Energy Aware Resource Management for MapReduce Jobs

by

Adam Gregory

A thesis submitted to the Faculty of Graduate and Postdoctoral
Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

© 2016, Adam Gregory

Abstract

Clouds which continue to garner interest from practitioners in industry and academia require effective energy aware resource managers to leverage processing power of underlying resources while minimizing energy consumption in global data centers. This thesis proposes several energy aware resource management techniques that can effectively perform matchmaking and scheduling of MapReduce jobs each of which is characterized by a Service Level Agreement (SLA) that includes a client specified earliest start time, execution time and a deadline with the objective of minimizing energy consumption. Techniques are proposed for both batch workloads and open systems subject to continuous job arrivals.

Simulation based experimental results presented in this thesis demonstrate the effectiveness of the proposed energy aware resource management techniques compared to alternative resource management techniques that do not consider energy consumption in task allocation and scheduling decisions.

Acknowledgements

I would like to express my sincere gratitude to Dr. Shikharesh Majumdar for the immeasurable support and guidance he has provided throughout completion of this thesis. Additionally, I would like to acknowledge Carleton University and the Natural Sciences and Engineering Research Council of Canada (NSERC) for providing financial support for this research. Finally, I would like to thank my girlfriend, my family, and my friends for keeping me grounded over the long research process.

Table of Contents

Abstract.....	ii
Acknowledgements	iii
Table of Contents	iv
List of Tables	ix
List of Figures.....	xi
List of Abbreviations and Symbols	xiii
Glossary of Terms	xvii
Chapter 1: Introduction.....	19
1.1 Motivation of the Thesis.....	20
1.2 Proposed Solution.....	21
1.3 Scope of the Thesis.....	23
1.4 Contributions of the Thesis	23
1.5 Thesis Outline.....	25
Chapter 2: Background and Related Work.....	27
2.1 MapReduce.....	27
2.2 Constraint Programming	29
2.2.1 Branch and Bound.....	30
2.2.1.1 Node Selection Strategy	31
2.2.2 Constructive Search with Constraint Propagation	32
2.2.2.1 Constraint Propagation	32
2.2.2.2 Constructive Search.....	34
2.3 Data Center Energy Consumption.....	35
2.3.1 Dynamic Voltage and Frequency Scaling.....	36

2.4	Related Work.....	37
2.4.1	Scheduling for fast completion time	37
2.4.2	Resource management for Jobs with SLAs.....	39
2.4.3	Energy aware resource management.....	40
2.4.3.1	Reducing Idle Resource Energy Consumption.....	41
2.4.3.2	Reducing Energy Consumption Due to Job Execution	43
2.4.3.3	Resource Management in Heterogeneous Cloud Environments	46
2.4.4	Multi-objective resource management	47
2.4.5	Comparison with Related Work.....	49
Chapter 3:	Mathematical Model and CP Formulation.....	51
3.1	Mathematical Models	51
3.1.1	Energy Model.....	51
3.1.2	System Model.....	55
3.1.3	Workload Model	56
3.2	Constraint Programming Formulation and Implementation.....	58
3.2.1	Constraint Programming Model Formulation	59
3.2.2	Constraint Programming Model Implementation.....	64
3.2.2.1	CP Optimizer Processing Time	69
Chapter 4:	Energy Aware Resource Management Techniques for Batch Workloads 73	
4.1	Energy Aware Batch Resource Manager.....	73
4.1.1	Overview	73
4.1.2	Performance Evaluation	74
4.1.2.1	Experimental Setup	75
4.1.2.2	Performance Metrics	76
4.1.2.3	System and Workload Parameters.....	79

4.1.2.4	Processor Description	82
4.1.3	Experimental Results	83
4.1.3.1	Impact of Laxity Factor	84
4.1.3.2	Impact of Missed Deadline Ratio Bound	85
4.1.3.3	Impact of Workload Size	87
4.1.3.4	Number of Missed Deadlines	90
4.1.4	Summary of Results	90
4.2	Configurable Batch Resource Manager	91
4.2.1	Overview	91
4.2.2	Resource Manager Design	93
4.2.2.1	Alternative Resource Manager Functionality	96
4.2.3	Performance Evaluation	98
4.2.3.1	Experimental Setup	98
4.2.3.2	Performance Metrics	99
4.2.3.3	System and Workload Parameters	100
4.2.3.4	Processor Description	102
4.2.4	Experimental Results	103
4.2.4.1	Impact of Batch Completion Time Bound	104
4.2.4.2	Impact of Missed Deadline Ratio Bound	106
4.2.4.3	Impact of Laxity Factor	107
4.2.4.4	Impact of Number of Resources	108
4.2.4.5	Impact of Map Task Execution Time	110
4.2.4.6	Impact of Number of Jobs	111
4.2.5	Determining Suitable Configuration Parameters	112
4.2.6	Summary of Results	113

Chapter 5: Energy Aware Resource Management Techniques for Open Systems

115

5.1	Energy Aware MapReduce Resource Manager for Open Systems	115
5.1.1	Overview	115
5.1.2	Resource Manager Design	118
5.1.2.1	EAMR-RM Algorithm	119
5.1.2.2	Scheduling Procedure	123
5.1.2.3	Matchmaking Algorithm	126
5.1.3	Performance Evaluation	128
5.1.3.1	Experimental Setup	128
5.1.3.2	Performance Metrics	129
5.1.3.3	System and Workload Parameters	130
5.1.3.4	Processor Description	131
5.1.4	Experimental Results	133
5.1.4.1	Impact of Arrival Rate	133
5.1.4.2	Impact of Laxity Factor	135
5.1.4.3	Impact of Map Task Execution Time	136
5.1.4.4	Impact of Number of Resources	137
5.1.4.5	Proportion of Rejected Jobs	138
5.1.4.6	Alternative Processor Model	139
5.1.4.7	Summary of Results	139
5.2	Hybrid Energy Aware MapReduce Resource Manager for Open Systems	141
5.2.1	Overview	142
5.2.2	Resource Manager Design	144
5.2.2.1	HEAMR-RM Algorithm	145
5.2.2.2	Scheduling Procedure	147

5.2.2.3	Matchmaking Algorithm	149
5.2.2.4	Resource Scaling Procedure	153
5.2.3	Performance Evaluation	159
5.2.3.1	Experimental Setup	160
5.2.3.2	Performance Metrics	160
5.2.3.3	System and Workload Parameters	161
5.2.3.4	Tuning Parameters	163
5.2.4	Experimental Results	165
5.2.4.1	Impact of Arrival Rate	165
5.2.4.2	Impact of Map Task Execution Time	168
5.2.4.3	Impact of Maximum Number of Resources	170
5.2.4.4	Impact of Earliest Start Time	172
5.2.4.5	Impact of Resource Enable Time	174
5.2.5	Summary of Results	174
Chapter 6:	Conclusions.....	177
6.1	Summary and Conclusions	177
6.1.1	Energy Aware Batch Resource Manager	178
6.1.2	Configurable Batch Resource Manager	179
6.1.3	Energy Aware MapReduce Resource Manager for Open Systems	180
6.1.4	Hybrid Energy Aware MapReduce Resource Manager for Open Systems	181
6.2	Future Work.....	183
References.....		184

List of Tables

Table 3.1: Base CP Formulation Constraints and Objective Function	61
Table 4.1 - System and Workload Parameters	79
Table 4.2 - Workload Parameter Variations	81
Table 4.3 - Intel Core i7-2760QM Power Measurements	82
Table 4.4 - Batch Completion Time and Energy Savings Tradeoff	87
Table 4.5 - Configurable Batch Resource Manager CP Formulation	95
Table 4.6 - Alternative CP Formulations	97
Table 4.7 - Configurable Batch Resource Manager System and Workload Parameters	101
Table 4.8 - AMD Opteron 2380 Power Measurements	103
Table 4.9 - Effect of db on $E_{savings}$ for $l=0.15$	107
Table 4.10 - Effect of Low l on $E_{savings}$	108
Table 5.1: EAMR-RM Algorithm	121
Table 5.2: EAMR-RM CP Formulation	124
Table 5.3: EAMR-RM Matchmaking Algorithm	127
Table 5.4: EAMR-RM System and Workload Parameters	130
Table 5.5: AMD Opteron 2380 and Intel Xeon E5 2687W Power Measurements	132
Table 5.6: HEAMR-RM Algorithm	146
Table 5.7: HEAMR-RM CP Formulation	148
Table 5.8: HEAMR-RM Matchmaking Algorithm - Part A	150
Table 5.9: HEAMR-RM Matchmaking Algorithm - Part B	151
Table 5.10: Resource Scaling Rules	156

Table 5.11: Resource Scaling Algorithm	157
Table 5.12: HEAMR-RM System and Workload Parameters	162
Table 5.13: HEAMR-RM Tuning Parameters	164

List of Figures

Figure 1.1 - Monthly Data Center Operating Costs [6]	20
Figure 2.1 - MapReduce WordCount Application.....	28
Figure 3.1 - Impact of Fail Limit on Normalized Energy Consumption	72
Figure 4.1 - Impact of l on C and $E_{savings}$ for Medium Workload	84
Figure 4.2 - Impact of db on C and $E_{savings}$ for Medium Workload	86
Figure 4.3 - Impact of l on C and $E_{savings}$ for Small Workload	88
Figure 4.4 - Impact of db on C and $E_{savings}$ for Small Workload	89
Figure 4.5 - Configurable Batch Resource Manager System Overview.....	93
Figure 4.6 - Impact of cb on $E_{savings}$ and C for 30 job batch	105
Figure 4.7 - Impact of db on $E_{savings}$ and D for 30 job batch.....	106
Figure 4.8 - Impact of l on $E_{savings}$ and D for 30 job batch.....	108
Figure 4.9 - Impact of m on $E_{savings}$ and C for 30 job batch	109
Figure 4.10 - Impact of cb on $E_{savings}$ and C for 30 job batch with $m=25$	110
Figure 4.11 - Impact of e^{max} on $E_{savings}$ and C for 30 job batch.....	111
Figure 4.12 - Impact of n on $E_{savings}$ and C	112
Figure 5.1: EAMR-RM System Overview	117
Figure 5.2: Impact of Arrival Rate.....	134
Figure 5.3: Impact of Laxity Factor	135
Figure 5.4: Impact of Map Task Execution Time.....	136
Figure 5.5: Impact of Number of Resources.....	137
Figure 5.6: Impact of Arrival Rate on Proportion of Rejected Jobs	138

Figure 5.7: Impact of Arrival Rate using Intel Xeon E5 2687W	139
Figure 5.8: Scaling Interval Task Overlap	154
Figure 5.9: Impact of Arrival Rate on Energy Savings	166
Figure 5.10: Impact of Arrival Rate on Average Job Turnaround Time	167
Figure 5.11: Impact of Arrival Rate on Proportion of Rejected Jobs	168
Figure 5.12: Impact of e^{max} on Energy Savings	169
Figure 5.13: Impact of e^{max} on Average Job Turnaround Time	170
Figure 5.14: Impact of Maximum Resources on Energy Savings	171
Figure 5.15: Impact of Maximum Resources on Average Job Turnaround Time	172
Figure 5.16: Impact of p on Energy Savings	173
Figure 5.17: Impact of p on Average Job Turnaround Time	173
Figure 5.18: Impact of Resource Enable Time	174

List of Abbreviations and Symbols

A	Percentage of Active CPU Gates
ACPI	Advanced Configuration and Power Interface
a_t	The scheduled start time of task t
BRM	Energy Aware MapReduce Batch Resource Manager
C	Batch Completion Time
C0	CPU Operating State
C1	CPU Halt State
cap	Total CPU Capacitance Load
cb	Batch completion time bound
C-BRM	Configurable MapReduce Batch Resource Manager
c_{diff}	Midpoint between c_{min} and c_{max}
c_{max}	Maximum estimated batch completion time
c_{min}	Minimum estimated batch completion time
CP	Constraint Programming
CPU	Central Processing Unit
c_r^{map}	Map task execution capacity of resource r
c_r^{red}	Reduce task execution capacity of resource r
c_t	The execution duration of task t considering frequency scaling
D	Proportion of Missed Deadlines
db	Missed deadline ratio bound
d_j	Deadline of job j
DU	Discrete uniform distribution
DVFS	Dynamic Voltage and Frequency Scaling
E	CPU Energy Consumption
EAMR-RM	Energy Aware MapReduce Resource Manager for Open Systems
eb	Energy consumption bound
ECG	Efficient Green Control
ECS	Energy Conscious Scheduling

EDF	Earliest Deadline First
E_{Idle}	Energy consumption due to idle resources
$E_{savings}$	Energy Savings
e_t	The execution time of task t at maximum permitted operating frequency f^{high}
E_{Task}	Energy consumption due to task execution
FCFS	First Come First Served
f^{high}	Minimum permitted CPU operating frequency
FIFO	First In First Out
f^{low}	Maximum permitted CPU operating frequency
GB	Gigabyte
GEMS	GrEen MapReduce Scheduler
h	Halt Percentage
HEAMR-RM	Hybrid Energy Aware MapReduce Resource Manager for Open Systems
HEFT	Heterogeneous Earliest Finish Time
i	Length of scaling interval
IO	Input/Output
IT	Information Technology
J	The set of jobs
j	A given job from the set J
J_{scaled}	Joules scaled by proportionality constant δ
k	Proportionality constant for calculating Total Power Dissipation
k_j^{map}	The number of map tasks in job j
k_j^{red}	The number of reduce tasks in job j
l	Laxity Factor
LP	Linear Programming
LT₁	Lower utilization threshold 1
LT₂	Lower utilization threshold 2
m	Number of resources in cloud environment
$maxTime$	Maximum Solve Time
me	Map task execution time

MHz	Megahertz
MILP	Mixed Integer Linear Programming
m_{min}	Minimum permitted number of active resources
MOLS	Multi-Objective List Scheduling
ms	Milliseconds
n	Number of jobs
N	Number of Missed Deadlines
NEA	Non Energy Aware Resource Manager
N_j	Missed deadline flag for job j
NP	Nondeterministic Polynomial Time
P	Total Power Dissipation
p	Proportion of requests with earliest start time greater than arrival time
P_d	Dynamic Power Dissipation
P_s	Static Power Dissipation
PUE	Power Usage Effectiveness
QoS	Quality of Service
R	The set of cloud resources
r	A given resource from the set R
R_{active}	The subset of all resources R which are active
re	Reduce task execution time
r_{enable}	The time (in seconds) to enable a resource
RSO	Resource Scaling Only Resource Manager
s	Seconds
s_j	Earliest start time of job j
SJF	Shortest Job First
SLA	Service Level Agreement
STF	Shortest Task First
SU^{map}	Map Slot Scaled Average Resource Utilization
SU^{red}	Reduce Slot Scaled Average Resource Utilization
s_x	A given task execution slot
T	The set of tasks of all jobs in J

t	A given task from the set T
TAT	Average Job Turnaround Time
TB	Terabyte
T_j	The set of tasks in job j
T_j^{map}	The set of map tasks in job j
T_j^{red}	The set of reduce tasks in job j
U	Uniform distribution
UT	Upper utilization threshold
v_j	The arrival time of job j
VM	Virtual Machine
W_{scaled}	Watts scaled by proportionality constant δ
x_{tr}	Matchmaking decision variable for task t and resource r
δ	Proportionality constant for calculating CPU Energy Consumption
λ	Arrival rate (in requests per second)
τ	Time
ω	Proportion of Rejected Jobs

Glossary of Terms

Average job turnaround time (TAT):

The average time taken to fulfill a request, measured from job earliest start time to completion.

Batch completion Time (C):

The completion time of the latest finishing reduce task in the batch workload.

Batch completion time bound (cb):

Upper bound on completion time the batch workload. Specified in the configuration parameters of the Configurable Batch Resource Manager as an additional scheduling requirement.

Big data analytics:

The process of examining large data sets to uncover hidden patterns and unknown correlations.

Dynamic Voltage and Frequency Scaling (DVFS):

The adjustment of operating frequency and supply voltage of a computing device's processor to save power and reduce energy consumption when the workload does not require execution at maximum speed.

Energy consumption bound (eb):

Upper bound on energy required to execute the batch workload. Specified in the configuration parameters of the Configurable Batch Resource Manager as an additional scheduling requirement.

Energy Savings ($E_{savings}$):

The CPU energy consumption of a proposed technique expressed relative to that of an alternative non-energy-aware resource management approach.

Fixed Task:

A task that cannot be migrated or rescheduled because it has already started executing or is scheduled to begin execution on a given processor before the current invocation of the resource management algorithm is expected to complete.

Halt Percentage (h):

Power dissipation of a halted CPU core defined as a percentage of the power dissipation of a CPU core operating at the maximum permitted frequency

Laxity:

A measure of the available slack time in the deadline of a request.

$$\text{Laxity} = \text{deadline} - \text{execution time} - \text{earliest start time}$$

MapReduce:

A leading computing framework devised by Google for performing big data analytics.

Matchmaking:

The resource management process of determining the allocation of tasks in the workload to resources in the system.

Maximum solve time (maxTime):

The maximum time allocated to the Constraint Programming Optimizer to solve the optimization problem.

Missed deadline ratio bound (eb):

Upper bound on ratio of the maximum number of jobs permitted to miss their client specified deadlines to total number of jobs in the batch workload. Specified in the configuration parameters of the Configurable Batch Resource Manager as an additional scheduling requirement.

Proportion of missed deadlines (D):

The fraction of jobs which miss their client specified deadlines relative to total number of jobs in the workload.

Proportion of rejected jobs (ω):

The fraction of jobs which are not guaranteed to meet their client specified deadlines and rejected by the system relative to the total number of job arrivals.

Scheduling:

The resource management process of determining the start time for tasks and the order in which tasks allocated to a given resource will be executed.

Service Level Agreement (SLA):

A contract between a service provider and a client that defines the level of service expected from the service provider for execution of a request. Comprises of a client specified earliest start time, execution time, and deadline for the request.

Chapter 1: Introduction

The advent of low cost and reliable mobile data sensors including cameras and microphones as well as extensive software logs has led to the collection of data sets which are too large and complex for traditional data processing applications. For example, the Pratt and Whitney Geared Turbo Fan engine is equipped with 5000 sensors that measure engine vibration, stress, and other factors and can generate up to 10 GB of data per second [1]. With an average of 12 hours of flight-time, a single twin-engine aircraft can generate 844 TB of data. By the end of 2014, popular social media website Facebook was estimated to collect around 600 TB of data related to user activity per day [1]. Advanced data processing techniques are required to perform data analytics that concerns the extraction of knowledge from raw data from such large data sets.

Big data analytics is the process of examining large data sets to uncover hidden patterns and unknown correlations. It is used, for example, in engineering to determine the health of, and schedule maintenance for, safety critical machinery and infrastructure [2] and in business to analyze market trends, spending patterns, and customer preferences [3]. Cloud computing has emerged as the leading platform for big data analytics. With cloud computing, clients have on-demand access to a large pool of computing resources to process these large and complex data sets in parallel. However, electricity usage in global data centers accounted for between 1.1% and 1.5% of energy usage worldwide in 2010 [4]. Data center energy consumption is expected to grow as cloud computing technologies continue to garner increased interest from industry professionals, academics, and the general public. This research concerns devising energy aware resource

management techniques for cloud data centers and clusters that are often used for conducting big data analytics.

1.1 Motivation of the Thesis

Due to increased energy costs and a trend towards more electricity-intensive information technology (IT) equipment [5], direct IT equipment acquisition costs are becoming a less significant factor in the economics of providing cloud computing services. A breakdown of monthly data center operating cost is shown in Figure 1.1.

Figure 1.1 - Monthly Data Center Operating Costs [6]

Figure 1.1 shows that server acquisition and maintenance costs account for the majority of monthly operating cost whereas power consumption of data center IT equipment directly accounts for an estimated 18.5% of operating cost. However, the vast majority of infrastructure costs are closely related to power. Power distribution infrastructure is functionally related to power in that sufficient power distribution equipment is required to

distribute the maximum amount of power consumed by the data center equipment. Cooling infrastructure is functionally related to power in that heat generated by power dissipation within the data center must be removed. Lower power consumption leads to power distribution and cooling infrastructure that is less expensive to acquire, operate, and maintain. Direct power consumption as well as power distribution and cooling infrastructure contribute to the impact of power on data center operating cost. Therefore, power costs directly account for or are related to 41.6% of monthly data center operating cost.

Substantial power consumption in global data centers not only results in large operating cost but also large carbon dioxide emissions [7]. Devising energy aware resource management techniques for cloud data centers is important not only in the context of increasing service provider profits and reducing cloud computing service costs to clients, but also in protecting the environment and limiting climate change.

1.2 Proposed Solution

Effective resource management strategies which leverage the underlying resource pool to meet client quality of service requirements while reducing energy consumption are required. Cloud resource managers must provide two important operations: *matchmaking* and *scheduling*. The matchmaking operation selects a resource or resources from a pool of available resources to execute requests. The scheduling operation determines time and order in which requests allocated to a resource will be executed. Matchmaking and scheduling for cloud environments is a well-known NP hard problem. Additionally, a trend towards latency sensitive big data applications such as live business analytics [8]

further increases complexity of resource management decisions by incorporating client Quality of Service (QoS) concerns.

This research concerns devising energy aware resource management techniques for matchmaking and scheduling MapReduce jobs each of which are characterized by multiple stages of execution and are subject to Service Level Agreements (SLAs) that comprise of a client specified execution time and deadline. Jobs submitted by clients are not permitted to start execution until the client specified earliest start time and should complete before their deadline. MapReduce is a programming framework developed by Google for processing large data sets in parallel [9] and is discussed in detail in Section 2.1. Big data analytics can be performed on batch workloads comprising a number of MapReduce jobs or in an open system subjected to a stream of MapReduce job arrivals. This thesis presents several resource management techniques for handling both batch workloads and open systems.

The proposed resource management techniques decrease energy consumption due to task execution using a Dynamic Voltage and Frequency Scaling (DVFS) based approach to reduce the CPU operating frequency. Since the service provider is only obligated to deliver results before the client specified deadline, task execution can be slowed down to save energy in some cases without violating client SLAs. The Hybrid Energy Aware MapReduce Resource Manager, a resource management technique presented in this thesis for open systems, also supports disabling unused resources to reduce energy consumption due to idle CPUs. The proposed techniques formulate the complex matchmaking and scheduling problem as an optimization problem using Constraint Programming (CP). Jobs are allocated to resources and scheduled with the goal of

minimizing data center energy consumption while ensuring that client QoS concerns captured by SLA deadline requirements are satisfied.

1.3 Scope of the Thesis

This thesis focuses on devising energy aware resource management techniques for open systems and batch workloads of compute intensive MapReduce jobs subject to SLAs which include deadlines. Compute intensive jobs are CPU bound. As in research performed in [10] and [11], CPU operating frequency is assumed to be inversely proportional to program execution time. Reducing average CPU operating frequency by half, for example, will cause program execution time to double. However, not all MapReduce jobs can be characterized as compute intensive. Resource management for IO intensive MapReduce jobs forms an interesting direction for future research.

In line with other research regarding the scheduling of jobs with deadlines (ex. [8], [10], [12], [13]), all task execution times are assumed to be specified by the client and known *a priori*. Incorporating MapReduce task execution time prediction models and devising techniques to handle error in task execution times estimates is discussed as future work in Section 6.2.

1.4 Contributions of the Thesis

The main contributions of the thesis are summarized next.

- A novel DVFS-based energy aware resource management technique for batch workloads of deadline constrained MapReduce jobs is introduced. The technique saves energy at the cost of increased batch completion time by leveraging slack in job deadlines and decreasing CPU operating frequency during execution of some

tasks. The resource manager ensures that the proportion of jobs that miss their deadlines does not exceed a specified upper bound.

- A configurable batch resource manager that enables service providers to achieve an effective tradeoff among batch completion time, SLA violations, and energy consumption is presented. Configuration parameters permit service providers to impact resource management decisions based on their requirements by specifying an upper bound on batch completion time and number of missed deadlines.
- An Energy Aware MapReduce resource manager, EAMR-RM, for open systems subjected to a stream of MapReduce jobs with SLAs is introduced. Algorithms and performance optimizations are devised to reduce solver overhead and manage a request queue as required for an open system subject to a continuous stream of job arrivals.
- Alternative energy savings approaches, which reduce idle energy consumption by disabling unused resources, are investigated. A hybrid strategy, HEAMR-RM, that combines the reduction in task execution energy consumption from DVFS with the reduction in idle energy consumption from disabling unused resources is discussed.
- The thesis provides various insights into system behavior and performance that are obtained from a rigorous simulation based analysis that includes the following:
 - A demonstration of the effectiveness of the proposed techniques for various workloads and system configurations is presented.

- The performance improvement of the proposed techniques compared to alternative non-energy-aware resource management techniques is discussed.

Three refereed papers ([14], [15], [16]) have resulted from the thesis and are indicated below:

- [14] A. Gregory and S. Majumdar, "A Constraint Programming Based Energy Aware Resource Management Middleware for Clouds Processing MapReduce Jobs with Deadlines," in *ICPE '16 Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering (2nd International Workshop on Performance Analysis of Big data Systems (PABS))*, Delft, The Netherlands, 2016, pp. 15-20.
- [15] A. Gregory and S. Majumdar, "A Configurable Energy Aware Resource Management Technique for Optimization of Performance and Energy Consumption on Clouds," in *8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Luxembourg City, Luxembourg, 2016. pp. 184-192.
- [16] A. Gregory and S. Majumdar, "Energy Aware Resource Management for MapReduce Jobs with Service Level Agreements in Cloud Data Centers," in *6th IEEE International Symposium on Cloud and Service Computing (SC2)*, Nadi, Fiji, 2016. pp. 568-577.

1.5 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 provides background information on MapReduce, Constraint Programming (CP), and data center energy consumption, and

discusses related work. Chapter 3 describes the mathematical model used for calculating energy consumption of the system and presents the base CP formulation and implementation used for matchmaking and scheduling. Chapter 4 discusses the algorithm details and performance evaluation of the energy aware resource management techniques for batch workloads. The energy aware resource management techniques for open systems, including algorithms and performance evaluation, are presented in Chapter 5. Finally, Chapter 6 summarizes and concludes the thesis and presents directions for future research.

Chapter 2: Background and Related Work

This chapter presents background information regarding the proposed energy aware resource management techniques and discusses related work from existing literature.

2.1 MapReduce

MapReduce is a programming model proposed by Google for handling large data sets [9] and has emerged as the leading computing framework for big data analytics. Due to the size of input data sets, computation is distributed among multiple resources to increase parallelism and reduce total processing time. The MapReduce programming model describes a multi-stage workflow with two key stages of execution: a map stage and a reduce stage [9]. In the map stage, independent map tasks each perform sorting and filtering on a portion of the set of input data in parallel. The resulting intermediate key/value pairs are then shuffled and merged before being passed to the reduce stage where they are processed as a set of parallel reduce tasks by an application specific summary operation to generate a more meaningful data set.

Many computations can be expressed using the MapReduce programming model. *WordCount*, a classic example of a MapReduce application and shown in Figure 2.1, counts the number of occurrences of each word in a given input data set. In the provided example shown in Figure 2.1, the input data set comprises a list of fruits. The output of the *WordCount* application will include the list of fruits that appear in the input data and the number of occurrences of each. The input data is first divided into blocks to be processed in parallel. Next, in the map phase, map tasks processes each of the blocks of input data. In the *WordCount* application, the user defined mapper function reads each word in the input block and generates a key/value pair in the form: {Word, 1}. This

key/value pair specifies that one instance of a particular word has appeared in the input data. The intermediate data generated in the map phase may contain many duplicate key/value pairs. For example, the key/value pair {apple, 1} may appear several times in the intermediate data set indicating that it was found more than once in the input data.

Figure 2.1 - MapReduce WordCount Application

Next, in the shuffle phase, intermediate key/value pairs are shuffled and merged as shown in Figure 2.1. The shuffled and merged key/value pairs are then passed to the reduce phase where they are processed by a user defined reducer function to produce the final output. The reduce phase typically performs a summary or aggregate operation to represent the input data in a smaller and more meaningful format. In general, the reduce phase cannot begin until all tasks in the map phase have completed execution. In the *WordCount* application, the reducer function sums the value of all pairs with the same key to generate output data in the form: {Word, Count}. Thus, the final output of the

WordCount application is a list of words with the total number of occurrences of each word in the input data.

MapReduce has become increasingly utilized for latency-sensitive applications such as live business intelligence and real-time data analysis [8] as well. For example, Facebook, an early adopter of the MapReduce programming model using open source implementation Hadoop [17], uses MapReduce to process terabytes of data to detect spam, analyze user activity, and measure the effectiveness of advertisements [18].

2.2 Constraint Programming

The energy aware MapReduce resource management techniques proposed in this thesis model the complex matchmaking and scheduling problem as an optimization problem using a Constraint Programming (CP) [19] methodology. Similar to Linear Programming (LP), CP is a powerful programming paradigm for solving combinatorial search problems. Unlike traditional optimization techniques such as LP, CP seeks a feasible solution rather than an optimal one. However, this distinction is superficial: CP techniques can be configured to achieve an optimal solution with regards to minimizing or maximizing a specified objective function by gradually tightening a bound on the objective function value [20]. The primary difference between the approaches is that traditional optimization techniques such as LP are based on numerical algebra whereas CP is based on logical inference. CP also natively supports logical constraints and includes specialized global constraints and decision variable types that improve solve efficiency for some problems [19]. The suitability of CP based techniques for solving scheduling problems is well documented (see [20] and [21] for example).

CP is a form of declarative programming wherein relationships between variables are stated in the form of constraints. A CP formulation has three major components: 1) decision variables—the set unknown variables to solve, 2) an objective function—the function which produces the value to be optimized (maximized or minimized) based on selected decision variable values, and 3) constraints—the set of rules which specify the requirements of decision variable values for feasible solutions. The CP formulation does not specify the process of finding a feasible solution, rather, the formulation describes the desired characteristics of the solution to be found. Values are assigned to the set of decision variables such that the solution is feasible (i.e. no constraints are violated) and the specified objective function is optimized. Combinatorial search algorithms such as branch and bound and constructive search with constraint propagation are used to search the solution space for high quality feasible solutions [19]. Each of these techniques is briefly discussed.

