SCHEDULING AND DESIGN IN CLOUD COMPUTING SYSTEMS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of the Ohio State University

By

Yousi Zheng, B.S., M.S.

Graduate Program in Electrical and Computer Engineering

The Ohio State University

2015

Dissertation Committee:

Prasun Sinha, Advisor
Ness B. Shroff, Advisor
Atilla Eryilmaz
Can Emre Koksal
© Copyright by
Yousi Zheng
2015
ABSTRACT

We are witnessing a major data revolution. Both the popularity and variety of cloud computing based applications are on the rise, driven in large part by lightweight data intensive devices and services. These services are inherently delay sensitive, hence it is critical that efficient scheduling algorithms are developed within the cloud to ensure that the end-users performance is not marred by high latencies. Moreover, the 3V (Volume, Velocity, Variety) properties of the data generated in the cloud require that the schedulers to run at high efficiencies, but low complexity in order to ensure good performance and scalability.

This dissertation investigates the development of efficient scheduling algorithms that provides rigorous performance guarantees for cloud computing systems. The research is conducted along the following major problems summarized below.

We first consider the MapReduce paradigm, which is one of the most important and popular frameworks used in cloud computing systems. We focus on the problem of minimizing the total delay of a sequence of jobs in the MapReduce framework, where the jobs arrive over time and need to be processed through both Map and Reduce procedures before leaving the system. For this problem for non-preemptive tasks, no on-line algorithm can achieve a constant competitive ratio. We then construct a slightly weaker metric of performance called the efficiency ratio. We then show a surprising property that, for the flow-time problem, any work-conserving scheduler
has a constant efficiency ratio in both preemptive and non-preemptive scenarios. More importantly, we are able to develop an online scheduler with a very small efficiency ratio of 2, and through simulations we show that it outperforms state-of-the-art schedulers.

Second, we explore ways to simplify scheduling algorithms when the number of servers in a cloud computing system is very large (a practical reality). For the problem of minimizing the total delay in the MapReduce framework, we show that any work-conserving scheduler is asymptotically optimal under a wide range of traffic loads, including the heavy traffic limit. This result implies, somewhat surprisingly, that when we have a large number of machines, there is little to be gained by optimizing beyond ensuring that a scheduler should be work-conserving. For long-running applications, we also study the relationship between the number of machines and the running time, and show sufficient conditions to guarantee the asymptotic optimality of work-conserving schedulers. Further, we run extensive simulations, that indeed verify that when the total number of machines are large, state-of-the-art work-conserving schedulers have similar and close-to-optimal delay performance.

Third, we study what is the proper size (number of machines) needed in the cloud computing system for given quality of service (QoS) requirements. We develop two new heavy traffic limit based results for GI/H/n queueing systems. We characterize the heavy traffic limits for a QoS requirement, and show how the required number of servers should scale with the traffic load, which turns out to be quite sensitive to the QoS requirement. We also conduct numerical studies to support our theoretical results, and show that the heavy traffic limits are accurate even for moderate system sizes. These results lead to good rules of thumb on how to dimension large power efficient cloud computing systems based on different QoS requirements.
Fourth, we study the efficient schedulers for interactive applications. The interactive jobs arrive to the system over time, and are allowed to be partially executed before their deadlines. Their deadlines vary across users and applications, which makes the job scheduling problem very challenging when the overall system performance needs to be optimized. In this part, our objective is to maximize the total utility gain. We focus on the preemptive scenario, where a job in service can be interrupted by other jobs, and its service can be resumed later. We propose a deadline agnostic scheduler, called ISPEED (Interactive Services with Partial ExEcution and Deadlines). Being deadline agnostic is an attractive property of ISPEED because data center schedulers are often not privy to individual job deadlines, and thus schedulers that are deadline dependent may not be amenable to practical implementation. We first prove that ISPEED achieves the maximum total utility when jobs have homogeneous deadlines and their utility functions are non-decreasing and concave. Then, in the case of heterogeneous job deadlines we prove that ISPEED achieves a competitive ratio of $2 + \alpha$, where $\alpha$ is a shape parameter for a large class of non-decreasing utility functions. In the special case of $\alpha = 0$, i.e., the utility functions are concave, ISPEED has a competitive ratio of 2, while no causal scheduler can achieve a competitive ratio smaller than $\frac{\sqrt{5}+1}{2}$. Finally, we show through trace-driven simulations that ISPEED outperforms the state-of-the-art schedulers in a wide range of scenarios.
ACKNOWLEDGMENTS

I would like to thank my Ph.D. advisors, Prof. Ness Shroff and Prof. Prasun Sinha for their guidance and support through my PhD pursuit. They always inspire me to find out better ideas and encourage me to conquer hard problems and situations. Their feedback, suggestions and comments are invaluable in completing my PhD study and this work. I am greatly benefitted from the research discussions with my advisors and other members in our research group.

I would also like to thank my candidacy and dissertation committee members, Prof. Andrea Serrani, Prof. Atilla Eryilmaz, Prof. Can Emre Koksal, for their valuable comments on this dissertation. Their comments and feedback makes this work significantly improved.

I am grateful to Prof. Srikant (University of Illinois at Urbana-Champaign), Prof. Bo Ji (Temple University), Dr. Zizhan Zheng (University of California, Davis), and Dr. Jian Tan (I.B.M. TJ Watson), for their insightful suggestions on research topics related to this dissertation. I would like to thank my lab colleagues (current and graduated) - Srikanth, Bo, Zhoujia, Ruogu, Wenzhuo, Shengbo, Swapna, Shuang, Bin, Yang, Dongyue, Fangzhou, Dong Meng, Irem, Ghada, Zhenzhi, Fei, Fang, Jiashang, Yin, Jia, Zizhan, Yoora, Arman, etc. for an colorful and memorable graduate student life at the Ohio State University. I also thank all other lab colleagues and friends in ECE and CSE department - Zhixue, Dong Li, Tarun, Gopi, Wenjie, Tanmoy, Rupam, etc. for all discussions and help in research and life. Special thanks are due to Jeri
and Patricia for their nice and patient help of coordinating PhD things, answering questions and many other administrative issues.

I am forever indebted to my family, academic advisors, and friends, for their constant support and encouragement. Finally, this dissertation is dedicated to my parents Qingmin Zheng and Junying Wang for their unselfish love, and my wife Siwei Dong for her consistent support and endless love.
VITA

2006 ................................. B.S., Electronics Engineering, Tsinghua University

2009 ................................. M.S., Electronics Engineering, Tsinghua University

2009-Present ......................... Graduate Research Associate, Graduate Teaching Associate, The Ohio State University

