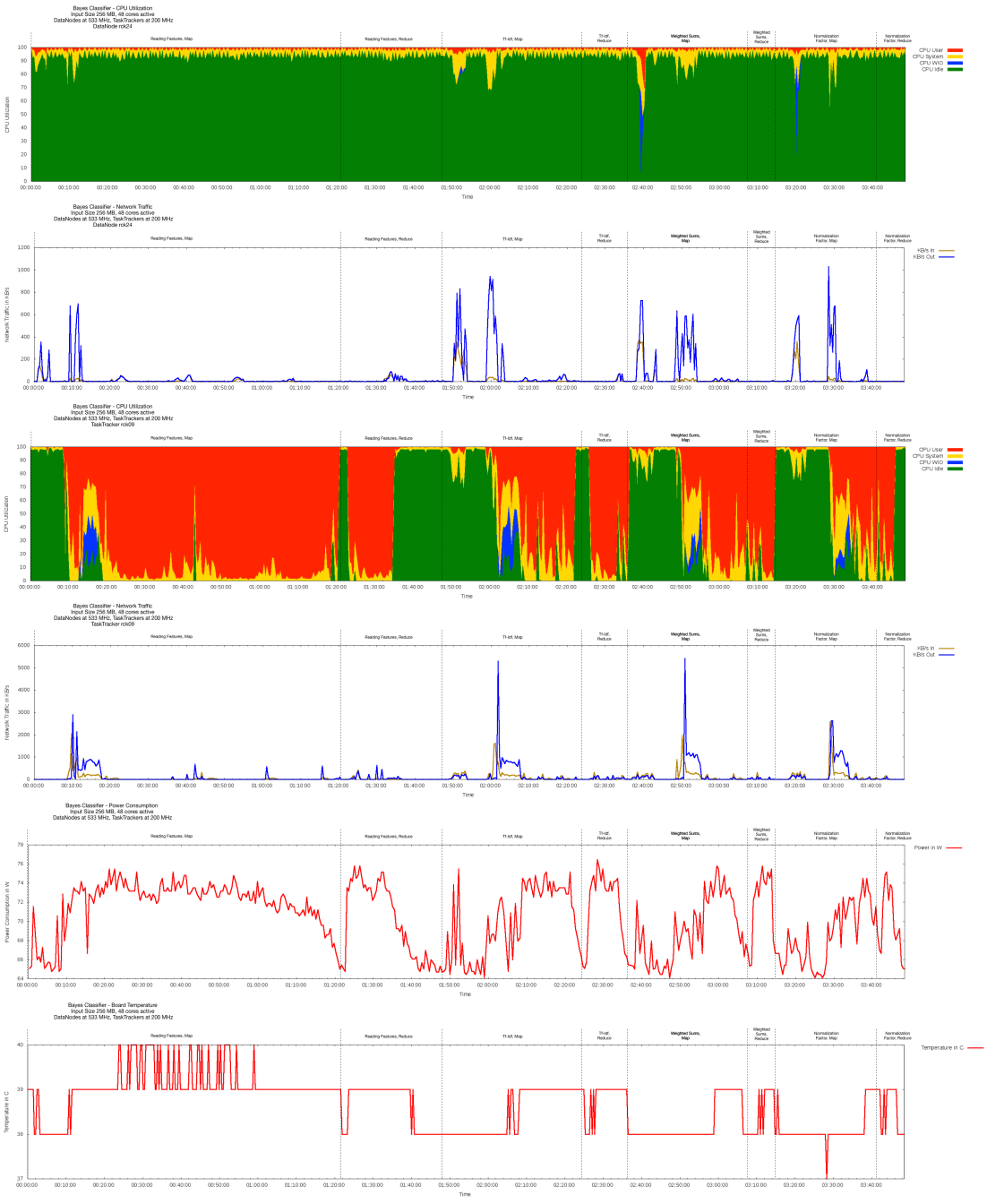
B.2.9 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 533 MHz



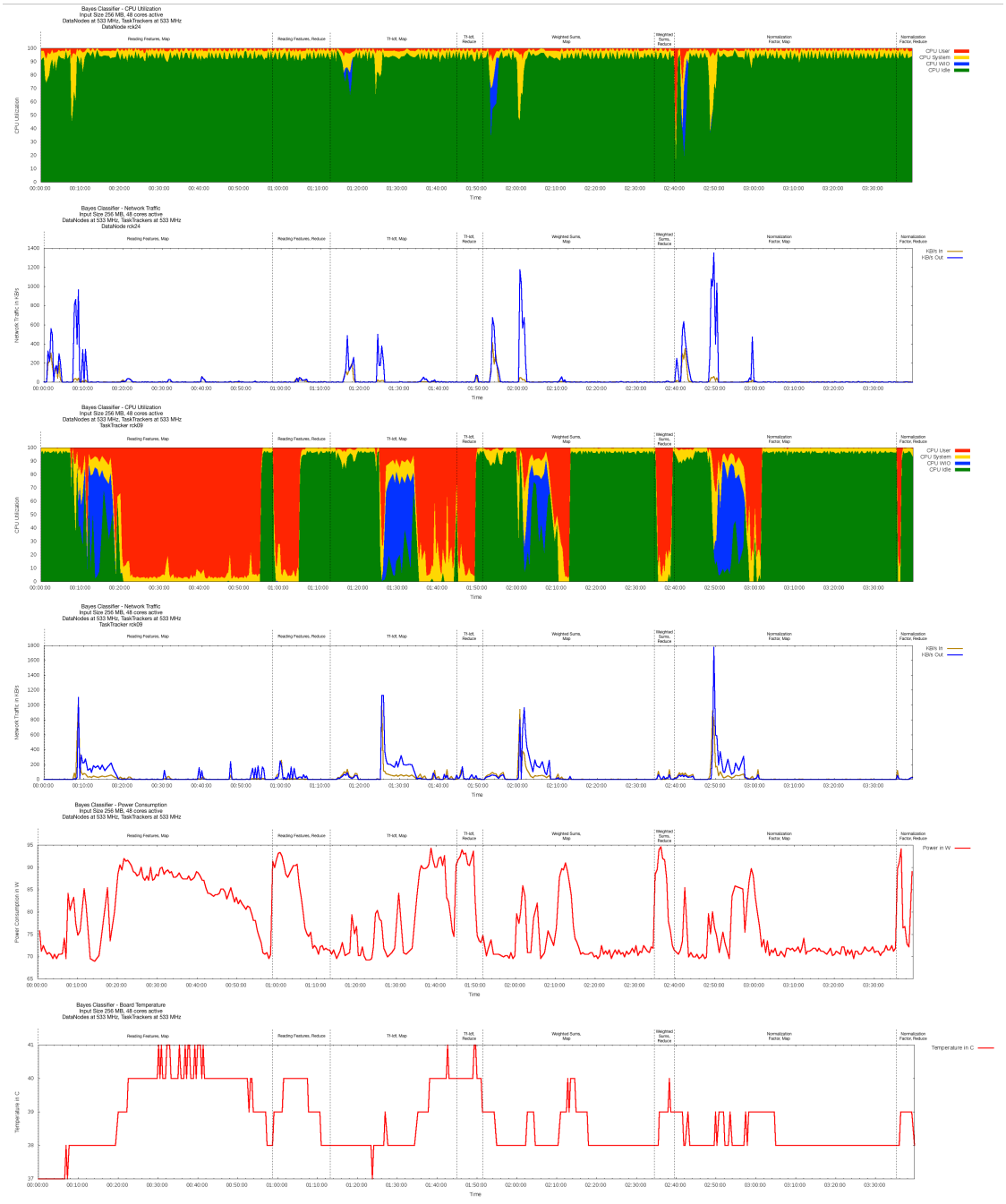
B.2.10 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 800 MHz



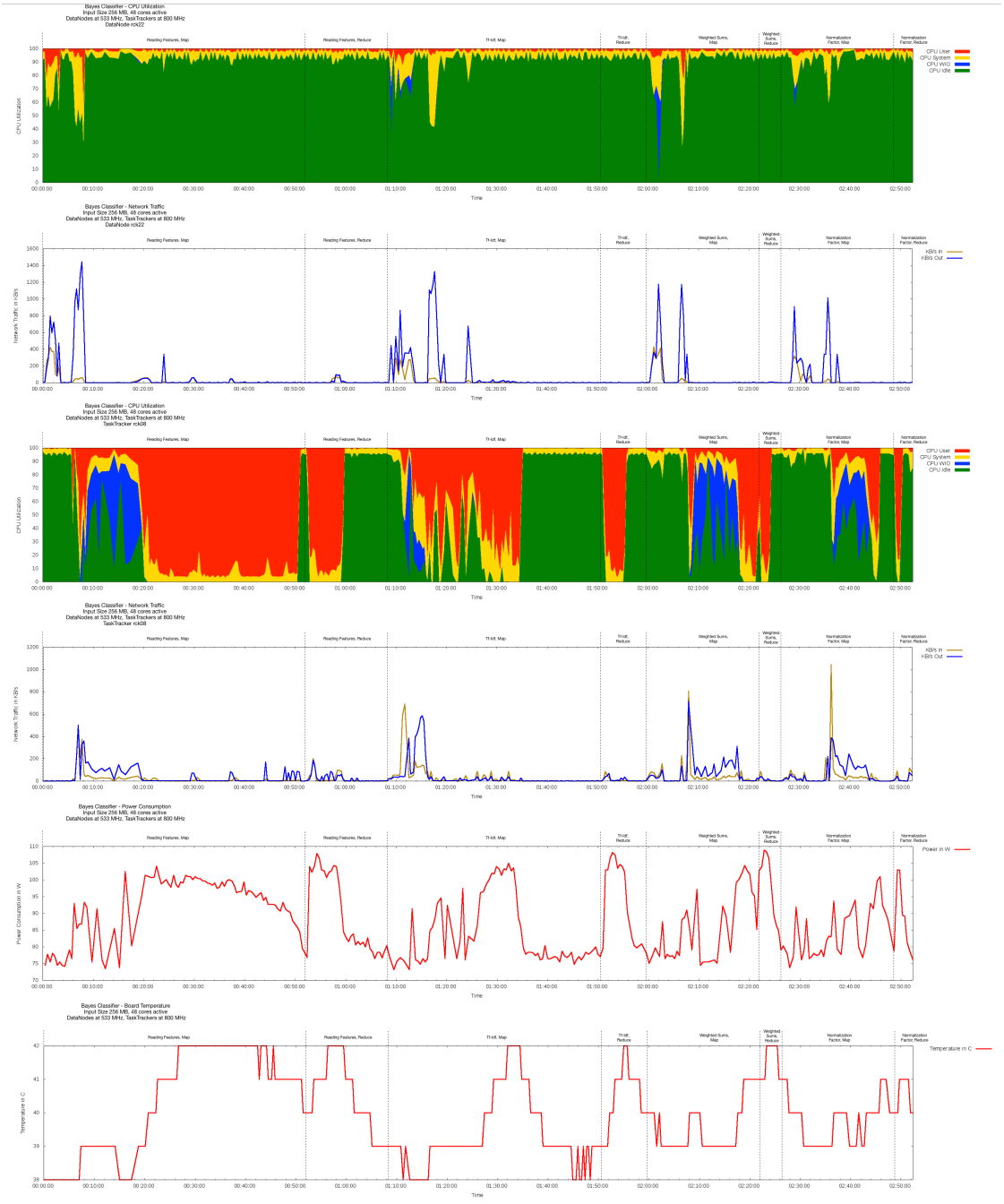
B.2.11 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 200 MHz