2.2.1 Branch and Bound

A branch and bound algorithm consists of a state space search through systematic enumeration of candidate solutions. The set of candidate solutions is thought of as forming a rooted tree and the algorithm explores branches of the tree which represent subsets of the solution set. During solving, the best solution found so far by the algorithm is known as the incumbent solution [22]. Before enumerating the candidate solutions of a branch, the value of the bounding function, which estimates the best possible objective function value of any leaf node in the branch, is compared to the incumbent solution. If the bounding function value is not better (higher for maximization problems or lower for minimization problems) than the incumbent solution, the branch is pruned from the

solution space and the candidate solutions corresponding to that branch are not enumerated. The bounding function is mathematically guaranteed by design to overestimate the quality of candidate solutions. This ensures that the optimal solution is not mistakenly pruned from the solution space before being enumerated.

A branch and bound algorithm's efficiency depends on accurate bounding function estimates [22]. A branch and bound algorithm with a low quality bounding function which severely overestimates the best possible objective function value of any leaf node in a branch approaches an exhaustive state space search because few or no branches can be pruned from the solution space before being enumerated.

2.2.1.1 Node Selection Strategy

The order in which branches are explored and candidate solutions are enumerated is based on the node selection strategy. The most common branch and bound node selection strategies are breadth-first search, depth-first search, and combined global best.

In breadth-first search, all branches on the current level are expanded before moving deeper in the tree. Using this node selection strategy often leads to long processing times before the first solution is found. Because all branches on the current level are explored before considering nodes deeper in the tree, many branches are explored before the first leaf node is enumerated. This behavior often leads to poor performance of the branch and bound algorithm because branches cannot be pruned before the first incumbent solution is found. Therefore, many branches with poor bounding function values which would otherwise be pruned may be needlessly enumerated.

In depth-first search, a single branch is fully expanded to the leaf nodes before other branches are considered. This node selection strategy results in the shortest processing

time before the first feasible solution is found. Depth-first search is especially effective for problems with very large solution spaces because it improves the chances of quickly finding an incumbent which can then be used for pruning. However, depth-first search with eschew promising nodes in other branches in favor of fully exploring the current branch even if the potential objective function values are significantly worse.

Combined global best provides a compromise between the willingness to consider alternatives of breadth-first search and the solution based focus of depth-first search [22]. The global best node selection strategy will always consider the node with the best bounding function value. This means that the branch and bound algorithm will always expand the branches and enumerate the solutions with the best expected objective function value first. Global best is the most effective of the three common node selection strategies for typical CP problems [22].

2.2.2 Constructive Search with Constraint Propagation

Constructive search with constraint propagation is an alternative method for solving constraint programming problems. Constraint propagation is used to reduce the size of the solution space before constructive search is used to efficiently search the remaining solution space for high quality feasible solutions [23].

2.2.2.1 Constraint Propagation

Constraint propagation refers to two separate processes: 1) decision variable domain reduction, and 2) propagating these reductions among the constraints. Each decision variable in the CP model has a domain of values permitted in the solution. However, not all of these domain values are possible in a feasible solution. A constraint stated over one or more decision variables restricts the assignment of values to these variables.

For example, consider the decision variables a and b , each of which has an initial permitted domain of $[1..20]$ and are subject to the constraint $a + b \leq 7$. The decision variables initially have a domain of the set of integer values between 1 and 20. However, an assignment of $a \geq 7$ does not yield a feasible solution as there is no value in the domain of b which will satisfy the constraint. This reasoning can be applied similarly to the domain of decision variable b . Thus, the domains of a and b can be reduced to $[1..6]$. Domain reduction can be performed using a constraint to remove values from the domain of decision variables which are not feasible in solutions of the constraint. When all inconsistent domain values are removed from a decision variable the process is known as full domain reduction. However, full domain reduction is costly, and, in practice, domain reduction does not typically remove all infeasible values [23].

Once domain reduction has been performed for each constraint individually, domain reductions are propagated among all appropriate constraints. This process can result in more decision variable domain reductions and continues until no domain can be further reduced or a domain becomes empty. An empty domain during initial constraint propagation indicates that no feasible solution exists.

For example, consider the decision variables x with initial domain $[0..10]$, y with initial domain $[0..1]$, and z with initial domain $[0..1]$ subject to the following constraints:

$$(1) y \neq x$$

$$(2) y \neq z$$

$$(3) x + 5y + z \leq 4$$

Constraints (1) and (2) cannot reduce the domain of the involved decision variables independently. The domain reduction for constraint (3) reduces the domain of x to $[0..4]$

and y to $[0]$. Thus decision variable y is fixed to a single value. Constraint propagation invokes domain reduction for every constraint involving either x or y . Domain reduction of constraint (1) further reduces the domain of x to $[1..4]$ and constraint (2) fixes the value of z to a single value. Constraint propagation invokes domain reduction for every constraint involving either x or z and the domain of x is further reduced to $[1..3]$ by constraint (3).

No more domain reduction can be achieved and the final domains are:

- $x = [1..3]$
- $y = [0]$ and
- $z = [1]$

2.2.2.2 Constructive Search

After decision variable domains have been reduced by constraint propagation, constructive search strategies efficiently search the remaining reduced solution space for the optimal solution to the constraint programming problem. Constructive search implicitly generates combinations of feasible decision variable values until the optimal solution is found [24]. Using an operation called branching, constructive search attempts to build a solution by choosing a non-fixed decision variable and value for that variable. The decision variable is fixed to the chosen value and constraint propagation is triggered to reduce the domain of the remaining decision variables and the size of the solution space. After constraint propagation terminates, another non-fixed decision variable is chosen, if one exists, and the process repeats until all decision variables are fixed and a solution is found. If fixing the decision variable fails because the domain of a non-fixed decision variable is made empty during constraint propagation and the chosen value

cannot lead to a solution, constructive search backtracks and chooses another value for decision variable [24]. After an incumbent solution is found, the process is repeated and alternative branches which may yield an improvement in objective function value are explored. The search terminates once the incumbent solution is proven to be optimal [23]. Constructive search includes a mechanism similar to the bounding function used by branch and bound algorithms to estimate the objective function quality of a branch's solutions. Branches which do not provide significant expected improvement compared to the incumbent solution will be avoided to further reduce the size of the solution space and decrease search time.

Data center energy consumption, an important performance metric for the resource management techniques proposed in this thesis, is discussed in the following section.

2.3 Data Center Energy Consumption

As discussed previously, currently electricity costs directly account for an estimated 18.5% of monthly data center operation costs and a further 23.1% of monthly data center operation costs are associated with infrastructure costs related to energy consumption (i.e. power distribution and cooling infrastructure) [6]. For example, a data center with a higher monthly energy consumption will need to pay more to support and maintain the more robust power distribution infrastructure necessary to deliver power to the data center components.

Due to energy savings in infrastructure and support systems such as cooling, a one watt savings in power dissipation of data center IT components (processor, memory, storage etc.) results in a greater reduction in overall facility power dissipation [25]. A data center with lower server level component energy consumption will also expend less energy

cooling the data center to maintain the required temperature. The difference between the total facility power consumption and the IT component power consumption is measured by the *power usage effectiveness PUE* as shown in Equation 2.1 [6]:

$$PUE = \frac{\text{Total Facility Power}}{\text{IT Component Power}} \quad 2.1$$

The PUE of a data center captures how many watts must be delivered to and consumed by the data center to get one watt to the critical load, the servers themselves. A PUE of 1.7 means that for every watt delivered to the IT components to process the workload, 0.7 watts is required by the cooling and power distribution infrastructure. PUEs of data centers vary greatly. Inefficient enterprise data centers can have a PUE as high as 2.0 or even 3.0. However, a PUE of around 1.7 is more common [6]. Techniques which focus on minimizing the energy consumption of these IT components offer the largest potential reduction in energy consumption and therefore, data center operating cost.

Measurements reported in [26] determined that the CPU has the largest impact on data center server energy consumption among these server level components. Thus, this research aims to reduce data center energy consumption by minimizing energy consumption of data center server processors.

2.3.1 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS) is a technique that is often used for reducing the energy consumption of data center server CPUs [27]. By decreasing the voltage supplied to the CPU, power dissipation can be reduced. However, the reduction in supply voltage also leads to a corresponding decrease in operating frequency and, therefore, a potential increase in program execution time. Reducing CPU supply voltage may not always yield lower overall energy consumption. For example, a program

executing for 10 seconds on a CPU with a power dissipation of 10 Watts will consume more energy than the same program executing for 5 seconds on a CPU with a power dissipation of 15 Watts. DVFS can be used to effectively decrease CPU energy consumption, however, it must be applied intelligently to ensure energy consumption is reduced as expected.

2.4 Related Work

This research focuses on the matchmaking and scheduling of multi-stage workflows such as MapReduce which are widely adopted among researchers and industry practitioners for big data analytics. A representative sample of the considerable body of work regarding resource management of MapReduce and other compute intensive workloads both with and without SLAs on cloud environments is discussed in this section.

2.4.1 Scheduling for fast completion time

Much of the existing research on matchmaking and scheduling for cloud environments focuses on reducing completion time. For batch workloads this goal is expressed as minimizing makespan where makespan is the total length of the schedule (i.e. when all jobs have finished processing). In the case of an open stream of job arrivals, reducing completion time is often expressed as minimizing average job turnaround time where turnaround time for a given job is the time between earliest start time and completion of the job. Scheduling workloads for reduced completion time can improve client satisfaction as well as increase service provider revenue. When jobs are completed quickly, clients will receive results in a timely manner and the cloud environment may be able to accept more work.

The authors of [28] propose two resource management strategies based on Linear Program (LP) relaxation: OFFA for a batch of MapReduce jobs and ONA for an open stream of MapReduce job arrivals. For both OFFA and ONA the scheduling problem is solved using a two-step algorithm. LP relaxation is used to determine a lower bound on the optimal completion time. The output of the LP relaxation itself may not be feasible because some scheduling constraints are relaxed, however, this solution is then used to derive a feasible workload schedule which is within a factor of 3 of the optimal. A simulation based analysis is used to compare the performance of OFFA and ONA with First-in First-out (FIFO), Shortest Job First (SJF), and Shortest Task First (STF) scheduling strategies. SJF is considered optimal for minimizing the average turnaround time of traditional jobs, however, it may not be optimal for scheduling jobs which are characterized by multiple stages of execution. Experimental results show that ONA achieves an up to 30% reduction in completion time compared to FIFO and consistently outperforms SJF and STF.

Researchers in [29] devise a preemptive resource management strategy for an open stream of MapReduce job arrivals. The proposed Fair Completion Scheduler reduces completion time by minimizing occurrences of prolonged execution of small jobs caused by starvation due to the monopolizing behavior of long reduce tasks from large jobs. Performance analysis with a Hadoop cluster and well-known MapReduce benchmark applications found that the proposed technique improves average job execution time by up to 39.7% and improves fairness by up to 66.7% compared to the Hadoop Fair Scheduler.

2.4.2 Resource management for Jobs with SLAs

A recent movement in cloud computing towards latency-sensitive applications such as live business intelligence, spam and fraud detection, and real-time data analysis [8] further increases resource management complexity. For these and other real-time data analytics applications, analyzing up-to-date data and delivering timely results is of critical importance. MapReduce jobs with an associated deadline on completion time have become increasingly utilized for such latency-sensitive applications. Client deadline requirements are often specified in Service Level Agreements (SLAs) with the service provider. A SLA defines a contract between the client and the service provider regarding the level of service required by a request. The service provider may be required to pay a penalty to the client if this level of service requirement is violated. Devising resource management strategies which minimize the number of jobs which miss their specified deadlines is an important research topic.

The authors of [21] investigate the suitability of various optimization techniques for matchmaking and scheduling a batch of MapReduce jobs such that the number of jobs which miss their deadlines is minimized. The authors compare the performance of three alternative approaches for solving the matchmaking and scheduling problem: Mixed Integer Linear Programming (MILP) model solved using LINGO (Approach 1), Constraint Programming (CP) model solved using MiniZinc and Gecode (Approach 2), and CP model solved using CPLEX (Approach 3). The simulation based performance analysis shows that all three approaches were able to determine schedules with no missed deadlines. “Approach 3” was shown to have lower solver overhead and was able to effectively handle larger workloads than the alternatives.

In [8], researchers propose a heuristic resource allocation policy based on Earliest Deadline First (EDF) called MinEDF-WC for an open stream of MapReduce jobs with deadlines. The assignment of tasks to execution slots is performed online using a greedy algorithm which allocates each job the minimum number of execution slots required to complete before its deadline. MinEDF-WC allocates spare resources to active jobs to improve cluster utilization and performance and can dynamically de-allocate these resources to service newly arrived jobs with earlier deadlines if necessary. An extensive performance analysis including experimentation using Facebook traces on a Hadoop cluster and simulations involving benchmark workloads shows that up to 20% fewer jobs missed their deadlines when scheduled using MinEDF-WC compared to EDF. However, a minor increase in average job turnaround time was observed for some tested workloads. Researchers in [12] extend their work on “Approach 3” from [21] and devise a MapReduce Constraint Programming (CP) based resource manager (MRCP-RM) for matchmaking and scheduling an open stream of MapReduce jobs with deadlines. The proposed technique builds the CP model incrementally as jobs arrive and invokes the CPLEX CP Optimizer to determine an optimal solution to the matchmaking and scheduling problem which minimizes the number of jobs which miss their deadlines. A discrete event simulation based performance analysis using the Facebook trace workload from [8] shows that MRCP-RM achieves an up to 93% reduction in number of missed deadlines and 7% reduction in average turnaround time compared to MinEDF-WC.

2.4.3 Energy aware resource management

High energy costs and a trend towards increasingly electricity-intensive information technology (IT) equipment [5] means that direct IT equipment acquisition costs are

becoming a less significant determinant in the economics of providing cloud computing services. Energy efficient resource management techniques have started to receive increased attention from researchers in order to reduce the data center operating costs for service providers and the costs of cloud computing services for clients. These techniques typically focus on mitigating energy consumption using one of several strategies: 1) reducing energy consumption due to idle resources, 2) reducing energy consumption of resources executing jobs, or 3) allocating jobs to the best suited resources in heterogeneous environments.

2.4.3.1 Reducing Idle Resource Energy Consumption

In [30], researchers present an energy efficient VM placement strategy for batch workloads of MapReduce jobs on clouds. The approach considers the tradeoff between efficient spatial fitting of VMs on servers to minimize the number of servers required to execute the batch workload and balanced temporal fitting of VMs having similar runtimes to ensure servers run at high utilization during their uptime. Resources not needed to execute the batch workload are disabled to reduce energy consumption. Furthermore, the approach incorporates techniques to dynamically scale cluster size to further reduce energy consumption or improve completion time as required. Performance analysis shows that the proposed approach reduces energy consumption by between 20% and 35% compared to alternative strategies which only consider spatial consolidation.

Researchers in [31] devise an energy efficient resource management method for an open stream of MapReduce job arrivals in virtualized cloud environments. The proposed VM placement strategy considers job runtimes and resource requirements when mapping VMs to host resources such that the minimum number of resources are needed. Unused

resources are disabled when possible to reduce energy consumption. A live mapping algorithm is used to migrate VMs dynamically and improve consolidation as additional jobs arrive. Simulation based evaluation shows that the proposed energy efficient resource management method reduces energy consumption by up to 40% compared to a First Fit VM placement approach.

The authors of [32] develop an energy aware resource management system for virtualized cloud data centers subject to continuous workload arrivals. Energy savings are achieved by continuous consolidation of VMs using live migration according to current CPU requirements, thermal state of computing resources, and task communication. Because aggressive consolidation of VMs can lead to a lack of available resources for future requests, heuristics that select VMs for migration balance the trade-off between performance and energy savings. The VM migration policy attempts to keep the total CPU utilization within a specified upper and lower utilization threshold. Performance analysis using an extended CloudSim toolkit demonstrates that the proposed dynamic VM migration and allocation policy reduces energy consumption by up to 83% compared to static non-power aware policies without significantly impacting system performance.

The authors of [33] propose a resource management technique for minimizing energy consumption of heterogeneous cloud environments subject to bursty MapReduce job arrivals. The GrEen MapReduce Scheduler (GEMS) models the datacenter topology as a network graph where servers and switches are nodes and links are edges with different capacities and power consumption based on operating mode. Incoming tasks are assigned to the minimum energy path which satisfies the CPU, memory, and network requirements. The data center is provisioned based on peak expected load and servers and

switches can be disabled or enabled as required to reduce energy consumption. Simulations based on Google MapReduce traces show that GEMS reduces energy consumption by up to 35% and reduces job turnaround time by up to 35% compared to alternative network agnostic policies which do not consider disabling unused resources. Researchers in [7] apply an N-policy based approach to optimizer operational cost and mitigate server idle power consumption for open cloud environments executing deadline constrained jobs without precedence relationships. Servers are transitioned to a sleep mode when idle and remain sleeping until the number of jobs waiting in the first come first serve queue exceeds the threshold N. Researchers propose an efficient green control (ECG) algorithm to solve constrained optimization problems and determine the minimum operational cost based on job arrival rate and N. Simulation results show that the proposed technique is effective for making tradeoffs between operational cost and performance and can reduce energy consumption compared to a typical system providing the same deadline guarantee.

2.4.3.2 Reducing Energy Consumption Due to Job Execution

In [10], researchers propose a power aware resource management technique for an open stream of “bag-of-tasks applications” with deadline constraints on cloud environments. Unlike MapReduce applications, bag-of-tasks applications are defined by a single stage of execution with no precedence relationships between tasks. The technique uses Dynamic Voltage and Frequency Scaling (DVFS) to vary resource supply voltage and corresponding operating frequency and reduce dynamic power consumption. Running resources at low supply voltage decreases energy consumption but may cause jobs to miss their deadlines due to low processor speeds. However, running resources at high

supply voltage to meet job deadlines incurs larger energy consumption. The approach aims to set the supply voltage of resources as low as possible to reduce energy consumption while still guaranteeing that all the deadlines of accepted jobs will be met. When a job arrives, each resource tests the schedulability of the newly arrived tasks and returns the estimated energy required to execute the task. Tasks are then assigned to the resource with the lowest energy estimate and scheduled based on EDF. Simulation based performance analysis shows that the proposed resource management technique reduces energy consumption by between 13.6% and 44.3% compared to static voltage schemes. Energy savings was accompanied by an up to 13.3% increase in the number of jobs rejected because deadline compliance could not be guaranteed.

In [34], researchers investigate the tradeoff between makespan and energy consumption for batch workloads of precedence constrained applications on clouds. The Energy Conscious Scheduling (ECS) heuristic uses DVFS to vary the energy consumption and completion time of tasks in the workload. Processor allocation and execution frequency for each task is determined using a utility function which attempts to balance energy consumption and makespan considerations. Utility for each combination of task, processor, and clock frequency are enumerated and the combination with the highest utility is chosen. An extensive performance analysis shows that ECS reduces makespan by up to 5% and reduces energy consumption by up to 12% compared to the well-known Heterogeneous Earliest Finish Time (HEFT) heuristic [35].

Researchers in [36] devise DVFS based energy efficient scheduling heuristics for batch workloads of precedence constrained parallel tasks on cloud environments. Executing applications are modeled as workflows with directed acyclic graphs and critical path

analysis is used to determine the sequence of tasks which governs the application completion time. DVFS is used to slow down execution of non-critical tasks to take advantage of slack time and reduce energy consumption without impacting the overall completion time of the application. Simulation based performance analysis using synthetic DAG workloads shows that the proposed critical path scheduling heuristic reduces energy consumption by up to 44.3% compared to alternatives which do not consider energy consumption and by up to 6.3% compared to the ECS approach described in [34].

The authors of [37] investigate the impact of dynamically scaling CPU supply voltage and operating frequency based on load to improve energy efficiency for computation intensive MapReduce batch workloads. The authors consider how MapReduce efficiency varies based on the number of available concurrent workers and DVFS that adjusts processor frequency based on the workloads' computational needs. Experimental results obtained using a small cluster connected to commercially available power monitors indicate that significant energy savings can be achieved from appropriate resource allocation and intelligent DVFS scheduling and that DVFS is a suitable approach for reducing energy consumption of MapReduce and other computation intensive applications.

In [11], researchers compare two alternative energy saving approaches for computation intensive applications on cloud environments: *race-to-halt* and *CP-aware slack reclamation*. *Race-to-halt* is a DVFS scheduling strategy that enforces resources to execute workloads at the highest working frequency and supply voltage of the hardware when jobs are available for processing and enter a low-power energy saving mode when

processing is complete. *CP-aware slack reclamation* uses DVFS to leverage available slack and slow down execution of jobs which are not on the critical path to save energy. Energy savings can be achieved effectively by both approaches without significantly impacting makespan. Previous formal proofs and conventional theory indicate that *CP-aware slack reclamation* is more energy efficient than *race-to-halt*, however, the presented experimental results show that the gap in energy savings capabilities of the two approaches has significantly narrowed due to advances in processor technologies leading to only minor variations in supply voltage as operating frequency of a processor is scaled. The authors of [38] propose an optimization based VM placement policy for batch workloads without precedence relationships between tasks on cloud environments. The approach simultaneously solves for three objective functions to determine the allocation which minimizes the energy required to execute the workload. The objectives are to 1) minimize the number of resources required to execute the workload, 2) minimize resource operating frequency using DVFS, and 3) maximize utilization of enabled resources. The authors apply NSGA-II, an evolutionary genetic algorithm, to simultaneously solve the three objective functions and determine the VM allocation and operating frequency. Simulation based performance analysis shows that the proposed technique is effective.

2.4.3.3 Resource Management in Heterogeneous Cloud Environments

Researchers in [13] devise two Energy aware MapReduce Scheduling Algorithms (called EMRSA-I and EMRSA-II) for heterogeneous cloud environments executing batch MapReduce workloads. EMRSA-I and EMRSA-II find the allocation of Map and Reduce tasks to resource execution slots in order to minimize the energy consumed when

executing the application while guaranteeing that the user specified completion time deadline is met. Since a heterogeneous cloud environment is considered, the execution time and energy required to execute a task on two different resources may not be the same. The authors define an *energy consumption rate* metric for each resource which is a measure of the resource's execution speed and energy consumption. Map and Reduce tasks are assigned to execution slots such that the overall workload energy consumption rate is minimized and all job deadlines are met. Performance analysis shows that the proposed heuristics generate high quality task mappings with approximately 40% lower energy consumption on average than a common practice resource management policy which minimizes makespan.

The authors of [39] investigate the tradeoff among energy consumption, deadline compliance, and service provider profit for virtualized data centers. The profit-centric approach models all facets of resource management as costs and compares the expected cost of each possible Virtual Machine (VM) allocation to the revenue for job completion to determine the most profitable solution. Simulations show that energy efficient resource management techniques are among the most profitable and that energy consumption can be significantly reduced without considerable degradation to deadline compliance compared to other non-energy aware resource management strategies.

2.4.4 Multi-objective resource management

Most resource management algorithms focus on the optimization of a single metric: energy consumption, SLA compliance, or completion time for example. In some situations, it may be worthwhile to manage resources to achieve a balance among several performance metrics rather than to optimize a single one. Such multi-objective resource

management strategies which handle both client and service provider considerations have no single optimal solution. Instead the solution space is a sometimes infinite set of Pareto optimal solutions [40]. A Pareto optimal solution is a solution for which no performance metric can be improved without a corresponding decrease in one or more of the other performance metrics. An effective tradeoff among correlated and competing performance metrics is necessary to simultaneously improve client satisfaction and service provider profit.

Researchers in [41] devise a resource management strategy based on an ant colony algorithm to simultaneously minimize completion time and client cost for an open stream of deadline constrained bag-of-tasks applications on cloud environments. The ant colony algorithm makes behavioral choices based on heuristic information and pheromone strength to allocate tasks to resources. The quality of feasible solutions found is evaluated using a fitness function and used as feedback for the algorithm behavior. When a high quality solution is found the pheromone strength for that solution is increased to encourage the algorithm to search for similar solutions in future iterations. After a given number of iterations, the algorithm completes and the solution with the highest fitness is returned as the result. Performance analysis using simulation and real world experiments show that the proposed ant colony technique achieves an up to 20% reduction in client cost and up to 56.6% reduction in completion time without incurring additional SLA violations compared to a First Come First Served (FCFS) scheduling policy.

The authors of [40] propose a multi-objective list scheduling (MOLS) algorithm for a single precedence constrained scientific workflow on cloud environments. The technique determines the set of Pareto optimal solutions which simultaneously minimize 1)

makespan, 2) client cost, and 3) energy consumption subject to a client specified constraint vector which defines soft constraints on the upper bound of the three objectives. The MOLS algorithm is based on three major phases. In phase 1, the algorithm estimates the objectives' sub-constraints for each task in the workflow using the constraint vector. In phase 2, the algorithm assigns a rank to each task based on its bottom-level, defined in graph theory as the longest path to the last node including the node itself. Tasks are sorted in ascending order by rank. Finally, in phase 3, the algorithm attempts to allocate each task in order of rank to the most appropriate resource based on the estimated sub-constraints. Performance analysis using real-world scientific workflow applications shows that in most cases the obtained solutions dominate the user specified constraint vector and the proposed technique outperforms a related bi-criteria scheduling heuristic and bi-criteria genetic algorithm.

2.4.5 Comparison with Related Work

The proposed energy aware MapReduce resource management techniques are also based on optimization techniques and model the problem using Constraint Programming as discussed in [21]. The success of "Approach 3" for matchmaking and scheduling MapReduce jobs with deadlines motivated both MRCP-RM in [12] and this research. However, the main difference between this research and the research performed in [21] and [12] is that this research focuses on minimizing data center energy consumption in addition to considering MapReduce jobs subject to SLAs which include deadlines.

Many of the described existing techniques also focus on minimizing data center energy consumption. However, to the best of our knowledge, none of the existing research is suited to manage the full range of factors that energy aware MapReduce resource

management techniques proposed in this thesis can effectively handle. For example, EMSRA-I and EMRSA-II discussed in [13] are most closely related by goals and capabilities to the proposed energy aware MapReduce resource manager for batch workloads. However, these techniques are only suited for use in heterogeneous cloud environments and are ineffective in homogeneous cloud environments. Furthermore, EMSRA-I and EMSRA-II are based on batch profiling techniques which may be problematic to extend for open systems subject to asynchronous MapReduce job arrivals which the resource management techniques proposed in this thesis can manage.

Existing research described earlier is deficient in that it does not consider energy consumption (ex. [28], [29]), does not consider workloads characterized by multiple stages of execution with precedence relationships (ex. [10], [41]), or does not consider jobs subject to SLAs which include deadlines (ex. [37], [40]). Additionally, much of the existing research focuses only on batch workloads (ex. [30], [34], [36], [13]) and are not suited for use in open systems subject to job arrivals considered in this thesis. This thesis presents energy aware resource management techniques for matchmaking and scheduling MapReduce jobs subject to SLAs which include deadlines with the objective of minimizing data center energy consumption. Two of the proposed techniques (see Chapter 4) are suited for batch workloads and the remaining two (see Chapter 5) are intended for open systems subject to continuous MapReduce job arrivals. To the best of our knowledge, none of the existing techniques concerns constraint programming based energy aware resource management of deadline constrained MapReduce jobs for both batch workloads and open systems subject to continuous job arrivals that this research focuses on.

Chapter 3: Mathematical Model and CP Formulation

This chapter describes the mathematical model used for calculating energy consumption of the system including the representation of cloud resources and MapReduce jobs. The Constraint Programming formulation based on these models for matchmaking and scheduling a set of MapReduce jobs with the objective of minimizing energy consumption is also presented. Mathematical models are described in Section 3.1 and the CP formulation is discussed in Section 3.2.

3.1 Mathematical Models

The energy model used to compute the CPU energy consumption is presented in Section 3.1.1. Accurately calculating energy required to execute a set of MapReduce jobs with a given resource allocation and schedule is critically important for the energy aware resource management techniques described in this research. The system and workload models required for matchmaking and scheduling a set of MapReduce jobs subject to SLAs comprising of user specified execution times and deadlines are described in Section 3.1.2 and Section 3.1.3 respectively.

3.1.1 Energy Model

The Advanced Configuration and Power Interface (ACPI) specification provides an open standard developed by industry which enables operating systems to configure underlying hardware components and describes the use of DVFS to manage CPU power dissipation [27]. ACPI compliant CPUs support a number of discrete voltage/frequency levels and permit the use of DVFS to vary operating frequency and corresponding supply voltage while computation is ongoing. We denote f^{high} and f^{low} as the maximum and minimum CPU operating frequencies respectively. Research performed in [42] shows that typical

DVFS transition times are of the order of tens of nanoseconds. Since these transition times are negligibly small compared to typical MapReduce task execution times, we assume that transitioning between two voltage/frequency levels is instantaneous. Although CPUs allow only a discrete set of fixed frequency/voltage levels, any desired average operating frequency between f^{high} and f^{low} can be obtained by using DVFS to vary the proportion of time each frequency/voltage level is used. The voltage/frequency level, and therefore power dissipation, of each core in a multicore CPU can be varied independently [42].

We adopt the energy model from [36] to determine the power dissipation for each of the frequency/voltage levels permitted by an ACPI compliant CPU. Power dissipation P is the sum of two components: static power dissipation P_s and dynamic power dissipation P_d . Static power dissipation is dependent on hardware architecture whereas dynamic power dissipation is based on system load. As in previous research [36], [43], and [44], static power dissipation is assumed to be proportional to dynamic power dissipation. Therefore, total CPU power dissipation P can be expressed as a function of dynamic power dissipation alone:

$$P = P_s + P_d \quad 3.1$$

$$P = kP_d + P_d \quad 3.2$$

$$P = (1 + k)P_d \quad 3.3$$

Where k is a proportionality constant. Piguet et al. show that k is solely dependent on system architecture and does not depend on the executing programs or system load [43].