PUBLICATIONS


FIELDS OF STUDY

Major Field: Electrical and Computer Engineering

Specialization: Network Science
# TABLE OF CONTENTS

Abstract ................................................................. ii

Acknowledgments ..................................................... v

Vita ................................................................. vii

List of Tables .......................................................... xii

List of Figures .......................................................... xiii

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction ................................. 1</td>
</tr>
<tr>
<td>1.1</td>
<td>Efficient Schedulers for MapReduce Framework ............................ 1</td>
</tr>
<tr>
<td>1.2</td>
<td>Scheduler Design for Large Scale Cloud Computing Systems ................. 3</td>
</tr>
<tr>
<td>1.3</td>
<td>Design of Cloud Computing Systems based on QoS ............................ 4</td>
</tr>
<tr>
<td>1.4</td>
<td>Maximizing Utility of Interactive Services in Cloud Computing Systems .... 5</td>
</tr>
<tr>
<td>2</td>
<td>A New Analytical Technique for Designing Provably Efficient MapReduce Schedulers .......................... 7</td>
</tr>
<tr>
<td>2.1</td>
<td>Introduction ................................. 7</td>
</tr>
<tr>
<td>2.2</td>
<td>Related Work .................................. 9</td>
</tr>
<tr>
<td>2.3</td>
<td>System Model .................................. 10</td>
</tr>
<tr>
<td>2.4</td>
<td>Efficiency Ratio ............................. 15</td>
</tr>
<tr>
<td>2.5</td>
<td>Work-Conserving Schedulers ..................... 19</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Preemptive Scenario .......................... 19</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Non-preemptive Scenario ........................ 29</td>
</tr>
<tr>
<td>2.6</td>
<td>Available-Shortest-Remaining-Processing-Time (ASRPT) Algorithm and Analysis ........................................ 38</td>
</tr>
<tr>
<td>2.6.1</td>
<td>ASRPT Algorithm ................................ 39</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Efficiency Ratio Analysis of ASRPT Algorithm .......................... 40</td>
</tr>
<tr>
<td>2.7</td>
<td>Simulation Results ............................ 46</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Simulation Setting ............................ 46</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.7.2 Efficiency Ratio</td>
<td>47</td>
</tr>
<tr>
<td>2.7.3 Cumulative Distribution Function (CDF)</td>
<td>52</td>
</tr>
<tr>
<td>2.8 Conclusion</td>
<td>52</td>
</tr>
<tr>
<td>3 Exploiting Large System Dynamics for Designing Simple Data Center</td>
<td>55</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>55</td>
</tr>
<tr>
<td>3.2 System Model and Asymptotic Optimality</td>
<td>57</td>
</tr>
<tr>
<td>3.2.1 System Model under MapReduce Framework</td>
<td>57</td>
</tr>
<tr>
<td>3.2.2 Asymptotic Optimality of Schedulers</td>
<td>60</td>
</tr>
<tr>
<td>3.2.3 Assumptions</td>
<td>61</td>
</tr>
<tr>
<td>3.3 Asymptotic Optimality of Work-conserving Schedulers in Preemptive</td>
<td>63</td>
</tr>
<tr>
<td>and Paralleled Scenario</td>
<td></td>
</tr>
<tr>
<td>3.4 Asymptotic Optimality of Work-conserving Schedulers in Non-preemptive</td>
<td>68</td>
</tr>
<tr>
<td>and Non-parallelizable Scenario</td>
<td></td>
</tr>
<tr>
<td>3.5 The Relationship between $T$ and $N$</td>
<td>74</td>
</tr>
<tr>
<td>3.6 Simulation Results</td>
<td>76</td>
</tr>
<tr>
<td>3.6.1 Simulation Setting</td>
<td>76</td>
</tr>
<tr>
<td>3.6.2 Fixed Traffic Intensity</td>
<td>78</td>
</tr>
<tr>
<td>3.6.3 Heavy Traffic Scenario</td>
<td>80</td>
</tr>
<tr>
<td>3.6.4 Exponentially Distributed Workload</td>
<td>80</td>
</tr>
<tr>
<td>3.6.5 Uniformly Distributed Workload</td>
<td>82</td>
</tr>
<tr>
<td>3.7 Conclusion</td>
<td>87</td>
</tr>
<tr>
<td>4 Design of a Power Efficient Cloud Computing Environment: Heavy Traffic</td>
<td>88</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>88</td>
</tr>
<tr>
<td>4.2 System Model and QoS Classes</td>
<td>90</td>
</tr>
<tr>
<td>4.2.1 System Model and Preliminaries</td>
<td>90</td>
</tr>
<tr>
<td>4.2.2 Definition of QoS Classes</td>
<td>91</td>
</tr>
<tr>
<td>4.3 Related Work</td>
<td>93</td>
</tr>
<tr>
<td>4.4 Heavy Traffic Limit for the MWT class</td>
<td>94</td>
</tr>
<tr>
<td>4.5 Heavy Traffic Limit for the BWT Class</td>
<td>108</td>
</tr>
<tr>
<td>4.6 Applications in Cloud Computing/Data Centers</td>
<td>111</td>
</tr>
<tr>
<td>4.6.1 Heavy Traffic Limits for Different Classes Summarized</td>
<td>114</td>
</tr>
<tr>
<td>4.6.2 Number of Operational Machines for Different Classes</td>
<td>115</td>
</tr>
<tr>
<td>4.7 Numerical Analysis</td>
<td>117</td>
</tr>
<tr>
<td>4.7.1 Evaluation Setup</td>
<td>117</td>
</tr>
<tr>
<td>4.7.2 Necessity of Class-based Design</td>
<td>117</td>
</tr>
<tr>
<td>4.7.3 Evaluation for the MWT and BWT Classes</td>
<td>120</td>
</tr>
<tr>
<td>4.8 Conclusion</td>
<td>125</td>
</tr>
</tbody>
</table>
5 Forget the Deadline: Scheduling Interactive Applications in Data Centers

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>126</td>
</tr>
<tr>
<td>5.2</td>
<td>System Model and ISPEED Scheduler</td>
<td>129</td>
</tr>
<tr>
<td>5.2.1</td>
<td>System Model</td>
<td>129</td>
</tr>
<tr>
<td>5.2.2</td>
<td>ISPEED scheduler</td>
<td>130</td>
</tr>
<tr>
<td>5.3</td>
<td>Performance Analysis of ISPEED with Homogeneous Deadlines</td>
<td>131</td>
</tr>
<tr>
<td>5.4</td>
<td>Performance Analysis of ISPEED with Heterogenous Deadlines</td>
<td>135</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Lower Bound on the Competitive Ratio for All Schedulers:</td>
<td>136</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Competitive Ratio of ISPEED</td>
<td>137</td>
</tr>
<tr>
<td>5.5</td>
<td>Practical Considerations</td>
<td>142</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Initialization Cost</td>
<td>142</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Multiple Tasks Per Job</td>
<td>143</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Cost of Parallelization and Interruption</td>
<td>143</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Robustness to Incomplete Information</td>
<td>144</td>
</tr>
<tr>
<td>5.5.5</td>
<td>System Overloading</td>
<td>145</td>
</tr>
<tr>
<td>5.6</td>
<td>Simulation Result</td>
<td>145</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Understanding the Nature of Utility Function</td>
<td>145</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Evaluation Setup</td>
<td>146</td>
</tr>
<tr>
<td>5.6.3</td>
<td>Homogeneous Deadlines</td>
<td>148</td>
</tr>
<tr>
<td>5.6.4</td>
<td>Heterogeneous Deadlines</td>
<td>149</td>
</tr>
<tr>
<td>5.6.5</td>
<td>Fairness of Different Schedulers</td>
<td>151</td>
</tr>
<tr>
<td>5.6.6</td>
<td>Different Cost Parameters</td>
<td>152</td>
</tr>
<tr>
<td>5.6.7</td>
<td>Job Initialization Cost</td>
<td>153</td>
</tr>
<tr>
<td>5.6.8</td>
<td>Multiple Tasks Per Job</td>
<td>154</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion</td>
<td>155</td>
</tr>
<tr>
<td>5.8</td>
<td>Related Work</td>
<td>156</td>
</tr>
</tbody>
</table>

6 Summary

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Summary of Contributions</td>
<td>158</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Efficient Schedulers for MapReduce Framework</td>
<td>158</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Scheduler Design for Large Scale Cloud Computing Systems</td>
<td>159</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Design of Cloud Computing Systems based on QoS</td>
<td>160</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Maximizing Utility of Interactive Services in Cloud Computing Systems</td>
<td>160</td>
</tr>
<tr>
<td>6.2</td>
<td>Future Directions</td>
<td>161</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Applications of Efficiency Ratio</td>
<td>161</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Algorithm Design with Incomplete Information</td>
<td>162</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Joint Design of Scheduling and Networking</td>
<td>163</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Scheduling with Multiple Complex Phases Precedence</td>
<td>164</td>
</tr>
</tbody>
</table>