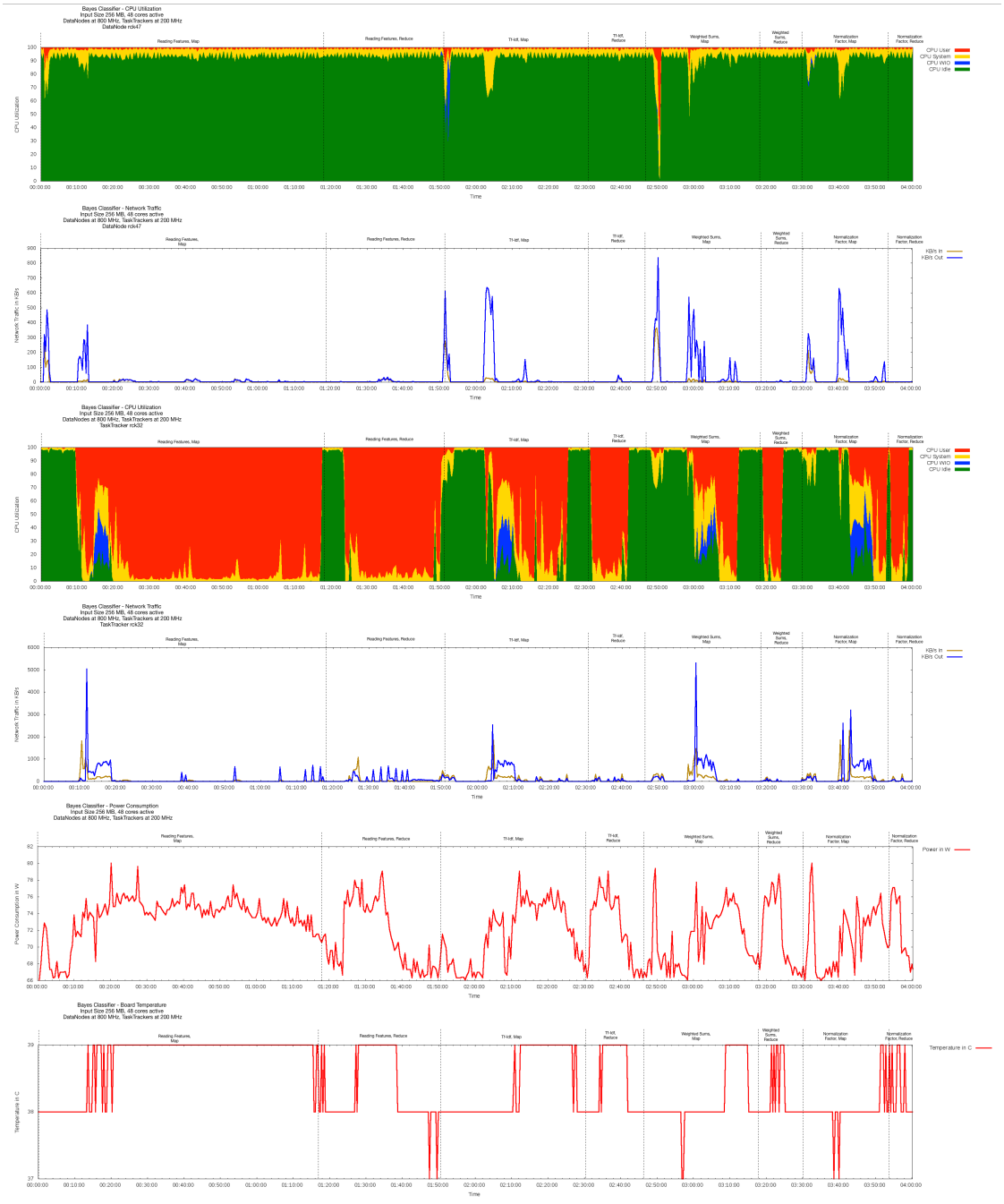
B.2.12 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 533 MHz



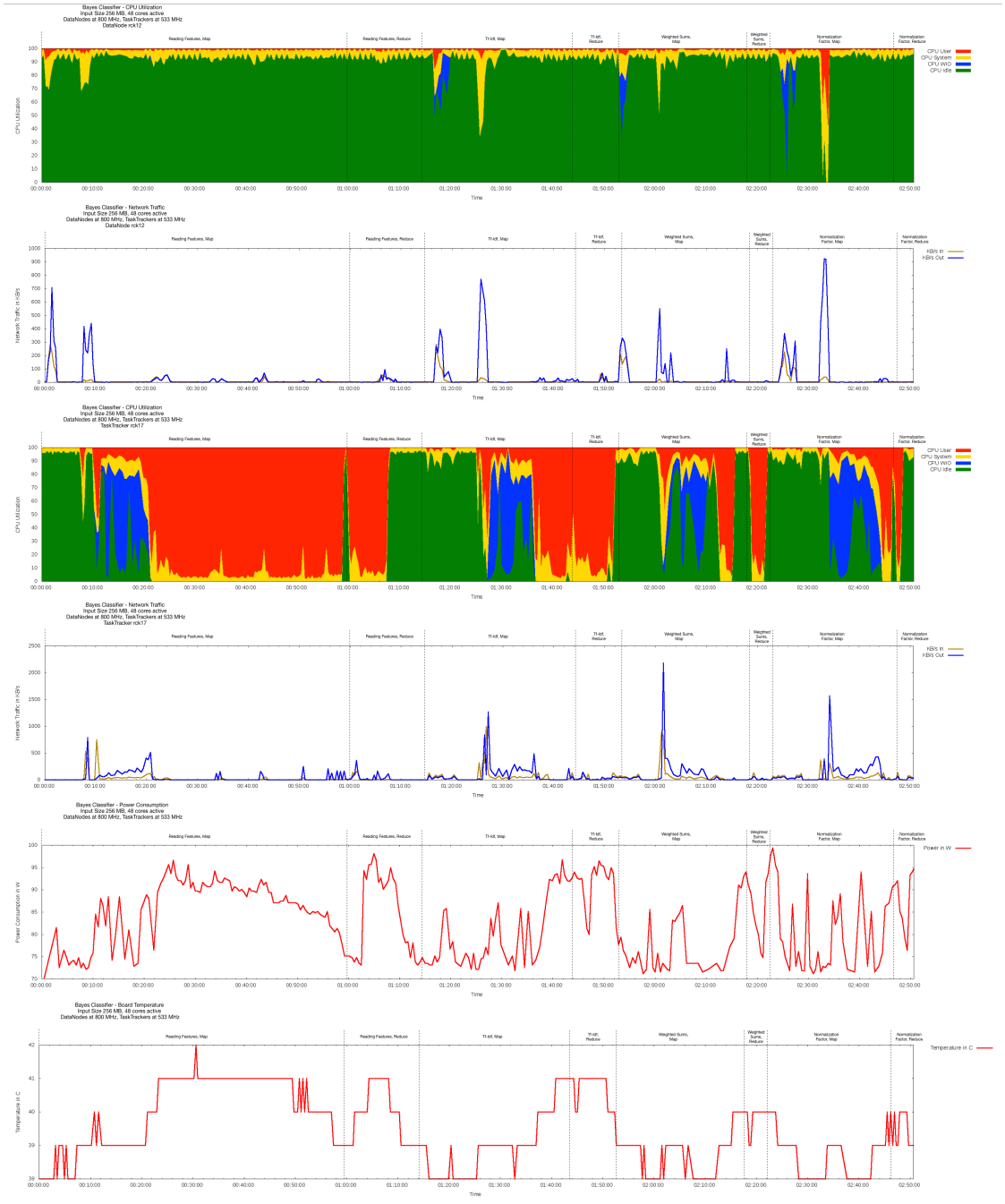
B.2.13 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 800 MHz



B.2.14 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 200 MHz



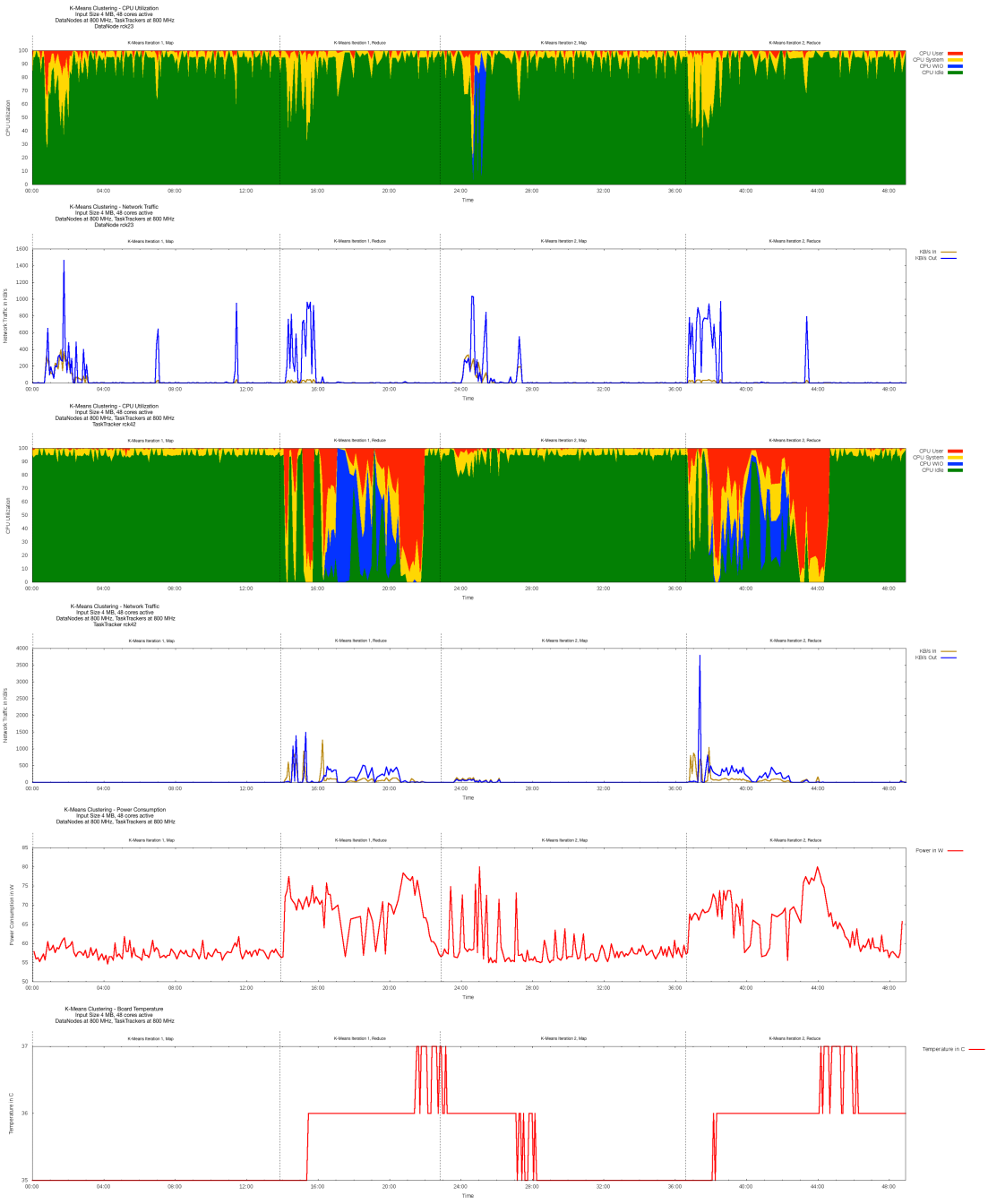
B.2.15 Input Size 256 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 533 MHz