Dynamic power dissipation can be expressed as the product of the percentage of active gates (A), the total capacitance load (cap), the square of the supply voltage (V), and the operating frequency (f) [44]:

$$P_d = A * cap * V^2 * f \quad 3.4$$

Note that dynamic power dissipation is dependent on machine characteristics (cap , V , and f) as well as program load which affects the value of A . Researchers in [44] demonstrated that the percentage of active gates for a given program can be reasonably approximated as being independent of CPU frequency and constant for the duration of program execution. Therefore, total energy consumed by the CPU, E , during execution of a given program can be expressed as:

$$P = \delta V^2 f \quad 3.5$$

$$E = \sum_{\Delta\tau} P * \Delta\tau \quad 3.6$$

Where $\Delta\tau$ is the duration the program is executed at a given voltage/frequency level with total power dissipation $P = \delta V^2 f$. The units for power and energy used in this research, denoted W_{scaled} and J_{scaled} respectively, are scaled by the proportionality constant δ which is based on system architecture and machine and program characteristics. Such proportional units are appropriate for this research that does not concern obtaining absolute values of performance and energy metrics but focuses on their general relationship with system and workload parameters. Furthermore, the impact of δ on total energy consumption can be factored out when comparing the performance of two alternative resource management strategies for matchmaking and scheduling the same set of MapReduce jobs on identical cloud environments. More precisely, the relative

performance improvement in energy consumption of the proposed techniques over alternative strategies will be independent of δ .

This research focuses on compute intensive jobs. As in [10] and [11], CPU operating frequency is assumed to be inversely proportional to program execution time. Reducing average CPU operating frequency by half, for example, will cause program execution time to double. This is based on the assumption that overheads such as those caused by cache misses are negligible compared to task execution times. Accounting for such overheads for workloads characterized by smaller task execution times forms an interesting direction for future research.

ACPI compliant CPUs support a number of power and sleep modes, collectively known as C-states [27], to manage energy consumption of idle CPU cores. Idle CPU cores are permitted to enter a power saving mode C1-C6 and can be transitioned back to the operating state C0 when necessary. C1 (commonly known as Halt) is used as the default energy saving mode for idle CPU cores in this research because the transition overhead between Halt state C1 and Operating state C0 is negligible compared to the scale of program execution times. Halted CPU cores are assumed to dissipate a percentage of the power required by cores operating at f^{high} in Operating state C0. This percentage, henceforth referred to as *halt percentage*, h , is used to determine the energy consumed by idle CPU cores. Usage of deeper sleep states (C2-C6) would reduce energy consumption of idle CPU cores and improve system performance, however, the large transition overhead between deep sleep states and the Operating state would impact matchmaking and scheduling decisions.

The proposed DVFS-based technique is suitable for use with conventional single CPU based systems as well as in cloud environments where multiple virtual machines (VMs) share a single physical node. If a single core is dedicated to a VM, the operating frequency of that core can be adjusted to the desired level. If time slicing is used to share a single core among multiple VMs, operating frequency of the respective core can be scaled to the level required by the new VM when a time slice expires and execution of the presently running VM is preempted. Note that because typical DVFS transition times are of the order of tens of nanoseconds [42], CPU cores can be scaled appropriately during context switching that follows VM preemption.

3.1.2 System Model

The system model describes the characteristics of the computing nodes used to execute the MapReduce workload. MapReduce jobs are allocated to and scheduled on the set of m distributed computing nodes or previously acquired cloud resources $R = \{r_1, r_2, \dots, r_m\}$. Resources are modelled with a static configuration of fixed execution slots similar to Hadoop MRv1 [17]; each resource has:

- a fixed number of map slots (map task capacity) c_r^{map} ,
- a fixed number of reduce slots (reduce task capacity) c_r^{red} .

The map task capacity and reduce task capacity of a given resource define the maximum number of map and reduce tasks, respectively, which can be executed in parallel by that resource.

An independent processor model is used to determine the CPU energy consumption of each resource in the set. The processor model includes:

- a set of permitted voltage/frequency levels,
- a halt percentage, h .

Power dissipation for each voltage/frequency level is calculated by Equation 3.5 and used to determine energy consumed by MapReduce task execution, whereas halt percentage is used to determine the energy consumed by idle CPU cores. Each MapReduce task execution slot is mapped directly to a physical CPU core to ensure that operating frequency of each MapReduce task is independent. Following similar research in this area [36], we investigate a homogenous cloud environment where each resource in R operates using the same processor model.

3.1.3 Workload Model

The workload model describes the characteristics of the MapReduce jobs to be executed. Similar to work performed in [21] and [13], the workload consists of a set of MapReduce jobs each of which is subject to an SLA comprising of an execution time and deadline. Each of the n jobs j in the set $J = \{j_1, j_2, \dots, j_n\}$ to be mapped and scheduled on the set of cloud resources has the following characteristics:

- A deadline, d_j , by which the job should be completed.
- An earliest start time, s_j , when the job is permitted to start.
- A set of map tasks $T_j^{map} = \{t_{j,1}^{map}, t_{j,2}^{map}, \dots, t_{j,k_j^{map}}^{map}\}$ where k_j^{map} is the total number of map tasks in job j .

- A set of reduce tasks $T_j^{red} = \{t_{j,1}^{red}, t_{j,2}^{red}, \dots, t_{j,k_j^{red}}^{red}\}$ where k_j^{red} is the total number of reduce tasks in job j .
- A set $T_j = \{T_j^{map}, T_j^{red}\}$ which includes all tasks in job j .

Each task t in the set T_j has a client specified execution time e_t which is defined as the completion time when executed at f^{high} . In line with other work regarding matchmaking and scheduling of jobs with deadlines ([8], [10], [12], [13]), task execution times are assumed to be known. Predicting MapReduce task execution times and accounting for error in execution time estimates is discussed further in Section 6.2. Tasks of all jobs are stored in the set $T = \{T_1, T_2, \dots, T_n\}$.

The general matchmaking and scheduling requirements for a set of n MapReduce jobs subject to SLAs comprising of a client specified earliest start time, execution time, and deadline on a set of m cloud resources are described next. Each task t in the set T must be scheduled on exactly one cloud resource r where it executes until completed. Each task in T_j cannot begin execution before its earliest start time, s_j , and should complete before the deadline for job j , d_j . In general, all tasks in T_j^{map} must complete before tasks in T_j^{red} are permitted to begin execution. Finally, the map and reduce task capacity of each resource in R cannot be exceeded at any point in time.

The Hadoop Slow Start configuration parameter can be used to permit reducers to begin transferring output data from map tasks before all of the job's map tasks have completed thus reducing job completion time [17]. Starting data transfer early is not expected to significantly improve performance in systems executing multiple jobs simultaneously. Reducers transferring data occupy resources and may become blocked waiting for map tasks to finish thereby preventing other jobs' reduce tasks from starting when ready.

Furthermore, systems executing many MapReduce jobs simultaneously may not often have additional available reduce task execution slots due to resource contention between jobs. Therefore, the Slow Start parameter is not used in this research which concerns batch workloads and open streams of job arrivals.

3.2 Constraint Programming Formulation and Implementation

This section describes the base Constraint Programming (CP) formulation and implementation used to model and solve the matchmaking and scheduling problem for a batch of MapReduce jobs subject to SLAs which comprise a client specified earliest start time, execution time, and deadline. The base CP formulation discussed in this section is intended for use with batch MapReduce workloads with soft deadline requirements. In systems with soft deadlines, requests are sometimes permitted to miss their specified deadlines. However, minimizing the number of missed deadlines is important quality of service consideration. A number of resource management techniques described in this research are devised for use in different situations. An example is an open system subject to continuous arrivals of MapReduce jobs. These techniques require minor variations to the CP formulation and implementation that are explained in subsequent chapters.

Soft deadline requirements are considered in the context of batch workloads since rejecting a workload because a few job deadlines are missed is not beneficial for either the service provider or client. Rejecting batch workloads will leave the service provider with fewer opportunities to earn revenue and may result in client dissatisfaction. However, reducing the number of jobs that miss their client specified deadlines is still an important consideration for service providers as it improves QoS and reduces SLA violation penalties. *Missed deadline ratio bound*, db , a QoS parameter, is used to capture

this requirement. Missed deadline ratio bound is defined as the upper bound on the ratio of the maximum number of jobs that are permitted to miss their client specified deadlines to the total number of jobs in the workload.

3.2.1 Constraint Programming Model Formulation

This section describes the CP formulation used to model and solve the matchmaking and scheduling problem with the objective of minimizing the energy required to execute the workload. The matchmaking and scheduling problem is modeled and solved in a joint step using CP. Recall from Section 2.2 that CP is a form of declarative programming wherein the properties of the solution to be found are specified in the form of inviolable constraints. A CP formulation has three major components: decision variables—the set of unknown variables to solve for, an objective function—the function which produces the value to be optimized (maximized or minimized) based on selected decision variable values, and constraints—the set of rules which specify the requirements of decision variable values for feasible solutions. The CP formulation does not specify the process of finding a feasible solution, rather, the formulation describes the desired characteristics of the solution to be found. One shortcoming of CP is that only discrete decision variables are permitted. For continuous variables, such as time, a higher degree of precision in scheduling decisions can be achieved with a change of units. For example, 1.234 s can be expressed as 1234 ms.

The CP formulation used to model and solve the matchmaking and scheduling problem for a set of MapReduce jobs characterized by SLAs which include a client specified earliest start time, execution time, and deadline such that CPU energy consumption is minimized is presented in Table 3.1. Recall from the description of the matchmaking and

scheduling model presented in Section 3.1.3 that a set of n jobs $J = \{j_1, j_2, \dots, j_n\}$ are allocated to and scheduled on a set of m cloud resources $R = \{r_1, r_2, \dots, r_m\}$. Furthermore, the tasks of all jobs are stored in the set $T = \{T_1, T_2, \dots, T_n\}$. The constraints presented in Table 3.1 are specifically designed for use with MapReduce jobs. However, a more general solution for multi-stage workflows with additional stages of execution is possible following the same approach.

The CP model has the following decision variables:

- A binary variable x_{tr} for each combination of t in T and r in R . x_{tr} is assigned a value of one if task t is assigned to resource r and a value of zero otherwise. These decision variables are used for matchmaking.
- An integer variable a_t for each task which specifies the scheduled start time for task t . The start time of each task is one component of the task scheduling operation.
- An integer variable c_t for each task which specifies the execution duration of task t . Note that execution duration will vary based on CPU frequency. The execution duration is the second component of task scheduling and can be used to derive task completion time and CPU operating frequency during execution.

A binary variable N_j , which is set to 1 if job j misses its deadline and set to 0 otherwise.

There is an N_j variable for each job in J .

A description of the CP formulation presented in Table 3.1 is provided. The objective of the CP formulation is to generate an output schedule, which includes the allocation of tasks to resources, scheduled start time of tasks, and scheduled end time of tasks, that minimizes the energy required to execute the set of MapReduce jobs. The calculation of

energy consumption, shown in the CP formulation objective function (see first row of Table 3.1), is broken down into two components: Task Energy and Idle Energy.

Table 3.1: Base CP Formulation Constraints and Objective Function

<i>minimize $E_{Task} + E_{Idle}$ subject to</i>	
$\sum_{r \in R} x_{tr} = 1 \forall t \in T$	(1)
$(a_t \geq s_j \forall t \in T_j^{map}) \forall j \in J$	(2)
$(a_t \geq \max_{t \in T_j^{map}} (a_t + c_t) \forall t \in T_j^{red}) \forall j \in J$	(3)
$(cumulative((a_t x_{tr} = 1), (c_t x_{tr} = 1), 1, c_r^{map}) \forall t \in T_r^{map}) \forall r \in R$	(4)
$(cumulative((a_t x_{tr} = 1), (c_t x_{tr} = 1), 1, c_r^{red}) \forall t \in T_r^{red}) \forall r \in R$	(5)
$\left(\max_{t \in T_j^{red}} (a_t + c_t) > d_j \Rightarrow N_j = 1 \right) \forall j \in J$	(6)
$\sum_{j \in J} N_j \leq J * db$	(7)
$(x_{tr} \in \{0,1\} \forall t \in T) \forall r \in R$	(8)
$N_j \in \{0,1\} \forall j \in J$	(9)
$\left(c_t \in \mathbb{Z} \mid e_t \leq c_t \leq e_t \frac{f^{high}}{f^{low}} \right) \forall t \in T$	(10)
$a_t \in \mathbb{Z} \forall t \in T$	(11)

Task Energy is the energy consumed by cloud resources to execute tasks. Energy required to execute a given task t is calculated as $E_{(for\ task\ t)} = P(f^{high} * e_t / c_t) * c_t$ where $P(x)$ is a function which gives CPU power when operating at frequency x .

The expression E_{Task} , the total energy consumption of all tasks in the workload, is shown in Equation 3.7.

$$E_{Task} = \sum_{t \in T} P\left(\frac{e_t}{c_t} f^{high}\right) * c_t \quad 3.7$$

Recall from Section 3.1.3 that e_t is the client specified execution time for task t which is defined as its execution time when executed at f^{high} .

Constraints (1) to (7) are based on the CP formulation for “Approach 3” specified in [21] and are modified to suit the research goals of minimizing energy consumption. These constraints enforce the general matchmaking and scheduling requirements described at the end of Section 3.1.3. Constraint (1) ensures that each task is allocated to exactly one resource using the matchmaking decision variables (x_{tr}). Thus, the sum of the matchmaking decision variables for each task (i.e. $x_{t1} + x_{t2} + \dots + x_{tm}$) must be exactly equal to one. To enforce the requirement that tasks in T_j cannot begin execution before s_j , Constraint (2) iterates over all tasks in T_j^{map} and forces their scheduled start time to be at or after s_j . Constraint (3) ensures that no tasks in the set T_j^{red} begin execution until all tasks in the set T_j^{map} have completed. This is achieved by iterating over all tasks in the set T_j^{red} and requiring that their scheduled start time is at or after the completion time of the latest finish task in the set T_j^{map} which is calculated using the max function.

Constraints (4) and (5) ensure that the resource capacity of each resource in R is not exceeded at any point in time. These constraints are expressed using the global constraint cumulative. The cumulative constraint requires two input parameters: a collection of tasks and a limit. Each task in the collection has an associated start time, duration, and resource requirement. The cumulative constraint ensures that the sum of the resource requirements of tasks in the collection which have an overlap in their scheduled execution time do not exceed the specified limit at any point in time. The resource requirement of each task is set to one because each task executes on exactly one slot.

As discussed previously, in systems with soft deadline requirements reducing the number of jobs that miss their client specified deadlines is an important consideration for service providers as it improves client satisfaction and reduces SLA violation penalties. Constraints (6) and (7) together guarantee that the output schedule meets a specified grade of service by requiring the ratio of jobs that miss their deadlines to total number of jobs to be less than the *missed deadline ratio bound*, db . Constraint (6) is used to determine which jobs miss their deadlines. If the end time of any task in t_j^{red} is greater than d_j , N_j is set to one. Constraint (7) requires that the number of jobs which miss their deadlines is less than the cardinality of J multiplied by db .

Constraints (8) to (11) specify the domain of the decision variables and are used to determine their respective range of valid values. Constraint (8) specifies that each of the x_{tr} matchmaking decision variables is a binary variable indicating whether or not task t is allocated to resource r . Constraint (9) specifies that each of the N_j decision variables is a binary variable indicating whether or not the deadline of job j is met. Constraint (10) specifies that each of the execution duration decision variables is an integer with a minimum value of e_t and a maximum value based on e_t and the ratio between f^{high} and f^{low} . Finally, Constraint (11) specifies that the start time of each task is an integer.

Verification of the base CP formulation presented in Table 3.1 is performed by performing matchmaking and scheduling manually for very small workloads and comparing the output with the allocation and schedule determined by the CP Optimizer. Additional verification of correct operation of the CP Optimizer for the derived CP formulation is achieved by permitting all jobs to miss their deadlines and varying the halt percentage used to determine idle CPU energy consumption. For low values of halt

percentage, the CP Optimizer is expected to schedule tasks to execute with the lowest permitted CPU operating frequency. For high values of halt percentage, the CP Optimizer is expected to schedule tasks to execute with the highest permitted CPU operating frequency such that they complete as quickly as possible. The results of all such tests indicated correct system operation.

3.2.2 Constraint Programming Model Implementation

The CP model formulation described in Section 3.2.1 for matchmaking and scheduling a set of MapReduce jobs with deadline constraints such that the energy required to execute the workload is minimized is implemented and solved using the commercially available optimization software IBM ILOG CPLEX Optimization Studio 12.6 [45]. The CP model formulation is expressed in Java using IBM ILOG CPLEX Concert Technology (referred to as *concert*) and solved using the CPLEX CP Optimizer Java API (referred to as *CP Optimizer*) [45].

Concert allows the resource management algorithms to call IBM ILOG CPLEX directly through the Java Native Interface (JNI) and supplies a rich interface for building optimization models using Java objects. The use of concert allows for a more compact and less complex representation of the CP model. The model is then solved using the CP Optimizer constraint programming solving engine which provides specialized variables, constraints, and expressions for modeling and solving scheduling problems efficiently [46].

The implementation takes advantage of these specialized mechanisms provided by CP Optimizer to efficiently solve the matchmaking and scheduling problem by reducing the memory and processing time requirements [47]. The CP Optimizer provides a composite

decision variable called *interval* specifically for scheduling problems. An interval decision variable represents a period of time during which an activity or task is performed whose position in time is an unknown in the scheduling problem. An interval is characterized by five attributes: 1) start time, 2) duration, 3) end time, 4) optionality, and 5) intensity. The optionality property indicates whether an interval must be present for the solution to be valid. Optional intervals may be used to represent activities that can execute on one of a set of alternative resources and are important for allocating tasks to resources. The intensity property models the resource usage or utility of a task during its interval as a stepwise integer function. For example, a construction worker who works a half-day on Wednesday and does not work on Sunday will have an intensity of 50% and 0% on those days respectively. For this worker 7 man-days of work will require more than 7 days to complete. The use of interval decision variables allows for optimized library expressions and constraints to further improve solver efficiency [47].

The CP model formulation matchmaking and scheduling decision variables, x_{tr} , a_t , and c_t , are implemented using the CPLEX interval data type. Each task t in the set T is expressed as an interval decision variable. Variable interval size is used to capture the potential increase in task execution time due to a decrease in execution frequency as a result of DVFS. Minimum interval size is defined as the execution duration at the maximum execution frequency, i.e. the user specified task execution time, e_t . Maximum interval size is defined as the execution duration at the minimum execution frequency and can be expressed as $\left\lceil e_t * \frac{f^{high}}{f^{low}} \right\rceil$. The ceiling function is used because only whole number interval sizes are permitted by CPLEX.

Ideally DVFS would be modeled using the interval intensity property rather than the interval size. Interval intensity when executing at the maximum possible frequency would be 100% and intensity at other permitted frequency levels would be calculated proportional to f^{high} . This approach would enable the output schedule to include the sequence and duration of frequencies required to execute each task thereby improving ease of scheduling based on solver output. Furthermore, the approach would be beneficial when considering an open stream of MapReduce job arrivals as it would allow for easier rescheduling of in progress tasks to accommodate additional arriving jobs. Unfortunately, a limitation in the design of the CP Optimizer does not permit stepwise integer functions which model interval intensity to be used as decision variables. Therefore, the approach is not currently possible [46].

The implementation of several constraints in the CP formulation is briefly described. An interval decision variable for each combination of task and resource in the system is used to implement the CP formulation matchmaking decision variables, x_{tr} . These matchmaking intervals are marked as *optional*, which indicates that they are not required to be present for the final solution to be feasible. As mentioned previously, optional intervals are suitable for scheduling problems which include allocation of tasks to one of a set of alternative resources. Constraint (1) is implemented using the optimized CPLEX `alternative` constraint for interval scheduling. The `alternative` constraint is a synchronization constraint with two input parameters: an interval a , and a set of optional intervals B . The `alternative` constraint specifies that if interval a is present in the solution, exactly one interval from B (denoted b) will also be present in the solution and the start and end time of intervals a and b will be synchronized. In this implementation,

an `alternative` relationship exists between each interval decision variable for task t in T and the subset of optional interval decision variables that correspond to the allocation of task t to each resource in R . This use of the `alternative` constraint ensures that each task is assigned to execute on exactly one resource.

Constraints (4) and (5) are implemented using the `pulse` function. `pulse` is a cumulative function expression which represents the contribution of an individual interval and covers the usage of renewable resources when an activity acquires resources at start time and releases resources when complete. Thus, `pulse` is suitable for modeling the resource usage of tasks which require resources only during execution. The `pulse` function has two input parameters: an interval a to represent the task, and a height h which represents the task resource requirement. Resource requirement for all tasks in this implementation is set to 1 because each task executes on exactly one task execution slot. The `pulse` function returns a value as a function of time which represents the resource usage of the task. When the task is active (i.e. between the start time and the end time of the interval) the `pulse` function returns a value of h . For all other points in time, the `pulse` function returns a value of 0 to represent that the task is not contributing to resource usage.

To implement Constraint (4), the `sum` expression is used to add the contribution of the subset of optional matchmaking decision variables which correspond to map tasks that could be allocated to a given resource r as generated using the `pulse` function. When invoked with an optional interval that is absent from the solution, indicating that the task is not allocated to r , the `pulse` function returns a value of 0 for all points in time. Thus, this summation only captures the resource requirement of map tasks which are allocated

to resource r . The less than or equal constraint is used to ensure that the sum of the resource requirement of map tasks executing on resource r does not exceed the map task capacity, c_r^{map} , at any point in time. This is repeated for each r in R . Implementation of Constraint (5) is similar except resource usage of reduce tasks and reduce task capacity are used.

Once the matchmaking and scheduling problem is solved by the CP Optimizer and the feasible solution which minimizes the energy consumption of the set of n MapReduce jobs with deadline constraints is found, values necessary to generate the task allocation and schedule are obtained using the CPLEX Concert element interface [45]. Task execution duration is used to determine the average CPU operating frequency f_t during execution of each task t where $f_t = f^{high} * \frac{e_t}{c_t}$. If this average operating frequency does not match directly with one of the frequency levels permitted by the DVFS enabled CPU, the exact required operating frequency can be achieved by varying the proportion of time permitted frequency levels are using during task execution.

For example, a task with a minimum execution time of 9 s has a scheduled execution duration of 12 s on a resource with a CPU with two permitted operating frequencies: 1000 MHz and 500 MHz. The required average operating frequency of 750 MHz can be achieved by executing the task at 1000 MHz for 6 s and 500 MHz for 6 s. This approach can be used because typical DVFS transition times are on the order of tens of nanoseconds which are negligibly small compared to MapReduce task execution times [42].

The final output schedule consists of the resource allocation, start time, and execution frequency of each task in the workload. This information is used to schedule the workload for execution on the set of available cloud resources.

3.2.2.1 CP Optimizer Processing Time

There are many factors which impact the processing time required for the CP Optimizer to determine the solution to the matchmaking and scheduling problem which minimizes energy consumption. As discussed earlier in Section 2.2, CP problems are solved by combinatorial search algorithms. Because these algorithms depend on a state space search, solving very complex scheduling problems to determine the optimal solution may require an extremely long processing time. Due to the size and complexity of the matchmaking and scheduling problem, in particular the use of DVFS to vary operating frequency during execution of all tasks, the CPLEX CP Optimizer is not always able to find the optimal solution or prove that a given feasible solution is optimal in a reasonable amount of time. Due to how algorithms search the solution space, it is possible that the true optimal solution is found early in the search and is set as the incumbent. However, the algorithm will not complete until the incumbent solution is proven to be optimal by exploring all or as many branches as required. CP Optimizer uses a variation of constructive search [23] with constraint propagation to solve CP models.

The CPLEX CP Optimizer incorporates additional techniques to reduce processing time. For example, the CP Optimizer utilizes multiple concurrent worker threads to take advantage of multicore processor technology to increase solve speed [46]. CP Optimizer also supports a number of node selection strategies that control the order in which branches are explored and may improve the processing time based on the scheduling

problem. The default node selection strategy used in the CP Optimizer is depth-first search with restart [45]. This search is similar to the traditional depth-first search in that it explores a branch down to the fully enumerated solutions before considering other branches. However, if the search encounters a large number of infeasible solutions the search will backtrack to avoid wasting further time on what could be an entirely infeasible branch. This type of search is appropriate for large and complex scheduling problems with large solution spaces because a depth-first approach improves the chances that a feasible incumbent solution is found quickly. Once an incumbent solution is found, branches with poor expected objective function quality can be avoided to reduce the size of the search space.

If necessary, CP Optimizer processing time can be further reduced by setting the solver to terminate before the optimal solution is found if one or more specified search conditions is reached. The CP Optimizer can be configured to terminate after a specified number of branches are explored, infeasible solutions are enumerated, or time has elapsed [45]. If the solver terminates this way, the incumbent solution is returned as the best found solution during the allowed processing time. Furthermore, there is no guarantee on the solution optimality when the CP Optimizer search is terminated based on a limit. However, this concession is sometimes necessary to ensure that the solver returns a feasible solution within a reasonable time frame.

Preliminary experimentation with the medium batch workload used in [21] showed that the CP Optimizer was unable to determine the optimal solution to the matchmaking and scheduling problem in a reasonable amount of time. Experiments were run for as long as one full week without terminating normally and were subsequently cancelled.

Matchmaking and scheduling a typical batch workload or open stream of MapReduce job arrivals requires much lower processing times.

Analysis of the CP Optimizer behavior when solving the matchmaking and scheduling problem shows that a high quality solution is found early in the search. When the solver is initially started, the incumbent solution changes quickly as new feasible solutions with better objective function values are found. However, as the solver continues, many candidate solutions are enumerated without finding a solution with a better objective function value. These results suggest that the mechanism used by the constructive search algorithm to estimate the best-case objective function value of a branch for this CP model substantially overestimates objective function quality. Recall from Section 2.2 that poor estimates of objective function quality lead to exhaustive search because the searches algorithm is unable to avoid exploring many or all branches in the solution space.

To address this issue and reduce the required processing time to acceptable levels, the CP Optimizer was configured to only enumerate a fixed number of infeasible solutions before terminating and returning the incumbent solution. The maximum number of enumerated infeasible solutions is configured using the CP Optimizer *fail limit*. The results for matchmaking and scheduling a single batch workload generated with the workload parameters for a “medium” workload specified in [21] using various fail limits are shown in Figure 3.1. Resource management for the generated workload is performed repeatedly using increasing fail limit to determine the impact of increased CP Optimizer processing time on solution quality. The energy consumption values shown in Figure 3.1 are normalized based on the energy consumption of the worst solution found by the CP Optimizer. As expected, processing time increases with increased fail limit. Normalized

energy consumption decreases rapidly as fail limit is initially increased but quickly approaches a knee value. Beyond this point, further increasing fail limit does not substantially reduce the normalized energy consumption—solution quality does not significantly increase beyond this point for the tested workload and fail limit values. The results show that the performance improvement of the solution found after 1000 seconds is negligible compared to the solution found after only 80 seconds. Although not guaranteed to be optimal, the CP Optimizer can be configured to deliver high quality solutions in a reasonable time frame and be made suitable for solving the matchmaking and scheduling problem for typical batch workloads and open streams of job arrivals by greatly limiting the permitted processing time.

Figure 3.1 - Impact of Fail Limit on Normalized Energy Consumption

Chapter 4: Energy Aware Resource Management Techniques for Batch Workloads

This chapter discusses the energy aware resource management techniques designed for batches of MapReduce jobs each of which are subject to an SLA comprising of a client specified earliest start time, a task execution time, and a soft deadline. As discussed previously in Section 3.2, jobs are sometimes permitted to miss their client specified deadlines in systems with soft deadline requirements. However, reducing the number of missed deadlines is an important consideration in matchmaking and scheduling the workload. The energy aware batch resource manager for minimizing energy consumption is described in Section 4.1. Section 4.2 presents the configurable batch resource manager that achieves an effective tradeoff among batch completion time, energy consumption, and number of missed deadlines for deadline constrained MapReduce batch workloads.

4.1 Energy Aware Batch Resource Manager

This section presents the *Energy Aware MapReduce Batch Resource Manager*, *BRM*, in detail. An overview of the system, including resource manager design, as well as the simulation based performance evaluation conducted on the resource manager are discussed. A paper containing a discussion of BRM was published in a refereed international workshop [14].

4.1.1 Overview

The energy aware MapReduce batch resource manager performs matchmaking and scheduling for batches of MapReduce jobs each of which is subject to an SLA comprising of a client specified execution time, and soft deadline. In systems with such soft deadlines, requests are sometimes permitted to miss their specified deadline.

However, minimizing the number of requests which miss their deadline, thereby minimizing the number of SLA violations, is an important quality of service consideration which impacts both client satisfaction and service provider profit.

BRM makes use of DVFS to reduce energy consumption by decreasing the CPU operating frequency during execution of some tasks. Careful selection of the level of CPU frequency reduction and the tasks to which it is applied permits the resource manager to take advantage of available slack time in the execution window of a request and reduce energy consumption without sacrificing service quality.

The matchmaking and scheduling problem is formulated as a CP model and solved in a joint step using the CPLEX CP Optimizer as described in Chapter 2. No variations on this base CP formulation or implementation are required for matchmaking and scheduling a batch of deadline constrained MapReduce jobs with the objective of minimizing energy consumption.

4.1.2 Performance Evaluation

This section contains a detailed analysis of the impact of various system and workload parameters on system performance. In line with related work (e.g. [8], [28], and [12]) a simulation-based approach is used to conduct the performance evaluation. A simulated system and workload allows for flexibility in systematically varying system and workload parameters. Simulation is suitable for this research which does not concern obtaining exact values of performance and energy metrics but focuses on their general relationship with system and workload parameters and relative improvement compared to alternative strategies.

The remainder of this section is organized as follows: the experimental setup is described in Section 4.1.2.1, the performance metrics used to compare BRM with alternative strategies are discussed in Section 4.1.2.2, the system and workload parameters for simulation based performance analysis are presented in Section 4.1.2.3, and the cloud resource processor model is described in Section 4.1.2.4.

4.1.2.1 Experimental Setup

The performance evaluation of the proposed MapReduce batch resource manager was accomplished by conducting the required resource management decisions for matchmaking and scheduling various deadline constrained MapReduce batch workloads. Experiments simulate the scheduling and execution of batch workloads on a closed system to evaluate the performance achieved by the resource manager. The CP model is generated based on the input batch workload and solved using the CPLEX CP Optimizer to determine the resource allocation and scheduled execution time of each task in the workload. Workload execution is simulated using the output task allocation and schedule to determine the batch completion time and energy consumption.