Bibliography 166
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Comparison between State-of-the-art Schedulers and ISPEED</td>
</tr>
<tr>
<td>5.2</td>
<td>Default parameters in our evaluations</td>
</tr>
</tbody>
</table>

# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The set of all schedulers can be classified into Case I and II.</td>
</tr>
<tr>
<td>2.2</td>
<td>The Quick Example</td>
</tr>
<tr>
<td>2.3</td>
<td>An example schedule of a work-conserving scheduler</td>
</tr>
<tr>
<td>2.4</td>
<td>The job $i$, which arrives in the $j^{th}$ interval, finishes its Map tasks in $K_j$ time slots and Reduce tasks in $K_j + K_{j+1} + R_i - 1$ time slots.</td>
</tr>
<tr>
<td>2.5</td>
<td>The construction of schedule for ASRPT and DSRPT based on the schedule for SRPT. Observe that the Map tasks are scheduled in the same way. But, roughly speaking, the Reduce tasks are delayed by one slot.</td>
</tr>
<tr>
<td>2.6</td>
<td>Efficiency Ratio (Exponential Distribution, Large Reduce)</td>
</tr>
<tr>
<td>2.7</td>
<td>Efficiency Ratio (Exponential Distribution, Small Reduce)</td>
</tr>
<tr>
<td>2.8</td>
<td>Efficiency Ratio (Uniform Distribution, Large Reduce)</td>
</tr>
<tr>
<td>2.9</td>
<td>Efficiency Ratio (Uniform Distribution, Small Reduce)</td>
</tr>
<tr>
<td>2.10</td>
<td>Convergence of Efficiency Ratios (Exponential Distribution, Large Reduce)</td>
</tr>
<tr>
<td>2.11</td>
<td>Convergence of Efficiency Ratios (Exponential Distribution, Small Reduce)</td>
</tr>
<tr>
<td>2.12</td>
<td>Convergence of Efficiency Ratios (Uniform Distribution, Large Reduce)</td>
</tr>
<tr>
<td>2.13</td>
<td>Convergence of Efficiency Ratios (Uniform Distribution, Small Reduce)</td>
</tr>
<tr>
<td>2.14</td>
<td>CDF of Schedulers (Exponential Distribution, Large Reduce)</td>
</tr>
<tr>
<td>2.15</td>
<td>CDF of Schedulers (Exponential Distribution, Small Reduce)</td>
</tr>
</tbody>
</table>
3.16 Ratio to the Lower Bound (Uniform Distribution, Heavy Traffic, Map and Reduce are Independent) .................................................. 85
3.17 Ratio to the Lower Bound (Uniform Distribution, Heavy Traffic, Map and Reduce are Positively Correlated) ........................................ 86
3.18 Ratio to the Lower Bound (Uniform Distribution, Heavy Traffic, Map and Reduce are Negatively Correlated) ........................................ 86
4.1 Original System and Artificial System .................................................. 96
4.2 The inter-arrival time of the separated queues ........................................... 97
4.3 Artificial System II ........................................................................... 101
4.4 Cloud Computing Architecture .............................................................. 113
4.5 Operational Number of Machines for Exponential Distributed Service Time ................................................................. 118
4.6 Operational Number of Machines for Hyper-exponential Distributed Service Time ............................................................. 119
4.7 Additional Operational Machines for Exponential Distributed Service Time ................................................................. 119
4.8 Additional Operational Machines for Hyper-exponential Distributed Service Time ............................................................. 120
4.9 Traffic Intensity for Exponential Distributed Service Time ...................... 121
4.10 Traffic Intensity for Hyper-exponential Distributed Service Time ............ 121
4.11 Simulation results for the queueing systems of the MWT Class (Log Y-Axis) ................................................................. 122
4.12 Ratio $r_1$ .................................................................................. 123
4.13 Simulation results for the queueing systems of the BWT Class ............... 124
4.14 Simulation Results for the MWT class with Other Inter-Arrival Processes (Log Y-Axis) ............................................................. 124
4.15 Simulation Results for the BWT class with Other Inter-Arrival Processes ................................................................. 125
5.1 Example Utility Function ................................................................. 131
5.2 An example to illustrate the proof of Theorem 5.4.1 .............................. 138
5.3 Utility Function with Setup ........................................... 142
5.4 Utility Function with Multiple Tasks ................................. 143
5.5 Utility Function for Search Engine ................................... 146
5.6 Homogeneous Deadlines ............................................. 148
5.7 Heterogeneous Customers and Deadlines ........................... 150
5.8 Heterogeneous Deadlines ........................................... 151
5.9 Cumulative Distribution Function (CDF) of Schedulers .......... 152
5.10 Parallelization and Interruption Cost ................................. 153
5.11 Job Initialization Cost ............................................... 154
5.12 Multiple Tasks .................................................. 155
CHAPTER 1
INTRODUCTION

Cloud computing systems are used to support a large variety of services, such as social networks (Facebook), web search (Google search), data storage (Dropbox), encyclopedia (Wikipedia), online shopping (Amazon and Ebay), and entertainment (Netflix). Many of these services have now become integral to our daily lives. The scheduling algorithm is a core technique that determines the performance of these individual applications and the cloud computing system as a whole. In this thesis we investigate a critical issue in cloud computing systems, which is to develop simple yet efficient scheduling algorithms that decide which job should be processed at which machines. Increasing size of such systems due to their popularity has presented unprecedented challenges for the design of such scheduling algorithms.

1.1 Efficient Schedulers for MapReduce Framework

We begin with the most popular framework, MapReduce, in cloud computing systems in Chapter 2. MapReduce is designed to process massive amounts of data in a cluster of machines [1]. Although it was first proposed by Google [1], today, many other companies including Microsoft, Yahoo, and Facebook also use this framework. In fact MapReduce has become virtually ubiquitous appearing in a wide variety of
applications such as search indexing, distributed searching, web statistics generation, and data mining.

MapReduce consists of two elemental processes: _Map_ and _Reduce_. For Map tasks, the inputs are divided into several small sets, and processed by different machines in parallel. The output of Map tasks is a set of pairs in <key, value> format. The Reduce tasks then operate on this intermediate data, possibly running the operation on multiple machines in parallel to generate the final result. The aim is to design a _scheduler_, which is responsible for making decisions on which task to execute at what time and on which machine. Typically, we expect this scheduling operation to be performed in a centralized fashion for a given data center. The key metric considered in this chapter is the _total delay in the system_ per job, which includes the time it takes for a job, since it arrives, until it is fully processed. This includes both the delay in waiting before the first task in the job begins to be processed, plus the time for processing all tasks in the job.

A critical consideration for the design of the scheduler is the _dependence_ between the Map and Reduce tasks. For each job, the Map tasks need to be finished before starting any of its Reduce tasks [1]. In Chapter 2, we define the problem of minimizing the total delay (flow-time) in the MapReduce framework and show that no on-line algorithm can achieve a constant competitive ratio under non-preemptive job scheduling. Then, to directly analyze the total delay in the system, we propose a new metric to measure the performance of schedulers, which we call the _efficiency ratio_. Under some weak assumptions, we then show a surprising property that for the flow-time problem any work-conserving scheduler has a constant efficiency ratio in both preemptive and non-preemptive scenarios. We also present an online scheduling algorithm called ASRPT (Available Shortest Remaining Processing Time) with a
very small (less than 2) efficiency ratio, and show that it outperforms state-of-the-art schedulers through simulations.