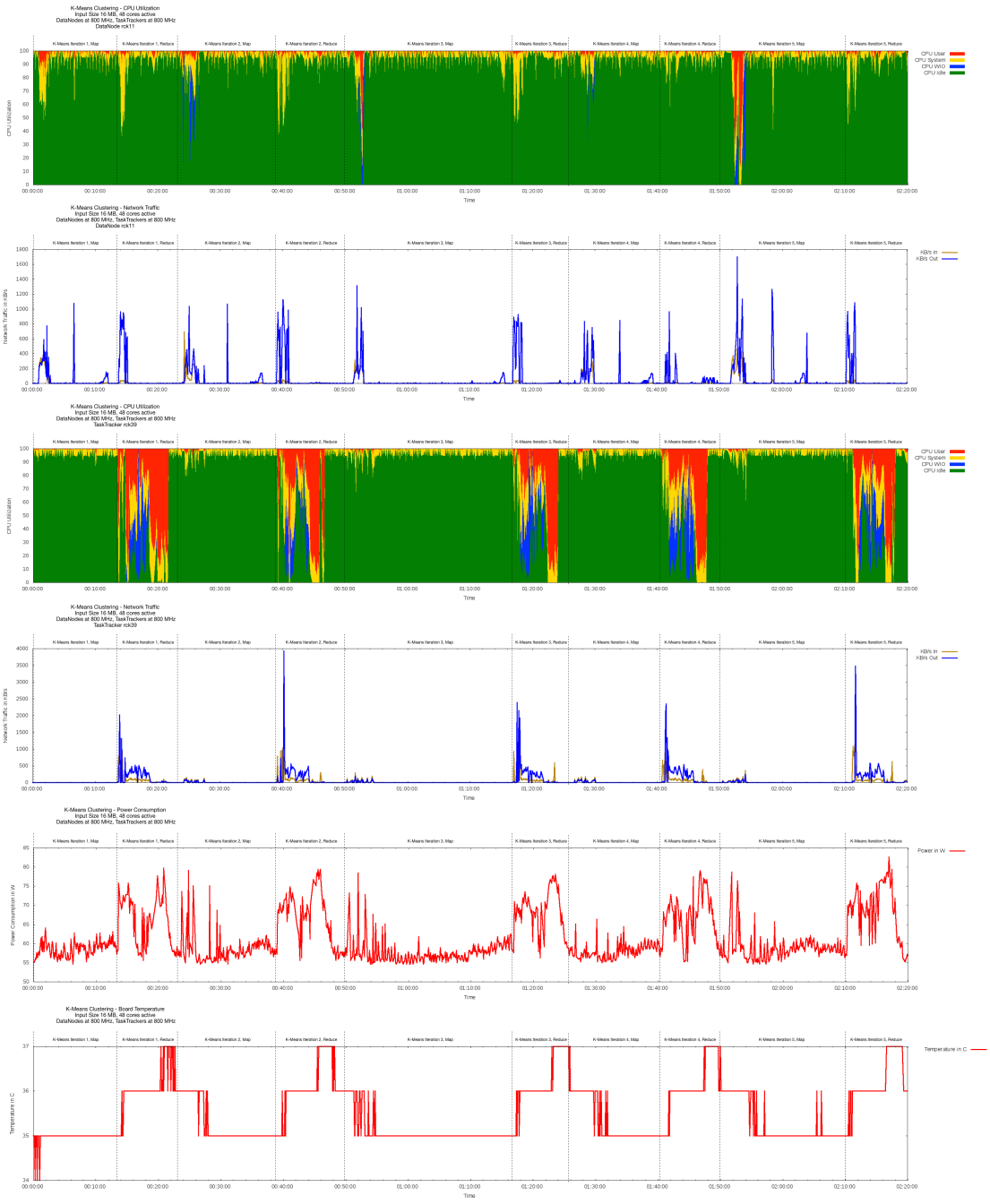
B.3 K-Means Clustering

B.3.1 Input Size 121 KB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

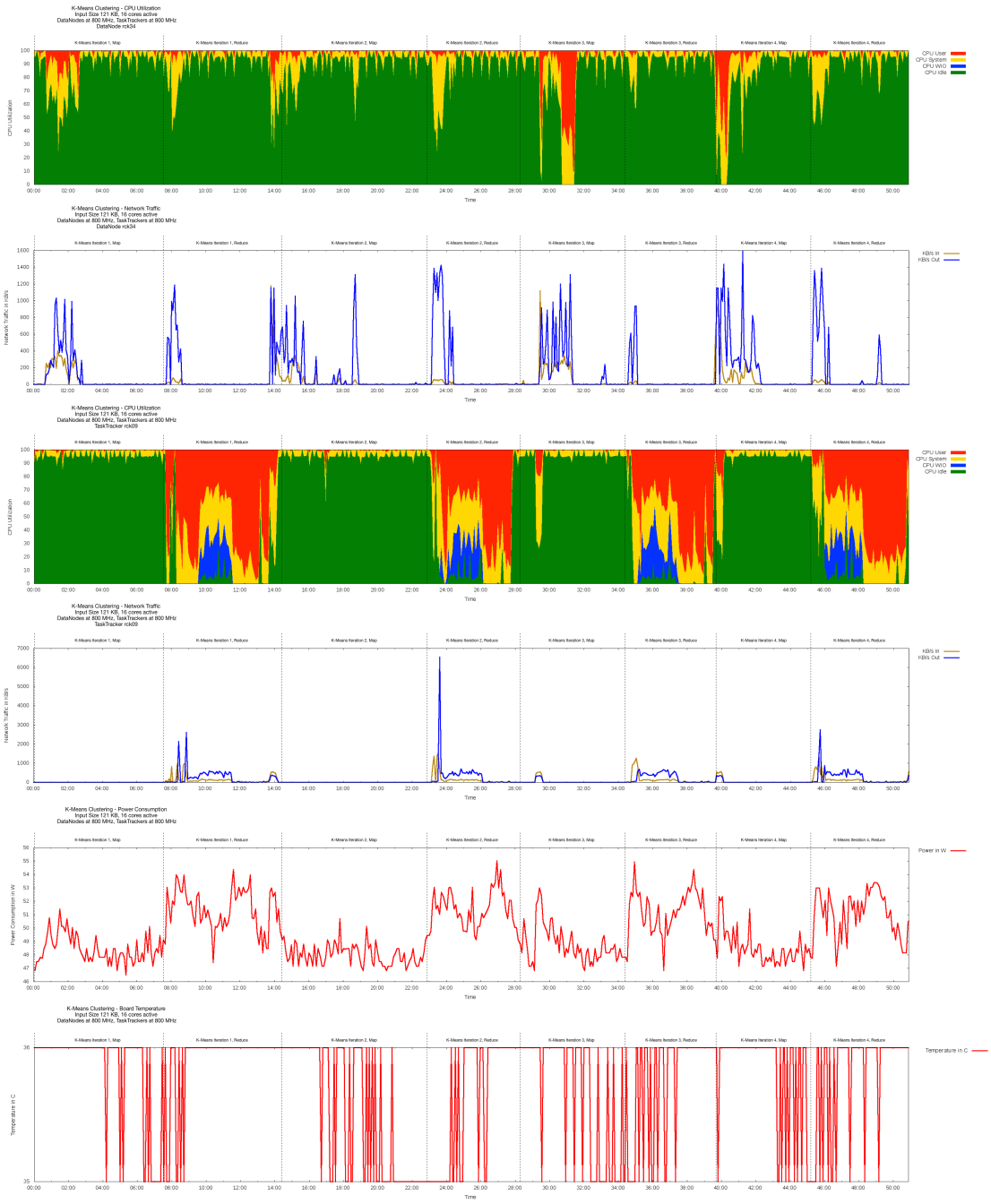
B.3.2 Input Size 4 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



B.3.3 Input Size 16 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

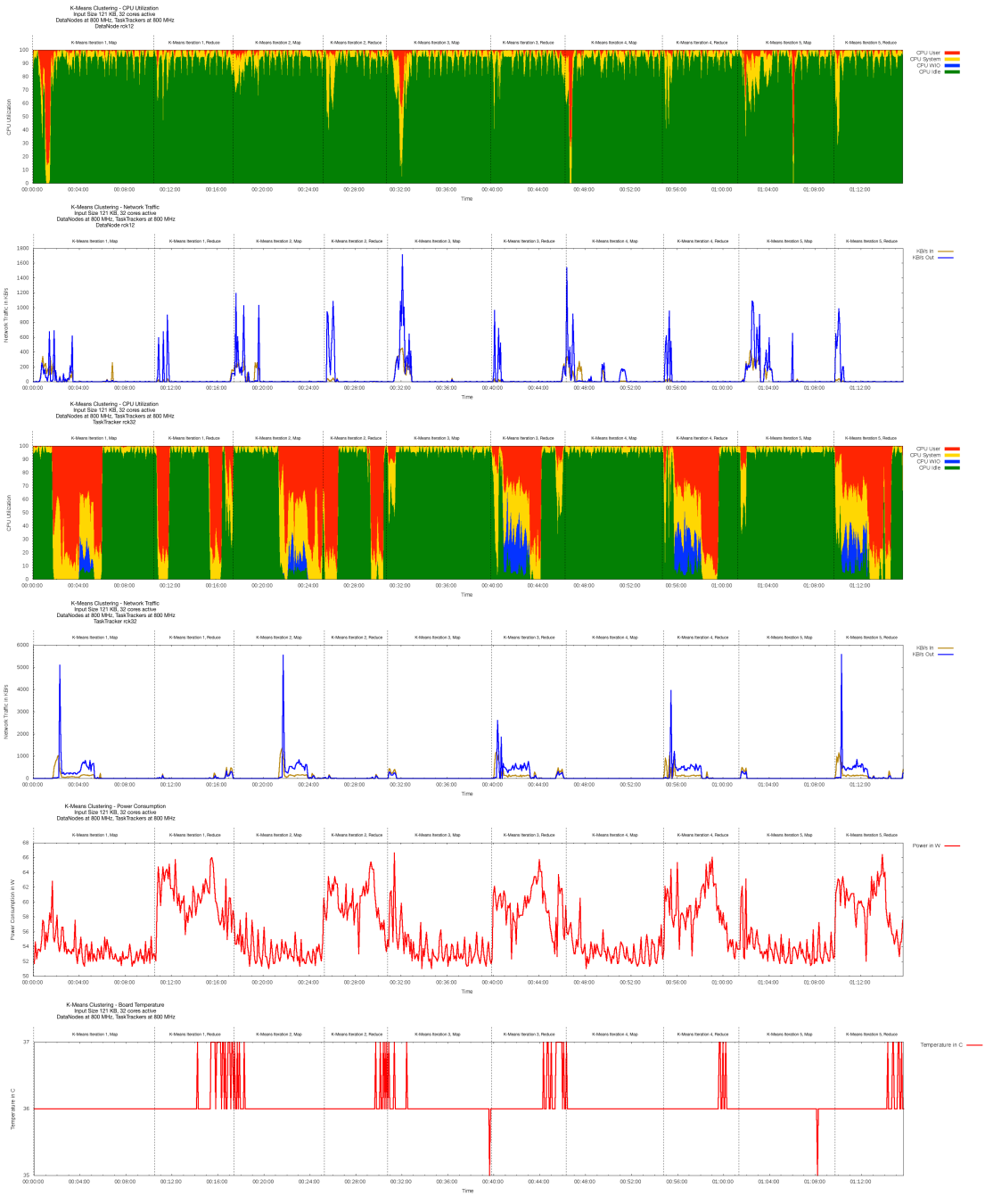


B.3.4 Input Size 121 KB, 16-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

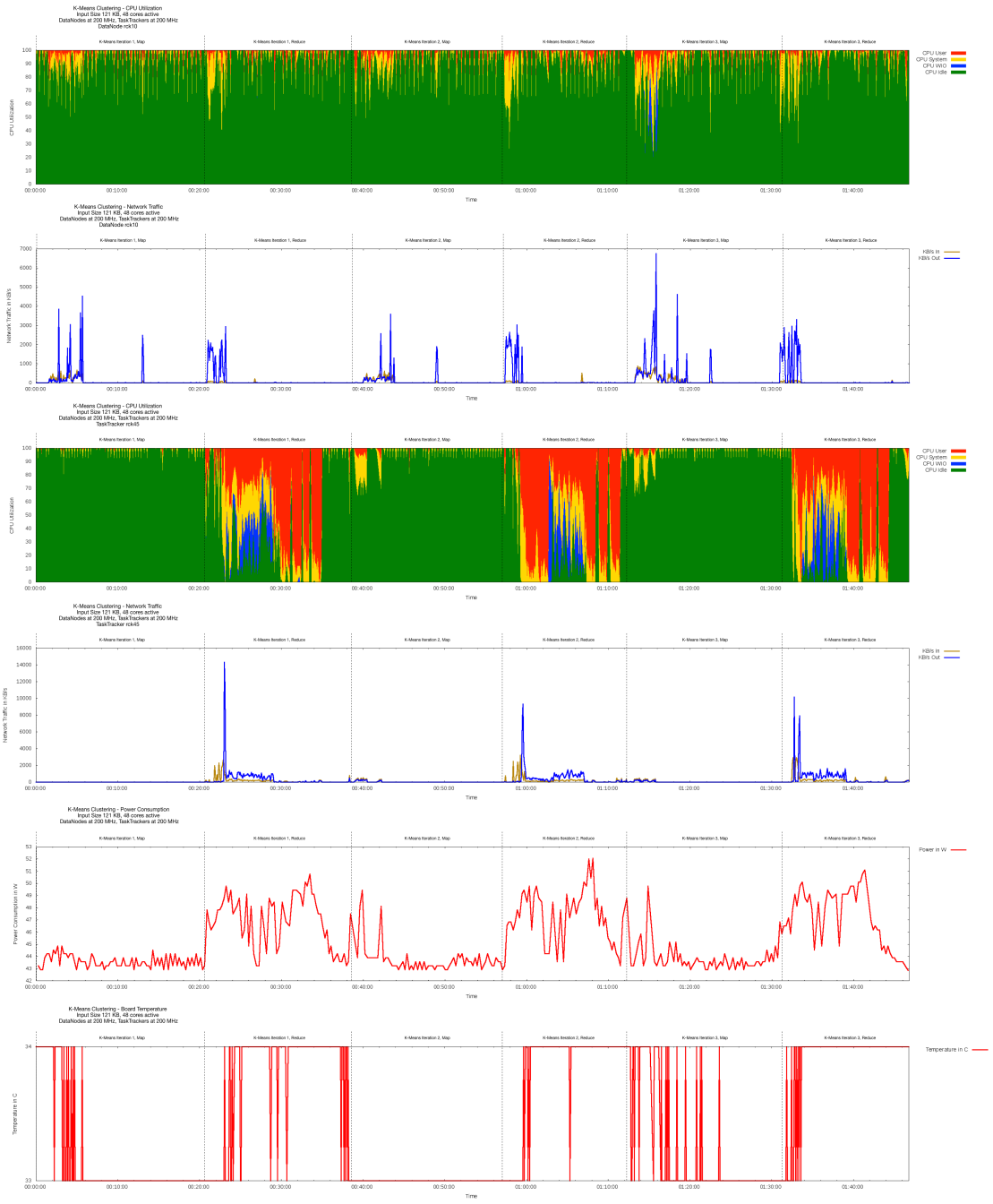


B.3.5 Input Size 121 KB, 24-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