Note that, as discussed in Section 3.2.2.1, due to the size and complexity of the matchmaking and scheduling problem, the CPLEX CP Optimizer is unable to prove that a solution is optimal in a timeframe that is appropriate for resource management of typical batch workloads. To ensure that a feasible solution is available within a reasonable amount of time, the CP Optimizer is configured to terminate after a specified time limit. The output schedule is not guaranteed to be optimal but is the best solution found within the limited time. An appropriate time limit was determined through preliminary experimentation. The knee of the curve of processing time vs. solution

quality (refer to Figure 3.1 in Section 3.2.2.1 for example) was found to occur when processing time of the CPLEX CP Optimizer is limited to 1% of the total execution time of all tasks in the workload. Experiments with lower time limits yielded significantly worse results whereas experiments with higher time limits seemed to yield negligible improvement in system performance.

The experiments were performed on a PC running under Windows 7 Professional with a 3.2GHz Intel Core i5 CPU and 4GB of RAM. The energy-aware MapReduce batch resource manager can be expected to generate a solution with lower required energy consumption when running the CP Optimizer on a PC with a faster CPU and more memory. However, these improvements have been observed to be negligibly small for tested processing time limit and input workloads.

4.1.2.2 Performance Metrics

The performance of the energy aware MapReduce batch resource manager is evaluated on the basis of energy required to execute the workload, system utilization and throughput, and deadline compliance. System performance of the BRM is compared to that of alternative techniques using the following three performance metrics:

Energy consumption (E): the total energy consumed to execute the batch workload.

Batch Completion time (C): the time at which the last task in the batch workload finishes execution.

Number of missed deadlines (N): the total number of jobs in the batch workload that miss their client specified deadlines.

Each of the performance metrics is calculated based on the final output schedule obtained from the CPLEX CP Optimizer using the CPLEX Concert element interface.

The energy required to execute the batch workload is simply the objective function value of the CP Optimizer solution, but can also be verified based on the output schedule by manually calculating the energy required to execute each task in the batch and the idle energy consumed by each resource. The energy consumed by the execution of a given task t is calculated using the task execution duration c_t and execution frequency f_t where $E_{(for\ task\ t)} = c_t * P(f_t)$. Recall from Section 3.2.1 that $P(x)$ is a function which gives CPU power when operating at frequency x . The idle energy consumption of a given resource r is based on the batch completion time, C , the resource task slot capacity, and the execution duration of all tasks allocated to r . The equation for idle energy consumption can be expressed as $E_{(for\ resource\ r)} = P(f^{high}) * h * (C * (c_r^{map} + c_r^{red}) - \sum_{t \in T} x_{tr} * c_t)$ where power dissipation while halted is a percentage, defined as halt percentage, h , of the power dissipation of a CPU core operating at f^{high} . Idle energy consumption for a given resource r is calculated as the product of idle power consumption and idle CPU core duration. As discussed previously in Section 3.2.1, idle CPU core duration for a given resource can be found by multiplying batch completion time by resource task slot capacity and subtracting the total execution duration of all tasks in T which are assigned to that resource (i.e. the sum of all c_t where $x_{tr} = 1$).

Batch completion time is obtained by calculating the completion time of the latest finishing task in the workload from the output schedule. Since a job's map tasks must finish execution before its reduce tasks can begin, the latest finishing task in the workload is guaranteed to be a reduce task. Completion time of a given task t is calculated as the sum of the scheduled start time a_t and scheduled execution duration c_t . The batch

completion time, C , is the maximum of the completion time of all reduce tasks in the workload.

The number of missed deadlines is computed from the CP Optimizer output schedule. The completion time of each job is calculated in a manner similar to the overall batch completion time. Again, only reduce tasks need to be considered in this calculation since the latest finishing task of each job is guaranteed to be a reduce task. For each job j in the workload, if the job completion time exceeds the client specified deadline d_j , the job has missed its deadline and the variable tracking the number of missed deadlines is incremented.

Performance of the proposed MapReduce batch resource manager (referred to as BRM) is compared with “Approach 3” (referred to as System I) from [21], a resource management approach for batch workloads subject to SLAs that include soft deadlines with the primary objective of minimizing the number of missed deadlines. System I uses a similar CP formulation to solve the matchmaking and scheduling problem but does not support the use of DVFS to vary the operating frequency during task execution and does not consider energy consumption when making resource management decisions.

Total energy consumption for the batch workload, obtained either from the objective function value of the CP Optimizer solution or calculated manually based on the output schedule, is scaled by the unknown proportionality constant δ as discussed in Section 3.1.1. To prevent δ from affecting performance results, the comparison of energy consumption between System I and BRM is expressed as *Energy Savings*, $E_{savings}$, of BRM over System I. The expression for energy savings is given by:

$$E_{savings} = \left(1 - \frac{E_{(for\ BRM)}}{E_{(for\ SystemI)}}\right) * 100\% \quad 4.1$$

This expression permits the unknown proportionality constant δ , which is based on system architecture and machine and program characteristics, to be factored out of the energy consumption comparison.

4.1.2.3 System and Workload Parameters

Table 4.1 presents the system and workload parameters used in simulation based experiments to compare the performance of System I and BRM. This synthetic workload is similar to those used by other researchers (see [28] for example) and is adapted from the small and medium workloads in [21]. Synthetic workloads permit easier investigation into the impact of various workload conditions on system behavior due to the flexibility and control over input parameters.

Table 4.1 - System and Workload Parameters

	Jobs (s_j and d_j in seconds)	Task execution times (in seconds)	Resources
medium	$n = 10$: $s_j \sim \text{DU}(1,50)$ $d_j = \lceil s_j + e_j^{tot} * l \rceil$ $k_j^{map} = 10$ $k_j^{red} = 5$	$e_t^{map} \sim \text{DU}(1,25)$ $e_t^{red} \sim \text{DU}(1,75)$	$m = 15$: $c_t^{map} = 2$ $c_t^{red} = 2$
small	$n = 5$: $s_j \sim \text{DU}(1,50)$ $d_j = \lceil s_j + e_j^{tot} * l \rceil$ $k_j^{map} = 10$ $k_j^{red} = 3$	$e_t^{map} \sim \text{DU}(1,15)$ $e_t^{red} \sim \text{DU}(1,75)$	$m = 10$: $c_t^{map} = 2$ $c_t^{red} = 2$

The resource characteristics and workload attributes for both the medium and small workload are described in Table 4.1. The earliest start time of each job and the minimum execution time of each task are generated using discrete uniform distributions. The discrete uniform distribution, presented in Table 4.1 using the notation $\text{DU}(a,b)$, produces

a uniformly distributed integer between a and b inclusive. Similar distributions are used to generate characteristics of MapReduce jobs in [21] and [28].

In practice, job deadlines will be specified by the client. However, in order to conduct the performance analysis of the proposed MapReduce batch resource manager, a deadline is generated for each job in the synthetic workload. Job deadlines are generated using the expression for d_j in Table 4.1 and are based on the job earliest start time, s_j , the execution time of all tasks in the job, e_j^{tot} , and the job *laxity factor*, l .

In general, laxity is a measure of the available slack time in the execution window of a job. Jobs with high laxity have relaxed deadlines, whereas jobs with low laxity have stricter deadlines. The expression for laxity is given in Equation 4.2. Laxity factor is defined as one plus the ratio of the laxity of job j to the sum of execution times of all tasks in j as shown in Equation 4.3.

$$Laxity(job\ j) = d_j - e_j^{tot} - s_j \quad 4.2$$

$$l = 1 + \frac{laxity}{e_j^{tot}} \quad 4.3$$

Typically, laxity of a job is calculated from the earliest start time, execution time, and deadline as a measure of the strictness of its deadline requirement. Conversely, deadlines of the desired strictness for a synthetic workload can be generated using an appropriate laxity factor. For each experiment, a fixed laxity factor is used to generate the deadline of all jobs in the workload. Workloads with a higher laxity factor allow for a greater degree of freedom in the scheduling of tasks due to relaxed job deadlines. The range of laxity factor values used to generate job deadlines for various experiments is presented in Table 4.2.

As discussed previously in Section 4.1.1, the energy aware MapReduce batch resource manager performs matchmaking and scheduling of workloads characterized by soft deadline requirements. However, reducing the number of jobs which miss their deadlines is an important quality of service consideration for both clients and service providers. This quality of service parameter is captured by the missed deadline ratio bound, db , which defines the ratio of the maximum number of jobs that are permitted to miss their deadlines to the total number of jobs in the workload. In practice, this parameter could be set by either the client or service provider. Performance of the energy aware MapReduce batch resource manager is investigated for the range of values of missed deadline ratio bounds presented in Table 4.2.

Table 4.2 - Workload Parameter Variations

Workload Parameter	Tested Values
Laxity factor, l	{0.4, 0.5, 0.6 , 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5}
Missed deadline ratio bound, db	{0.0, 0.1 , 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}

The resource management process does not depend significantly on the size of input data. As input data size increases, there is a corresponding increase in either number of map tasks or map task execution times. Performance for various input data sizes can be investigated by varying either the mean of the discrete uniform distribution which determines map task execution time or the number of map tasks.

The performance of the energy aware resource manager was evaluated for small and medium scale workloads. Because the energy aware resource manager takes advantage of excess slack in the execution window of jobs to reduce the processor operating frequency during the execution of some tasks it was believed that workloads with a small number of tasks may not benefit substantially from this approach. Investigation of the performance for small workloads was deemed worthwhile to investigate this hypothesis. Resource

management for the large workload specified in [21] is discussed for the configurable batch resource manager in Section 4.2.

4.1.2.4 Processor Description

The processor model used in the BRM experimentation is based on the frequency/voltage levels for the Intel Core i7-2760QM presented in [11] and shown in the first two columns of Table 4.3. The Intel Core i7-2760QM is a recent multicore consumer class processor that could be used in private clusters executing MapReduce workloads [48]. Experimental results for the DVFS-based approach using alternative processor models, including models based on processors from different manufacturers and server class processors more common in large data centers, are presented in subsequent chapters. The power dissipation of each frequency/voltage level is calculated using Equation 3.5 and shown in the “Power” column of Table 4.3. Note that because δ is not known, power dissipation is expressed in proportional unit W_{scaled} .

Table 4.3 - Intel Core i7-2760QM Power Measurements

Frequency (MHz)	Voltage (V)	Power (W_{scaled})
2400	1.060	2696.64
2000	0.970	1881.8
1600	0.890	1267.36
800	0.760	462.08

A linear regression of the power dissipation in each frequency/voltage level is used to express power dissipation as a function of operating frequency:

$$P = \delta * (1.364f - 741.81) \quad 4.4$$

The coefficient of determination for this linear regression was found to be $R^2 = 0.9699$. Note that the power estimate presented in Equation 4.4 is based on empirical measurements of processor supply voltage performed by researchers in [11] and is valid

only for frequencies between f^{high} and f^{low} . Equation 4.4 is used by the CPLEX CP Optimizer to determine the objective function value of candidate solutions and can be used to manually calculate the energy required by each task and idle energy consumption of each resource based on the output schedule.

4.1.3 Experimental Results

This section describes the experimental results for the MapReduce batch resource manager. An investigation into the impact of changes to system and workload parameters is performed using a *factor-at-a-time* approach. System performance is observed while one parameter is varied and the other parameters are kept at their default values. The default value of each parameter is shown in boldface in the “Value” column of Table 4.2. Halt percentage, used to determine energy consumption of idle CPU cores, is set to 5% for all experiments. This value was selected based on the minimum permitted supply voltage (0.25 V) for the Intel Core i7-2760QM given in [49]. To ensure statistically significant results, each experiment is repeated a number of times until a confidence interval on the energy savings at a 95% confidence level with upper and lower bounds within $\pm 5\%$ of the mean is achieved. The confidence level and interval are computed by using the student’s t -distribution based formulation presented in [50] with the results of the multiple experimental runs. Each experimental run used a unique starting seed value for the random number generator leading to the generation of a different workload based on the given set of workload parameters.

Recall from Section 4.1.2.2 that the energy consumption of the proposed MapReduce batch resource manager (BRM) is expressed relative to that of a *non-energy-aware*

MapReduce resource manager (referred to as System I) based on “Approach 3” from [21].

4.1.3.1 Impact of Laxity Factor

The impact of varying laxity factor, l , on batch completion time and energy savings for the medium workload is shown in Figure 4.1. It can be seen that increasing laxity factor results in an increase in energy savings and batch completion time for BRM.

Figure 4.1 - Impact of l on C and $E_{savings}$ for Medium Workload

As laxity factor increases, the slack time in the execution window of jobs in the workload increases and each job in the workload can be completed later without incurring additional deadline violations. BRM takes advantage of this slack by reducing execution frequency of some tasks to minimize energy consumption. Energy savings and batch completion time for BRM do not significantly increase for values of laxity factor above 1.0. Additional slack does not seem to further reduce energy consumption or increase batch completion time for BRM once all tasks in the workload are executing at the

minimum frequency. In all experiments for laxity factor the number of jobs that miss their deadlines is less than the maximum number permitted by the default value of missed deadline ratio bound ($db=0.1$). However, few or no deadlines need to be missed to fully reduce CPU frequency during execution of all tasks for high values of laxity factor.

Batch completion time for System I is observed to be independent of laxity factor. Because the objective of System I is to minimize the number of missed deadlines and DVFS is not used to decrease the energy required to execute the workload, the batch workload is completed as quickly as possible. Relaxing job deadlines by increasing laxity factor has no impact on how System I schedules the workload.

4.1.3.2 Impact of Missed Deadline Ratio Bound

The impact of varying missed deadline ratio bound, db , on batch completion time and energy savings for the medium workload can be seen in Figure 4.2. As expected, the results show that as missed deadline ratio bound increases, batch completion time and energy savings for BRM increase.

The tasks belonging to jobs that will miss their deadlines can be executed at the minimum permitted operating frequency without incurring additional SLA violation penalties. Since the job deadline will be missed regardless, completion time is further delayed by BRM to reduce energy consumption. As more jobs are permitted to miss their deadlines, additional tasks can be executed at a reduced operating frequency and energy savings and batch completion time increase. Energy savings and batch completion time for BRM do not significantly increase for missed deadline ratio bound values above 0.5. As db increases, the resource manager is able to distribute task execution over a longer period of time and reduce resource contention. When resource contention is low, the majority of

tasks can be executed at low CPU operating frequency. Further increasing the number of jobs permitted to miss their deadlines does not seem to further reduce energy consumption or increase batch completion time for BRM once all tasks in the workload are executing at the minimum frequency.

A missed deadline ratio bound value of 1 implies that deadline constraints are not a concern and jobs are executed on a best effort basis as is the case in bulk data analytics. Figure 4.2 shows that the proposed technique can achieve an up to 45% reduction in energy consumption for these types of workloads. For typical batch workloads where deadline constraints are a concern for service providers and a missed deadline ratio bound between 0.0 and 0.2 would be more acceptable, BRM still saves a substantial amount of energy compared to System I.

Figure 4.2 - Impact of db on C and $E_{savings}$ for Medium Workload

Although the output schedule for BRM results in a later batch completion time in all cases compared to System I this does not indicate a reduction in service quality because the total number of missed deadlines never exceeds the limit specified by the missed

deadline ratio bound. A summary of the tradeoff between increase in batch completion time and energy savings is presented in Table 4.4. Increase in batch completion time for BRM with respect to System I can be expressed as:

$$\Delta C = C_{(for\ BRM)} - C_{(for\ System\ I)} \quad 4.5$$

Table 4.4 - Batch Completion Time and Energy Savings Tradeoff

Increase in Batch Completion Time (s)	110	125	130	145	160
Energy Savings (%)	25	30	33	40	42

BRM takes advantage of the relaxation of the grade of service parameter to save energy by reducing execution frequency of some tasks thereby delaying completion of the workload. Note that BRM offers substantial energy savings compared to System I even when few or no deadlines are permitted to be missed.

The results show that the batch completion time for System I is not dependent on missed deadline ratio bound. This behavior is expected because the primary objective of System I is to minimize the number of jobs which miss their client specified soft deadlines. Increasing the number of jobs permitted to miss their deadlines does not impact the scheduling decisions of System I—jobs will be scheduled by System I such that the minimum number of deadlines are missed regardless of db .

4.1.3.3 Impact of Workload Size

The impact of varying laxity factor, l , on batch completion time and energy savings for the small workload can be seen in Figure 4.3. Refer to Table 4.1 in Section 4.1.2.3 for a description of the small workload. A similar trend compared to the medium workload can be observed: increasing laxity factor causes a corresponding increase in both energy savings and batch completion time for BRM.

It is interesting to note that the knee of the energy savings graph is observed at laxity factor approximately equal to 0.7 for the small workload compared to approximately 1.0 in the medium workload. After this point, further increasing the laxity factor does not substantially impact either energy savings or batch completion time of BRM. It seems that at this knee point the majority of workload tasks are scheduled to execute at the minimum allowed processor operating frequency and further increasing laxity in job deadlines has only marginal impact on performance. The observation that a lower laxity factor is required to execute the majority of tasks in the small workload at the minimum permitted operating frequency suggests that although fewer resources are used, the resources are more lightly loaded in the small workload experiments compared to in the medium workload experiments.

Figure 4.3 - Impact of l on C and $E_{savings}$ for Small Workload

A feasible solution could not be found using either System I or BRM when laxity factor was set to 0.4. For low values of laxity factor, it is possible that the minimum execution time of the longest task in a job is longer than the job's permitted execution window. In

this case the job cannot be scheduled without missing its deadline. If the system is unable to meet the requirement of a given d scheduling the workload will not be possible. As discussed in Section 4.1.3.1 regarding the medium workload, increasing the laxity factor does not increase the batch completion time for System I.

The impact of varying missed deadline ratio bound, db , on batch completion time and energy savings can be seen in Figure 4.4. A similar trend as in the case of the medium workload is observed: increasing db leads to an increase in energy savings and batch completion time for BRM. Again, batch completion time for System I is observed to be independent of missed deadline ratio bound.

Figure 4.4 - Impact of db on C and $E_{savings}$ for Small Workload

The possible energy savings and relative increase in batch completion time for BRM are smaller in the small workload than observed in the medium workload when both laxity factor (see Figure 4.1 and Figure 4.3) and missed deadline ratio bound (see Figure 4.2 and Figure 4.4) are varied. Because the small workload contains significantly fewer tasks compared to the medium workload, there are substantially fewer opportunities to save

energy by reducing the processor frequency during task execution. Furthermore, because of the lighter workload, idle energy is a higher proportion of total energy consumption. Since idle energy is not reduced using this technique, the total possible energy savings is reduced. Resource management for larger batch workloads is discussed for the configurable batch resource manager in Section 4.2.

4.1.3.4 Number of Missed Deadlines

For experiments where laxity factor is varied, the missed deadline ratio bound is fixed to the default value $db=0.1$. When analyzing the impact of number of missed deadlines, missed deadline ratio bound is varied between 0 and 1. It should be noted that because the primary objective of System I is to minimize the number of missed deadlines rather than reduce energy consumption, fewer jobs will miss their deadlines under System I than BRM. However, in both cases, the ratio of jobs that miss their deadlines to total number of jobs in the workload will never exceed the grade of service parameter missed deadline ratio bound. Furthermore, db specifies an upper limit to the number of jobs which are permitted to miss their deadlines. In some cases, (specifically for high values of laxity factor) fewer jobs than permitted will need to miss their deadlines under BRM in order to minimize energy consumption due to excessive slack in job deadlines.

4.1.4 Summary of Results

The energy aware MapReduce batch resource manager performs matchmaking and scheduling for a batch workload of deadline constrained MapReduce jobs. The technique makes use of available slack time in the execution window of jobs and applies a DVFS-based processor frequency reduction to the execution of some tasks to reduce energy consumption *without violating SLAs*. Performance analysis demonstrates energy savings

between 25% and 45% for varying values of missed deadline ratio bound and laxity factor using the DVFS-based approach compared to previous resource management approaches which do not consider energy. Smaller, but still substantial, energy savings are observed for workloads with fewer total tasks. The energy savings are accompanied by small to moderate increases in batch completion times.

4.2 Configurable Batch Resource Manager

This section describes the details of the *Configurable MapReduce Batch Resource Manager*, *C-BRM*. An overview of the system, design of the resource manager, performance evaluation, and experimental results are discussed. A paper containing a discussion of the configurable batch resource manager is accepted for publication in a refereed international conference [15].

4.2.1 Overview

The proposed configurable resource management technique is for matchmaking and scheduling of batch workloads comprising of MapReduce jobs each of which are characterized by multiple stages of execution and are subject to SLAs that comprise of a client specified earliest start time, execution time, and (soft) deadline.

Energy consumption, batch completion time, and number of SLA violations are three important metrics for cloud service providers. The first affects the data center operation cost directly whereas the second and third are intimately related with revenue earned. A lower batch completion time enables the service provider to run more batches per unit time thereby increasing revenue. Having a small number of SLA violations (as captured by the ratio of jobs that miss their deadlines to total number of jobs) is important for minimizing the penalty associated with violating SLAs as well as increasing customer

satisfaction which aids in customer retention. Service provider profit can be increased by achieving an effective tradeoff among these three correlated parameters.

Most existing resource management algorithms focus on the optimization of a single metric: energy consumption (e.g. [10], [37], [13]), number of missed deadlines (e.g. [8], [21]), or batch completion time (e.g. [28]). The configurable resource manager considers several performance metrics when making scheduling decisions and minimizes energy consumption while guaranteeing that service provider specified upper bounds on missed deadline ratio and batch completion time are satisfied.

The proposed technique reduces energy consumption by applying a DFVS based technique to reduce the CPU operating frequency during execution of some tasks. Reducing CPU operating frequency will increase batch completion time and may reduce the number of batch workloads a data center can accept thereby limiting service provider revenue. Furthermore, aggressively reducing operating frequency of all tasks to save energy may increase the number of jobs which miss their deadlines. Service providers unwilling to accept the implications of SLA violations or an unconstrained increase in batch completion times can configure the resource manager with an upper bound on missed deadline ratio and an upper bound on batch completion time. The configurable batch resource manager will determine the output schedule which minimizes energy consumption while satisfying all service provider and client scheduling requirements.

An overview of the proposed technique is shown in Figure 4.5. A client submits a batch workload comprising of deadline-constrained MapReduce jobs to the resource manager which executes on a physical or virtual resource in the cloud. Acceptable upper bounds on missed deadline ratio and batch completion time are set as configuration parameters

by the service provider. The matchmaking and scheduling problem is formulated using CP and solved using the IBM CPLEX CP Optimizer [45]. This commercially available software uses constructive search with constraint propagation to solve the CP formulation as discussed in Section 2.2.2. The output schedule, which comprises of a scheduled start time, operating frequency, and resource allocation for each task in the workload, is then transferred to the cloud resources.

Figure 4.5 - Configurable Batch Resource Manager System Overview

4.2.2 Resource Manager Design

Similar to the energy aware batch resource manager, the configurable batch resource manager formulates the matchmaking and scheduling problem using CP and solves the CP formulation using the CPLEX CP Optimizer. This section describes the changes to the base CP formulation and implementation discussed in Chapter 2 required by the configurable batch resource manager.

Matchmaking and scheduling of input batch workloads is performed in accordance with the service provider specified resource manager configuration parameters. Configuration

parameters include an upper bound on performance metrics (batch completion time and missed deadline ratio). We denote *batch completion time bound*, cb , as the upper bound on batch completion time, and *missed deadline ratio bound*, db , as the upper bound on missed deadline ratio. Both of these metrics of interest may also be unbounded if desired by the service provider.

The configurable MapReduce batch resource manager is devised for handling larger workloads than used in Section 4.1 with the energy aware MapReduce batch resource manager. Because the energy aware MapReduce batch resource manager CP formulation requires a matchmaking decision variable for every combination of task and resource, poor optimizer performance is possible for systems with a large number of present tasks and resources. When tested with large workloads, BRM executes slowly, runs out of memory, and sometimes crashes due to the large number of decision variables created for matchmaking. To address this problem, the C-BRM performs matchmaking and scheduling independently rather than in one joint step: the CP Optimizer performs the scheduling operation based on a modified CP formulation and an additional matchmaking algorithm is devised to allocate tasks to resources using the output schedule.

Similar to research described in [12], the scheduling operation is performed first by considering the resources in R as a single resource pool with combined map and reduce task execution slots equal to the total number of map and reduce task execution slots among all resources. The CP Optimizer schedules the batch workload to run on this combined resource pool and determines an output schedule which includes the start time and execution duration of each task.

Matchmaking of tasks in the output schedule is then performed using a best fit approach which considers each execution slot independently. Tasks in the output schedule are sorted by increasing start time and assigned to execution slots such that the gap between the completion of the previous task and the start of the new task is minimized. It is important to note that map tasks may only be assigned to map task execution slots and reduce tasks may only be assigned to reduce task execution slots. After all tasks in the workload are allocated to task execution slots, a solution which contains the number of resources in R is generated. Task slots are randomly assigned to resources until each resource is assigned a number of map and reduce task execution slots equal to its capacity.

Table 4.5 - Configurable Batch Resource Manager CP Formulation

$minimize E_{Task} + E_{Idle} \text{ subject to}$	(a)
$(a_t \geq s_j \forall t \in T_j^{map}) \forall j \in J$	(2)
$(a_t \geq \max_{t \in T_j^{map}} (a_t + c_t) \forall t \in T_j^{red}) \forall j \in J$	(3)
$(\max_{t \in T_j^{red}} (a_t + c_t) > d_j \Rightarrow N_j = 1) \forall j \in J$	(4)
$(cumulative((a_t x_{tr} = 1), (c_t x_{tr} = 1), 1, c_r^{map}) \forall t \in T_r^{map}) \forall r \in R$	(5)
$(cumulative((a_t x_{tr} = 1), (c_t x_{tr} = 1), 1, c_r^{red}) \forall t \in T_r^{red}) \forall r \in R$	(6)
$\sum_{j \in J} N_j \leq J * db$	(7a)
$\max_{t \in T} (a_t + c_t) \leq cb$	(8a)
$N_j \in \{0,1\} \forall j \in J$	(9)
$(c_t \in \mathbb{Z} \mid e_t \leq c_t \leq e_t \frac{f^{high}}{f^{low}}) \forall t \in T$	(10)
$a_t \in \mathbb{Z} \forall t \in T$	(11)

Due to this performance optimization, the configurable MapReduce batch resource manager CP formulation does not require the set of x_{tr} decision variables used for matchmaking by BRM described in Section 4.1. However, the sets of a_t and c_t decision variables for scheduling and N_j decision variables for missed deadlines are still needed by the CP Optimizer to schedule the input workload. The configurable MapReduce batch resource manager uses the base CP formulation described in Chapter 2 with the following changes: 1) the matchmaking constraints are no longer required, and 2) an additional constrain which limits batch completion time to be less than the specified upper bound on batch completion time is needed. The CP formulation is shown in Table 4.5.

Constraint (7) and Constraint (8) support the functionality of the configurable MapReduce batch resource manager. Constraint (7) ensures that the resulting schedule meets the required grade of service by limiting the proportion of jobs which are permitted to miss their deadlines. Constraint (8) guarantees the batch will complete before the specified upper bound on completion time, cb . The explanations of the remaining constraints are the same as for the base CP formulation discussed in Section 3.2. For a full discussion on the CP formulation and implementation using CPLEX CP Optimizer, refer to Chapter 3. Verification of the changes to the CP formulation required by the configurable batch resource manager is performed using an approach similar to that described for the base CP formulation in Section 3.2.1.

4.2.2.1 Alternative Resource Manager Functionality

With only minor modifications to the CP formulation presented in Table 4.5 and the introduction of an upper bound on energy consumption, eb , it is possible to permit service providers to selectively minimize energy consumption, batch completion time, or missed

deadline ratio using the configurable MapReduce batch resource manager. The additional objective functions and constraints necessary for this selective operation are shown in Table 4.6. To minimize batch completion time subject to upper bounds on energy consumption and missed deadline ratio, for example, (a) and Constraint (8a) from Table 4.5 are replaced with (b) and Constraint (8b) from Table 4.6 respectively.

Table 4.6 - Alternative CP Formulations

Minimization of Batch Completion Time	
$minimize \max_{t \in T} (a_t + c_t) \text{ subject to}$	(b)
$E_{Task} + E_{Idle} \leq eb$	(8b)
Minimization of Missed Deadline Ratio	
$minimize \sum_{j \in J} N_j \text{ subject to}$	(c)
$E_{Task} + E_{Idle} \leq eb$	(7c)

The difficulty with this proposed alternative functionality is in generating reasonable upper bounds on energy consumption. Because energy consumption is dependent on unknown proportionality constant δ , which is based on hardware and software characteristics, actual energy consumption to execute a batch cannot be determined without extensive offline batch profiling to determine the value of δ for each task in the workload. Therefore, without performing such offline operations *a priori*, it is difficult to determine reasonable upper bounds on energy consumption for a workload. However, this problem can be avoided if the service provider permits the energy consumption to be unbounded and the proposed technique can be used to minimize batch completion time subject to a given bound on db or minimize missed deadline ratio subject to a given bound on cb .

One possible solution to this problem is to permit the service provider to specify the upper bound on energy consumption as a minimum permitted energy savings compared to the largest reasonable energy consumption. An estimate of the maximum energy required to execute a workload can be obtained by configuring the resource manager to schedule the workload such that completion time is minimized subject to unbounded energy consumption with DVFS disabled.

4.2.3 Performance Evaluation

This section describes the simulation based performance analysis conducted for the configurable MapReduce batch resource manager. Use of a simulated system and workload, similar to work performed in [31], [39], and [33], permits easier investigation into the impact of system and workload parameters on overall performance.