1.2 Scheduler Design for Large Scale Cloud Computing Systems

The number of large-scale data centers (e.g., those with tens of thousands of machines) is rapidly increasing. In an attempt to minimize the total delay, a new metric to analyze the performance of schedulers called efficiency ratio was introduced in Chapter 2, where loose bounds were provided on the performance of general work-conserving schedulers, and under the special case of preemptive and parallelizable tasks, it was shown that a scheduler called ASRPT can guarantee an efficiency ratio of two. However, ASRPT needs to execute a sorting operation based on the information of workload, which decrease the efficiency of the scheduler when the scale of the system is large.

In Chapter 3, we show that the scheduler design can be significantly simplified by taking advantage of scale. This scale also allows us to design simple schedulers that not only provide a bounded efficiency ratio as in [2, 3], but in fact approach optimality (i.e., the efficiency ratio goes to 1) as the number of machines grows large. Thus, by exploiting the scale inherent in current data centers, we can improve both the complexity of scheduler design in MapReduce-style multiple-phases system and the performance guarantees that we can provide. Under certain weak assumptions, we show that for the flow-time minimization problem any work-conserving scheduler is asymptotically optimal in two classes of scenarios: Preemptive and Parallelizable, and Non-Preemptive and Non-Parallelizable. Our results provide the following two
surprising properties: First, in a large system, it is not necessary to implement complex schedulers, as long as they honor the work-conserving principle, thus ensuring both high performance and scalability. Second, under appropriate and general assumptions, work-conserving schedulers can guarantee asymptotic optimality under both the *Noah Effect* [4] (a large amount of workload arrives into the system in the preemptive and parallelizable scenario) and *Joseph Effect* [4] (a large number of cumulative running jobs remain in the system in the non-preemptive and non-parallelizable scenario).

### 1.3 Design of Cloud Computing Systems based on QoS

Although we know that when the scale of cloud computing systems is large, simple schedulers can achieve asymptotically optimal performance guarantee, we still need to figure out how many machines are needed to design cloud computing systems. This chapter focuses on establishing new heavy traffic limits, and using these limits to figure out the necessary number of machines needed in cloud computing systems based on different QoS requirements.

In Chapter 4, we consider four different classes of QoS, from the most stringent to the weakest. As will be explained later, for two of these QoS classes, we develop new heavy traffic limits. What we will show is that the number of servers required to satisfy the QoS of each of these classes scales differently with respect to traffic intensity. This chapter makes new contributions to heavy traffic analysis, in that it derives new heavy traffic limits for two important QoS classes for queueing systems with general arrival processes and hyper-exponential service time distribution. Using the heavy traffic limits results, this chapter answers the important question for enabling a power efficient data center as an application: How many machines should
a data center have to sustain a specific system load and a certain level of QoS, or equivalently how many machines should be kept “awake” at any given time?

1.4 Maximizing Utility of Interactive Services in Cloud Computing Systems

Most interactive services such as web-search, online gaming, and financial services are now heavily dependent on computations at cloud computing systems because their demands for computing resources are both high and variable. Although fully completed jobs are preferred, partially completed jobs are also acceptable to the end user. For example, in a web search, users mostly care about the top few search results which can be obtained without completing the full job. This is one key difference of interactive services from the scenarios studied in the previous chapters. In Chapter 5, we study the problem of scheduling interactive jobs in a data center with the goal of maximizing the total utility of all jobs. This problem is particularly challenging because future arrivals of jobs and their requirements are unknown, which renders it difficult to make the right scheduling decisions for jobs currently in the system. In Chapter 5, we propose a deadline agnostic scheduler, called ISPEED, and prove that ISPEED maximizes the total utility when the jobs have homogeneous deadlines and their utility functions are non-decreasing and concave. When the jobs have heterogeneous deadlines, we prove that IPSEED achieves a competitive ratio of $2+\alpha$, where $\alpha$ is a shape parameter for a large class of non-decreasing utility functions. In the special case of $\alpha = 0$, i.e., the utility functions are non-decreasing and concave, the competitive ratio of ISPEED can be improved to 2. Since our solution does not require the knowledge of deadlines, it is suitable for real data centers. Indeed, in practice it is non-trivial to obtain the exact information about individual user
deadlines. The performance gap between our solution and the best causal scheduler is small in terms of the competitive ratio (2 vs. $\frac{\sqrt{5} + 1}{2}$), which shows that there is a limited space for further improvement upon our solution. We also show how this basic solution based on theoretical foundations can be enhanced to work well in real settings when a number of practical issues need to be addressed.
CHAPTER 2
A NEW ANALYTICAL TECHNIQUE FOR DESIGNING
PROVABLY EFFICIENT MAPREDUCE SCHEDULERS

2.1 Introduction
MapReduce is a framework designed to process massive amounts of data in a cluster of machines [1]. Although it was first proposed by Google [1], today, many other companies including Microsoft, Yahoo, and Facebook also use this framework. Currently this framework is widely used for applications such as search indexing, distributed searching, web statistics generation, and data mining.

MapReduce has two elemental processes: Map and Reduce. For the Map tasks, the inputs are divided into several small sets, and processed by different machines in parallel. The output of Map tasks is a set of pairs in <key, value> format. The Reduce tasks then operate on this intermediate data, possibly running the operation on multiple machines in parallel to generate the final result.

Each arriving job consists of a set of Map tasks and Reduce tasks. The scheduler is centralized and responsible for making decisions on which task will be executed at what time and on which machine. The key metric considered in this chapter is the total delay in the system per job, which includes the time it takes for a job, since it arrives, until it is fully processed. This includes both the delay in waiting before the
first task in the job begins to be processed, plus the time for processing all tasks in
the job.

A critical consideration for the design of the scheduler is the dependence between
the Map and Reduce tasks. For each job, the Map tasks need to be finished before starting any of its Reduce tasks\(^1\) [1, 6]. Various scheduling solutions has been proposed within the MapReduce framework [7, 8, 6, 5, 9], but analytical bounds on performance have been derived only in some of these works [6, 5, 9]. However, rather than focusing directly on the flow-time, for deriving performance bounds, [6, 5] have considered a slightly different problem of minimizing the total completion time and [9] has assumed speed-up of the machines. Further discussion of these schedulers is given in Section 2.2.

The contributions of this chapter are as follows:

- For the problem of minimizing the total delay (flow-time) in the MapReduce framework under non-preemptive job scheduling, we show that no on-line algorithm can achieve a constant competitive ratio. (Sections 2.3 and 2.4)

- To directly analyze the total delay in the system, we propose a new metric to measure the performance of schedulers, which we call the efficiency ratio. (Section 2.4)

- Under some weak assumptions, we then show a surprising property that for the flow-time problem any work-conserving scheduler has a constant efficiency ratio in both preemptive and non-preemptive scenarios (precise definitions provided in Section 2.3). (Section 2.5)

- We present an online scheduling algorithm called ASRPT (Available Shortest

\(^1\)Here, we consider the most popular case in reality without the Shuffle phase. For discussion about the Shuffle phase, see Section 2.2 and [5].
Remaining Processing Time) with a very small (less than 2) efficiency ratio (Section VI), and show that it outperforms state-of-the-art schedulers through simulations (Section 2.7).

2.2 Related Work

In Hadoop [10], the most widely used implementation, the default scheduling method is First In First Out (FIFO). FIFO suffers from the well known head-of-line blocking problem, which is mitigated in [7] by using the Fair scheduler.