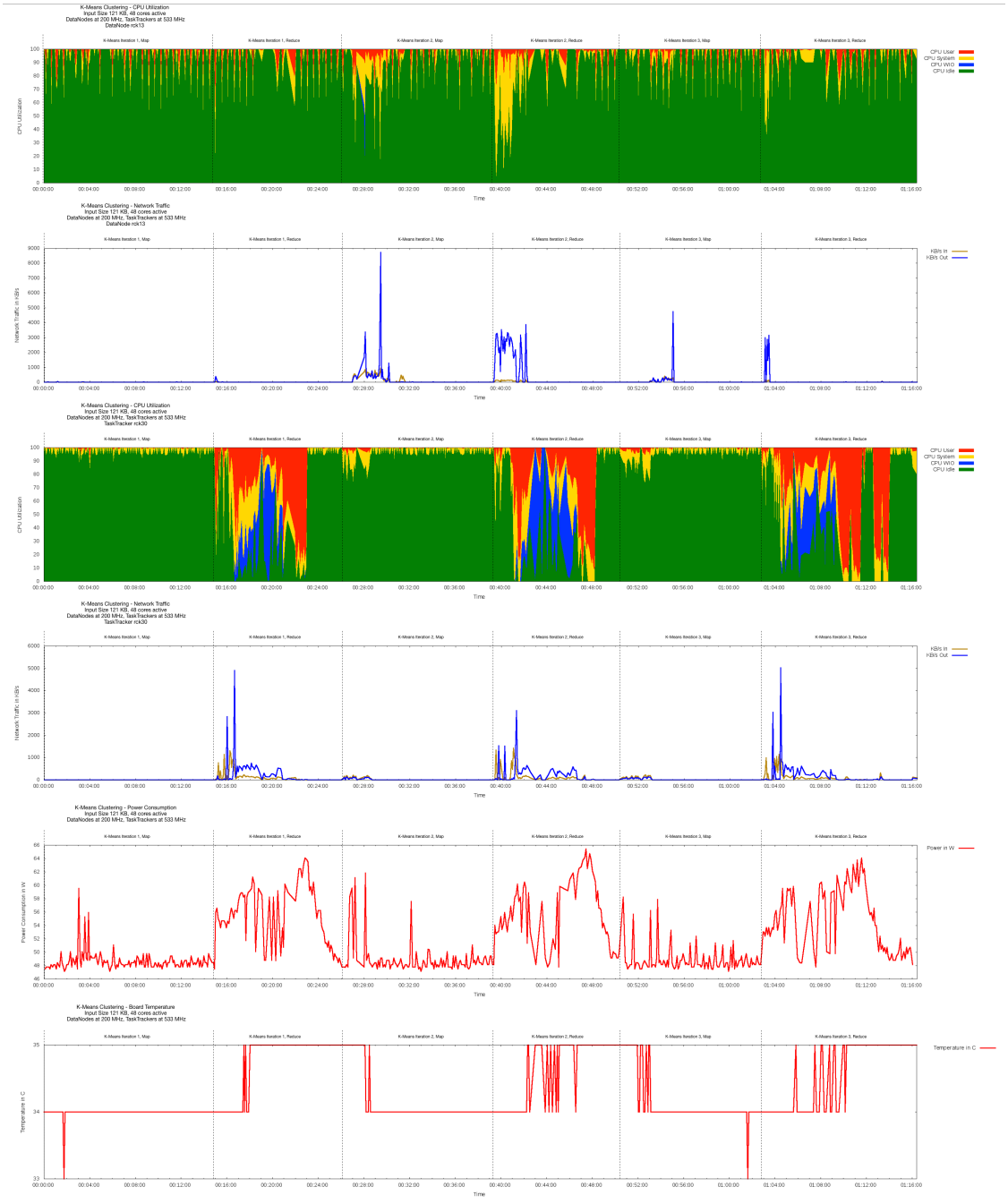
B.3.6 Input Size 121 KB, 32-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



B.3.7 Input Size 121 KB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 200 MHz

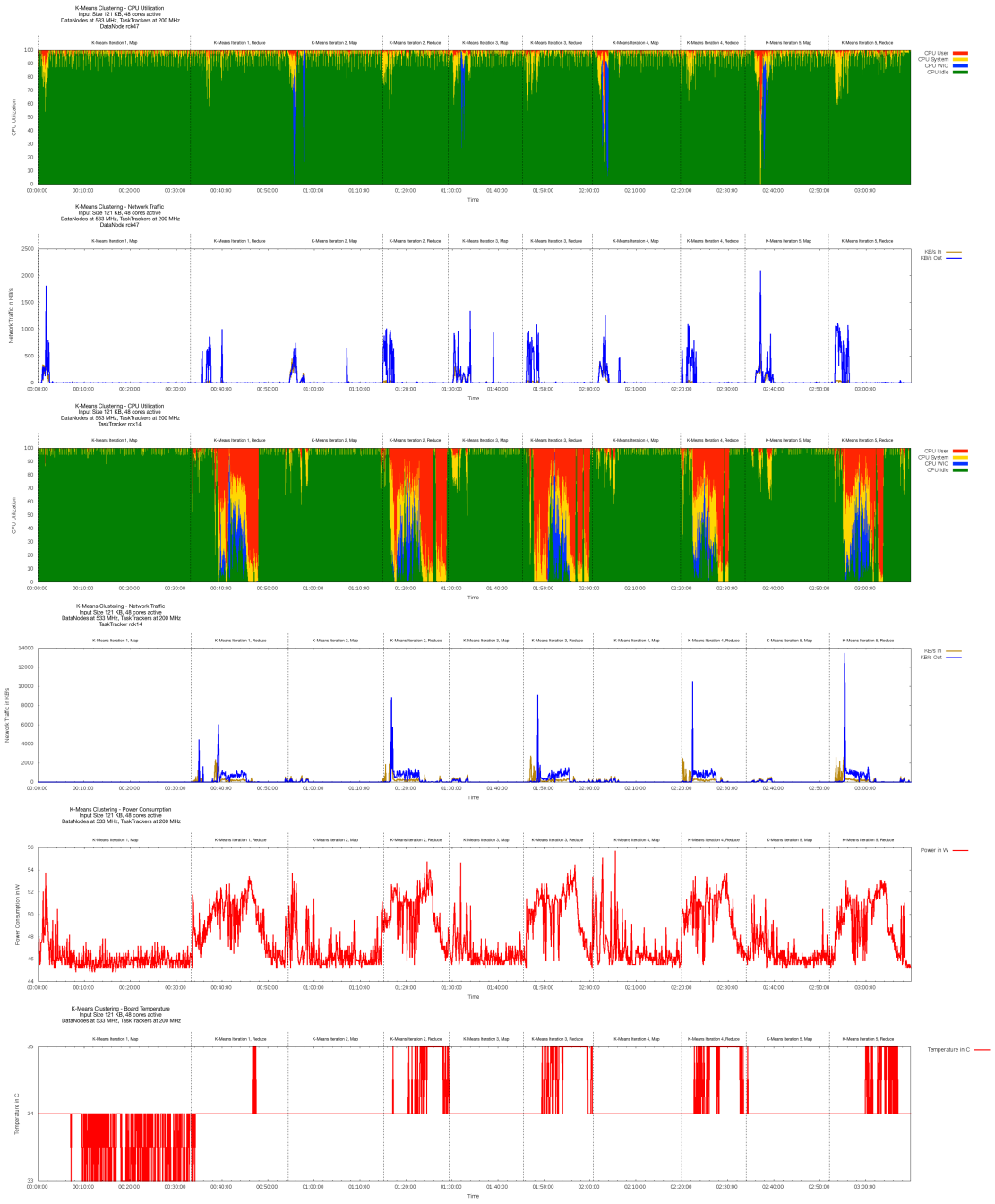


B.3.8 Input Size 121 KB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 533 MHz

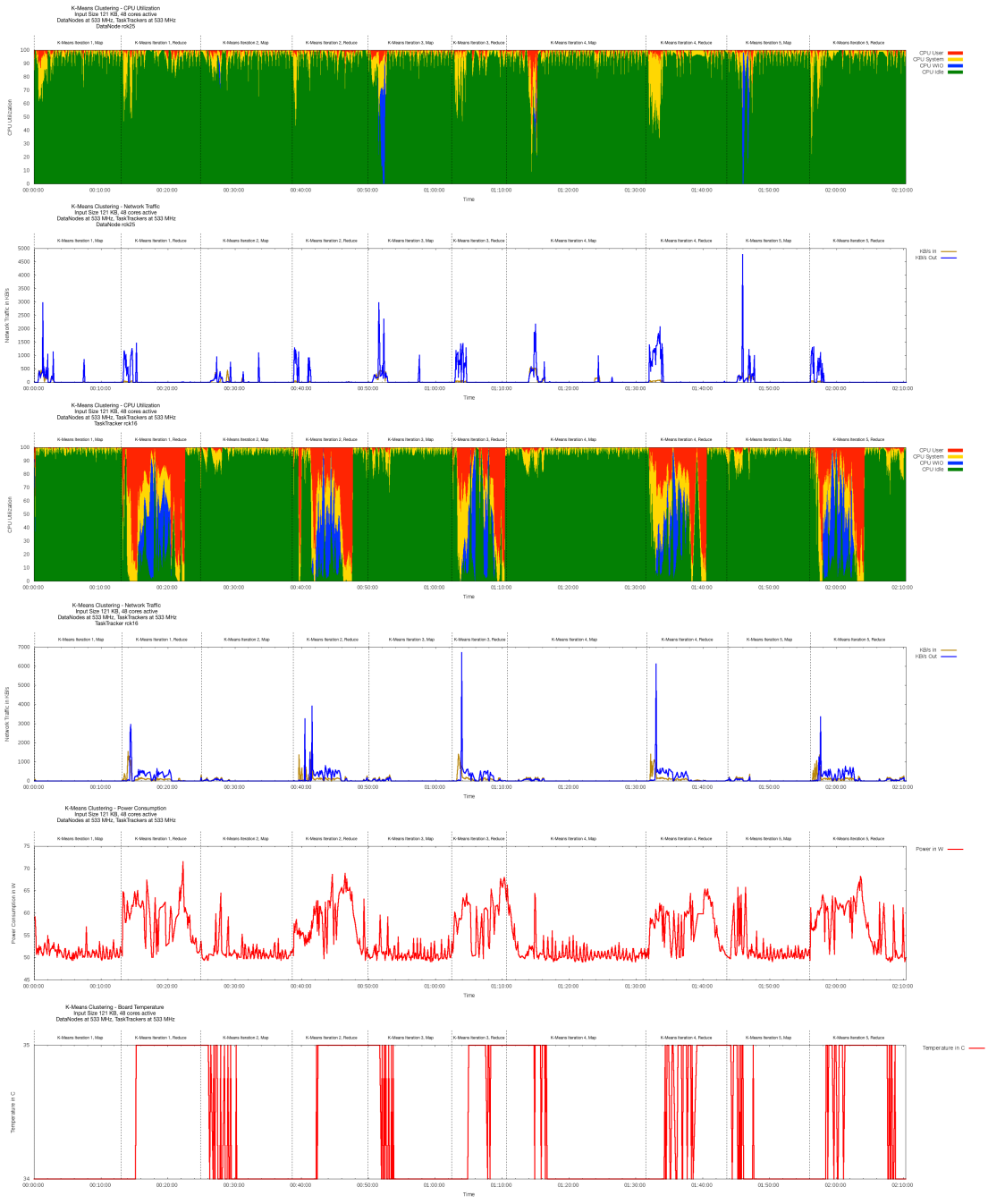


B.3.9 Input Size 121 KB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 800 MHz