4.2.3.1 Experimental Setup

Results for matchmaking and scheduling various input workloads were used to analyze performance of the configurable MapReduce batch resource management technique. The CP Optimizer and matchmaking routine perform matchmaking and scheduling on a single PC running Windows 7 Professional with a 2.93GHz Intel Core i7 CPU and 8GB of RAM to determine the output schedule of input workloads. Note that additional processing power and memory are required for C-BRM compared to BRM described earlier due to the larger workloads investigated. Task execution on the cloud environment is then simulated using the output schedule to determine energy usage, batch completion time, and missed deadline ratio.

As discussed previously in Section 3.2.2.1, due to the size and complexity of the formulated CP model, the CPLEX CP Optimizer is unable to prove that a given solution

is optimal using branch and bound techniques in a reasonable amount of time. To ensure a solution is generated in an appropriate timeframe, the CP Optimizer processing time is limited to 5% of the total execution time of all tasks in the workload. Note that processing time for the configurable MapReduce batch resource manager is limited to 5% of the total execution time of all tasks compared to the 1% time limit used for the energy aware MapReduce batch resource manager discussed in Section 4.1. This increase in processing time is necessary due to the increase in the size of investigated workloads. Because the size of the solution space does not increase linearly with the size of the problem, solution quality cannot be maintained by simply linearly scaling processing time with workload size. That is to say, the time required to obtain a high quality solution to the matchmaking and scheduling problem does not scale linearly with problem size.

4.2.3.2 Performance Metrics

Similar to the performance metrics described in Section 4.1.2.2 for the energy aware MapReduce batch resource manager, the performance of the configurable MapReduce batch resource manager is evaluated based on the following metrics: energy consumption E , batch completion time C , and number of missed deadlines N . Number of missed deadlines can also be expressed as *Proportion of Missed Deadlines*, D , which is defined as the fraction of jobs which miss their deadlines relative to total number of jobs in the workload.

Performance of the proposed configurable MapReduce batch resource manager (referred to as C-BRM) is compared to “Approach 3” from [21] (referred to as System I). System I is a CP-based resource manager that does not consider energy consumption when making matchmaking and scheduling decisions. More specifically, System I determines an output

schedule which minimizes the proportion of missed deadlines and does not support the use of DVFS to vary CPU operating frequency.

Because total energy consumption for the batch workload is scaled based on the unknown proportionality constant δ as discussed in Section 3.1.1 and Section 4.1.2.2, energy consumption for System I and C-BRM are compared based on Energy Savings, $E_{savings}$, of C-BRM over System I. The expression for energy savings which permits unknown proportionality constant δ to be factored out of the comparison is given in Equation 4.6:

$$E_{savings} = \left(1 - \frac{E_{(for\ C-BRM)}}{E_{(for\ SystemI)}}\right) * 100\% \quad 4.6$$

4.2.3.3 System and Workload Parameters

Performance of the configurable MapReduce batch resource manager is evaluated using a synthetic workload similar to those used by other researchers (e.g. [21], [28], [41]) and is adapted from workload Large 2 in [21]. Workload and system parameters used to analyze performance of the resource manager are presented in Table 4.7. The systems parameters section of Table 4.7 describes the cloud resources and the details of the resource manager configuration used to schedule the workload. Suitable upper bounds on batch completion time depend on workload and system parameters. Upper bounds to be used in simulation are calculated using an approximation of the minimum and maximum batch completion time for a given workload. The expressions for these approximations are given in the following equations:

$$c_{min} = \frac{n * average\ job\ execution\ time}{m * (c_r^{map} + c_r^{red})} \quad 4.7$$

$$c_{max} = c_{min} * f^{high} / f^{low} \quad 4.8$$

Where *average job execution time* is calculated using the mean number of map and reduce tasks with mean task execution times (computed from the distributions shown in Table 4.7). Recall from Section 3.1.2 that m denotes the number of pre-acquired resources available in the system and c_r^{map} and c_r^{red} denote the map and reduce task capacity respectively of each resource. The number of jobs in the batch is denoted by n whereas f^{high} and f^{low} are the maximum and minimum permitted CPU operating frequency respectively. c_{diff} (used in Table 4.7) is the difference between c_{max} and c_{min} . In some cases, upper bounds on batch completion time are rounded slightly to whole numbers for convenience.

Table 4.7 - Configurable Batch Resource Manager System and Workload Parameters

Name	Value
System Parameters	
Number of resources, m -Map task capacity, c_r^{map} -Reduce task capacity, c_r^{red}	$m = \{25, \mathbf{50}, 75, 100\}$ $c_r^{map} = 2, c_r^{red} = 2$
Bounds -Completion time bound, cb -Missed deadline ratio bound, db	$cb = \{c_{min}, c_{min} + 0.25c_{diff}, \mathbf{c_{min} + 0.5c_{diff}}, c_{min} + 0.75c_{diff}, c_{max}\}$ $db = \{0, 0.1, \mathbf{0.2}, 0.3, 0.4\}$
Workload Parameters	
Number of jobs n:	$n = \{10, 20, \mathbf{30}, 40, 50\}$
Job j: -Number of map tasks, k_j^{map} -Number of reduce tasks k_j^{red}	$k_j^{map} \sim DU(1, 100)$ $k_j^{red} \sim DU(1, k_j^{map})$
Task execution times (s) -Map task, me -Reduce task, re	$me \sim DU(1, e^{max})$ where $e^{max} = \{5, \mathbf{10}, 20\}$ $re = \frac{\left(\sum_{t \in T_j^{map}} e_t\right)}{k_j^{red}}$
Job start time, s_j (s)	<i>batch start</i>
Job deadline, d_j (s)	$d_j = s_j + (l * \sum_{t \in T_j} e_t)$ where $l = \{0.4, 0.8, \mathbf{1.2}, 1.6\}$

The workload parameters section of Table 4.7 describes the characteristics of the workload to be executed. Batch size is defined in the first row. Similar to [21] and [28], discrete uniform distributions are used to generate the number of map and reduce tasks in a job as well as the execution time of all map and reduce tasks in the workload. All jobs are free to begin execution once the batch starts. In a production environment, job deadlines will be specified by the client. However, for simulation-based performance analysis, *laxity factor*, l , is used to generate job deadlines as discussed in Section 4.1.2.3. As in [21], a job deadline is computed as the sum of two components: (i) the earliest start time for the job and (ii) the product of l and the sum of the execution time of all tasks in the job.

The goal of the experiments is to use various workloads with different characteristics such as the size of the batch, mean length of tasks, laxity factor, and number of resources to analyze the impact of system and workload parameters on performance. The smallest batch with 10 jobs has, on average, 500 map tasks and 250 reduce tasks. The largest batch with 50 jobs has, on average, 2500 map tasks and 1250 reduce tasks. Experimental workloads are representative of typical MapReduce batch workloads and are similar in size to workloads used in [21] and [13].

4.2.3.4 Processor Description

The CPU model used in this paper to determine task and idle energy consumed by the batch workload is based on measurements of the frequency/voltage pairs for the AMD Opteron 2380 performed by researchers in [11]. Experimentation with a variety of processor models demonstrates that the efficacy of the proposed DVFS-based technique for reducing energy consumption is not dependent on the use of a particular processor

model. These measurements are presented in column 1 and 2 of Table 4.8. Column 3 presents the power requirement of each frequency/voltage level calculated using Equation 3.5. Note that because δ is not known, power dissipation is expressed in proportional unit W_{scaled} .

Table 4.8 - AMD Opteron 2380 Power Measurements

Frequency (MHz)	Voltage (V)	Power (W_{scaled})
2500	1.300	4225.00
1800	1.200	2592.00
1300	1.100	1573.00
800	1.025	840.50

A linear regression of the values presented in column 3 was used to express power as a function of frequency as shown in Equation 4.9.

$$P(f) = \delta * (2.01f - 909.19) \quad 4.9$$

The coefficient of determination for this linear regression is $R^2 = 0.9903$. Note that this expression for power is based on empirical measurements of CPU supply voltage performed by researchers in [11] and is valid only for values between f^{low} and f^{high} .

4.2.4 Experimental Results

Experiments are performed using a *factor-at-a-time* methodology to analyze the impact of various system and workload parameters on overall performance. System performance is evaluated while one parameter is varied and the remaining parameters are held at their default values. The default value of each parameter is shown in boldface in the “Value” column of Table 4.7. In all experiments, halt percentage, h , is set to 10% based on the power measurements for the AMD Opteron 2380 from [51]. Each experiment was repeated a sufficient number of times to generate a confidence interval for energy savings at a 95% confidence level with upper and lower bounds for the intervals within $\pm 5\%$ of the mean.

Matchmaking and scheduling is performed offline by the resource manager by invoking the CP Optimizer once per batch workload to solve the CP formulation. An algorithmic complexity analysis of the constructive search with constraint propagation technique used by the CP Optimizer to solve CP formulations cannot be performed due to the proprietary nature of the software. However, as discussed earlier in Section 3.2.2.1, the maximum solve time can be limited to ensure a solution is available in a reasonable time frame. For, all experiments performed, the maximum solve time was limited to be less than or equal to 5% of the total execution time of all tasks in the workload.

Recall from Section 4.2.3.2 that the energy consumption of the proposed configurable MapReduce batch resource manager (C-BRM) is expressed relative to that of a *non-energy-aware* MapReduce resource manager (referred to as System I) based on “Approach 3” from [21].

4.2.4.1 Impact of Batch Completion Time Bound

The impact of varying the upper bound on batch completion time, cb , on Energy Savings, $E_{savings}$, and batch completion time, C , is shown in Figure 4.6.

When the upper bound on batch completion time, cb , is low Energy Savings is limited because C-BRM is unable to significantly reduce the CPU operating frequency using DVFS while completing the batch before the specified batch completion time. As cb is relaxed, $E_{savings}$ increases because C-BRM is able to use a lower CPU operating frequency during execution of more tasks while meeting the required batch completion time bound. No further improvement in $E_{savings}$ is observed after cb reaches 300.

The main objective of the proposed resource management technique is to minimize energy required to execute a batch of MapReduce jobs. Although secondary, lower batch

completion times can lead to higher throughput and increased revenue for cloud service providers. As expected, batch completion time, C , increases for both System I and C-BRM as the cb increases. C for C-BRM increases because the resource manager takes advantage of the additional time to reduce the CPU operating frequency during execution of tasks where possible to minimize energy consumption. Relaxing cb results in higher batch completion times for System I because the optimizer will terminate when a satisfactory schedule that minimizes the proportion of missed deadlines is found. Since all schedules with the same proportion of missed deadlines, regardless of batch completion time, are considered equally good by System I, other feasible schedules with smaller batch completion times may be overlooked. Note that although C-BRM results in a higher C compared to System I in all cases, SLA compliance is not sacrificed.

Figure 4.6 - Impact of cb on $E_{savings}$ and C for 30 job batch

4.2.4.2 Impact of Missed Deadline Ratio Bound

The impact of varying the upper bound on missed deadline ratio, db , on Energy Savings, $E_{savings}$, and proportion of missed deadlines, D , is shown in Figure 4.7.

There is approximately 35% savings in energy achieved by C-BRM for all values of db . For the default values of system and workload parameters used in this experiment, $E_{savings}$ is largely unaffected by variations in the upper bound on missed deadline ratio. It seems that the default value of laxity factor is sufficiently relaxed to permit a substantial reduction in energy consumption without incurring any missed deadlines. Although there is little variation in $E_{savings}$, D for C-BRM increases as db increases. C-BRM only requires that D is less than the upper bound on missed deadline ratio, db , and tries to take advantage of the relaxed constraint to save energy. System I is independent of variations in db because the resource manager objective is to minimize the number of jobs which miss their deadlines.

Figure 4.7 - Impact of db on $E_{savings}$ and D for 30 job batch

The relationship between db and $E_{savings}$ was investigated for values of laxity factor lower than the default value presented in Table 4.7. The impact of db on $E_{savings}$ when $l=0.15$ is shown in Table 4.9. When laxity factor is low and job deadlines are tight, increasing db increases $E_{savings}$ because there is insufficient slack to substantially reduce energy consumption without missing job deadlines. C-BRM saves energy by reducing the CPU frequency during execution of tasks belonging to jobs that miss their deadline. When laxity factor is extremely low, System I and C-BRM may be unable to determine a feasible schedule where few or no deadlines are missed (see Table 4.9 for $db=0$ for example). Energy savings cannot be calculated in these cases.

Table 4.9 - Effect of db on $E_{savings}$ for $l=0.15$

db	0.0	0.2	0.4	0.6	0.8	0.9	1.0
$E_{savings}$ (%)	N/A	11.3	18.2	24.2	31.7	36.1	36.4

4.2.4.3 Impact of Laxity Factor

The impact of laxity factor, l , on $E_{savings}$ and proportion of missed deadlines, D , for System I and C-BRM is presented in Figure 4.8. When system and workload parameters are set to their default values, Energy Savings does not change significantly as laxity factor is varied. D for both System I and C-BRM decreases as laxity factor increases. As job deadlines are relaxed and more slack time is available, schedules where fewer jobs miss their deadlines are possible.

Figure 4.8 - Impact of l on $E_{savings}$ and D for 30 job batch

The effect of using values of laxity factor lower than 0.4 on $E_{savings}$ is shown in Table 4.10. When l is low and job deadlines are tight, Energy Savings is low because C-BRM must execute tasks at a high operating frequency to ensure that enough jobs meet their specified deadlines to satisfy the upper bound on missed deadline ratio. As laxity factor is increased, $E_{savings}$ is observed to increase rapidly. As captured in Figure 4.8, increasing l beyond 0.4 does not increase $E_{savings}$. For extremely low values of laxity factor System I and C-BRM may be unable to determine a feasible solution to the matchmaking and scheduling problem (see Table 4.10 for $l=0.1$ for example).

Table 4.10 - Effect of Low l on $E_{savings}$

l	0.1	0.15	0.2	0.25	0.3	0.35	0.4
$E_{savings}$ (%)	N/A	12.9	26.5	29.0	30.4	33.2	34.4

4.2.4.4 Impact of Number of Resources

The impact of number of resources on $E_{savings}$ and C is shown in Figure 4.9. As expected for both C-BRM and System I, batch completion time decreases with an increase in the

number of resources, m . Energy Savings has a non-monotonic relationship with number of resources. When the number of resources is low, C-BRM is unable to use DVFS to reduce the CPU operating frequency of most tasks while completing the batch before the required deadline. As m is increased from low to moderate values the DVFS-based approach of C-BRM works well to minimize the energy required to execute the workload and $E_{savings}$ is high. Once there are a sufficient number of resources available to reduce the CPU frequency during execution of all tasks in the workload, adding additional resources will not further decrease energy consumption. Additional resources may reduce C , but will also incur additional idle energy consumption. Since idle energy cannot be reduced using the DVFS-based technique, energy savings decreases as m is increased beyond $m=75$. As expected C for System I and C-BRM decreases as m is increased.

Figure 4.9 - Impact of m on $E_{savings}$ and C for 30 job batch

Figure 4.10 - Impact of cb on $E_{savings}$ and C for 30 job batch with $m=25$

The observation of low Energy Savings for $m=25$ motivated further investigation. Figure 4.10 shows the effect of varying the upper bound on batch completion time for a 30 job batch scheduled on 25 resources. Significant energy savings of up to 40% compared to System I are possible when larger batch completion times are permitted. As shown in Figure 4.10, C for System I and C-BRM increases as cb increases. The rationale is similar to that provided for Figure 4.6 in Section 4.2.4.1.

4.2.4.5 Impact of Map Task Execution Time

The impact of the upper bound on map task execution time, e^{max} , on $E_{savings}$ and C is presented in Figure 4.11. Energy Savings decreases from approximately 40% to approximately 10% as e^{max} increases from 5 to 20. When e^{max} is low, C-BRM is able to significantly reduce the execution frequency of tasks to save energy while still meeting the required bound on batch completion time. As e^{max} increases, C-BRM must execute tasks at a higher frequency to guarantee that the upper bound on batch completion time is

satisfied. Batch completion time, C , increases as e^{max} increases for both System I and C-BRM.

Additional experiments for $e^{max}=20$ were performed. Substantial energy savings were possible for $e^{max}=20$ when using higher values of cb .

Figure 4.11 - Impact of e^{max} on $E_{savings}$ and C for 30 job batch

4.2.4.6 Impact of Number of Jobs

The impact of number of jobs, n , on $E_{savings}$ and C is shown in Figure 4.12. Recall from Section 4.2.3.3 that the upper bound on batch completion time for experimentation is calculated using c_{min} (see Equation 4.7) and c_{max} (see Equation 4.8) which are based on the number of jobs in the batch. A suitable upper bound on batch completion time for a 30 job batch would result in a scheduling problem that is overly relaxed for small batch workloads and infeasible for large batch workloads. Therefore, a suitable upper bound on batch completion time is calculated individually for each workload size as the midpoint

between c_{min} (see Equation 4.7) and c_{max} (see Equation 4.8). cb for each value of n is indicated on Figure 4.12 using data labels.

Figure 4.12 - Impact of n on $E_{savings}$ and C

Figure 4.12 shows that the C-BRM is able to save a significant amount of energy compared to System I regardless of workload size. The proposed DVFS-based technique scales well from small workloads with several hundred tasks (10 job batch) to large workloads with many thousand tasks (50 job batch). Number of jobs does not significantly affect $E_{savings}$. As expected, batch completion time for System I and C-BRM increases as n increases.

4.2.5 Determining Suitable Configuration Parameters

Section 4.2.3.3 discusses how the upper bounds on batch completion time and missed deadline ratio are selected for performance analysis, however, in practice these configuration parameters are to be selected by cloud service providers based on their requirements. Service providers may select configuration parameters to maximize profit based on system and workload parameters and external factors.

To select an appropriate upper bound on batch completion time, service providers may consider: the size of the batch workload (i.e. the number of jobs and execution time of tasks), the number of resources in the system, the job deadlines, the relationship between energy consumption and batch completion time, and the cost of energy, which may depend on other factors such as time of day. Furthermore, service providers may also consider any potential conflict between the current workload and previously accepted batch workloads which are scheduled to start in the future and the probability that the service provider will be unable to accept a batch which arrives before execution of the current workload is complete. When selecting an appropriate bound on missed deadline ratio, service providers may consider the laxity available in job deadlines, the impact of missed deadline ratio on energy consumption, the penalty for violating client SLAs requirements, and the cost of a potential reduction in future business due to lower client satisfaction as a result of missed deadlines. Selecting appropriate configuration parameters based on these and other system and workload parameters and external factors forms an interesting direction for future research.

4.2.6 Summary of Results

The proposed Configurable MapReduce Batch Resource Manager, C-BRM, is suitable for scheduling batch workloads of MapReduce jobs subject to SLAs and enables service providers to make resource management decisions based on multiple performance and energy metrics. The configurable resource manager presented in this chapter enables cloud service providers to achieve a desired tradeoff between energy consumption (that directly affects operating costs), missed deadline ratio (governing the quality of service) and batch completion time (governing the revenue earned by cloud service providers).

Results of experiments based on the system and workload parameters described in this paper have led to insights into system behavior and performance that include the following:

- Energy Savings, $E_{savings}$, can be increased at the cost of increased batch completion time (see Figure 4.6).
- Except for when very low values of laxity factor are used, increasing the proportion of jobs which are permitted to miss their deadlines does not increase $E_{savings}$ for system and workload parameter values used in the experiments.
- $E_{savings}$ exhibits a non-monotonic relationship with number of resources. Increasing m causes $E_{savings}$ to initially increase, however, $E_{savings}$ decreases for values of m above 75 (see Figure 4.9).
- The proposed technique is observed to scale well and provides significant Energy Savings independent of workload size (See Figure 4.12)

Chapter 5: Energy Aware Resource Management Techniques for Open Systems

This chapter discusses the energy aware resource management techniques devised for open systems that are subject to arrivals of MapReduce jobs each of which characterized by an SLA comprising of a client specified task execution time, earliest start time, and hard deadline. Unlike the soft deadline requirements discussed previously in Chapter 3, systems with hard deadline constraints guarantee that accepted jobs will complete before their deadlines. Rejecting jobs which are not guaranteed to complete before their deadline is a common approach in existing literature focusing on resource management for open systems subject to deadline constrained job arrivals (See [10], [52] for example). The *Energy Aware MapReduce Resource Manager, EAMR-RM*, for open systems is discussed in Section 5.1. Section 5.2 presents the *Hybrid Energy Aware MapReduce Resource Manager, HEAMR-RM*, for open system which incorporates additional energy savings techniques to reduce energy consumption due to idle resources.

5.1 Energy Aware MapReduce Resource Manager for Open Systems

This section describes the details of the *Energy Aware MapReduce Resource Manager for open systems EAMR-RM*. The system overview, resource manager design, and performance evaluation are described. A paper based on this research has been accepted for publication in a refereed international conference [16].

5.1.1 Overview

The proposed Energy Aware MapReduce Resource Manager for open systems is devised for matchmaking and scheduling an open stream of MapReduce jobs characterized by SLAs such that energy consumption is minimized. Unlike the batch resource managers

described in Chapter 3, EAMR-RM considers hard deadline constraints. Jobs which are accepted by the system are guaranteed to meet their client specified deadlines. In rare cases the workload and state of resources may prevent the system from guaranteeing that a given arriving job will complete before its deadline. In such cases, the job will be rejected by the system; all accepted jobs are guaranteed to complete before their specified deadlines.

Much of the existing research on MapReduce resource management has focused on improving the average job turnaround time (e.g. [28], [29]). Job turnaround time is the time taken to fulfill a request, measured from job earliest start time to completion, and is an important performance metric for open systems. Job turnaround time for job j can be expressed as:

$$\text{Turnaround Time}_j = \text{Completion Time}_j - \text{Earliest Start Time}_j \quad 5.1$$

Turnaround time minimization is not necessarily the best strategy for data center operators who are only obligated to deliver services by the agreed deadline specified in the SLA. These strategies fail to consider optimization opportunities which can reduce data center energy consumption. EAMR-RM leverages a DVFS based approach to scale CPU performance and energy consumption dependent on system load. The approach makes use of slack time in the execution window of a request by applying a reduced CPU frequency during the execution of some tasks. Careful selection of the level of CPU frequency reduction and the tasks to which it is applied ensures that energy consumption is reduced without violating SLAs.

Similar to the resource managers previously discussed in Chapter 3, EAMR-RM formulates the complex matchmaking and scheduling problem as an optimization

problem using CP. The CPLEX CP Optimizer is used to solve the CP formulation such that energy required to execute the input workload is minimized.

A general overview of EAMR-RM is presented in Figure 5.1. MapReduce jobs submitted by clients are placed in a job queue. When the resource manager is not busy, matchmaking and scheduling of all jobs in the queue is performed by invoking the EAMR-RM algorithm (described in Section 5.1.2.1). The resource management algorithm allocates and schedules all tasks on the cloud environment by invoking the EAMR-RM algorithm. Scheduling is performed using the CPLEX CP Optimizer to solve the new CP formulation (presented in Table 5.2 in Section 5.1.2.2) before the matchmaking algorithm (presented in Section 5.1.2.3) allocates tasks to resources. The current schedule is updated to reflect the changes due to the new job arrival.

Figure 5.1: EAMR-RM System Overview

The proposed technique is best suited for use in clusters and private clouds with pre-acquired resources dedicated to MapReduce workloads or in public clouds with dedicated MapReduce services. This thesis focuses on MapReduce workloads; for public clouds with multitenancy, additional energy saving mechanisms should be applied to reduce the energy consumption of resources allocated to non-MapReduce requests.

5.1.2 Resource Manager Design

The workload and system entities described in Section 3.1.3 and Section 3.1.2 respectively are represented by Java classes with similar characteristics. An additional attribute necessary for scheduling the workload is added to the Task class: *isFixed*, a Boolean variable, captures whether or not a task is permitted to be rescheduled. EAMR-RM provides flexibility in resource allocation and scheduling decisions by permitting tasks which have not started execution to be moved and rescheduled. Doing so allows the resource manager to respond to changes in workload characteristics; task scheduling can be reordered to account for newly arriving jobs with stricter deadlines and CPU operating frequency can be adjusted to handle an increase in system load. Once a task has started execution on a resource, it will continue executing as scheduled until completed. Tasks which are currently executing or are scheduled to start execution within the expected solve time of the CP Optimizer are marked as *fixed* to ensure that they are not rescheduled and in progress tasks and virtual machines are never preempted or migrated. To manage *fixed tasks*, EAMR-RM tracks the current resource allocation and schedule of all tasks in the system.

As discussed in Section 3.2.2.1, due to the potential size and complexity of the matchmaking and scheduling problem, the CPLEX CP Optimizer is not always able to prove that a feasible solution is optimal in a reasonably short amount of time as required for an open system. An upper bound on the CP Optimizer solve time is added to manage this issue. Because of this time limit there is no guarantee that the output resource allocation and task schedule has the absolute minimal energy consumption. However, this concession is necessary for proper operation of the EAMR-RM. The resource manager

uses a static solve time upper bound, henceforth referred to as *maximum solve time* (*maxTime*), of 10 seconds. Preliminary results showed that this value for *maxTime* was reasonable for tested arrival rates. Average EAMR-RM algorithm runtime for typical system and workload parameters was found to be less than 50% of *maxTime*. In the majority of cases, the resource manager is able to determine the optimal schedule which minimizes energy consumption within this specified time. The upper bound on solve time is reached only when the system is heavily loaded. Tasks which currently executing or are scheduled to start within *maxTime* from the time the EAMR-RM algorithm (discussed in Section 5.1.2.1) is invoked are considered fixed by the resource manager. Increasing *maxTime* beyond 10 seconds did not produce a significant improvement in system performance and detracts from the suitability of the proposed technique to handle open systems. When the system is heavily loaded, arriving jobs will need to wait for longer before they are scheduled and can begin execution thus further contributing to resource contention. Additionally, with higher values of *maxTime* the system is less capable of responding to changes in workload characteristics since more of the existing tasks on the system are fixed.

The EAMR-RM system consists of three main components: (1) the EAMR-RM algorithm, (2) the Scheduling procedure, and (3) the Matchmaking algorithm.

5.1.2.1 EAMR-RM Algorithm

The EAMR-RM algorithm presented in Table 5.1 manages the current schedule and fixed tasks and controls invocation of the Scheduling procedure and Matchmaking algorithm to perform matchmaking and scheduling of the workload. Lines 1-17 concern the management of jobs and associated tasks present in the system from previous invocations

of the matchmaking and scheduling algorithm (if any). EAMR-RM checks all tasks scheduled to execute on each resource to determine if that task has already started or is scheduled to start execution before the current invocation of the algorithm is expected to complete (Lines 1-3). Tasks which are not scheduled to start execution until after the algorithm is expected to complete (i.e. scheduled start time is greater than current time plus *maxTime*) require no additional processing and will be rescheduled. Tasks are sorted in non-decreasing order of scheduled start time, and thus once one task that requires reallocation and rescheduling is found, the remaining tasks assigned to that resource will also require rescheduling.

Tasks that are scheduled to start execution before the current invocation of the algorithm is expected to finish are marked as fixed (Lines 5-6). If the task has not finished executing (Line 7) additional constraints are added to the CP formulation to fix the task's start time and end time (Line 8). These constraints prevent the CP Optimizer from scheduling another task to execute on this resource slot during the same time period. For example, if the execution of t_3 is fixed from 15 to 23 time units the following constraints are added to the CP formulation:

$$a_3 = 15$$

$$c_3 = 8$$

If the task has already finished execution, it is removed from the corresponding job's task list. If the job's task list is now empty, the job is removed from J (Lines 9-12) to eliminate unnecessary decision variables and constraints and reduce CP Optimizer processing time.

Table 5.1: EAMR-RM Algorithm

Input: A set of MapReduce jobs J to map and schedule on a set of cloud resources R	
<hr/>	
1:	for each r in R
2:	for each task in current schedule mapped to r (sorted by increasing start time)
3:	if start time is greater than current time plus $maxTime$
4:	break
5:	else
6:	set $isFixed$ to true
7:	if end time is greater than current time
8:	add constraints to CP formulation to fix start time and end time of task
9:	else
10:	remove task from the job's task list
11:	if the job task list is empty
12:	remove job from J
13:	end if
14:	end if
15:	end if
16:	end for
17:	end for
18:	for each job j in J and queue
19:	if s_j is less than current time plus $maxTime$
20:	set s_j to current time plus $maxTime$
21:	end if
22:	end for
23:	create new CP model based on CP formulation for jobs in J and queue, resources in R , and fixed tasks
24:	invoke Scheduling procedure to solve CP formulation using CP Optimizer
25:	if a solution is found
26:	remove jobs from queue and add jobs and corresponding tasks to current list
27:	extract decision variables from solution
28:	invoke matchmaking algorithm to allocate tasks
29:	update schedule
30:	else
31:	reject and discard arriving job from queue and notify client
32:	end if

To guarantee that the CP Optimizer does not incorrectly schedule tasks to begin execution in the “past” or before the current invocation of the algorithm is expected to complete, the earliest start time of each job in the system is set to the maximum of its existing earliest start time and the current time plus $maxTime$ (Lines 18-22). This

prevents the CP Optimizer from generating an output schedule that requires tasks which have not yet started executing to begin execution before the current time. Fixed tasks which are on the system are excluded from this additional scheduling requirement.

EAMR-RM invokes the scheduling procedure (described in Section 5.1.2.2) to solve the CP formulation and determine the start time, end time, and execution duration of all tasks on the system (Line 24). EAMR-RM imports the Java packages *ilog.cp* and *ilog.concert* to build the CP model (Line 23), add constraints (Line 8) and extract the output schedule (Line 27). Concert is used to express the CP formulation in Java. A CP model based on the CP formulation given in Table 5.2 in Section 5.1.2.2 for jobs in J and the job queue, resources in R , and fixed tasks is generated each time the EAMR-RM algorithm is invoked. The resulting CP model is solved by the scheduling procedure using the CPLEX CP Optimizer Java API and the output schedule is extracted using the Concert element interface as discussed in Section 3.2.1. Tasks in the output schedule are allocated to resources by invoking the matchmaking algorithm (see Table 5.3 in Section 5.1.2.3) and the current schedule is updated to reflect the new start time, end time, execution duration, and resource allocation of all tasks in the system (Lines 28-29).

When invoked, the EAMR-RM algorithm executes in parallel with tasks running on the system. CP Optimizer solve time is included in the computation of mean job turnaround time. Resource manager energy consumption is negligible compared to that used during workload execution.