In the case of the Fair scheduler [7], one problem is that jobs stick to the machines on which they are initially scheduled, which could result in significant performance degradation. The solution of this problem given by [7] is delayed scheduling. However, the fair scheduler could cause a starvation problem (referred to [7, 8]). In [8], the authors propose a Coupling scheduler to mitigate this problem, and analyze its performance.

In [6], the authors assume that all Reduce tasks are non-preemptive. They design a scheduler in order to minimize the weighted sum of the job completion times by determining the ordering of the tasks on each processor. The authors show that this problem is NP-hard even in the offline case, and propose approximation algorithms that work within a factor of 3 of the optimal. However, as the authors point out in the article, they ignore the dependency between Map and Reduce tasks, which is a critical property of the MapReduce framework. Based on the work of [6], the authors in [5] add a precedence graph to describe the precedence between Map and Reduce tasks, and consider the effect of the Shuffle phase between the Map and Reduce tasks. They break the structure of the MapReduce framework from job level to task level using the Shuffle phase, and seek to minimize the total completion time of tasks instead of jobs. In both [6] and [5], the schedulers use an LP based lower bound which need to
be recomputed frequently. However, the practical cost corresponding to the delay (or storage) of jobs are directly related to the total flow-time, not the completion time. Although the optimal solution is the same for these two optimization problems, the efficiency ratio obtained from minimizing the total flow-time will be much looser than the efficiency ratio obtained from minimizing the total completion time. For example, if the flow time of each job is increasing with its arriving time, then the total flow time (delay) does not have a constant efficiency ratio, but the completion time may have a constant efficiency ratio. That means that even the total completion time has constant efficiency ratio, it cannot guarantee the performance of real delay.

In [9], the authors study the problem of minimizing the total flow-time of all jobs. They propose an $O(1/\epsilon^5)$ competitive algorithm with $(1 + \epsilon)$ speed for the online case, where $0 < \epsilon \leq 1$. However, speed-up of machines is necessary in the algorithm; otherwise, there is no guarantee on the competitive ratio (if $\epsilon$ decreases to 0, the competitive ratio will increase to infinity correspondingly).

### 2.3 System Model

Consider a data center with $N$ machines. We assume that each machine can only process one job at a time. A machine could represent a processor, a core in a multi-core processor or a virtual machine. Assume that there are $n$ jobs arriving into the system. We assume the scheduler periodically collects the information on the state of the jobs running on the machines, which is used to make scheduling decisions. Such a time slot structure can efficiently reduce the performance degradation caused by data locality (see [7][11]). We assume that the number of job arrivals in each time slot is i.i.d., and the arrival rate is $\lambda$. Each job $i$ brings $M_i$ units of workload for its Map tasks and $R_i$ units of workload for its Reduce tasks. Each Map task has 1 unit
of workload\(^2\), however, each Reduce task can have multiple units of workload. Time is slotted and each machine can run one unit of workload in each time slot. Assume that \(\{M_i\}\) are i.i.d. with expectation \(\bar{M}\), and \(\{R_i\}\) are i.i.d. with expectation \(\bar{R}\). We assume that the traffic intensity \(\rho < 1\), i.e., \(\lambda < \frac{N}{\bar{M} + \bar{R}}\). Assume the moment generating function of workload of arriving jobs in a time slot has finite value in some neighborhood of 0. In time slot \(t\) for job \(i\), \(m_{i,t}\) and \(r_{i,t}\) machines are scheduled for Map and Reduce tasks, respectively. As we know, each job contains several tasks. We assume that job \(i\) contains \(K_i\) tasks, and the workload of the Reduce task \(k\) of job \(i\) is \(R_i^{(k)}\). Thus, for any job \(i\), \(\sum_{k=1}^{K_i} R_i^{(k)} = R_i\). In time slot \(t\) for job \(i\), \(r_{i,t}^{(k)}\) machines are scheduled for the Reduce task \(k\). As each Reduce task may consist of multiple units of workload, it can be processed in either preemptive or non-preemptive fashion based on the type of scheduler.

**Definition 1.** A scheduler is called **preemptive** if Reduce tasks belonging to the same job can run in parallel on multiple machines, can be interrupted by any other task, and can be rescheduled to different machines in different time slots.

A scheduler is called **non-preemptive** if each Reduce task can only be scheduled on one machine and, once started, it must keep running without any interruption.

In any time slot \(t\), the number of assigned machines must be less than or equal to the total number of machines \(N\), i.e., \(\sum_{i=1}^{n} (m_{i,t} + r_{i,t}) \leq N\), \(\forall t\).

Let the arrival time of job \(i\) be \(a_i\), the time slot in which the last Map task finishes execution be \(f_i^{(m)}\), and the time slot in which all Reduce tasks are completed be \(f_i^{(r)}\).

For any job \(i\), the workload of its Map tasks should be processed by the assigned number of machines between time slot \(a_i\) and \(f_i^{(m)}\), i.e., \(\sum_{t=a_i}^{f_i^{(m)}} m_{i,t} = M_i\), \(\forall i \in \{1, \ldots, n\}\).

\(^2\)Because the Map tasks are independent and have small workload [8], such assumption is valid.
For any job $i$, if $t < a_i$ or $t > f_i^{(m)}$, then $m_{i,t} = 0$. Similarly, for any job $i$, the workload of its Reduce tasks should be processed by the assigned number of machines between time slot $f_i^{(m)} + 1$ and $f_i^{(r)}$, i.e., $\sum_{t=f_i^{(m)}+1}^{f_i^{(r)}} r_{i,t} = R_i, \forall i \in \{1, ..., n\}$.

Since any Reduce task of job $i$ cannot start before the finishing time slot of the Map tasks, if $t < f_i^{(m)} + 1$ or $t > f_i^{(r)}$, then $r_{i,t} = 0$.

The waiting and processing time $S_{i,t}$ of job $i$ in time slot $t$ is represented by the indicator function $1_{\{a_i \leq t \leq f_i^{(r)}\}}$. Thus, we define the delay as $\sum_{t=1}^{\infty} \sum_{i=1}^{n} S_{i,t}$, which is equal to $\sum_{i=1}^{n} \left(f_i^{(r)} - a_i + 1\right)$. The flow-time $F_i$ of job $i$ is equal to $f_i^{(r)} - a_i + 1$. The objective of the scheduler is to determine the assignment of jobs in each time slot, such that the cost of delaying the jobs or equivalently the flow-time is minimized.

For the preemptive scenario, the problem definition is as follows:

$$\min_{m_{i,t}, r_{i,t}} \sum_{i=1}^{n} \left(f_i^{(r)} - a_i + 1\right)$$

s.t. $\sum_{i=1}^{n} (m_{i,t} + r_{i,t}) \leq N$, $r_{i,t} \geq 0$, $m_{i,t} \geq 0$, $\forall t$, \hspace{1cm} (2.3.1)

$$\sum_{t=a_i}^{f_i^{(m)}} m_{i,t} = M_i, \sum_{t=f_i^{(m)}+1}^{f_i^{(r)}} r_{i,t} = R_i, \forall i \in \{1, ..., n\}.$$

In the non-preemptive scenario, the Reduce tasks cannot be interrupted by other jobs. Once a Reduce task begins execution on a machine, it has to keep running on that machine without interruption until all its workload is finished. Also, the optimization problem in this scenario is similar to Eq. (2.3.1), with additional constraints representing the non-preemptive nature, as shown below:
\[
\min \sum_{i=1}^{n} \left( f_i^{(r)} - a_i + 1 \right)
\]
\[
s.t. \sum_{i=1}^{n} \left( m_{i,t} + \sum_{k=1}^{K_i} r_{i,t}^{(k)} \right) \leq N, \forall t,
\]
\[
\sum_{t=a_i}^{f_i^{(m)}} m_{i,t} = M_i, m_{i,t} \geq 0, \forall i \in \{1, \ldots, n\},
\]
\[
\sum_{t=f_i^{(m)}+1}^{f_i^{(r)}} r_{i,t}^{(k)} = R_i^{(k)}, \forall i \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, K_i\},
\]
\[
r_{i,t}^{(k)} = 0 \text{ or } 1, \text{ if } 0 < \sum_{s=0}^{t-1} r_{i,s}^{(k)} < R_i^{(k)}.
\]

We show that the offline scheduling problem is strongly NP-hard by reducing 3-PARTITION to the decision version of the scheduling problem.