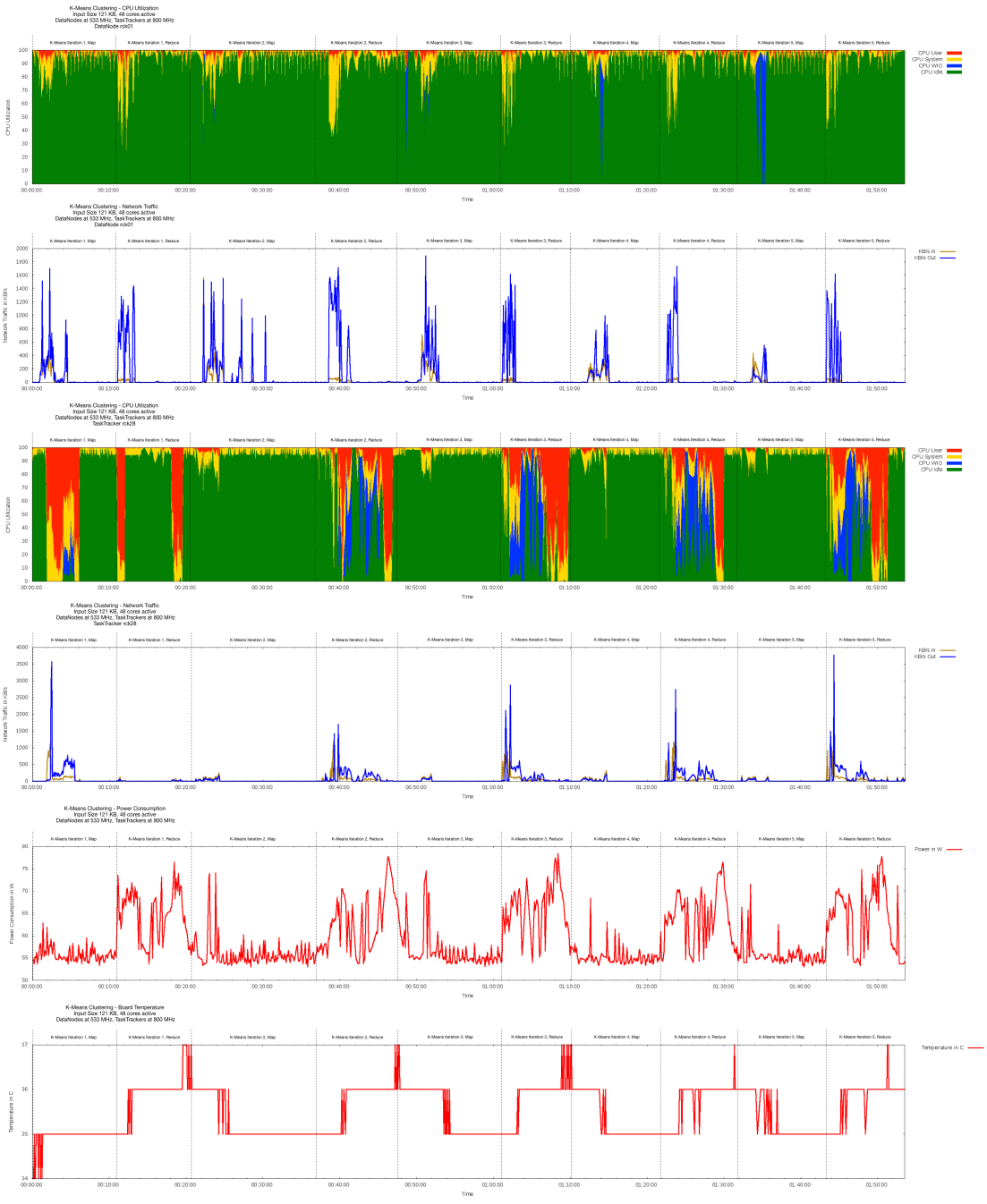
B.3.10 Input Size 121 KB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 200 MHz



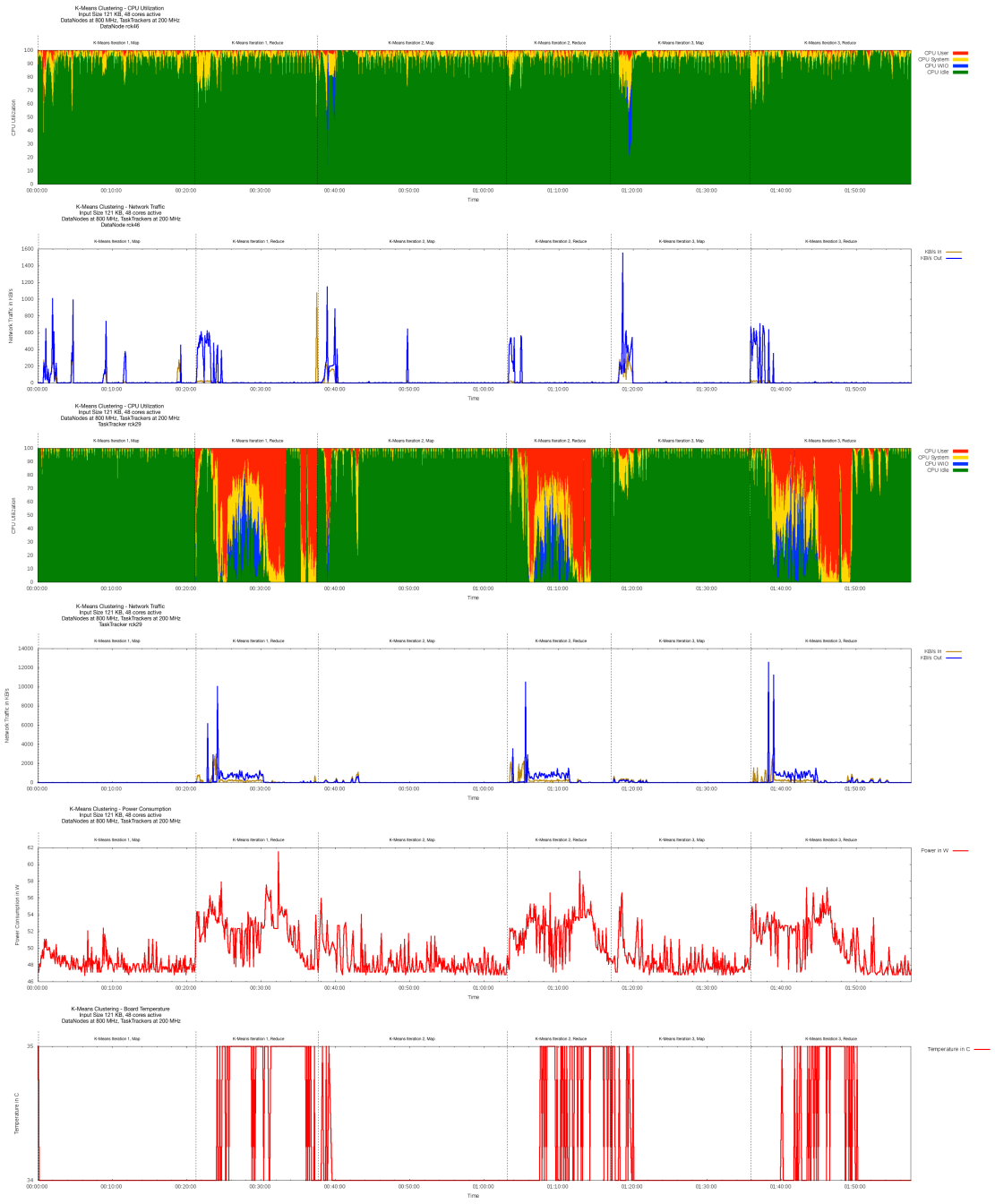
B.3.11 Input Size 121 KB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 533 MHz



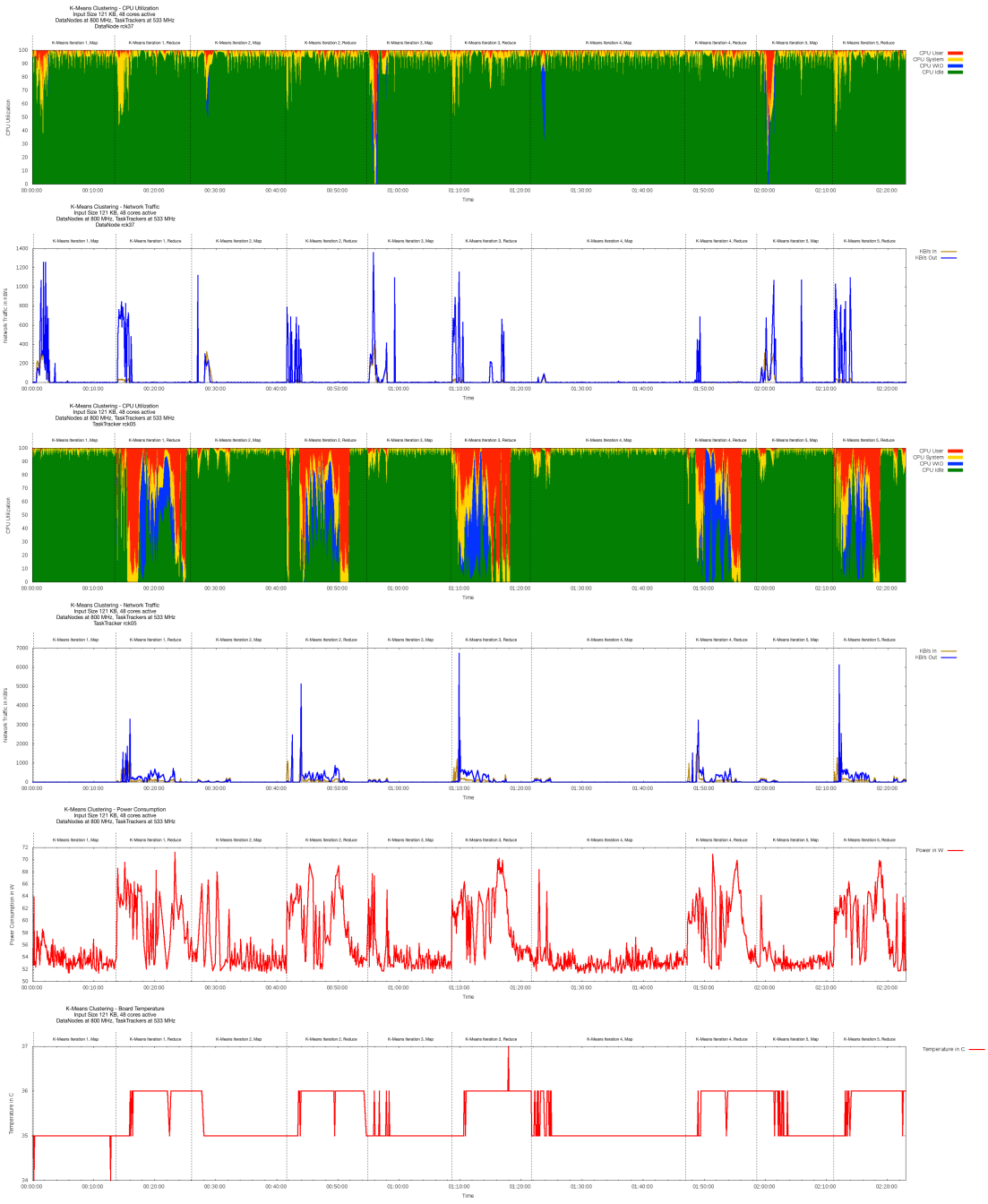
B.3.12 Input Size 121 KB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 800 MHz



B.3.13 Input Size 121 KB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 200 MHz