In rare occasions when the CP Scheduler is unable to determine a feasible solution which guarantees all job deadlines are met, the arriving job that triggered the invocation of the EAMR-RM algorithm is rejected and the client is notified (Line 31). The rejected job is

removed from the queue and the previously generated output schedule will continue to be used. EAMR-RM places a higher priority on client satisfaction than minimizing energy consumption. Client jobs are only rejected if specified deadlines cannot be met. Although designed for deadline constrained MapReduce jobs, EAMR-RM can be used for workloads including jobs without deadline requirements by assigning these jobs arbitrarily large deadlines.

EAMR-RM is built on the energy aware MapReduce batch resource manager discussed in Section 4.1 and models the matchmaking and scheduling problem using the base CP formulation discussed in Section 3.2.1 with minor changes discussed later in Section 5.1.2.2. As discussed previously with respect to the configurable MapReduce batch resource manager discussed in Section 4.2, because the base CP formulation requires a matchmaking decision variable for every combination of task and resource pool optimizer performance is possible for systems with a large number of active tasks and resources. A significant performance boost is possible for such systems by performing matchmaking and scheduling separately rather than in a single joint step. Doing so permits the matchmaking decision variables to be removed from the CP model which significantly reduces memory requirement and processing time of the CP Optimizer. Task scheduling is performed by CP Optimizer using the scheduling procedure discussed in Section 5.1.2.2. An additional matchmaking routine presented in Section 5.1.2.3 is introduced to allocate tasks to cloud resources.

5.1.2.2 Scheduling Procedure

In line with performance optimizations devised in [12], the scheduling procedure is performed before matchmaking by considering the resources in R as a single resource

pool with combined map and reduce task execution slots equal to the total number of map and reduce task execution slots among all resources. For example, consider a scenario where a system has 50 resources with 2 map task execution slots and 2 reduce task execution slots each. In this scenario, the combined resource pool will have a total of 100 map task execution slots and 100 reduce task execution slots to execute the workload. The combined resource pool scheduling problem is modeled as an optimization problem using constraint programming as shown in Table 5.2 and solved using the CPLEX CP Optimizer to determine the start time, end time, and execution duration of all tasks in the workload.

Table 5.2: EAMR-RM CP Formulation

$minimize E_{task} + h * P(f^{high}) * (\sum_{r \in R} (\max_{j \in J} (d_j) * (c_r^{map} + c_r^{red}) - \sum_{t \in T} x_{tr} * c_t))$	<i>subject to</i>
$(a_t \geq s_j \forall t \in T_j^{map}) \forall j \in J$	(1)
$(a_t \geq \max_{t \in T_j^{map}} (a_t + c_t) \forall t \in T_j^{red}) \forall j \in J$	(2)
$(\max_{t \in T_j^{red}} (a_t + c_t) \leq d_j) \forall j \in J$	(3)
$(cumulative(a_t, c_t, 1, \sum_{r \in R} c_r^{map}) \forall t \in T^{map})$	(4)
$(cumulative(a_t, c_t, 1, \sum_{r \in R} c_r^{red}) \forall t \in T^{red})$	(5)
$(c_t \in \mathbb{Z} \mid e_t \leq c_t \leq e_t \frac{f^{high}}{f_{low}}) \forall t \in T$	(6)
$a_t \in \mathbb{Z} \forall t \in T$	(7)

The differences between the base CP formulation discussed in Section 3.2.1 and the CP formulation for EAMR-RM are described next. Since EAMR-RM performs matchmaking and scheduling separately rather than in a single joint step, the x_{tr} decision variables can be removed from the CP formulation. This modification reduces the number of decision variables in the CP model and substantially reduces CP Optimizer processing time, particularly for systems with a large number of present tasks and resources. Constraints

(4) and (5) which model the resource capacity of the system are modified to be consistent with the removal of the matchmaking decision variables and the use of a combined resource pool.

The component of total energy consumed by idle resources is calculated differently for EAMR-RM in comparison to the base CP formulation. The total idle duration of all CPU cores during execution of the set of MapReduce jobs can be found by multiplying the latest deadline of all jobs in J by the total number of map and reduce task execution slots in R and subtracting the total execution duration of all tasks currently scheduled to run on the system. Since future arrivals are unknown the CP formulation only considers Idle Energy consumption up to the latest deadline of all jobs currently present on the system; future Idle Energy consumption cannot be predicted. Because all resources in R will remain active to handle future job arrivals, Idle Energy cannot be reduced by minimizing completion time. Idle Energy is calculated up to the latest deadline of all jobs in J rather than the latest completion time to ensure that the solver does not attempt to reduce Idle Energy consumption by scheduling tasks such that execution is completed earlier than necessary.

Note that the deadline requirement modeled by Constraint (3) is modified for hard deadlines. In systems with hard deadlines, such as EAMR-RM, accepted requests are never permitted to miss their specified deadlines. The explanation for the remaining constraints is the same as that presented in Section 3.2.1. Verification of the changes to the CP formulation required by EAMR-RM is performed using an approach similar to that described for the base CP formulation in Section 3.2.1.

5.1.2.3 Matchmaking Algorithm

Matchmaking is then performed based on the start time, end time, and execution duration of all tasks in the workload as given by the output schedule of the scheduling procedure for the combined resource pool. Allocation of tasks to resources is performed in a similar manner to the approach described in Section 4.2.2 for the configurable energy aware MapReduce batch resource manager. However, task allocation for EAMR-RM is more complex due to the added consideration of fixed tasks since these tasks cannot be migrated to other resources. The matchmaking algorithm is presented in Table 5.3.

Matchmaking of tasks in the output schedule is performed using a best fit approach. Each execution slot is considered independently and is initially populated with its assigned fixed tasks specified in the current schedule (Lines 1-3). Tasks in the output schedule are sorted in non-decreasing order of start times and assigned to task execution slots such that the gap between the completion of the previous task and the start of the new task is minimized (Lines 4-15). For example, consider a scenario where there are two task execution slots: s_1 and s_2 , and a task t_3 which must be allocated to one of these slots from time 17 to 23. Slot s_1 is scheduled to execute task t_1 from time 8 to 12, and slot s_2 is scheduled to execute task t_2 from time 10 to 15. In this case task t_3 would be allocated to slot s_2 since the resulting gap ($17-15=2$) is less than the gap for s_1 ($17-12=5$). It is important to note that map and reduce tasks are considered separately. Map tasks are assigned to map task execution slots in Lines 4-9. Reduce tasks are assigned to reduce task execution slots in Lines 10-15.

After all tasks from the output schedule have been assigned to task execution slots, a solution which contains the number of resources in R is generated. Task slots with

specific assignment requirements due to the presence of fixed tasks are assigned to the appropriate resource (Lines 16-20 and Lines 23-27). Task execution slots which do not execute any fixed tasks are then randomly assigned to resources until each resource is assigned a number of map and reduce task execution slots equal to its capacity (Lines 21-23 and Lines 28-30).

Table 5.3: EAMR-RM Matchmaking Algorithm

1:	for each t in T_{fixed}
2:	allocate t to the appropriate task execution slot based on the current schedule
3:	end for
4:	for each map task t_{map} in T (sorted in non-decreasing order of start time)
5:	for each map task execution slot s_{map} in R
6:	determine the difference between start time of t_{map} and end time of the last task allocated to s_{map}
7:	end for
8:	assign t_{map} to the slot with the minimum time difference
9:	end for
10:	for each reduce task t_{red} in T (sorted in non-decreasing order of start time)
11:	for each map task execution slot s_{red} in R
12:	determine the difference between the start time of t_{red} and end time of the last task allocated to s_{red}
13:	end for
14:	assign t_{red} to the slot with the minimum time difference
15:	end for
16:	for each map task execution slot s_{map} in R
17:	if s_{map} executes fixed tasks
18:	allocate s_{map} to the appropriate resource r in R based on fixed tasks
19:	end if
20:	end for
21:	for each unallocated map task execution slot s_{map} in R
22:	allocate s_{map} to a random resource r in R with assigned map slots less than map slot capacity c_r^{map}
23:	for each reduce task execution slot s_{red} in R
24:	if s_{red} executes fixed tasks
25:	allocate s_{red} to the appropriate resource r in R based on fixed tasks
26:	end if
27:	end for
28:	for each unallocated reduce task execution slot s_{red} in R
29:	allocate s_{red} to a random resource r in R with assigned reduce slots less than reduce slot capacity c_r^{red}
30:	end for

5.1.3 Performance Evaluation

This section describes the simulation based performance analysis conducted for EAMR-RM. A simulated system and workload allows for flexibility in systematically varying system and workload parameters. Simulation is suitable for this research which does not concern obtaining exact values of performance and energy metrics but focuses on their general relationship with system and workload parameters and relative improvement compared to alternative strategies. Simulation based performance analysis has been used extensively by other researchers for investigating similar matchmaking and scheduling problems (ex. [28], [12]).

5.1.3.1 Experimental Setup

Results for matchmaking and scheduling open streams of MapReduce job arrivals were used to analyze the performance of the energy aware MapReduce resource manager for open systems. The CP Optimizer, matchmaking routine, and EAMR-RM algorithm were executed on a single PC running Windows 7 Professional with a 2.93GHz Intel Core i7 CPU and 8GB of RAM. MapReduce job arrivals are simulated during the discrete event simulation and the EAMR-RM algorithm is invoked to perform matchmaking and scheduling for each arriving job. Task execution on the cloud environment is simulated to determine the proportion of jobs rejected by the system, average job turnaround time, and energy consumption for the arriving MapReduce job workload. Each experiment was run long enough to ensure that the system reached steady state and the impact of transient behaviour on system performance is negligible.

5.1.3.2 Performance Metrics

To the best of our knowledge there exist no competing strategies for minimizing energy consumption of an open stream of MapReduce job arrivals subject to SLAs which include deadlines. Thus the performance of EAMR-RM is compared to a *non-energy-aware* (NEA) resource manager based on MRCP-RM described in [12]. NEA does not consider energy consumption when making resource management decisions to minimize average job turnaround time. Furthermore, NEA does not use DVFS to vary CPU operating frequency based on system load—all tasks are executed at the maximum permitted operating frequency.

Resource manager performance is evaluated and compared based on the following metrics:

- *Energy Savings* ($E_{savings}$), the percentage reduction in total energy required to execute an open stream of arriving MapReduce jobs using EAMR-RM compared to NEA. The expression for Energy Savings, which permits the unknown proportionality constant δ to be factored out of the comparison, is given in Equation 5.2.

$$E_{savings} = \left(1 - \frac{E_{(for\ EAMR-RM)}}{E_{(for\ NEA)}}\right) * 100\% \quad 5.2$$

- *Average turnaround time (TAT)*, the average time between earliest start time and completion of accepted MapReduce jobs.
- *Proportion of Rejected Jobs* (ω), the proportion of jobs that were not guaranteed to complete before their deadlines and thus rejected by the system.

Similar to the energy aware MapReduce resource manager for batch workloads, performance of EAMR-RM is evaluated based on energy consumption, system utilization and throughput, and deadline compliance. Energy required to execute the workload is

expressed as energy savings to permit easier comparison to alternative resource management techniques. Because EAMR-RM rejects jobs which are not guaranteed to complete before their client specified deadlines, deadline compliance is evaluated based on the proportion of jobs rejected by the system rather than the number of jobs that miss their deadlines.

5.1.3.3 System and Workload Parameters

Table 5.4 outlines the system and workload parameters used in the experiments. The system parameters section describes the characteristics of the resources in the cloud environment. The number of resources used (m) as well as the map and reduce task capacity of each resource is described in the table.

Table 5.4: EAMR-RM System and Workload Parameters

Name	Value
System Parameters	
Number of resources, m -Map task capacity, c_r^{map} -Reduce task capacity, c_r^{red}	$m = \{25, \mathbf{50}, 75, 100\}$ $c_r^{map} = 2, c_r^{red} = 2$
Workload Parameters	
Job arrival rate, λ (Jobs/s)	$\lambda = \{0.001, 0.005, \mathbf{0.01}, 0.015, 0.02, 0.025\}$ Inter-arrival times $\sim$ exponentially distributed
Job j: -Number of map tasks, k_j^{map} -Number of reduce tasks k_j^{red}	$k_j^{map} \sim DU(1, 100)$ $k_j^{red} \sim DU(1, k_j^{map})$
Task execution times (s) -Map task (me) -Reduce task (re)	$me \sim DU(1, e^{max})$ where $e^{max} = \{25, \mathbf{50}, 75, 100\}$ $re = \frac{(3 * \sum_{t \in T_j^{map}} e_t)}{k_j^{red}} + DU(1, 10)$
Job start time, s_j (s)	<i>arrival time</i>
Job deadline, d_j (s)	$d_j = s_j + (U(1, l) * \sum_{t \in T_j} e_t)$ where $l = \{1, 2, \mathbf{3}, 4\}$

The attributes of the arriving MapReduce jobs are described in the Workload Parameters section of Table 5.4. The first row of this section describes the job arrival rate; the job

arrivals are assumed to be characterized by a Poisson process. The use of a Poisson arrival process is in line with work performed in [8] and [29] among others. The second row defines the number of map tasks (k_j^{map}) and the number of reduce tasks (k_r^{red}) for each job, whereas the third row specifies the execution time of map tasks (me) and reduce tasks (re). Note that each job has different values for k_j^{map} and k_r^{red} which are generated using discrete uniform (DU) distributions as in [28] and [12]. Furthermore, each task in a job has a different execution time. Similar to work performed in [28] and [12], me is generated using a DU distribution where e^{max} is the maximum map task execution time, and re is generated based on the total execution time of all map tasks in the job. The earliest start time (s_j) of each job is equal to the job arrival time. Job deadline (d_j) is calculated as the job arrival time plus the sum of execution times of all the tasks in the job ($\sum_{t \in T_j} e_t$) multiplied by *laxity factor*. Consistent with work performed in [12], laxity factor is generated using a uniform (U) distribution where l is the upper bound of the distribution. Recall from Section 4.1.2.3, higher values of l result in lenient deadlines with more slack time.

5.1.3.4 Processor Description

The processor model used in this performance analysis is based on the frequency/voltage levels for the AMD Opteron 2380 presented in [11] and shown in the first two columns of Table 5.5. The power dissipation of each frequency/voltage level is calculated using Equation 3.5 and shown in the “Power” column of Table 5.5. Note that because δ is not known, power dissipation is expressed in proportional unit W_{scaled} defined in Section 3.1.1.

Table 5.5: AMD Opteron 2380 and Intel Xeon E5 2687W Power Measurements

Frequency (MHz)	Voltage (V)	Power (W_{scaled})
AMD Opteron 2380		
2500	1.300	4225.00
1800	1.200	2592.00
1300	1.100	1573.00
800	1.025	840.50
Intel Xeon E5 2687W		
3100	1.200	4464.00
1200	0.840	846.72

A linear regression of the power dissipation in each frequency/voltage level is used to express power dissipation as a function of operating frequency:

$$P = \delta * (2.01f - 909.19) \quad 5.3$$

The coefficient of determination for this linear regression was found to be $R^2 = 0.9903$. Note that the power estimate presented in Equation 5.3 is based on empirical measurements of processor supply voltage performed by researchers in [11] and is valid only for frequencies between f^{high} and f^{low} .

To validate the conclusions of performance analysis for other processors, select experiments are performed using an alternative processor model based on the measurements for the Intel Xeon E5 2687W presented in [11] and shown in Table 5.5. The AMD Opteron 2380 is a recent multicore consumer class processor that could be used in private clusters executing MapReduce workloads [48] whereas the Intel Xeon E5 2687W is a server class processor such as those more commonly found in large data centers. Experimentation using the Intel Xeon E5 2687W demonstrates that the proposed DVFS-based technique is effective for reducing energy consumption of server class processors typically found in large data centers. Although only measurements of supply voltage at f^{high} and f^{low} are provided for the Intel Xeon E5 2687W, a linear relationship

between frequency and power dissipation is assumed. The coefficient of determination for the linear regression of frequency and power measurements for the AMD Opteron 2380 and the linear relationship between frequency and power presented for alternative processor models suggested in [10] and [53] demonstrate that this assumption is reasonable.

5.1.4 Experimental Results

An investigation into the impact of changes to system and workload parameters is performed using a *factor-at-a-time* approach. System performance is observed while one parameter is varied and the other parameters are kept at their default values. The default value of each parameter is shown in boldface in the “Value” column of Table 5.4. Halt percentage, used to determine energy consumption of idle CPU cores, is set to 10% for all experiments using the processor model from the AMD Opteron 2380 based on the power measurements for the AMD Opteron 2380 from [51]. Halt percentage is set to 5% for all experiments using the processor model from the Intel Xeon E5 2687W based on the power measurements from [54]. As in performance analysis for BRM and C-BRM discussed in Chapter 4, each experiment was repeated a sufficient number of times to generate a confidence interval for Energy Savings (see Equation 5.2) at a 95% confidence level with upper and lower bounds for the interval within $\pm 5\%$ of the mean.

5.1.4.1 Impact of Arrival Rate

Figure 5.2 shows the impact of arrival rate on system performance. Energy Savings ($E_{savings}$) has a non-monotonic relationship with arrival rate. At moderate arrival rates EAMR-RM is able to save a substantial amount of energy compared to NEA by applying a reduced operating frequency during the execution of some tasks. An energy savings of

up to 41.4% is observed at $\lambda=0.005$. The average task operating frequency for moderate arrivals rates is close to f^{low} . At high arrival rates, there is greater contention for system resources and EAMR-RM is unable to apply a significantly reduced operating frequency during the execution of tasks while satisfying client deadline requirements. $E_{savings}$ is small for very low arrival rates. When arrival rate is low and time between job arrivals is long, a higher proportion of total energy required is consumed by idle CPU cores. Since the DVFS based approach does not reduce energy consumption of idle CPU cores, EAMR-RM does not provide significant benefit compared to NEA.

Figure 5.2: Impact of Arrival Rate

Average Turnaround Time, TAT , for NEA increases as arrival rate increases. Due to contention for system resources, some tasks cannot begin execution immediately. TAT for EAMR-RM has a non-monotonic relationship with arrival rate. For low arrival rates, TAT is small because resource contention is low. At moderate arrival rates, TAT is high because the system is able to apply an operating frequency close to f^{low} to the execution of most tasks while still meeting all job deadlines. Resource contention prevents jobs

from beginning execution immediately after they arrive. At high arrival rates, the system must execute tasks at higher CPU operating frequency to keep up with frequent job arrivals thus leading to a smaller mean TAT compared to moderate arrival rates.

TAT for EAMR-RM is greater than NAE for all cases because the resource manager applies a reduced CPU operating frequency when possible to minimize energy consumption. However, EAMR-RM does not incur any SLA violations because of this increased average turnaround time because the resource manager guarantees that all deadlines of accepted jobs are met.

5.1.4.2 Impact of Laxity Factor

Figure 5.3: Impact of Laxity Factor

The impact of laxity factor on system performance is shown in Figure 5.3. Significant energy savings of approximately 40% were observed for all values of l . EAMR-RM was able to execute most tasks in the workload at an operating frequency close to f^{low} for all values of l when other parameters were at their default values.

Average Turnaround Time of NEA is not impacted by laxity factor, l , because jobs are completed as fast as possible. Figure 5.3 shows that as l increases, TAT for EAMR-RM increases since the scheduler has more flexibility in scheduling tasks. Energy Savings, $E_{savings}$, is largely insensitive to variations in l for typical parameter values because job frequency, rather than deadlines, limits the minimum CPU operating frequency.

5.1.4.3 Impact of Map Task Execution Time

The impact of map task execution time on system performance can be seen in Figure 5.4. As expected, TAT for both EAMR-RM and NEA increases as e^{max} , the upper bound on map task execution time, increases. $E_{savings}$ decreases from 38.2% to 8.4% as average map task execution time increases. For low values of e^{max} , the system can apply a CPU operating frequency close to f^{low} to the execution of most tasks while still meeting all deadlines. When e^{max} is high, Energy Savings is low because tasks must be executed at a higher CPU operating frequency due to the higher job service times and increased resource contention.

Figure 5.4: Impact of Map Task Execution Time

5.1.4.4 Impact of Number of Resources

The impact of number of resources on TAT and $E_{savings}$ is shown in Figure 5.5. TAT for both EAMR-RM and NEA decreases as the number of resources increases. This result is intuitive; jobs can be completed faster when tasks are distributed over more available resources. $E_{savings}$ has a non-monotonic relationship with number of resources. When number of resources is low, contention often prevents tasks from being executed at the minimum permitted operating frequency. As number of resources increases to a moderate value, $E_{savings}$ increases since tasks can be executed at a lower frequency while still meeting all deadlines. An $E_{savings}$ of up to 41.0% is achieved for $m=50$. When a given workload is executed on a higher number of resources the proportion of total energy consumption due to idle CPU cores increases; system capacity increases while workload intensity remains constant. As discussed previously in Section 5.1.4.1, EAMR-RM is less effective in these circumstances.

Figure 5.5: Impact of Number of Resources

5.1.4.5 Proportion of Rejected Jobs

The workload and system state may prevent the resource manager from guaranteeing that an arriving job will complete before its deadline. In such cases the job will be rejected to ensure that client QoS requirements defined by SLAs are never violated. Since EAMR-RM places a higher priority on client satisfaction than minimizing energy consumption and only rejects jobs if scheduling is infeasible, EAMR-RM does not reject more jobs than NEA. The proportion of rejected jobs, ω , is less than 0.01 for all tested system and workload parameters. The impact of arrival rate on ω is shown in Figure 5.6. As expected, when arrival rate and resource contention are high, a larger proportion of jobs are rejected. When resource contention is low due to a low arrival rate, few or no jobs need to be rejected to guarantee the generation of a schedule where all deadlines are met.

Figure 5.6: Impact of Arrival Rate on Proportion of Rejected Jobs

5.1.4.6 Alternative Processor Model

To validate the conclusions of our performance analysis for other processors, select experiments are performed using an alternative processor model based on the measurements for the Intel Xeon E5 2687W. Figure 5.7 shows the performance of EAMR-RM compared to NEA for various arrival rates when the Intel Xeon E5 2687W processor model is used. The overall trends in energy savings and average turnaround time are comparable to those observed for the AMD Opteron 2380 (See Figure 5.2). Due to its higher maximum frequency, the Intel Xeon E5 2687W resulted in a lower *TAT* for all tested arrival rates.

Figure 5.7: Impact of Arrival Rate using Intel Xeon E5 2687W

5.1.4.7 Summary of Results

This section describes an effective cloud resource management technique for jobs which are characterized by multiple stages of execution and subject to SLAs that include hard deadlines. More specifically, this section presents a constraint programming (CP) based

approach and resource manager, EAMR-RM, for matchmaking and scheduling an open stream of MapReduce jobs with hard deadlines such that data center processor energy consumption is minimized and client SLAs are not violated. This research is directed to address the needs of data center operators and cloud service providers for energy aware resource management techniques which reduce data center energy consumption without violating client SLAs.

EAMR-RM is implemented in Java and uses a two-step approach to solve the matchmaking and scheduling problem. The scheduling problem is formulated as a CP optimization problem and solved using the IBM CPLEX CP Optimizer. Tasks are allocated to resources by invoking the best-fit based matchmaking algorithm presented in Table 5.3 in Section 5.1.2.3. Simulation-based factor-at-a-time experiments demonstrate the effectiveness of the proposed technique for a variety of system and workload parameters. A comparison with a *non-energy-aware* (NEA) resource management technique based on MRCP-RM described in [12] is included. Results show that the proposed DVFS based approach can reduce data center CPU energy consumption by close to 45% over NEA for various workloads. Experimentation yielded a number of important observations of system behavior and the key insights are summarized next:

- EAMR-RM performed best and achieved an up to 45% reduction in CPU energy consumption compared to NEA when the proportion of idle CPUs and the average frequency for the active CPUs are small. A combination of these two factors occurs, for example, for a medium value of λ (0.01 in Figure 5.2) and a medium value of e^{max} (50 in Figure 5.4). Lesser energy savings were achieved for low and high system loads and are explained next:

- Under light load, the proportion of idle CPUs is high. Idle energy consumption, which is not reduced by the DVFS based approach, is a significant proportion of total energy consumption.
- When system load is high, average task execution frequency is high. Tasks must be executed at frequencies close to f^{high} to ensure all deadlines are met, thus DVFS cannot be used to reduce the CPU operating frequency and save energy.
- Energy savings were often accompanied by increases in average job turnaround time. However, client SLAs were not violated since all specified deadlines were met.
- Non-monotonic relationships between $E_{savings}$ and average map task execution time as well as between $E_{savings}$ and arrival rate were observed. The system performed best for moderate values of these parameters.
- Energy savings is observed to be largely insensitive to laxity factor regardless of system load.
- EAMR-RM rejected less than 1% of arriving jobs for all tested system and workload parameters.

5.2 Hybrid Energy Aware MapReduce Resource Manager for Open Systems

This section describes the details of the technique and algorithm for the *Hybrid Energy Aware MapReduce Resource Manager for open systems*, HEAMR-RM. Details of the technique including the resource manager and performance evaluation of the technique are discussed.

5.2.1 Overview

The Hybrid Energy Aware MapReduce Resource Manager for open systems is designed for matchmaking and scheduling an open stream of deadline constrained MapReduce job arrivals such that energy consumption is minimized. Performance analysis for EAMR-RM discussed in Section 5.1.4.2 showed that the resource manager is unable to save a substantial amount of energy compared to NEA when the cloud environment is lightly loaded because the DVFS based scheduling technique cannot reduce resource idle energy consumption. This observation motivated the design of HEAMR-RM which combines the technique of collocating tasks with similar execution times to reduce idle energy consumption by permitting unused resources to be disabled as discussed in [7], [30], and [33] with the DVFS based scheduling approach used by EAMR-RM.

Similar to EAMR-RM discussed in Section 5.1, HEAMR-RM considers MapReduce jobs with hard deadline constraints. Jobs which are accepted by the system are guaranteed to complete before their client specified deadlines. In rare cases, the workload and state of resources may prevent the system from guaranteeing that an arriving job will complete before its deadline. In such cases, the job is rejected to ensure that the SLA requirements of accepted jobs are never violated.

The proposed technique minimizes energy consumption by considering task energy consumption and idle energy consumption independently. Task energy consumption is reduced using a DVFS based scheduling approach to decrease the processor operating frequency during execution of some tasks. By leveraging available slack in job deadlines, CPU operating frequency can be reduced, thereby decreasing energy consumption, while still guaranteeing that all deadline requirements are met. Idle energy consumption is

decreased using a matchmaking approach which collocates tasks with similar execution times to reduce the number of resources required to execute the workload. When the system is lightly loaded and there is a surplus of available resources, unused resources are disabled to reduce idle energy consumption. If system load increases, resources are enabled to handle the additional load and ensure that jobs are not needlessly rejected by the system.

A resource which is enabled and capable of executing tasks is said to be *active*. The resource manager maintains a subset R_{active} of the set of resources R to track how many and which resources in the pre-acquired cloud environment are active. Since enabling a resource takes a significant amount of time, only active resources are considered for matchmaking and scheduling tasks in the workload.

An overview of the HEAMR-RM operation is presented next. Client submitted MapReduce jobs are placed in a job queue. When the resource manager is not busy, matchmaking and scheduling of all jobs in the queue is performed by invoking the HEAMR-RM algorithm (discussed in Section 5.2.2.1). HEAMR-RM formulates the complex matchmaking and scheduling problem as an optimization problem using CP. The CPLEX CP Optimizer is used to solve the CP formulation such that energy required to execute the input workload is minimized. The resource management algorithm allocates and schedules all tasks on active resources and updates the current schedule. If the current number of active resources is not suited to the system load, the resource manager will adjust the number of active resources to improve system performance.

5.2.2 Resource Manager Design

HEAMR-RM provides flexibility in resource management decisions by permitting tasks which have not yet started execution to be rescheduled or allocated to different resources. Doing so allows the resource manager to adjust to changes in workload characteristics; CPU operating frequency can be increased to accommodate an increase in system load and task execution can be reordered to account for newly arriving jobs with stricter deadlines. Furthermore, tasks can be allocated to different resources to better match tasks with similar execution times to reduce the number of resources required to execute the workload. As discussed previously in Section 3.2.1, all tasks run on exactly one task execution slot and have a resource requirement of 1. Thus, only task execution time is considered when collocating tasks to reduce the number of resources required. Similar to EAMR-RM, tasks that have started execution or are scheduled to start execution before the current invocation of the HEAMR-RM algorithm (discussed in Section 5.2.2.1) are marked as fixed. Fixed tasks execute until completion as scheduled and are never migrated to another resource or preempted. HEAMR-RM tracks the resource allocation and schedule of all tasks in the workload to manage fixed tasks.

As discussed previously in Section 3.2.2.1, due to the size and complexity of the resource management problem, the CPLEX CP Optimizer is sometimes unable to prove that a given solution is optimal within a reasonable time frame as required by an open system. To manage this issue, CP Optimizer solve time is required to be less than a specified upper limit. Similar to EAMR-RM, a static solve time limit or *maximum solve time*, *maxTime*, of 10 seconds is used. Due to this limit, there is no guarantee that the task schedule and resource allocation determined by the resource manager achieve the optimal

energy consumption. However, this concession is necessary for proper operation of the resource manager. *maxTime* is used to determine which tasks are fixed based on the current schedule. Tasks which have already started executing or are scheduled to start within *maxTime* from the time the HEAMR-RM algorithm (discussed in Section 5.2.2.1) is invoked are considered fixed by the resource manager and cannot be rescheduled or reallocated.

The HEAMR-RM system consists of four main components: (1) the HEAMR-RM algorithm, (2) the Scheduling procedure, (3) the Matchmaking algorithm, and (4) the Resource Scaling procedure.

5.2.2.1 HEAMR-RM Algorithm

Similar to the EAMR-RM algorithm described in Section 5.1.2.1, the HEAMR-RM algorithm is responsible for managing the job queue as well as jobs and associated tasks present in the system from previous invocations of the matchmaking and scheduling procedure (if any). The HEAMR-RM algorithm presented in Table 5.6 uses the same process for managing fixed tasks and invoking the matchmaking and scheduling procedures as the EAMR-RM algorithm. Only key actions of the HEAMR-RM algorithm are described in this section. For a full discussion of the basic algorithm and management of fixed tasks, refer to Section 5.1.2.1.