Consider an instance of 3-PARTITION: Given a set \( A \) of 3\( m \) elements \( a_i \in \mathbb{Z}^+, i = 1, \ldots, 3m \), and a bound \( N \in \mathbb{Z}^+ \), such that \( N/4 < a_i < N/2, \forall i \) and \( \sum a_i = mN \). The problem is to decide if \( A \) can be partitioned into \( m \) disjoint sets \( A_1, \ldots, A_m \) such that \( \sum_{a_i \in A_k} a_i = N, \forall k \). Note that by the range of \( a_i \)'s, every such \( A_k \) must contain exactly 3 elements.

Given an instance of 3-PARTITION, we consider the following instance of the scheduling problem. There are \( N \) machines in the system (without loss of generality, we assume \( N \mod 3 = 0 \)), and 3\( m \) jobs, \( J_1, \ldots, J_{3m} \), where each job arrives at time 0, and \( M_i = a_i, R_i = \frac{2}{3} N - a_i \). Furthermore, let \( \omega = 3m^2 + 3m \). The problem is to decide if there is a feasible schedule with a flow time no more than \( \omega \). This mapping can clearly be done in polynomial time.

Let \( S \) denote an arbitrary feasible schedule to the above problem, and \( F(S) \) its flow-time. Let \( n_k^S \) denote the number of jobs that finish in the \( k \)-th time slot in schedule \( S \). We then have \( F(S) = \sum_{k=1}^{\infty} k \times n_k^S \). We drop the superscript in the
following when there is no confusion. It is clear that \( n_1 = 0 \). Let \( N_k = \sum_{j=1}^{k} n_j \) denote the number of jobs finished by time \( k \). The following lemma provides a lower bound on \( N_k \).

**Lemma 2.3.1.** (1) \( N_{2k} \leq 3k \) and \( N_{2k+1} \leq 3k + 1, \forall k \). (2) \( \sum_{k=1}^{2m} N_k \leq 3m^2 \) and the lower bound is attained only if the following condition holds:

\[
N_{2k} = 3k, \forall k \in \{1, ..., m\}, N_{2k+1} = 3k, \forall k \in \{1, ..., m - 1\}.
\] (2.3.3)

**Proof.** Since \( M_i + R_i = 2N/3 \), at most \( 3k \) jobs can finish by time slot \( 2k \), hence \( N_{2k} \leq 3k \). Similarly, \( N_{2k+1} \leq \left\lfloor \frac{3(2k+1)}{2} \right\rfloor = 3k + 1 \). Now consider the second part.

First if Eq. (2.3.3) holds, we have \( \sum_{k=1}^{2m} N_k = \sum_{k=1}^{m} 3k + \sum_{k=1}^{m-1} 3k = 3m^2 \). Now assume \( N_{2k+1} = 3k + 1 \) for some \( k \in \{1, ..., m - 1\} \). Then we must have \( N_{2k} < 3k \). Otherwise the first \( 2k \) time slots will be fully occupied by the first \( 3k \) jobs by the fact that \( M_i + R_i = 2N/3 \), and hence both the \( M \)-task and \( R \)-task of the \((3k+1)\)-th job have to be done in the \((2k+1)\)-th time slot, a contradiction. By a similar argument, we also have \( N_{2k+2} < 3k \). Hence if \( N_{2k+1} = 3k + 1 \) for one or more \( k \in \{1, ..., m - 1\} \), we have \( \sum_{k=1}^{2m} N_k < 3m^2 \). Therefore, \( 3m^2 \) is a lower bound of \( \sum_{k=1}^{2m} N_k \), which is attained only if Eq. (2.3.3) holds. ■

**Lemma 2.3.2.** \( F(S) \geq 3m^2 + 3m \) and the lower bound is attained only if Eq. (2.3.3) holds.

**Proof.** We note that \( F(S) = \sum_{k=1}^{\infty} k \times n_k = \sum_{k=1}^{\infty} [(2m + 1) - (2m + 1 - k)] \times n_k \geq (2m + 1) \sum_{k=1}^{\infty} n_k - \sum_{k=1}^{2m} (2m + 1 - k)n_k = (2m + 1)3m - \sum_{k=1}^{2m} N_k \geq 6m^2 + 3m - 3m^2 = 3m^2 + 3m \), where the last inequality follows from Lemma 2.3.1. The lower bound is attained only if \( n_k = 0 \) for \( k > 2m \) and \( \sum_{k=1}^{2m} N_k = 3m^2 \), which holds only when Eq. (2.3.3) holds. ■
Theorem 2.3.3. The scheduling problem (both preemptive and non-preemptive) is NP-complete in the strong sense.

Proof. The scheduling problem is clearly in NP. We reduce 3-Partition to the scheduling problem using the above mapping. If the answer to the partition problem is ‘yes’, then $A$ can be partitioned into disjoint sets $A_1, ..., A_m$ each containing $3$ elements. Consider the schedule $S$ where for $k = 1, ..., m$, the $M$-tasks of the three jobs corresponding to $A_k$ are scheduled in the $k$-th time slot, and the $R$-tasks of them are scheduled in the $(k + 1)$-th time slot. The schedule is clearly feasible. Furthermore, Eq. (2.3.3) holds in this schedule. Therefore, $F(S) = 3m^2 + 3m$ by Lemma 2.3.2. On the other hand, if there is a schedule $S$ for the scheduling problem such that $F(S) = 3m^2 + 3m$, then Eq. (2.3.3) holds by Lemma 2.3.2. An induction on $k$ then shows that for each $k = 1, ..., m$, the $(2k - 1)$-th time slot must be fully occupied by the $M$-tasks of three jobs, and the $2k$-th time slot must be fully occupied by the corresponding $R$-tasks, which corresponds a feasible partition to $A$. Since 3-PARTITION is strongly NP-complete, so is the scheduling problem.

2.4 Efficiency Ratio

The competitive ratio is often used as a measure of performance in a wide variety of scheduling problems. For our problem, the scheduling algorithm $S$ has a competitive ratio of $c$, if for any total time $T$, any number of arrivals $n$ in the time $T$, any arrival time $a_i$ of each job $i$, any workload $M_i$ and $R_i$ of Map and Reduce tasks with respect to each arrival job $i$, the total flow-time $F^S(T, n, \{a_i, M_i, R_i; i = 1...n\})$ of scheduling algorithm $S$ satisfies the following:

$$\frac{F^S(T, n, \{a_i, M_i, R_i; i = 1...n\})}{F^*(T, n, \{a_i, M_i, R_i; i = 1...n\})} \leq c,$$

(2.4.1)
where
\[
F^∗(T, n, \{a_i, M_i, R_i; i = 1...n\})
= \min_S F^S(T, n, \{a_i, M_i, R_i; i = 1...n\}),
\] (2.4.2)
is the minimum flow time of an optimal off-line algorithm.