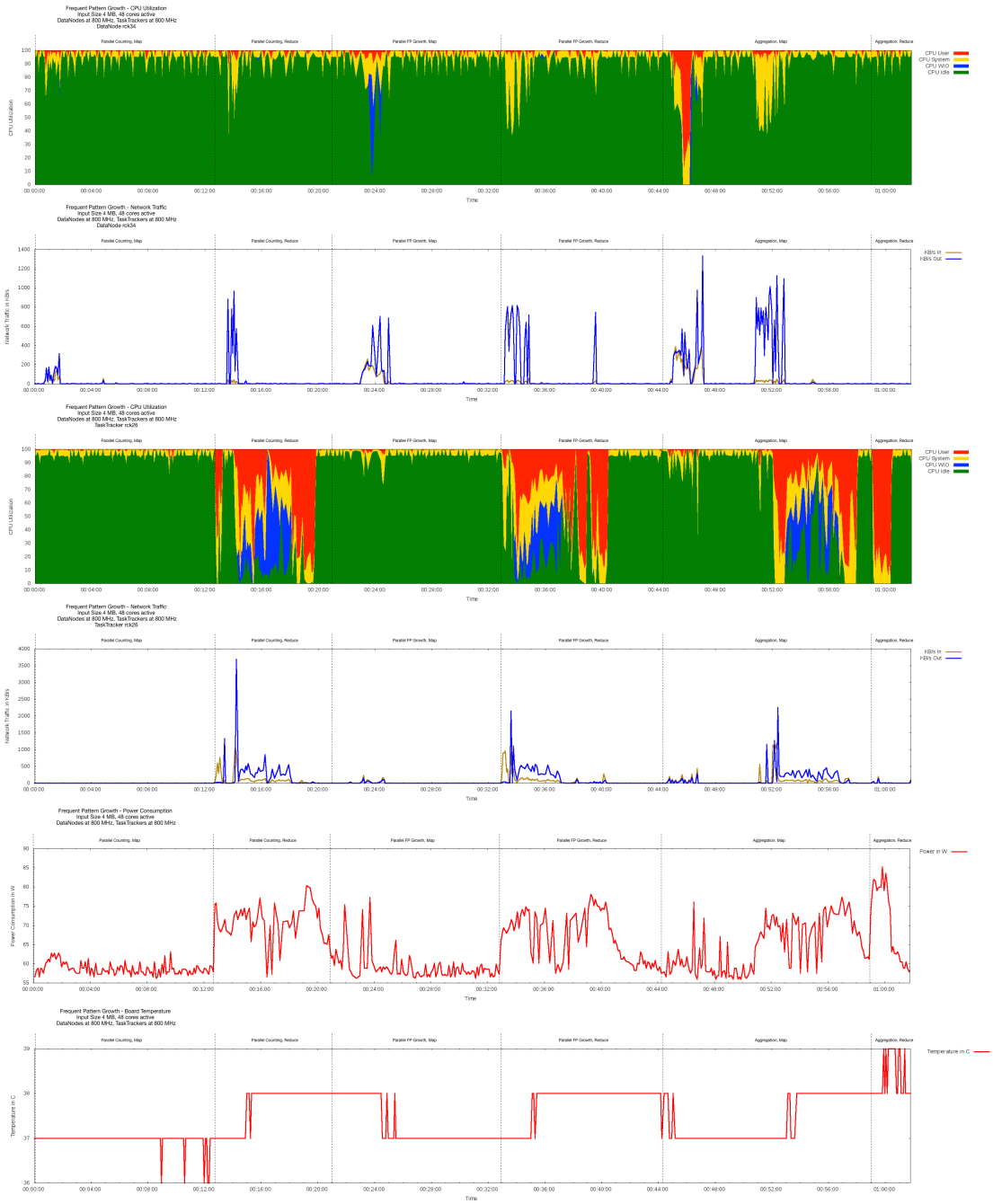


B.3.14 Input Size 121 KB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 533 MHz



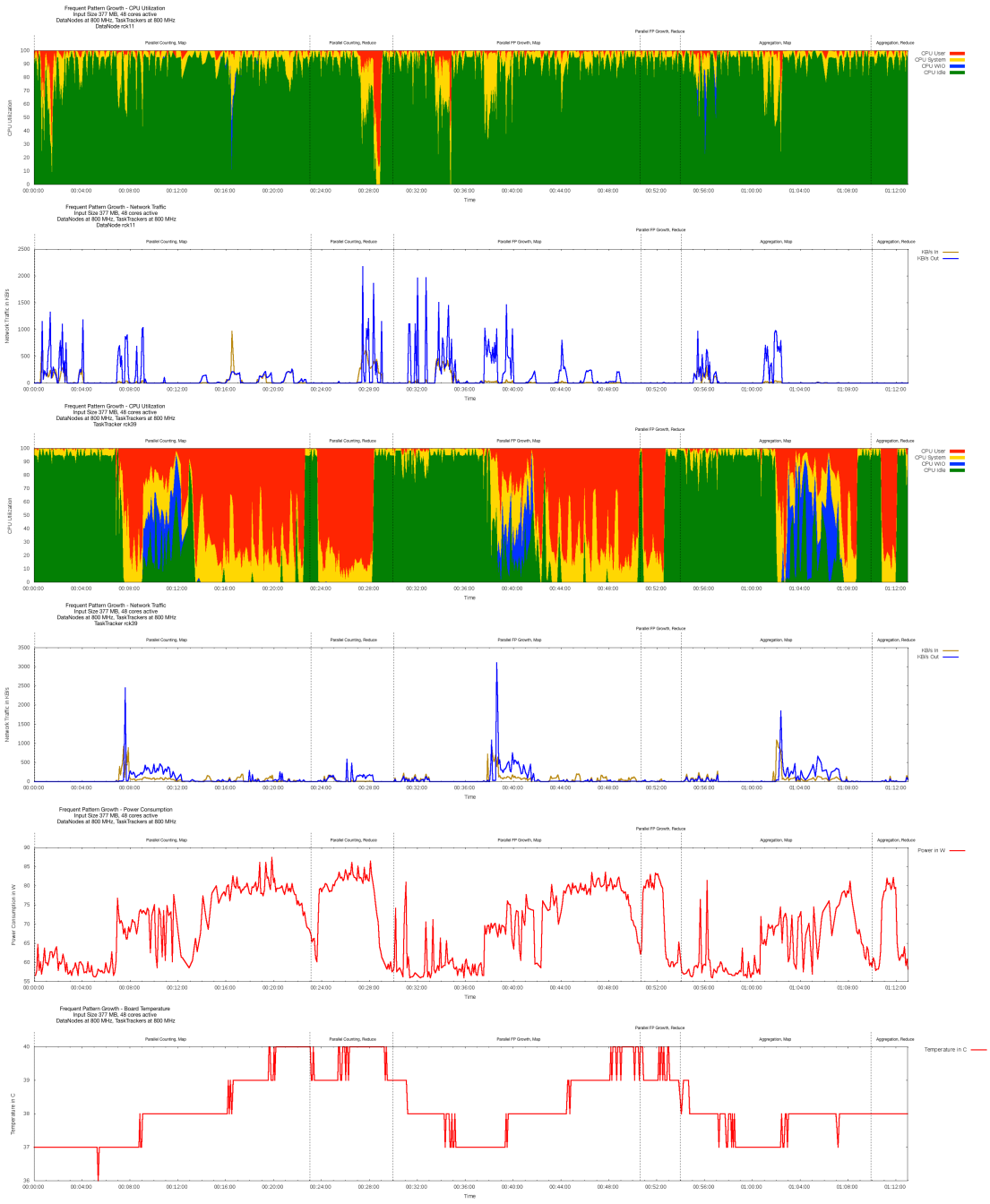
B.4 Frequent Pattern Growth

B.4.1 Input Size 4 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



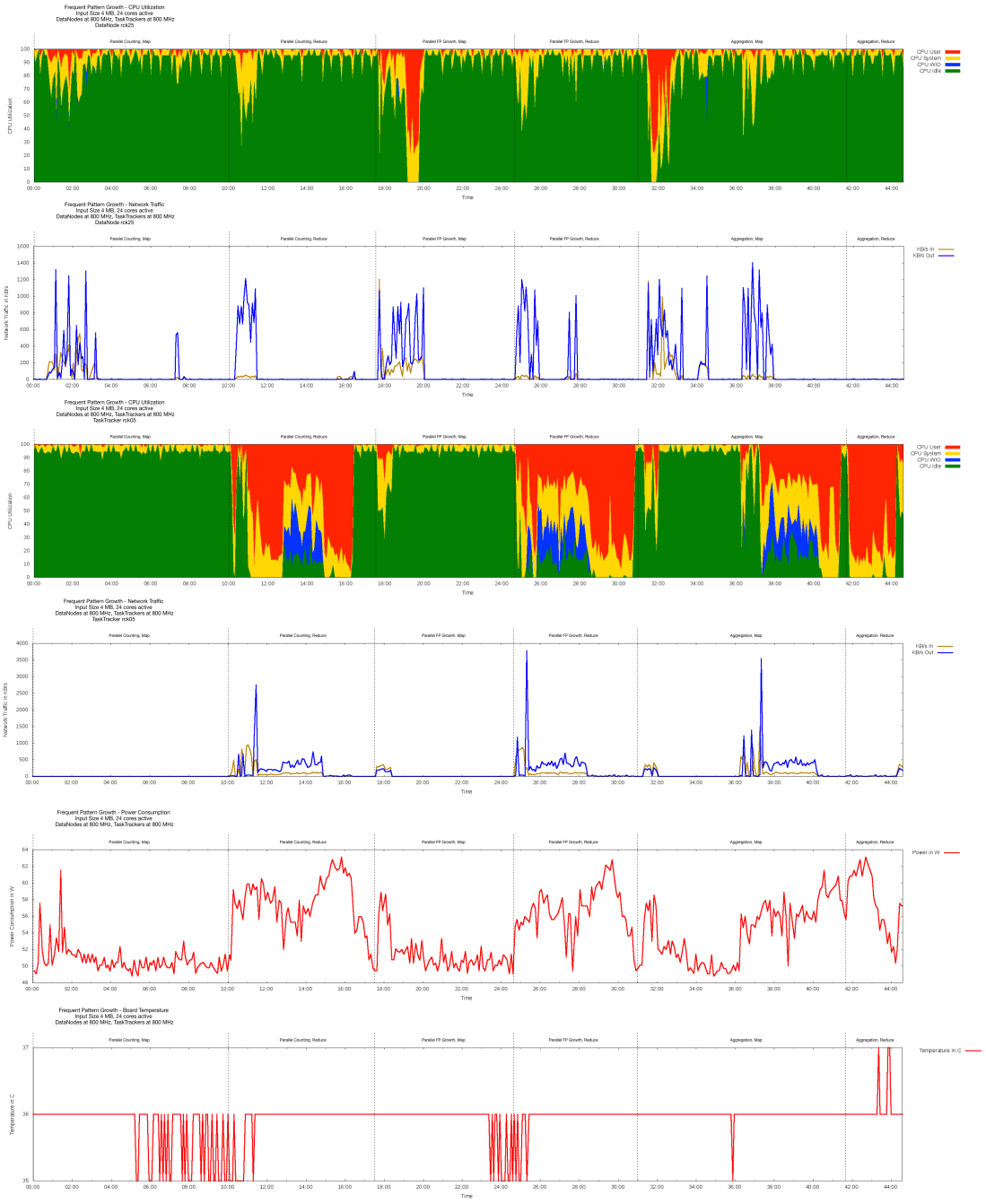
B.4.2 Input Size 34 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