The CPLEX CP Optimizer is invoked to solve the CP formulation and determine the output schedule which includes the start time, execution frequency, and end time of all tasks on the system (Lines 23-27). The matchmaking algorithm is invoked to determine the allocation of tasks to resources based on the output schedule (Line 28). After the current schedule is updated with the start time, execution frequency, end time, and

resource allocation of all tasks in the system, the resource scaling procedure is invoked to adjust the number of active resources based on system load (Line 33).

Table 5.6: HEAMR-RM Algorithm

Input: A set of MapReduce jobs J to map and schedule on a set of cloud resources R	
<hr/>	
1:	for each r in R
2:	for each task in current schedule mapped to r (sorted by increasing start time)
3:	if start time is greater than current time plus $maxTime$
4:	break
5:	else
6:	set $isFixed$ to true
7:	if end time is greater than current time
8:	add constraints to CP formulation to fix start time and end time of task
9:	else
10:	remove task from the job's task list
11:	if the job task list is empty
12:	remove job from J
13:	end if
14:	end if
15:	end if
16:	end for
17:	end for
18:	for each job j in J and queue
19:	if s_j is less than current time plus $maxTime$
20:	set s_j to current time plus $maxTime$
21:	end if
22:	end for
23:	create new CP model based on CP formulation for jobs in J and queue, resources in R , and fixed tasks
24:	invoke Scheduling procedure to solve CP formulation using CP Optimizer
25:	if a solution is found
26:	remove jobs from queue and add jobs and corresponding tasks to current list
27:	extract decision variables from solution
28:	invoke matchmaking algorithm to allocate tasks
29:	update schedule
30:	else
31:	reject and discard arriving job from queue and notify client
32:	end if
33:	invoke resource scaling procedure

The enhancements added to the HEAMR-RM algorithm over the EAMR-RM algorithm

presented in Table 5.1 in Section 5.1.2.1 are described next. Enhancements to the

HEAMR-RM algorithm permit the resource manager to allocate and schedule arriving MapReduce jobs characterized by SLAs which include a client specified earliest start time in the future in addition to the deadline.

The HEAMR-RM system maintains a secondary job queue to handle requests with earliest start times greater than the respective arrival times. These requests are not immediately scheduled when they arrive at the system. Instead, they are placed in the secondary job queue until close to their client specified earliest start times to ensure they are not repeatedly considered in the scheduling procedure. Once the current time is sufficiently close to the client specified earliest start time, the request is returned to the primary job queue where it will be scheduled and allocated to resources as normal by the HEAMR-RM algorithm. This procedure ensures good performance of the CP Optimizer by limiting the number of decision variables in the CP formulation used for scheduling.

The primary enhancement of HEAMR-RM compared to EAMR-RM is the capability to enable or disable resources from the pre-acquired cloud environment based on system load to reduce energy consumption due to idle resources. This approach is referred to as *Resource Scaling* due to its similarities to cloud auto-scaling techniques. The Scheduling procedure presented in Section 5.2.2.2 and Matchmaking algorithm presented in Section 5.2.2.3 are invoked by HEAMR-RM for matchmaking and scheduling the input workload. The Resource Scaling procedure, which is invoked after the Matchmaking algorithm completes and the current schedule is updated, is described in Section 5.2.2.4.

5.2.2.2 Scheduling Procedure

Similar to EAMR-RM, HEAMR-RM performs matchmaking and scheduling separately rather than in a single joint step to reduce the complexity of the scheduling through the

removal of the matchmaking decision variables. Scheduling is performed first by formulating the problem as an optimization problem using CP. The CP formulation for scheduling in HEAMR-RM is shown in Table 5.7. The CP formulation is solved by the CPLEX CP Optimizer to determine the output schedule which includes the start time, end time, and execution duration of all tasks in the workload.

Table 5.7: HEAMR-RM CP Formulation

$minimize E_{task} + h * P(f^{high}) * (\sum_{r \in R_{active}} (\max_{j \in J}(d_j) * (c_r^{map} + c_r^{red}) - \sum_{t \in T} x_{tr} * c_t))$	<i>subject to</i>
$(a_t \geq s_j \ \forall t \in T_j^{map}) \ \forall j \in J$	(1)
$(a_t \geq \max_{t \in T_j^{map}} (a_t + c_t) \ \forall t \in T_j^{red}) \ \forall j \in J$	(2)
$(\max_{t \in T_j^{red}} (a_t + c_t) \leq d_j) \ \forall j \in J$	(3)
$(cumulative(a_t, c_t, 1, \sum_{r \in R_{active}} c_r^{map}) \ \forall t \in T^{map})$	(4)
$(cumulative(a_t, c_t, 1, \sum_{r \in R_{active}} c_r^{red}) \ \forall t \in T^{red})$	(5)
$(c_t \in \mathbb{Z} \mid e_t \leq c_t \leq e_t \frac{f^{high}}{f_{low}}) \ \forall t \in T$	(6)
$a_t \in \mathbb{Z} \ \forall t \in T$	(7)

The HEAMR-RM CP formulation for scheduling is based on the CP formulation for EAMR-RM shown in Table 5.2 of Section 5.1.2.2. The changes required for the HEAMR-RM are discussed next. For a full discussion of the CP formulation refer to the description of the base CP formulation presented in Section 3.2.1. Verification of the changes to the CP formulation required by HEAMR-RM is performed using an approach similar to that described for the base CP formulation in Section 3.2.1.

Because HEAMR-RM performs matchmaking and scheduling separately to reduce the number of decision variables and improve CP Optimizer efficiency, the task execution capabilities of the pre-acquired cloud resources are treated as a single combined resource

pool. However, since HEAMR-RM permits unused resources to be disabled to reduce idle energy consumption, only map and reduce task execution slots among active resources are considered as part of the resource pool. For example, consider a scenario where a system has 50 resources, 20 of which are currently active, with 2 map task execution slots and 2 reduce task execution slots each. In this scenario, the combined resource pool has 40 map task execution slots and 40 reduce task execution slots available to schedule the workload. Constraints (4) and (5) which model the resource capacity of the system are modified to be consistent with the addition of active resources. The expression for idle energy consumption in the objective function (see first row of Table 5.7) is also modified to accommodate the addition of active resources. As in research performed in [7], energy consumption of disabled resources is assumed to be negligible. Therefore, only resources which are active and contribute to the combined resource pool are considered when determining idle energy consumption during scheduling.

5.2.2.3 Matchmaking Algorithm

Matchmaking is then performed to allocate tasks to resources based on the output schedule determined by the CP Optimizer. The matchmaking algorithm for HEAMR-RM is presented in two parts: Part A, presented in Table 5.8, determines the allocation of tasks to execution slots and Part B, presented in Table 5.9, determines the allocation of execution slots to resources.

The allocation of tasks to execution slots in matchmaking algorithm - Part A uses a best fit approach similar to the one described for EAMR-RM in Table 5.3 of Section 5.1.2.3 and includes additional actions to manage disabled resources. Only active resources (i.e.

those in the subset R_{active}) are considered by the matchmaking algorithm for HEAMR-RM. Execution slots of active resources are considered independently and initially populated with the appropriate fixed tasks (Lines 1A-3A). The remaining tasks in the output schedule are sorted in non-decreasing order of start time and assigned to slots of active resources such that the gap between the end of the previous task and the start of the new task is minimized. Note that map and reduce tasks are considered separately. Map tasks are assigned to map task execution slots (Lines 4A-9A) and reduce tasks are assigned to reduce task execution slots (Lines 10A-15A). This best fit approach ensures that the resource manager attempts to allocate tasks to execution slots which are currently in use before considering unused task execution slots. Thus, the final allocation is less likely to require all available resources and some resources may be disabled to reduce energy consumption.

Table 5.8: HEAMR-RM Matchmaking Algorithm - Part A

1A:	for each t in T_{fixed}
2A:	allocate t to the appropriate task execution slot based on the current schedule
3A:	end for
4A:	for each map task t_{map} in T (sorted in non-decreasing order of start time)
5A:	for each map task execution slot s_{map} in R_{active}
6A:	determine the difference between start time of t_{map} and end time of the last task allocated to s_{map}
7A:	end for
8A:	assign t_{map} to the slot with the minimum time difference
9A:	end for
10A:	for each reduce task t_{red} in T (sorted in non-decreasing order of start time)
11A:	for each reduce task execution slot s_{red} in R_{active}
12A:	determine the difference between the start time of t_{red} and end time of the last task allocated to s_{red}
13A:	end for
14A:	assign t_{red} to the slot with the minimum time difference
15A:	end for
16A:	Invoke matchmaking algorithm - Part B

Table 5.9: HEAMR-RM Matchmaking Algorithm - Part B

1B:	for each map task execution slot s_{map} in R_{active}
2B:	if s_{map} executes fixed tasks
3B:	allocate s_{map} to the appropriate resource r in R based on fixed tasks
4B:	end if
5B:	end for
6B:	for each reduce task execution slot s_{red} in R_{active}
7B:	if s_{red} executes fixed tasks
8B:	allocate s_{red} to the appropriate resource r in R_{active} based on fixed tasks
9B:	end if
10B:	end for
11B:	for each unallocated map task execution slot s_{map} in R_{active}
12B:	for each r in R_{active} with assigned map slots less than map slot capacity c_r^{map}
13B:	for each execution slot s_r assigned to resource r
14B:	determine the difference between the latest end time of s_{map} and the latest end time of s_r
15B:	end for
16B:	sum difference in latest end times
17B:	end for
18B:	assign s_{map} to the resource with the minimum difference in sum of end times
19B:	end for
20B:	for each unallocated reduce task execution slot s_{red} in R_{active}
21B:	for each r in R_{active} with assigned reduce slots less than reduce slot capacity c_r^{red}
22B:	for each execution slot s_r assigned to resource r
23B:	determine the difference between the latest end time of s_{red} and the latest end time of s_r
24B:	end for
25B:	sum difference in latest end times
26B:	end for
27B:	assign s_{red} to the resource with the minimum difference in sum of end times
28B:	end for

Once all tasks in the output schedule have been allocated to task execution slots, execution slots are assigned to resources to generate the final resource allocation using matchmaking algorithm - Part B presented in Table 5.9. Execution slots with specific allocation requirements due to the presence of fixed tasks are assigned to the appropriate resources (Lines 1B-5B for map task execution slots and Lines 6B-10B for reduce task execution slots). Unlike the matchmaking algorithm for EAMR-RM, task execution slots which do not execute any fixed tasks are not assigned to resources randomly until each

resource is assigned a number of map and reduce task execution slots equal to its capacity. Instead, task execution slots are sorted in non-decreasing order of end time of their latest finishing assigned task, or latest end time, and allocated to resources such that the difference in latest end time among all assigned slots (both map task execution slots and reduce task execution slots) is minimized.

For example, consider a scenario where task slot s_4 with latest end time of 30 is to be assigned to either resource r_1 or resource r_2 . Task slot s_1 with latest end time of 25 and task slot s_2 with latest end time of 35 are assigned to resource r_1 . Task slot s_3 with latest end time of 42 is assigned to resource r_2 . In this scenario, task slot s_4 will be assigned to resource r_1 because the total difference among latest end times of assigned slots ($|30-25|+|30-35|=10$) is less than the difference for resource r_2 ($|30-42|=12$). When assigning unallocated map and reduce task execution slots to active resources, only those active resources with fewer assigned map and reduce task execution slots than permitted by their map and reduce slot capacity are considered as potential candidates. Unallocated map task execution slots are assigned to resources in Lines 11B-19B. Unallocated reduce task execution slots are assigned to resources in Lines 20B-28B. When calculating the difference in latest end times, a latest end time of 0 is used for resources with no assigned task execution slots and execution slots with no assigned tasks. This ensures that unused task execution slots will be grouped together and assigned to the same resources when possible.

The goal of this allocation procedure is to reduce the number of resources required to execute the tasks present on the system by assigning task execution slots which execute tasks to resources which are already being used and assigning task execution slots which

do not execute any tasks to resources which are unused. Furthermore, task execution slots are assigned to resources in such a way that the difference among latest end times is minimized. Because of this, resources are more likely to transition from a state where all assigned slots are utilized to a state where the resource is fully unused than to be in a state where some slots are utilized and others are unused for an extended period of time.

5.2.2.4 Resource Scaling Procedure

The HEAMR-RM algorithm scales the number of active resources to reduce the energy consumption of idle resources. When the system is lightly loaded, unused resources may be disabled to save energy. If system load increases, resources may be enabled to ensure that arriving jobs are not needlessly rejected by the system. The resource scaling procedure is invoked after matchmaking completes and all tasks are allocated to resources and the current schedule is updated.

The number of active resources is scaled using a reactive threshold based approach. After all tasks are scheduled and assigned to resources and the current schedule is updated, average resource utilization is calculated over the upcoming *scaling interval*. Resource utilization is averaged over a scaling interval to determine what changes are required to the number of active resources for the immediate future. The length of the scaling interval is set to the length of time required to enable a resource. By measuring resource utilization over the length of time required to enable a resource, HEAMR-RM can determine if the resource will still be required by the time it is enabled. If the average resource utilization exceeds an upper threshold, additional resources are enabled (if available) to accommodate the load. If the average resource utilization is below a lower threshold, resources are disabled to reduce idle energy consumption. Non-fixed tasks

assigned to resources which are disabled will be rescheduled and reallocated to the remaining active resources.

Average resource utilization is expressed as *Scaled Average Resource Utilization*, SU , as shown in Equation 5.4 where τ is the current time and i is the length of the scaling interval. Recall from Chapter 3 that a_t is the scheduled start time for task t , e_t is the execution time for task t when executing at the maximum permitted frequency f^{high} , c_t is the scheduled execution time for task t including the impact of frequency scaling, and c_r^{map} is the map task capacity of resource r . Note that scaled average resource utilization is calculated separately for map and reduce task execution slots.

$$SU^{map} = \frac{\sum_{t \in T^{map}} (\max(0, \min(\text{time} + i, a_t + c_t) - \max(\text{time}, a_t)) * \frac{e_t}{c_t})}{\sum_{r \in R_{active}} c_r^{map} * i} \quad 5.4$$

The expression in Equation 5.4 calculates the overlap between the scaling interval and all scheduled tasks. For a visual representation of this procedure, refer to Figure 5.8. The contribution of each task to the resource utilization is scaled proportional to the ratio of the scheduled execution frequency and the maximum permitted operating frequency. Tasks executing at higher frequency have a larger impact on resource utilization.

Figure 5.8: Scaling Interval Task Overlap

Scaled average resource utilization is a measure of the total work done per unit time by task execution slots of all active resources during the scaling interval. A scaled average resource utilization of 1 indicates that all execution slots of active resources are executing tasks at the maximum permitted operating frequency during the entire scaling interval. A scaled average resource utilization of 0 indicates that no execution slot among active resources is executing any task during the scaling interval.

The resource scaling procedure determines how many resources should be enabled or disabled based on the resource scaling rules presented in Table 5.10. If either SU^{map} or SU^{red} exceed the upper threshold UT , one additional resource is enabled if possible (Rule 1). In rare cases where the CP Optimizer is unable to determine a feasible schedule where all job deadlines are met and an arriving job is rejected, two additional resources are enabled if possible regardless of the scaled average resource utilization to reduce the likelihood of additional job rejections in the future (Rule 2). Although it may be possible to accept jobs that would otherwise be rejected by enabling additional resources, HEAMR-RM rejects arriving jobs that are not guaranteed to complete before their deadlines using the resources that are currently active to reduce complexity of the resource management algorithms. Experimental results presented in Section 5.2.4.1 demonstrate that this decision does not result in HEAMR-RM rejecting substantially more jobs than alternative approaches for tested arrival distributions. Performance analysis of the proposed techniques using alternative arrival distributions is discussed as future work in Section 6.2. The number of active resources cannot exceed the total number of pre-acquired resources m (Rule 3).

Enabling or disabling multiple resources at a time permits the resource manager to more quickly adjust to significant changes in system load. Two lower thresholds are used to determine if resources should be disabled. If both SU^{map} and SU^{red} fall below the lower threshold LT_1 two resources are disabled if possible (Rule 4). Otherwise, if both SU^{map} and SU^{red} fall below the lower threshold LT_2 one resource is disabled if possible (Rule 5). To handle arriving MapReduce jobs, a minimum number of active resources, m_{min} , must be maintained (Rule 6). The number of resources which must remain active can be configured as a resource manager tuning parameter. The time required to transition a resource from active to inactive is assumed to be negligible. The exact values of UT , LT_1 , and LT_2 are resource manager tuning parameters.

Table 5.10: Resource Scaling Rules

$SU^{map} > UT \text{ or } SU^{red} > UT \rightarrow \text{enable 1 resource}$	(Rule 1)
$job \text{ rejection} \rightarrow \text{enable 2 resources}$	(Rule 2)
$ R_{active} \leq R $	(Rule 3)
$SU^{map} < LT_1 \text{ and } SU^{red} < LT_1 \rightarrow \text{disable 2 resources}$	(Rule 4)
$SU^{map} < LT_2 \text{ and } SU^{red} < LT_2 \rightarrow \text{disable 1 resource}$	(Rule 5)
$ R_{active} \geq m_{min}$	(Rule 6)

The resource scaling algorithm is presented in Table 5.11. When invoked by the HEAMR-RM algorithm, the resource scaling algorithm calculates the scaled average map and reduce resource utilization using Equation 5.4 (Line 1) and determines how many resources should be enabled or disabled based on the resource scaling rules presented in Table 5.10 (Line 2). The resource scaling algorithm will perform different actions depending on if resources are enabled or disabled.

Any resource which is not already active or in the process of transitioning from inactive to active can be chosen to be enabled (Line 3). Resources which are enabled by the

resource scaling procedure are not available for matchmaking and scheduling for some period of time while the resource transitions from an inactive to active state. HEAMR-RM tracks the activation process and updates R_{active} when the resource is fully enabled and ready to execute tasks (Line 5). The exact length of time required to enable a resource depends on the system architecture of the cloud resources. Although tasks cannot be allocated to a resource until it is active, idle energy consumption is measured from the start of the activation process to model the energy cost of enabling the resource (Line 4).

Table 5.11: Resource Scaling Algorithm

1:	calculate SU^{map} and SU^{red}
2:	determine how many resources to enable or disable based on resource scaling rules
When enabling resources:	
3:	randomly select an inactive resource r in R to enable
4:	begin tracking idle energy consumption of resource r
5:	configure HEAMR-RM to add r to R_{active} after resource enable time
When disabling resources:	
6:	sort resources in non-decreasing order of latest fixed task finishing time among all assigned task execution slots
7:	select resource r from top of list to disable
8:	remove r from R_{active} and fixed tasks assigned to r from T
9:	solve CP feasibility problem
10:	if feasible solution exists
11:	configure HEAMR-RM to disable r after assigned fixed tasks are completed and stop tracking idle energy consumption once disabled
12:	invoke HEAMR-RM algorithm to reschedule and reallocate non-fixed tasks which were previously assigned to r
13:	else
14:	add r to R_{active} and fixed tasks assigned to r to T

Disabling resources is more challenging. When the resource scaling procedure determines one or more resources should be disabled based on the scaled average resource utilization and threshold values, the resource manager uses the current schedule

to determine which resources to disable. Resources are sorted in non-decreasing order of latest fixed task finishing time among all assigned task execution slots (Line 6). Any resources which are not currently executing any tasks (i.e. do not execute any fixed tasks) are selected first as they can be disabled immediately. If all resources are currently busy executing fixed tasks, the resource manager selects to disable the resource which will complete all of its fixed tasks first (line 7). These resources are scheduled to be disabled once execution of the assigned fixed tasks is complete.

Once the resources are chosen but before they are disabled, the CP Optimizer is used to determine if a feasible solution exists to schedule the workload without the selected resources (Line 9). The CP formulation for HEAMR-RM presented in Table 5.7 of Section 5.2.2.2 is used without the objective function to determine the feasibility of the problem. When solved without an objective function, the CP Optimizer will return the first feasible solution found. The selected resources are removed from R_{active} and fixed tasks allocated to those resources are not considered because they will be completed as scheduled before the resource is disabled (Line 8). The result is returned quickly since the CP Optimizer is searching for any feasible solution rather than the solution with minimum energy consumption.

If a feasible solution exists (i.e. the CP Optimizer is able to schedule the tasks in the system without the selected resources) the resources are disabled (Line 11). The HEAMR-RM algorithm is invoked again to manage any newly arriving jobs and to reschedule and reallocate tasks that were previously assigned to resources which are now disabled (Line 12). Since the disabled resources have been removed from R_{active} their task execution slots no longer contribute to the combined resource pool. Note that re-

invocation of the HEAMR-RM algorithm is only required when the resource scaling procedure invalidates the current schedule by disabling one or more resources with assigned tasks. This ensures that the HEAMR-RM algorithm does not repeat indefinitely when further resource scaling is not required. Although in some cases a resource may be removed from the active set before the resource has completed executing fixed tasks, energy consumption is calculated up to the time when the resource is disabled.

If a feasible solution does not exist, the selected resources remain active to guarantee that accepted jobs will complete before their client specified deadlines. The resources are re-added to the active set and the previous schedule and resource allocated is maintained (Line 14).

5.2.3 Performance Evaluation

This section describes the performance evaluation conducted for HEAMR-RM. Similar to other research in this area (ex. [8], [39]) a simulation based analysis is used to evaluate the performance of the proposed technique. As in the case of EAMR-RM discussed in Section 5.1, simulation is suitable for this research which does not concern exact values of performance and energy metrics but instead focuses on their general relationship with system and workload parameters. Furthermore, simulation is effective for evaluating the relative improvement in performance and energy metrics of a resource manager compared to alternative strategies.

The remainder of this section discusses the experimental setup, performance metrics, system and workload parameters, resource manager turning parameters, and experimental results.

5.2.3.1 Experimental Setup

Resource management for an open stream of deadline constrained MapReduce jobs was used to evaluate the performance of HEAMR-RM. The CP Optimizer, matchmaking routine, and HEAMR-RM algorithm, including resource scaling procedure, were executed on a single PC running Windows 7 Professional with a 2.93GHz Intel Core i7 CPU and 8GB of RAM. Job arrivals were simulated during discrete event simulation and the HEAMR-RM algorithm was invoked to perform matchmaking and scheduling for each arriving job. Execution of the workload was simulated for a cluster or an environment of pre-acquired cloud resources to determine the task and idle energy consumption, average job turnaround time, and number of jobs rejected by the system.

Experiments were run long enough to ensure that the system reaches steady state. Note that experiments including requests characterized by client specified earliest start times in the future must be run for significantly longer to reach steady state compared to experiments which do not include such tasks.

5.2.3.2 Performance Metrics

Similar to the performance metrics discussed for EAMR-RM in Section 5.1.3.2, performance of HEAMR-RM is evaluated on the basis of energy consumption, expressed as Energy Savings ($E_{savings}$), Average Job Turnaround Time (TAT), and Proportion of Rejected Jobs (ω) (See Section 5.1.3.2).

The performance of HEAMR-RM is compared to that of three alternative resource management strategies for an open stream of deadline constrained MapReduce jobs that are discussed next:

- (1) A *non-energy-aware* (NEA) resource manager based on MRCP-RM described in [12]. NEA does not incorporate any strategies to reduce energy consumption and does not consider energy consumption in resource management decisions.
- (2) A *resource-scaling-only* (RSO) resource manager based on the techniques described in [7], [30], and [33]. RSO attempts to collocate tasks with similar execution times to reduce the number of resource required to execute a workload and permits unused resources to be disabled to reduce idle energy consumption. RSO does not include strategies to reduce energy consumption due to task execution.
- (3) The Energy Aware MapReduce Resource Manager for open systems (EAMR-RM) described in Section 5.1. EAMR-RM utilizes a DVFS based approach to reduce the energy consumption due to task execute but does not incorporate techniques to reduce energy consumption due to idle resources. Energy Savings for all resource managers is expressed relative to the energy consumption of NEA.

5.2.3.3 System and Workload Parameters

Table 5.12 outlines the system and workload parameters used in the experiments. To allow for easier comparison in resource manager performance, system and workload parameters are similar to those presented for EAMR-RM in Table 5.4 of Section 5.1.3.3 that are based on the synthetic workloads used by [28] and [12]. The additional parameters required in the evaluation of HEAMR-RM are briefly discussed.

Since HEAMR-RM uses matchmaking algorithm - Part B presented in Table 5.9 in Section 5.2.2.3 to collocate tasks with similar execution times and permits unused resources to be disabled to save energy, m specifies the maximum number of resources which can be active at a given time (i.e. the number of pre-acquired resources in the

cluster or cloud environment). When conducting performance evaluation of alternative resource management strategies which do not perform resource scaling (NEA and EAMR-RM) the number of active resources will always be fixed to the maximum value. As in research performed in [33], the time required to enable a resources is set to 30 seconds. Note that because the resources are pre-acquired, it is not necessary to deploy a new virtual machine on the cloud each time a resource is enabled. r_{enable} is merely the time required to transition the resource from a disabled or energy saving sleep state to an operational state. Some experiments are performed with $r_{enable} = 60$ seconds to observe if longer resource enable times negatively impact system performance.

Table 5.12: HEAMR-RM System and Workload Parameters

Name	Value
System Parameters	
Maximum number of resources, m -Map task capacity, c_r^{map} -Reduce task capacity, c_r^{red} -Resource enable time (s), r_{enable}	$m = \{25, \mathbf{50}, 75, 100\}$ $c_r^{map} = 2, c_r^{red} = 2$ $r_{enable} = \{\mathbf{30}, 60\}$
Workload Parameters	
Job arrival rate, λ (Jobs/s)	$\lambda = \{0.001, 0.005, \mathbf{0.01}, 0.015, 0.02, 0.025\}$ Inter-arrival times $\sim$ exponentially distributed
Job j: -Number of map tasks, k_j^{map} -Number of reduce tasks k_j^{red}	$k_j^{map} \sim DU(1, 100)$ $k_j^{red} \sim DU(1, k_j^{map})$
Task execution times (s) -Map task (me) -Reduce task (re)	$me \sim DU(1, e^{max})$ where $e^{max} = \{25, \mathbf{50}, 75, 100\}$ $re = \frac{(3 * \sum_{t \in T_j^{map}} e_t)}{k_j^{red}} + DU(1, 10)$
Job earliest start time, s_j (s)	$s_j = \begin{cases} v_j, & x = 0 \\ v_j + DU(1, s_{max}), & x = 1 \end{cases}$ where v_j is the arrival time of job j $x \sim \text{Bernoulli}(p), p = \{\mathbf{0}, 0.2, 0.4, 0.6, 0.8, 1\}$ $s_{max} = 5000$
Job deadline, d_j (s)	$d_j = s_j + (U(1, l) * \sum_{t \in T_j} e_t)$ where $l = 3$

HEAMR-RM includes the necessary enhancements required to handle requests with earliest start time greater than arrival time. To model these requests, the earliest start time of a job may either be considered equal to its arrival time (v_j) or a time in the future. Similar to work performed in [12], a Bernoulli distribution is used to determine whether or not the earliest start time of a job is equal to its arrival time. The probability that an arriving job will have an earliest start time which is not equal to its arrival time is denoted by p . The parameter s_{max} is the upper bound on the discrete uniform distribution used to determine the value added to v_j for generating s_j when $x=1$.

Because experiments performed in Section 5.1.4.2 for EAMR-RM determined that moderate variations of laxity factor did not impact Energy Savings, laxity factor is not varied.

Performance analysis of the HEAMR-RM is conducted using the processor model based on the frequency/voltage levels of the AMD Opteron 2380 measured by researchers in [11] and shown in Table 5.5 of Section 5.1.3.4. A linear regression of the power dissipation in each frequency/voltage level is used to express power dissipation as a function of operating frequency (see Equation 5.3 in Section 5.1.3.4).

5.2.3.4 Tuning Parameters

HEAMR-RM operation and performance depends on the configured values of various tuning parameters. Tuning parameters and values used in experimentation are shown in Table 5.13.

As discussed previously in Section 5.2.2.4, a minimum number of resource must remain active to handle arriving jobs when system load is low to ensure jobs are not needlessly rejected. The minimum number of active resources is set to 10% of the maximum number

of resources available to the system. 50% of the maximum number of available resources are initially active to handle burst of arrivals when the system starts. A reasonable number of resources are required during this period to ensure jobs are not needlessly rejected while the system reaches steady state. The impact of idle energy consumption due to over-provisioning of resources during this transient state is negligible on final performance measurements.

Table 5.13: HEAMR-RM Tuning Parameters

Parameter name	Value
Minimum active resources, m_{min}	$m_{min} = 0.1 * m$
Upper Threshold, UT	0.50
Lower Threshold 1, LT_1	0.25
Lower Threshold 2, LT_2	0.10

Recall from Section 5.2.2.4 that the upper and lower threshold values for scaled average resource utilization determine when the system should enable additional resources to handle an increase in system load or disable resources due to decreased load to reduce idle energy consumption. Based on the linear relationship between frequency and power for the AMD Opteron 2380 shown in Equation 5.3 in Section 5.1.3.4, minimum energy consumption for a workload will occur when all slots of all active resources are executing tasks at the minimum possible operating frequency. Therefore, the ideal scaled average resource utilization in this case is:

$$ideal\ SU = \frac{f^{low}}{f^{high}} = \frac{800}{2500} = 0.32 \quad 5.5$$

The values of UT , LT_1 , and LT_2 are selected to keep scaled average utilization close to this value while ensuring that additional resources are enabled when necessary to limit the number of rejected jobs. Systems using alternative processor models may require

different threshold values for proper operation of the resource manager and good system performance.