For our problem, it is easy to construct an arrival pattern such that no scheduler can achieve a constant competitive ratio when jobs are non-preemptive. The basic idea is as follows. Consider a single machine and the question is when should a Reduce task be scheduled so as to minimize the flow time. If the scheduler chooses to schedule the Reduce task early, then future arrivals with very small workload could be unnecessarily delayed until this workload is completed. Similarly, if the scheduler chooses to delay scheduling this Reduce task and no future arrivals were to occur, then the delay is also unnecessarily large.

Consider the simplest example of non-preemptive scenario in which the data center only has one machine, i.e., \(N = 1\). There is a job with 1 Map task and 1 Reduce task. The workload of Map and Reduce tasks are 1 and \(2Q + 1\), respectively. Without loss of generality, we assume that this job arrives in time slot 1. We can show that, for any given constant \(c_0\), and any scheduling algorithm, there exists a special sequence of future arrivals and workload, such that the competitive ratio \(c\) is greater than \(c_0\).

For any scheduler \(S\), we assume that the Map task of the job is scheduled in time slot \(H + 1\), and the Reduce task is scheduled in the \((H + L + 1)st\) time slot \((H, L \geq 0)\). Then the Reduce task will last for \(2Q + 1\) time slots because the Reduce task cannot be interrupted. This scheduler’s operation is given in Fig. 2.2(a). Observe that any arbitrary scheduler’s operation can be represented by choosing appropriate values for \(H\) and \(L\) possibilities: \(H + L > 2(c_0 - 1)(Q + 1) + 1\) or \(H + L \leq 2(c_0 - 1)(Q + 1) + 1\). We show the set of all schedulers as Case I and II in Fig. 2.1.

Now let us construct an arrival pattern \(A\) and a scheduler \(S_1\). When \(H + L >
Figure 2.1: The set of all schedulers can be classified into Case I and II.

$2(c_0 - 1)(Q + 1) + 1$, the arrival pattern $A$ only has this unique job arrive in the system. When $H + L \leq 2(c_0 - 1)(Q + 1) + 1$, $A$ has $Q$ additional jobs arrive in time slots $H + L + 2$, $H + L + 4$, ..., $H + L + 2Q$, and the Map and Reduce workload of each arrival is 1. The scheduler $S_1$ schedules the Map task in the first time slot, and schedules the Reduce task in the second time slot when $H + L > 2(c_0 - 1)(Q + 1) + 1$. When $H + L \leq 2(c_0 - 1)(Q + 1) + 1$, $S_1$ schedules the Map function of the first arrival in the same time slot as $S$, and schedules the last $Q$ arrivals before scheduling the Reduce task of the first arrival. The operation of the scheduler $S_1$ is depicted in Figs. 2.2(b) and 2.2(d).

**Case I: $H + L > 2(c_0 - 1)(Q + 1) + 1$.**

In this case, the scheduler $S$ and $S_1$ are shown in Figs. 2.2(a) and 2.2(b), respectively. The flow-time of $S$ is $L + 2Q + 1$, and the flow-time of $S_1$ is $2Q + 2$. So, the competitive ratio $c$ of $S$ must satisfy the following:

$$c \geq \frac{H + L + 2Q + 1}{2Q + 2} > c_0. \quad (2.4.3)$$

**Case II: $H + L \leq 2(c_0 - 1)(Q + 1) + 1$.**

Then for the scheduler $S$, the total flow-time is greater than $(H + L + 2Q + 1) + (2Q + 1)Q$, no matter how the last $Q$ jobs are scheduled. In this case, the scheduler $S$ and $S_1$ are shown in Fig. 2.2(c) and 2.2(d), respectively.
Then, for the scheduler $S$, the competitive ratio $c$ satisfies the following:

$$
c \geq \frac{(H + L + 2Q + 1) + (2Q + 1)Q}{2Q + (H + L + 2Q + 2Q + 1)} \\
\geq \frac{(2Q + 1)(Q + 1)}{6Q + 1 + (2(c_0 - 1)(Q + 1) + 1)} \\
> \frac{(2Q + 1)(Q + 1)}{2(c_0 + 2)(Q + 1)} \geq \frac{Q}{c_0 + 2}.
$$

(2.4.4)

By selecting $Q > c^2_0 + 2c_0$, using Eq. 2.4.4, we can get $c > c_0$.

Thus, we show that for any constant $c_0$ and scheduler $S$, there are sequences of arrivals and workloads, such that the competitive ratio $c$ is greater than $c_0$. In other words, in this scenario, there does not exist a constant competitive ratio $c$. This is because the scheduler does not know the information of future arrivals, i.e., it only makes causal decisions. In fact, even if the scheduler only knows a limited amount of future information, it can still be shown that no constant competitive ratio will hold by increasing the value of $Q$. 

18
We now introduce a slightly weaker notion of performance, called the **efficiency ratio**.

**Definition 2.** We say that the scheduling algorithm $S$ has an efficiency ratio $\gamma$, if the total flow-time $F^S(T, n, \{a_i, M_i, R_i; i = 1...n\})$ of scheduling algorithm $S$ satisfies the following:

$$\lim_{T \to \infty} \frac{F^S(T, n, \{a_i, M_i, R_i; i = 1...n\})}{F^*(T, n, \{a_i, M_i, R_i; i = 1...n\})} \leq \gamma, \text{ with probability 1.} \quad (2.4.5)$$

Later, we will show that for the quick example, a constant efficiency ratio $\gamma$ still can exist (e.g., the non-preemptive scenario with light-tailed distributed Reduce workload in Section 2.5).

## 2.5 Work-Conserving Schedulers

In this section, we analyze the performance of work-conserving schedulers in both preemptive and non-preemptive scenarios.

### 2.5.1 Preemptive Scenario

We first study the case in which the workload of Reduce tasks of each job is bounded by a constant, i.e., there exists a constant $R_{\text{max}}$, s.t., $R_i \leq R_{\text{max}}$, $\forall i$.

Consider the total scheduled number of machines in all the time slots. If all the $N$ machines are scheduled in a time slot, we call this time slot a “developed” time slot; otherwise, we call this time slot a “developing” time slot. We call the time slot a “no-arrival” time slot, if there is no arrival in this time slot. The probability that a time slot is no-arrival time slot only depends on the distribution of arrivals. Since the job arrivals in each time slot are i.i.d., the probability that a time slot is no arrival time slot is equal to $p_0$, and $p_0 < 1$. We define the $j^{th}$ “interval” to be the interval
between the \((j - 1)^{th}\) developing time slot and the \(j^{th}\) developing time slot. (We define the first interval as the interval from the first time slot to the first developing time slot.) Thus, the last time slot of each interval is the only developing time slot in the interval. Let \(K_j\) be the length of the \(j^{th}\) interval, as shown in Fig. 2.3.

Firstly, we show that any moment of the length \(K_j\) of interval is bounded by some constants, as shown in Lemma 2.5.1. Then, we show the expression of first and second moment of \(K_j\) in Corollary 2.5.2. For the artificial system with dummy Reduce workload, the similar results can be achieved as shown in Corollary 2.5.3. Moreover, the internal length are i.i.d. in the artificial system as shown in Remark 2.5.13. Based on all above results, we can get Theorem 2.5.5, which show the performance of any work-conserving scheduler.

**Lemma 2.5.1.** If the algorithm is work-conserving, then for any given number \(H\), there exists a constant \(B_H\), such that \(E[K_j^H] < B_H\), \(\forall j\).

**Proof.** Since \(K_j\) is a positive random variable, then \(E[K_j^H] \leq 1\) if \(H \leq 0\). Thus, we only focus on the case that \(H > 0\).