B.4.3 Input Size 377 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

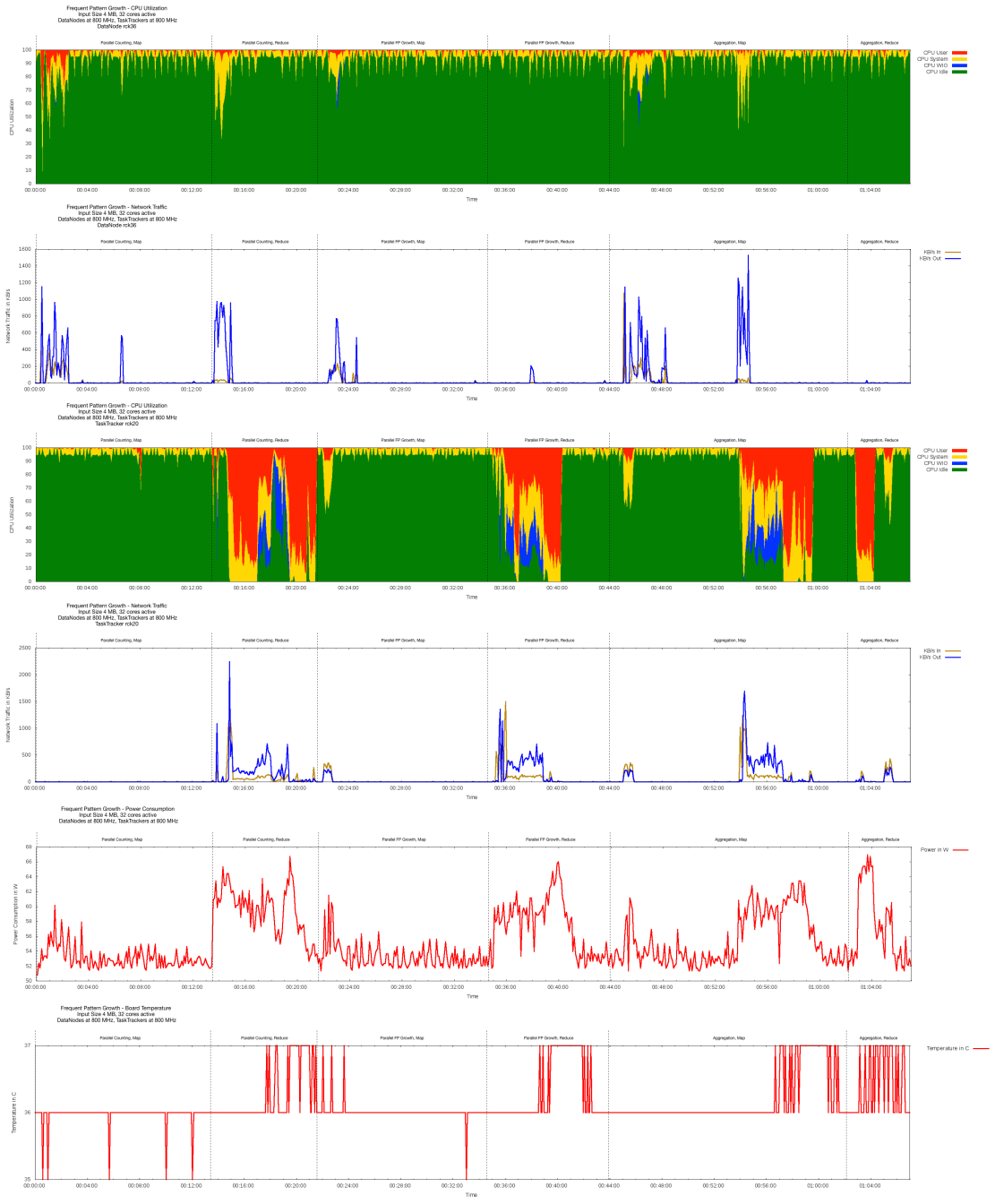


B.4.4 Input Size 4 MB, 16-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

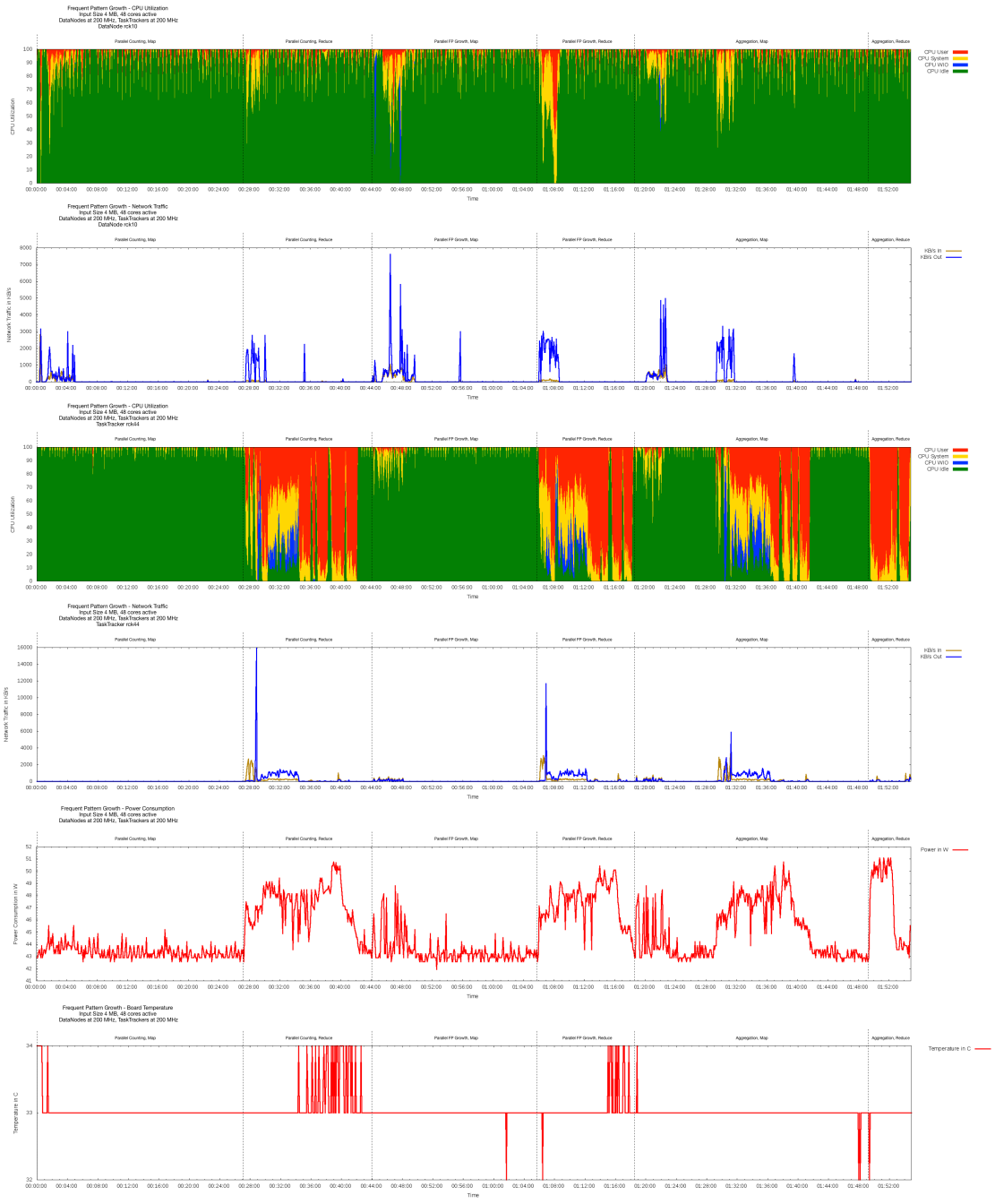
B.4.5 Input Size 4 MB, 24-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz



B.4.6 Input Size 4 MB, 32-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 800 MHz

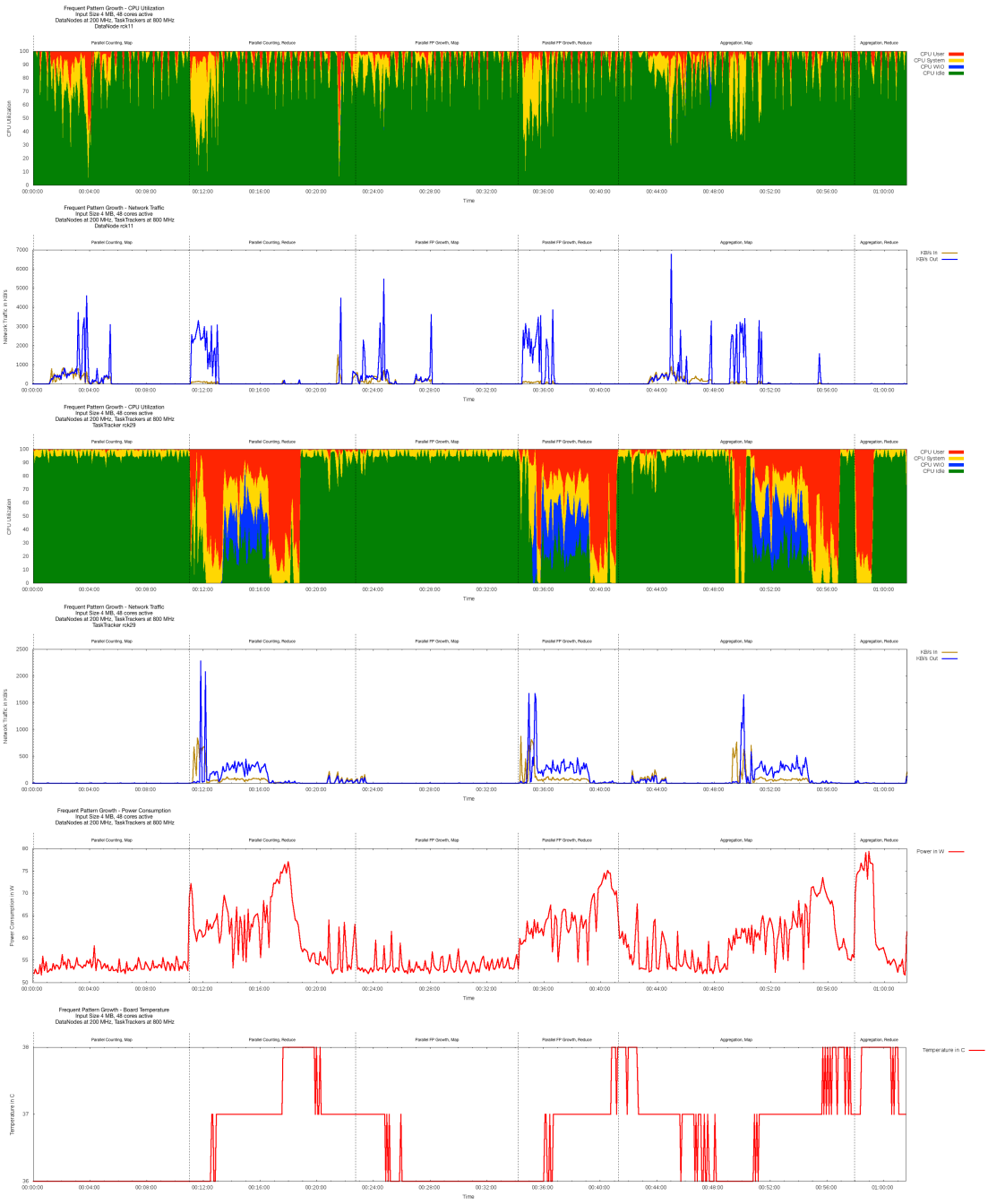


B.4.7 Input Size 4 MB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 200 MHz

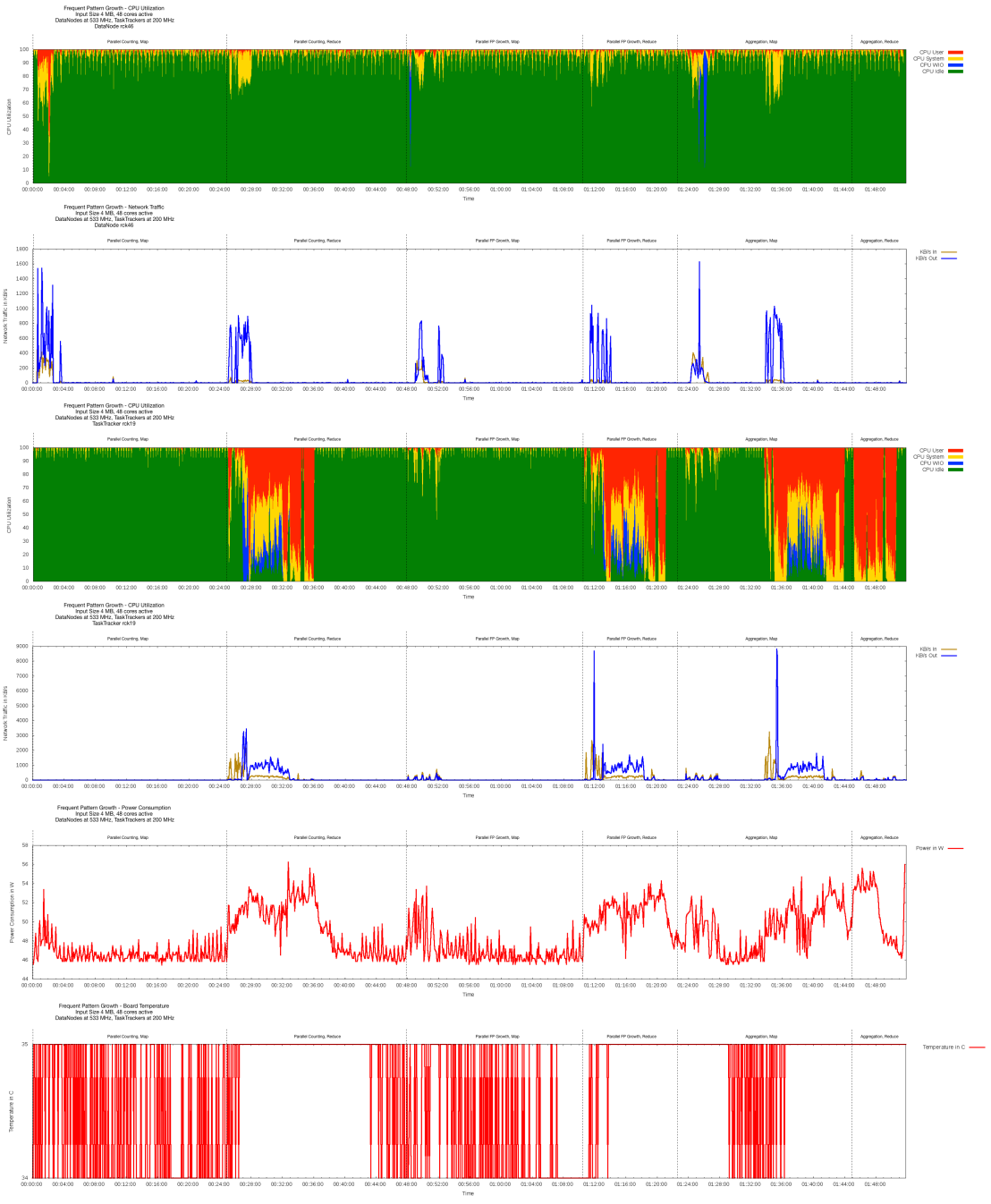


B.4.8 Input Size 4 MB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 533 MHz