5.2.4 Experimental Results

Performance analysis was conducted using a *factor-at-a-time* approach. System performance is observed while one parameter is varied and the others are held constant at their default values. The default value of each parameter is shown in boldface in the “Value” columns of Table 5.12 and Table 5.13. Halt percentage, used to determine the idle energy consumption of resources as discussed in Section 3.1.1, is set to 10% for all experiments based on the power measurements for the AMD Opteron 2380 from [51]. Each experiment was performed a sufficient number of times to generate a confidence interval for Energy Savings (refer to Equation 5.2 in Section 5.1.3.2) at a 95% confidence level with upper and lower bounds within $\pm 5\%$ of the mean.

5.2.4.1 Impact of Arrival Rate

Figure 5.9 shows the impact of arrival rate on Energy Savings ($E_{savings}$) for the tested resource managers. As arrival rate increases, Energy Savings achieved by HEAMR-RM decreases. At very low arrival rates, HEAMR-RM and RSO save a substantial amount of energy compared to NEA and significantly outperform EAMR-RM. When requests arrive slowly, the system is lightly loaded and few resources are required to execute the workload. HEAMR-RM and RSO take advantage of this to disable unused resources and substantially reduce energy consumption. HEAMR-RM outperforms RSO at low to moderate arrival rates because, although a less significant proportion of total energy consumption, energy consumption due to task execution can be reduced using DVFS. At moderate arrival rates HEAMR-RM and EAMR-RM have similar performance that is

superior to that of RSO. As arrival rate increases to moderate levels, energy consumption due to task execution, which can be effectively reduced using DVFS, becomes a more significant component of total energy consumption and the DVFS based techniques perform better than RSO. At high arrival rates, none of the energy aware resource managers save significant energy compared to NEA; all resources are required to be active and execute tasks close to f^{high} to ensure client QoS requirements are satisfied. The performance of HEAMR-RM and EAMR-RM is slightly better than that of RSO.

Figure 5.9: Impact of Arrival Rate on Energy Savings

The impact of arrival rate on Average Job Turnaround Time (TAT) is shown in Figure 5.10. Similar to EARM-RM, TAT for HEAMR-RM has a non-monotonic relationship with arrival rate. At low arrival rates, TAT is small because resource contention is low. At moderate arrival rates, TAT is larger because the system uses DVFS to apply a reduced operating frequency close to f^{low} to the execution of most tasks to save energy while still ensuring all client specified deadlines are met. At high arrival rates, Average job Turnaround Time is low because the system must execute tasks at a higher CPU

frequency to keep up with more frequent job arrivals. Average job Turnaround Time for HEAMR-RM is typically higher than for EAMR-RM because resource scaling leads to higher contention on active resources. *TAT* for NEA and RSO increases as arrival rate increases due to increased resource contention; jobs may not be able to start as early as permitted when arrival rate is high. For a given arrival rate, NEA and RSO that do not perform frequency scaling seem to substantially outperform HEAMR-RM and EAMR-RM in terms of average turnaround time. Note that although HEAMR-RM and EAMR-RM produce a higher *TAT* in comparison to NEA and RSO, they do not incur additional SLA violations because of this increased average turnaround time because the resource managers generate a schedule that guarantees that the deadlines of all accepted jobs will be met.

Figure 5.10: Impact of Arrival Rate on Average Job Turnaround Time

The impact of arrival rate on Proportion of Rejected Jobs (ω) is shown in Figure 5.11. In rare cases where the system state prevents the resource manager from guaranteeing that an arriving job will complete before its deadline the job must be rejected to ensure that

SLA requirements of accepted jobs are never violated. HEAMR-RM and RSO may have higher proportion of rejected jobs than EAMR-RM because resource scaling can lead to scenarios where insufficient resources are active to handle an arriving job. However, experimental results show that configured resource scaling threshold values enable additional resources when necessary and ω does not increase significantly compared to EAMR-RM. As expected, ω increases as arrival rate increases. As arrival rate and resource contention increase, additional jobs are rejected to generate a schedule that guarantees that the deadlines of all accepted jobs are met. ω was found to be less than 0.01 for all tested system and workload parameters.

Figure 5.11: Impact of Arrival Rate on Proportion of Rejected Jobs

5.2.4.2 Impact of Map Task Execution Time

The impact of e^{max} , the upper bound on map task execution time, on Energy Savings and Average Job Turnaround Time can be seen in Figure 5.12 and Figure 5.13 respectively. $E_{savings}$ decreases as average map task execution time increases. For low values of e^{max} , the system can apply a CPU operating frequency close to f^{low} to the execution of most

tasks and disable unused resources to save energy. When e^{max} is high, Energy Savings is low because additional resources are required to execute the workload and tasks must be executed at higher frequencies due to resource contention. HEAMR-RM performs the best with EAMR-RM performing at comparable levels for e^{max} greater than 50. The performance of both these strategies are observed to be superior to that of RSO. As expected, TAT for all resource managers increases as e^{max} increases. This result is intuitive; longer tasks are expected to lead to longer turnaround times. Relative performance of the four techniques follows the same pattern as observed in the case of Figure 5.10: resource management techniques that do not perform frequency scaling (NEA and RSO) significantly outperform the DVFS-based techniques (HEAMR-RM and EAMR-RM) in terms of average turnaround time.

Figure 5.12: Impact of e^{max} on Energy Savings

Figure 5.13: Impact of e^{max} on Average Job Turnaround Time

5.2.4.3 Impact of Maximum Number of Resources

Figure 5.14 shows the impact of maximum number of resources on Energy Savings. $E_{savings}$ increases as maximum number of resources increases. For low values of m , many or all resources need to be active at all times and executing tasks at a high operating frequency is required to ensure that jobs are completed before their client specified deadlines. As m is increased to moderate levels, most or all of the available resources may be required to execute the workload, but tasks can be executed at lower CPU frequency to reduce energy consumption due to task execution while still guaranteeing that all deadlines are met. As m is increased to higher levels, additional resources may not be required to execute the workload. HEAMR-RM outperforms EAMR-RM for large values of m because unused resources are disabled to save energy. For all tested values of m , both HEAMR-RM and EAMR-RM demonstrate higher energy savings than RSO.

Figure 5.14: Impact of Maximum Resources on Energy Savings

The impact of maximum number of resources on Average Job Turnaround Time is shown in Figure 5.15. TAT for all resource managers decreases as maximum number of available resources increases. This result is intuitive; jobs can be completed more quickly when tasks are spread out over a larger number of resources. Recall from Section 5.2.2.4 that the goal of the resource scaling procedure in the HEAMR-RM algorithm is to maintain the number of active resources such that all execution slots of active resources are executing tasks at the minimum permitted operating frequency. HEAMR-RM will use resources as necessary to keep utilization within the ideal range, thus decreasing resource contention and Average Job Turnaround Time, and disable resources which are not required. Further Increasing m will have no impact on TAT once all tasks are able to begin execution as soon as permitted due to low resource contention. These additional resources will go unused and will be disabled by the resource manager to reduce energy consumption. As discussed in Section 5.2.4.1, the DVFS-based resource management techniques (HEAMR-RM and EAMR-RM) have higher average job turnaround times

compared to the resource management techniques that do not scale CPU frequency to reduce energy consumption (RSO and NEA).

Figure 5.15: Impact of Maximum Resources on Average Job Turnaround Time

5.2.4.4 Impact of Earliest Start Time

The impact of the proportion of arriving requests with earliest start time greater than arrival time, p , on Energy Savings and Average Job Turnaround Time is shown in Figure 5.16 and Figure 5.17 respectively. Note that since EAMR-RM is not devised for use with these requests, it is not included in this comparison. The presence and proportion of requests with earliest start time greater than arrival time has negligible impact on Energy Savings and Average Job Turnaround Time for all resource managers. HEAMR-RM is able to effectively handle resource management of such requests. Once again, for a given p , HEAMR-RM leads to significantly higher energy savings in comparison to RSO. The energy savings of HEAMR-RM comes from a tradeoff with average job turnaround time compared to RSO and NEA. However, as discussed in Section 5.2.4.1, this increase in

average job turnaround time does not correspond to an increase in the number of SLA violations because HEAM-RM guarantees that the deadline of all accepted jobs will be met.

Figure 5.16: Impact of p on Energy Savings

Figure 5.17: Impact of p on Average Job Turnaround Time

5.2.4.5 Impact of Resource Enable Time

Figure 5.18 shows the impact of arrival rate on $E_{savings}$ and TAT for various resource enable times. Performance results show that HEAMR-RM continues to perform well for a higher resource enable time of 60 seconds. For any given arrival rate, longer resource enable times lead to negligible increases in Average Job Turnaround Time and negligible decreases in Energy Savings in most cases. Results show that longer resource enable times do not have a substantial impact on the capabilities of HEAMR-RM to function properly and save energy.

Figure 5.18: Impact of Resource Enable Time

5.2.5 Summary of Results

The Hybrid Energy Aware MapReduce resource manager is an effective resource management technique for matchmaking and scheduling an open stream of deadline constrained MapReduce jobs such that processor energy consumption is minimized and client specified SLAs are not violated. HEAMR-RM combines the resource management approaches discussed in [7], [30], and [33] which collocate tasks with similar execution

times and permit unused resources to be disabled to reduce idle energy consumption with the DVFS based approach used by EAMR-RM to reduce energy consumption due to task execution. The resulting hybrid approach provides significant energy savings compared to the independent techniques: RSO and EAMR-RM.

HEAMR-RM is implemented in Java and performs matchmaking and scheduling separately. The scheduling problem is formulated as an optimization using CP and solved first using the CPLEX CP Optimizer. Matchmaking is performed next using a best-fit allocation which groups tasks with similar execution times on the same resource to reduce the number of resources required to execute the workload. A resource scaling routine then calculates utilization of active resources and determines if additional resources should be enabled to manage an increase in system load or if resources should be disabled to reduce idle energy consumption.

Simulation based factor-at-a-time experiments compare the effectiveness of the proposed technique for a variety of system and workload parameters with several alternative resource management strategies. Performance analysis shows that the proposed technique achieves the desired goal of improving system performance over EAMR-RM when the environment is lightly loaded and resource contention is low. Furthermore, the DVFS-based scheduling procedure permits HEAMR-RM to reduce energy consumption due to task execution and outperform RSO, which only considers reducing energy consumption due to idle resources, regardless of system load. Important observations and key insights into system performance are summarized next:

- HEAMR-RM performs well when system resources are lightly or moderately loaded (see Figure 5.9)

- Under light load, resource scaling is an effective strategy for reducing energy consumption due to idle energy requirement of unused resources. HEAMR-RM reduces energy consumption by up to 70% over NEA and 50% over EAMR-RM when resource contention is low.
- Under moderate load, energy consumption due to task execution is a more significant component of total energy consumption and can be substantially reduced using DVFS. HEAMR-RM and EAMR-RM have similar performance under moderate system load.
- Under high load, none of these resource management strategies offer substantial energy savings over NEA; many or all resources must be active and most tasks must be executed at CPU frequencies close to f^{high} to manage the workload.
- Energy savings were accompanied by increases in Average Job Turnaround Time relative to NEA. However, no SLAs are violated since all client specified deadlines of accepted jobs are guaranteed to be met.
- Tests with longer resource enable times show only minor differences in performance and demonstrate that HEAMR-RM operation is not dependent on excessively short resource enable times.
- HEAMR-RM does not reject significantly more jobs than the alternative resource management techniques used in the performance evaluation. Less than 1% of jobs were rejected for all tested system and workload parameters.

Chapter 6: Conclusions

This chapter summarizes the contributions of this thesis and presents conclusions. Directions for future research are also discussed.

6.1 Summary and Conclusions

This thesis introduces four energy aware resource management techniques for matchmaking and scheduling MapReduce workloads comprising jobs each of which are characterized by SLAs that include client specified deadlines. The complex matchmaking and scheduling problem is formulated as an optimization problem using Constraint Programming (CP) with the objective of minimizing data center server energy consumption and solved using commercial optimization software IBM CPLEX.

The proposed techniques reduce energy consumption by applying a Dynamic Voltage and Frequency Scaling (DVFS) based approach to reduce the CPU supply voltage during execution of some tasks. Limiting CPU supply voltage decreases energy consumption at the cost of reduced CPU operating frequency and thus increased task execution times. Careful selection of the level of CPU supply voltage reduction and the tasks to which it should be applied based on available slack in job deadlines ensures that energy consumption is reduced without violating client specified SLAs.

Performance analysis for all techniques is conducted using a simulation based approach with synthetic workloads similar to other researchers in this area (ex. [8], [12], [28]). Synthetic workloads are useful as they enable a greater degree of control over experimental conditions and permit easier investigation into the impact of varying various system and workload parameters on overall performance. Simulation is suitable for this research which does not concern obtaining exact values of performance and

energy metrics but instead focuses on their general relationship with system and workload parameters and comparison with alternative resource management strategies.

6.1.1 Energy Aware Batch Resource Manager

The energy aware MapReduce resource manager for batch workloads presented in Section 4.1 performs matchmaking and scheduling with the goal of minimizing data center server energy consumption. This technique considers workloads characterized by soft deadline requirements. The upper bound on the ratio of the maximum number of jobs that are permitted to miss their client specified deadlines to the total number of jobs in the workload is submitted by the client as part of the request. The energy aware batch MapReduce resource manager is only suitable for batch workloads where all jobs and workload characteristics are known *a priori*. The matchmaking and scheduling problem for the entire workload is solved in a single step using the CP formulation and implementation discussed in Sections 3.2.1 and 3.2.2 respectively.

Simulation based performance analysis shows that the proposed technique saves a considerable amount of energy compared to System I, a *non-energy-aware* batch MapReduce resource management technique based on “Approach 3” from [21]. Energy savings between 25% and 45% were observed for various tested values of missed deadline ratio bound and laxity factor. Smaller but still substantial energy savings were observed for workloads with fewer total tasks. Energy savings were accompanied by small to moderate increases in batch completion time. However, increased batch completion time does not correspond to additional SLA violations; the ratio of the number of jobs which miss their deadlines to the total number of jobs is always less than the maximum missed deadline ratio bound submitted by the client.

6.1.2 Configurable Batch Resource Manager

The performance results for the energy aware batch MapReduce resource manager presented in Section 4.1 demonstrate the tradeoff among batch completion time, number of missed deadlines, and energy consumption. These three correlated parameters are intimately related with service provider profit. A lower batch completion time enables the service provider to accept and execute more batches. Reducing the number of missed deadlines is important for increasing client satisfaction which aids in customer retention. Energy consumption is a significant component of data center operating cost. The configurable batch resource manager attempts to achieve an effective tradeoff among batch completion time, number of missed deadlines, and energy consumption by permitting the service provider to specify upper bounds on batch completion time and missed deadline ratio. The proposed technique determines a resource allocation and task schedule for the input batch workload which minimizes data center energy consumption while satisfying all MapReduce task scheduling requirements and additional service provider specified upper bounds.

The simulation based performance analysis shows that the proposed technique achieves an effective tradeoff among the three correlated parameters of interest. Energy savings relative to System I, a *non-energy-aware* resource management technique for batch workloads based on “Approach 3” from [21], can be increased at the cost of increased batch completion time or increased proportion of missed deadlines. Tight job deadline and batch completion time requirements were found to have a similar impact on task scheduling; whichever requirement is stricter will dominate system behavior. For example, permitting a large number of deadlines to be missed but requiring the batch to

be completed quickly will result in an output schedule with low to moderate energy savings and few missed deadlines. The proposed technique was observed to scale well from workloads with hundreds of tasks to workloads with several thousand tasks; energy savings was independent on workload size.

6.1.3 Energy Aware MapReduce Resource Manager for Open Systems

The *Energy Aware MapReduce Resource Manager*, *EAMR-RM*, for open systems includes additional resource management techniques and performance optimization required to make the technique suitable for an open stream of deadline constrained MapReduce job arrivals rather than a batch workload. Arriving MapReduce jobs trigger invocation of the EAMR-RM algorithm which performs matchmaking and scheduling for the new tasks and considers rescheduling existing tasks which have not yet started execution. Rescheduling tasks which have not started execution enables the resource manager to respond to changes in workload characteristics; tasks can be reordered to accommodate arriving jobs with stricter deadlines and CPU frequency can be adjusted to handle an increase in system load. Tasks which have started execution are never migrated and will run as scheduled until completed.

Resource manager performance is compared to NEA, a *non-energy-aware* resource manager based on MRCP-RM from [12] which does not consider energy consumption when making scheduling decisions and minimizes average job turnaround time. Turnaround time minimization is not necessarily the best strategy for data center operators who are only obligated to deliver services by the agreed deadline specified in the SLA. Resource management strategies that focus only on minimizing average job

turnaround time fail to consider optimization opportunities which can reduce data center energy consumption.

Simulation based performance analysis shows that EAMR-RM achieves up to 45% energy savings compared to NEA. Non-monotonic relationships between energy savings and e^{max} , the upper bound on map task execution time, as well as energy savings and arrival rate were observed. The system performed best for moderate values of these parameters. When these parameters are low, system load is low, and idle energy consumption, which is not reduced by the DVFS based approach, is a significant proportion of total energy consumption. When arrival rate and e^{max} are high, system load is high and tasks must be executed at frequencies close to f^{high} to ensure all deadlines are met.

Energy savings were accompanied by increases in average job turnaround time. However, the increase in average turnaround time does not correspond to an increase in SLA violations; all client specified deadlines are met.

6.1.4 Hybrid Energy Aware MapReduce Resource Manager for Open Systems

Performance analysis for EAMR-RM shows that the system does not perform well when lightly loaded since the DVFS based approach cannot reduce energy consumption due to idle resources. The *Hybrid Energy Aware MapReduce Resource Manager*, *HEAMR-RM*, for open systems aims to address this deficiency by incorporating techniques described in [7], [30], and [33] to colocate tasks with similar execution times and disable unused resources. HEAMR-RM includes reactive threshold based resource scaling procedure to control the number of active resources on the system. When system load is low, resources

are disabled to reduce idle energy consumption. When system load is high, resources are enabled to accommodate additional job arrivals.

A preliminary performance analysis shows that the proposed technique achieves its goal. HEAMR-RM achieves an up to 70% reduction in energy consumption over NEA and an up to 50% reduction in energy consumption over EAMR-RM when system load is low. Unlike EAMR-RM where non-monotonic relationships were observed, energy savings for HEAMR-RM increases as maximum number of resources increases, decreases as arrival rate increases, and decreases as e^{max} , the upper bound on map task execution time, increases. When the system is lightly loaded, energy is saved primarily by disabling unused resources. When under moderate load, most resources are active and the majority energy savings comes from decreasing task operating frequency using DVFS. HEAMR-RM does not significantly reduce energy consumption when system load is high; all resources must be active and executing tasks at close to the maximum permitted operating frequency.

Energy savings are accompanied by a minor increase in average turnaround time over EAMR-RM due to increased resource contention because of disabled resources. This increase in average turnaround time does not correspond to an increase in the number of SLA violations.

6.2 Future Work

Directions for future work are discussed next. Research issues that are worth of investigation include:

- Incorporating MapReduce task execution time prediction models and devising an approach to handle error in predicted execution times caused by errors in estimates provided by clients.
- Enhancing the proposed resource management techniques to consider data locality in matchmaking and scheduling. Such an enhanced technique would attempt to place a task on a resource close to the disk containing the data processed by the task such that delays due to data transfer over the interconnection network are reduced.
- Investigating the performance of prototypes of the proposed resource management techniques on real Hadoop clusters.
- Conducting an extension of the performance evaluation of HEAMR-RM to further investigate the impact of variations in additional workload, system, and tuning parameters on performance.
- Evaluating performance of EAMR-RM and HEAMR-RM for alternate arrival processes including different arrival rates for different client classes.
- Devising resource management strategies suitable for IO intensive MapReduce jobs based on the detailed IO intensive task execution time models presented in [37] and [53].

References

- [1] B. Rapolu, "Analytics and the Internet of Things in Aerospace: How is the IoT affecting the aerospace industry?," The Innovation Enterprise Ltd., 5 June 2015. [Online]. Available: https://channels.theinnovationenterprise.com/articles/7319-analytics-and-the-internet-of-things-in-aerospace?utm_content=buffer629a0&. [Accessed October 2016].
- [2] D. Lau, J. Liu, S. Majumdar, B. Nandy, M. St-Hilaire and C. S. Yang, "A cloud-based approach for smart facilities management," in *'13 IEEE Conference on Prognostics and Health Management (PHM)*, Gaithersburg, MD, 2013, pp. 1-8.
- [3] J. Liebowitz, *Big Data and Business Analytics*, Boca Raton, FL: CRC Press, 2013.
- [4] J. G. Koomey, "Growth in data center electricity use 2005 to 2010," Analytics Press, Oakland, CA, 2011.
- [5] J. G. Koomey, C. Belady, M. Patterson, A. Santos and K. Lange, "Assessing trends over time in performance, costs, and energy use for servers," Analytics Press, Oakland, CA, 2009.
- [6] J. Hamilton, "Cooperative Expendable Micro-Slice Servers (CEMS): Low Cost, Low Power Servers for Internet-Scale Services," in *4th Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, 2009, pp. 1-8.
- [7] Y. J. Chiang, Y. C. Ouyang and C. H. Hsu, "An Efficient Green Control Algorithm in Cloud Computing for Cost Optimization," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 145-155, 2015.

- [8] A. Verma, L. Cherkasova, V. S. Kumar and R. H. Campbell, "Deadline-based workload management for MapReduce environments: Pieces of the performance puzzle," in *2012 IEEE Network Operations and Management Symposium*, Maui, HI, 2012, pp. 900-905.
- [9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on Large Clusters," in *OSDI '04 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, 2004, pp. 137-149.
- [10] K. H. Kim, R. Buyya and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters," in *7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, Rio De Janeiro, 2007, pp. 541-548.
- [11] L. Tan and Z. Chen, "Slow Down or Halt: Saving the Optimal Energy for Scalable HPC Systems," in *ICPE '15 Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, Austin, TX, 2015, pp. 241-244.
- [12] N. Lim, S. Majumdar and P. Ashwood-Smith, "A Constraint Programming-based resource management technique for processing MapReduce jobs with SLAs on clouds," in *2014 43rd International Conference on Parallel Processing (ICPP)*, Minneapolis MN, 2014, pp. 411-421.
- [13] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang and W. Shi, "Energy-Aware Scheduling of MapReduce Jobs for Big Data Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 10, pp. 2720-2733, 2015.
- [14] A. Gregory and S. Majumdar, "A Constraint Programming Based Energy Aware

- Resource Management Middleware for Clouds Processing MapReduce Jobs with Deadlines," in *ICPE '16 Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering (2nd International Workshop on Performance Analysis of Big data Systems (PABS))*, Delft, The Netherlands, 2016, pp. 15-20.
- [15] A. Gregory and S. Majumdar, "A Configurable Energy Aware Resource Management Technique for Optimization of Performance and Energy Consumption on Clouds," in *8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Luxembourg City, Luxembourg, 2016, pp. 184-192.
- [16] A. Gregory and S. Majumdar, "Energy Aware Resource Management for MapReduce Jobs with Service Level Agreements in Cloud Data Centers," in *6th IEEE International Symposium on Cloud and Service Computing (SC2)*, Nadi, Fiji, 2016, pp. 568-577.
- [17] Apache Software Foundation, "Hadoop Wiki," Apache, 20 January 2016. [Online]. Available: <https://wiki.apache.org/hadoop/>. [Accessed February 2016].
- [18] S. Baker, "The Two Flavors of Google," *Bloomberg Businessweek Magazine*, 12 December 2017.
- [19] F. Rossi, P. van Beek and T. Walsh, "Chapter 4: Constraint Programming," in *Handbook of Knowledge Representation*, San Diego, CA, Elsevier Science, 2004, pp. 181-211.
- [20] J. Hooker, "Logic, optimization and constraint programming," *INFORMS Journal on Computing*, vol. 14, no. 4, pp. 295-321, 2002.

- [21] N. Lim, S. Majumdar and P. Ashwood-Smith, "Engineering resource management middleware for optimizing the performance of clouds processing MapReduce jobs with deadlines," in *ICPE '14 Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, Dublin, 2014, pp. 161-172.
- [22] D. Morrison, S. Jacobson, J. Sauppe and E. Sewell, "Branch-and-bound algorithms," *Discrete Optimization*, vol. 19, no. C, pp. 79-102, 2016.
- [23] IBM, "IBM ILOG CPLEX Optimization Studio CP Optimizer User's Manual," IBM Corporation, 2015.
- [24] A. Bockmayr and J. Hooker, "Constraint Programming," in *Handbook of Discrete Optimization*, K. Aardal, G. Nemhauser and R. Weismantle, Eds., San Diego, CA, Elsevier Science, 2005, pp. 559-600.
- [25] Emerson Electric, "Energy Logic: Reducing Data Center Energy Consumption by Creating Savings that Cascade Across Systems," White Paper, Emerson Electric, 2009.
- [26] J. A. Aroca, A. Chatzipapas, A. F. Anta and V. Mancuso, "A measurement-based analysis of the energy consumption of data center servers," in *Proceedings of the 5th international conference on Future energy systems*, Cambridge, United Kingdom, 2014, pp. 63-74.
- [27] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation, "Advanced Configuration and Power Interface Specification, Revision 5.0a," December 6, 2011.
- [28] H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee and S.

- Mukherjee, "Scheduling in mapreduce-like systems for fast completion time," in *INFOCOM, 2011 Proceedings IEEE*, Shanghai, 2011, pp. 3074-3082.
- [29] Y. Wang, J. Tan, W. Yu, L. Zhang, X. Meng and X. Li, "Preemptive ReduceTask Scheduling for Fair and Fast Job Completion," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, San Jose, CA, 2013, pp. 279-289.
- [30] M. Cardosa, A. Singh, H. Pucha and A. Chandra, "Exploiting Spatio-Temporal Tradeoffs for Energy-Aware MapReduce in the Cloud," *IEEE Transactions on Computers*, vol. 61, no. 12, pp. 1737-1751, 2012.
- [31] Yanfei Li, Ying Wang, Bo Yin and Lu Guan, "An energy efficient resource management method in virtualized cloud environment," in *Network Operations and Management Symposium (APNOMS), 2012 14th Asia-Pacific*, Seoul, 2012, pp. 1-8.
- [32] A. Beloglazov and R. Buyya, "Energy efficient resource management in Virtualized Cloud Data Centers," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, Melbourne, Australia, 2010, pp. 826-831.
- [33] D. Çavdar, L. Y. Chen and F. Alagöz, "Green MapReduce for heterogeneous data centers," in *2014 IEEE Global Communications Conference*, Austin, TX, 2014, pp. 1120-1126.
- [34] Y. C. Lee and A. Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, Shanghai, 2009, pp. 92-99.

- [35] H. Topcuoglu, S. Hariri and Min-You Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, 2002.
- [36] L. Wang, G. von Laszewski, J. Dayal and F. Wang, "Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster with DVFS," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, Melbourne, VIC, 2010, pp. 368-377.
- [37] T. Wirtz and R. Ge, "Improving MapReduce energy efficiency for computation intensive workloads," in *Green Computing Conference and Workshops (IGCC), 2011 International*, Orlando, FL, 2011, pp. 1-8.
- [38] L. Xu, Z. Zeng and X. Ye, "Multi-objective optimization based virtual resource allocation strategy for cloud computing," in *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, Shanghai, 2012, pp. 56-61.
- [39] í. Goiri, J. L. Berral, J. O. Fitó, F. Juliá, R. Nou, J. Guitart, R. Gavaldí and J. Torres, "Energy-efficient and multifaceted resource management for profit-driven virtualized data centers," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 718-731, 2012.
- [40] H. M. Fard, R. Prodan, J. J. D. Barrionuevo and T. Fahringer, "A multi-objective approach for workflow scheduling in heterogeneous environments," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, Ottawa, ON, 2012, pp. 300-309.

- [41] L. Zuo, L. Shu, S. Dong, C. Zhu and T. Hara, "A Multi-Objective Optimization Scheduling Method Based on the Ant Colony Algorithm in Cloud Computing," *IEEE Access*, vol. 3, pp. 2687-2699, 2015.
- [42] W. Kim, M. S. Gupta, G. Y. Wei and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, Salt Lake City, UT, 2008, pp. 123-134.
- [43] C. Piguet, C. Schuster and J. L. Nagel, "Optimizing architecture activity and logic depth for static and dynamic power reduction," in *Circuits and Systems, 2004. NEWCAS 2004. The 2nd Annual IEEE Northeast Workshop on*, 2004, pp. 41-44.
- [44] J. P. Halimi, B. Pradelle, A. Guermouche, N. Triquenaux, A. Laurent, J. C. Beyler and W. Jalby, "Reactive DVFS Control for Multicore Processors," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCoM), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, Beijing, 2013, pp. 102-109.
- [45] IBM, "IBM ILOG CPLEX Optimization Studio," IBM Corporation, January 2016.
 [Online]. Available: <http://www-03.ibm.com/software/products/us/en/ibmilogcpleoptistud>.
- [46] IBM, "Detailed Scheduling in IBM ILOG CPLEX Optimization Studio with IBM ILOG CPLEX CP Optimizer," White Paper, IBM Corporation, 2010.
- [47] T. Dong, "Efficient modeling with the IBM ILOG OPL CPLEX Development Bundles," White Paper, IBM Corporation, 2009.

- [48] L. A. Barroso, J. Dean and U. Hölzle, "Web Search for a Planet: The Google Cluster Architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22-28, 2003.
- [49] Intel Corporation, "DC Specifications, 2nd Gen Intel® Core™ Processor Family Desktop Datasheet, Vol. 2," 2011.
- [50] B. L. Agarwal, Basic Statistics, New Delhi: New Age International, 2006.
- [51] AMD Incorporated, "AMD 10h server and workstation processor power and thermal data sheet," 2010.
- [52] E. Hwang and K. H. Kim, "Minimizing Cost of Virtual Machines for Deadline-Constrained MapReduce Applications in the Cloud," in *2012 ACM/IEEE 13th International Conference on Grid Computing*, Beijing, 2012, pp. 130-138.
- [53] S. P. T. Srinivasan and U. Bellur, "WattTime: Novel System Power Model and Completion Time Model for DVFS-Enabled Servers," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, Melbourne, VIC, 2015, pp. 448-455.
- [54] Intel Corporation, "Intel® Xeon Processor E5-1600/E5-2600/E5-4600 Product Families Datasheet, Vol. 1," 2012.