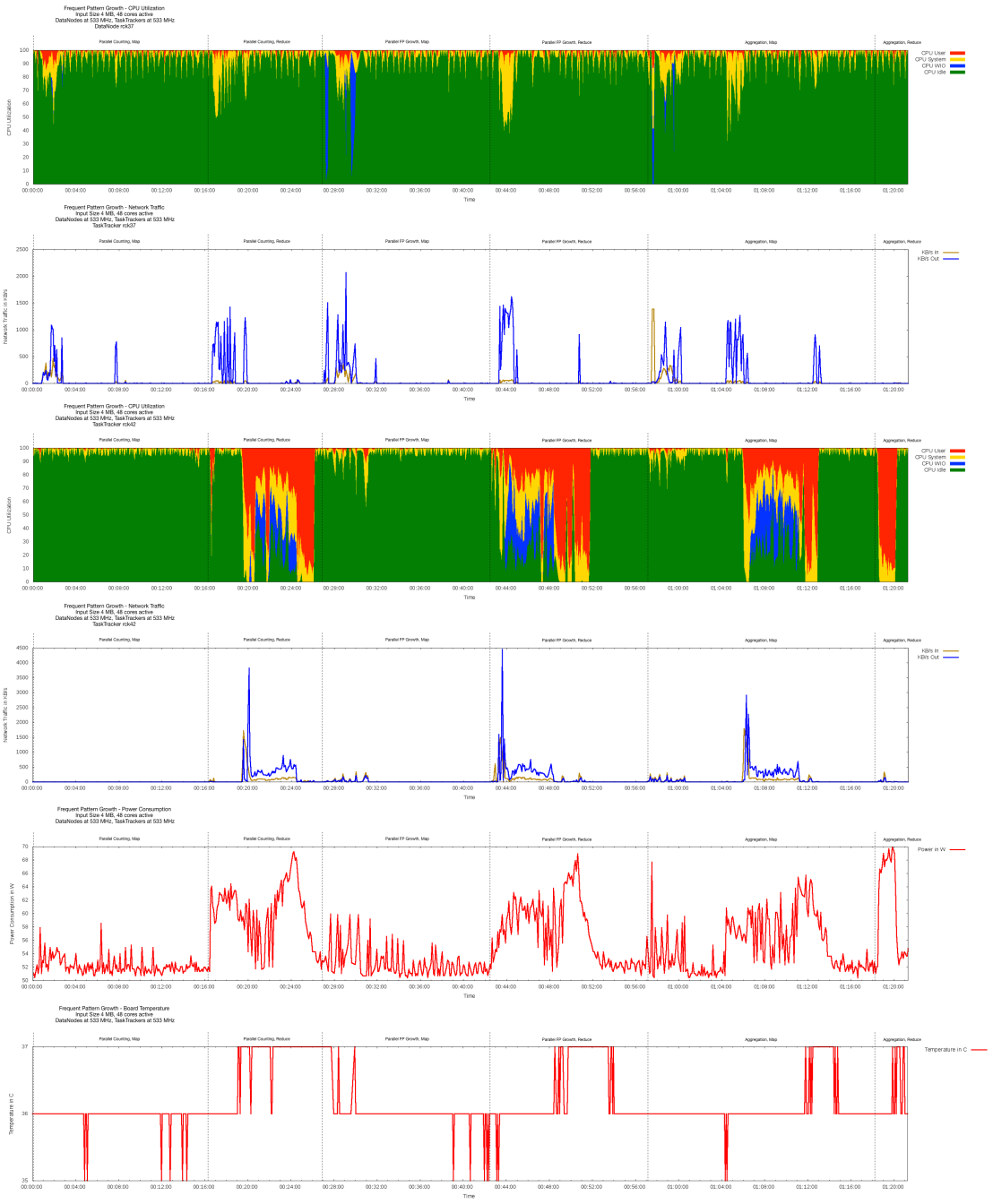
B.4.9 Input Size 4 MB, 48-Node Cluster Topology, DataNodes at 200 MHz -TaskTrackers at 800 MHz



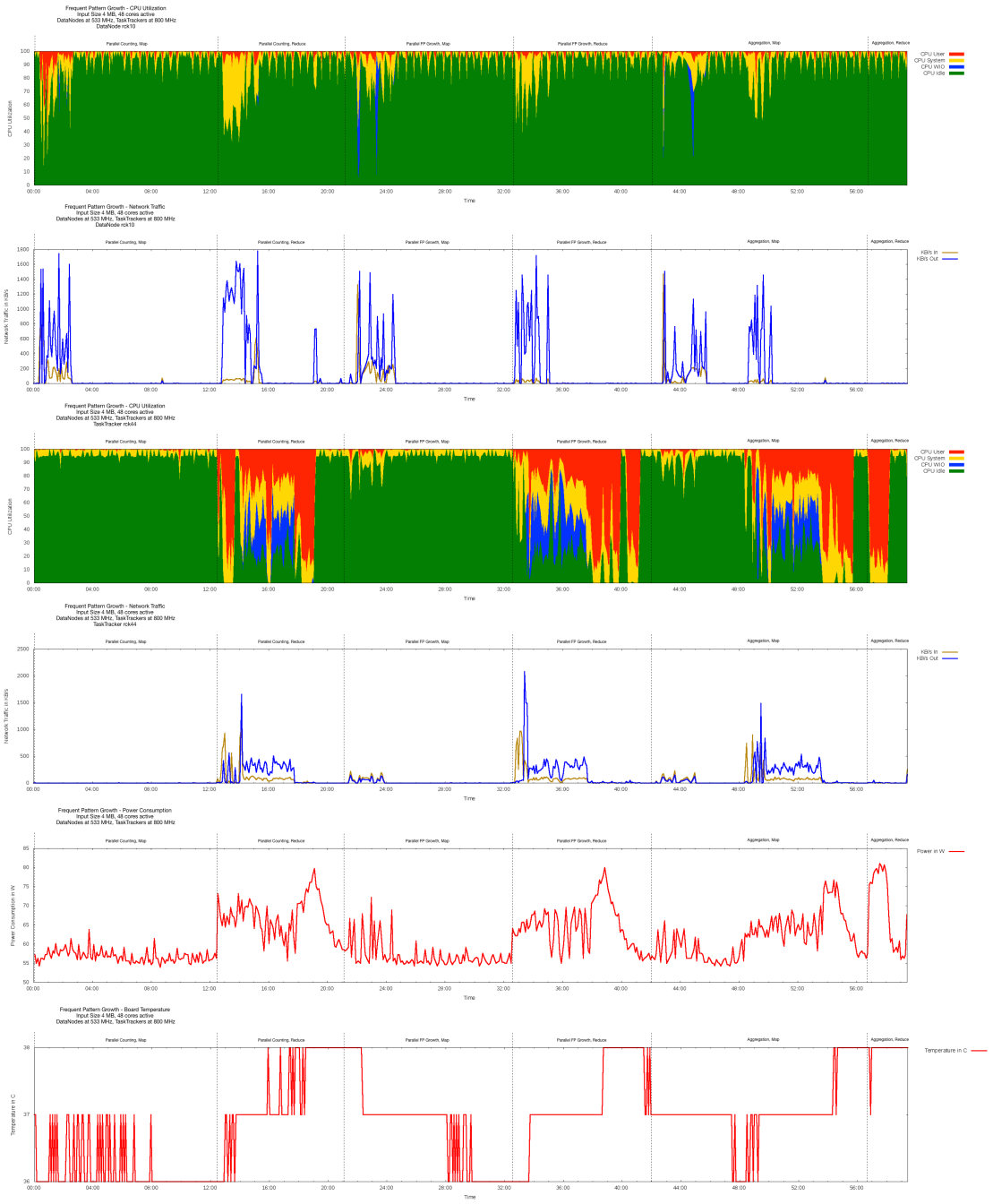
B.4.10 Input Size 4 MB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 200 MHz



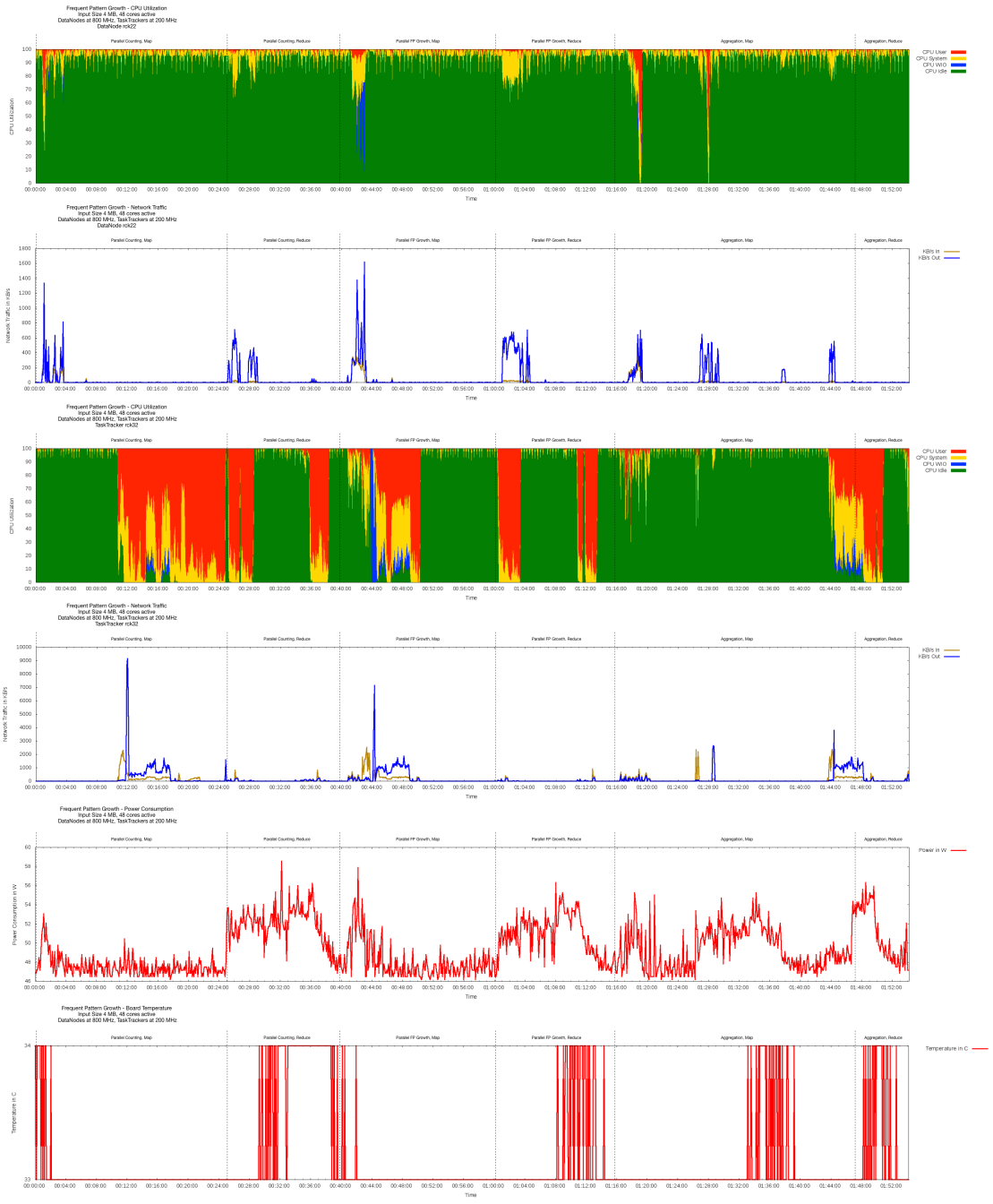
B.4.11 Input Size 4 MB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 533 MHz



B.4.12 Input Size 4 MB, 48-Node Cluster Topology, DataNodes at 533 MHz -TaskTrackers at 800 MHz



B.4.13 Input Size 4 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 200 MHz



B.4.14 Input Size 4 MB, 48-Node Cluster Topology, DataNodes at 800 MHz -TaskTrackers at 533 MHz