In the \(j^{th}\) interval, consider the \(t^{th}\) time slot in this interval. (Assume \(t=1\) is the first time slot in this interval.) Assume that the total workload of the arrivals in the \(t^{th}\) time slot of this interval is \(W_{j,t}\). Also assume that the unavailable workload of the Reduce function at the end of the \(t^{th}\) time slot in this interval is \(R_{j,t}\), i.e., the workload of the Reduce functions whose corresponding Map functions are finished in the \(t^{th}\) time slot in this interval is \(R_{j,t}\). The remaining workload of the Reduce function left by the previous interval is \(R_{j,0}\). Then, the distribution of \(K_j\) can be shown as below.
\begin{align*}
P(K_j = k) &= P(W_{j,1} + R_{j,0} - R_{j,1} \geq N, \\
& \quad W_{j,1} + W_{j,2} + R_{j,0} - R_{j,2} \geq 2N, \\
& \quad \ldots \quad (2.5.1) \\
& \quad \sum_{t=1}^{k-1} W_{j,t} + R_{j,0} - R_{j,k-1} \geq (k-1)N, \\
& \quad \sum_{t=1}^{k} W_{j,t} + R_{j,0} - R_{j,k} < kN)
\end{align*}

By the assumptions, for each job, \( R_i \leq R_{\text{max}} \), for all \( i \). Also, the unit of allocation is one machine, thus, in the last slot of the previous interval, the maximum number of finished Map functions is \( N - 1 \). Thus, \( R_{j,0} \leq (N - 1)R_{\text{max}} \). Now, let’s consider the case of \( k > 1 \).

\begin{align*}
P(K_j = k) &\leq P \left( \sum_{t=1}^{k-1} W_{j,t} + R_{j,0} - R_{j,k-1} \geq (k-1)N \right) \\
& \leq P \left( \sum_{t=1}^{k-1} W_{j,t} + (N - 1)R_{\text{max}} \geq (k-1)N \right) \\
& = P \left( \frac{\sum_{t=1}^{k-1} W_{j,t}}{k-1} \geq N - \frac{(N - 1)R_{\text{max}}}{k-1} \right). \\
& \quad (2.5.2)
\end{align*}

By the assumptions, the total (both Map and Reduce) arriving workload \( W_{j,t} \) in each time slot is i.i.d., and the distribution does not depend on the interval index \( j \) or the index of time slot \( t \) in this interval. Then, for any interval \( j \), we know that

\begin{align*}
P(K_j = k) &\leq P \left( \frac{\sum_{s=1}^{k-1} W_s}{k-1} \geq N - \frac{(N - 1)R_{\text{max}}}{k-1} \right), \\
& \quad (2.5.3)
\end{align*}

where \( \{W_s\} \) is a sequence of i.i.d. random variables with the same distribution as \( W_{j,t} \).
Since the traffic intensity $\rho < 1$, i.e., $E[W_s] = \lambda(M + R) < N$.

For any $\epsilon > 0$, there exists a $k_0$, such that for any $k > k_0$, \( \frac{(N-1)R_{\text{max}}}{k-1} \leq \epsilon \).

Then for any $k > k_0$, we have

\[
P(K_j = k) \leq P \left( \frac{\sum_{s=1}^{k-1} W_s}{k-1} \geq N - \epsilon \right), \quad (2.5.4)
\]

We take $\epsilon < N - E[W_s]$ and corresponding $k_0$. For any $k > k_0 = 1 + \left\lceil \frac{(N-1)R_{\text{max}}}{\epsilon} \right\rceil$, by Chernoff’s theorem in the large deviation theory [12], we can achieve that

\[
P \left( \frac{\sum_{s=1}^{k-1} W_s}{k-1} \geq N - \epsilon \right) \leq e^{-kl(N-\epsilon)}, \quad (2.5.5)
\]

where the rate function $l(a)$ is shown as below:

\[
l(a) = \sup_{\theta \geq 0} \left( \theta a - \log(E[e^{\theta W_s}]) \right). \quad (2.5.6)
\]

Since the moment generating function of workload in a time slot has finite value in some neighborhood around 0, for any $0 < \epsilon < N - \lambda(M + R)$, we know that $l(N - \epsilon) > 0$. Thus, we can achieve

\[
E[K_j^H] = \sum_{k=0}^{\infty} k^H P(K_j = k)
\]

\[
= \sum_{k=0}^{k_0} k^H P(K_j = k) + \sum_{k=k_0+1}^{\infty} k^H P(K_j = k)
\]

\[
\leq k_0^H + \sum_{k=2}^{\infty} k^H e^{-kl(N-\epsilon)}
\]

\[
= \left( 1 + \left\lceil \frac{(N-1)R_{\text{max}}}{\epsilon} \right\rceil \right)^H + \sum_{k=2}^{\infty} k^H e^{-kl(N-\epsilon)}. \quad (2.5.7)
\]

Since we know that $l(N - \epsilon) > 0$ in this scenario, $\sum_{k=2}^{\infty} k^H e^{-kl(N-\epsilon)}$ is bounded for given $H$. 22
Thus, given \( H \), \( E[K^H_j] \) is bounded by a constant \( B_H \), for any \( j \), where \( B_H \) is given as below.

\[
B_H \triangleq \min_{\epsilon \in (0, N - \lambda(M + R))] \left\{ \left(1 + \left[\frac{(N - 1)R_{\text{max}}}{\epsilon}\right]\right)^H + \sum_{k=2}^{\infty} k^H e^{-k\epsilon (N - \epsilon)} \right\}.
\]  

(2.5.8)

**Corollary 2.5.2.** \( E[K_j] \) is bounded by a constant \( B_1 \), \( E[K^2_j] \) is bounded by a constant \( B_2 \), for any \( j \). The expressions of \( B_1 \) and \( B_2 \) are shown as below, where the rate function \( l(a) \) is defined in Eq. (2.5.6).

\[
B_1 = \min_{\epsilon \in (0, N - \lambda(M + R))] \left\{ \frac{1 + \left[\frac{(N - 1)R_{\text{max}}}{\epsilon}\right]}{2e^{l(N - \epsilon)} - 1} + \frac{3e^{l(N - \epsilon)} - 1}{e^{l(N - \epsilon)} - 1} \right\},
\]

(2.5.9)

\[
B_2 = \min_{\epsilon \in (0, N - \lambda(M + R))] \left\{ \frac{1 + \left[\frac{(N - 1)R_{\text{max}}}{\epsilon}\right]^2}{4e^{2l(N - \epsilon)} - 3e^{l(N - \epsilon)} + 1} + \frac{3e^{l(N - \epsilon)} - 1}{e^{l(N - \epsilon)} - 1} \right\}.
\]

(2.5.10)

**Proof.** By substituting \( H = 2 \) and \( H = 1 \) in Eq. (2.5.8), we directly achieve Eqs. (2.5.9) and (2.5.10). □

We add some dummy Reduce workload in the beginning time slot of each interval, such that the remaining workload of the previous interval is equal to \((N - 1)R_{\text{max}}\). The added workload are only scheduled in the last time slot of each interval in the original system. Then, in the new artificial system, the total flow time of the real jobs doesn’t change. However, the total number of intervals may decrease, because the added workload may merge some adjacent intervals.

**Corollary 2.5.3.** In the artificial system, the length of new intervals \( \{\tilde{K}_j, j = 1, 2, \ldots\} \) also satisfies the same results of Lemma 2.5.1 and Corollary 2.5.2.
Proof. The proof is similar to the proofs of Lemma 2.5.1 and Corollary 2.5.2. ■

**Remark 2.5.4.** In the artificial system constructed in Corollary 2.5.3, the length of each interval has no effect on the length of other intervals, and it is only dependent on the job arrivals and the workload of each job in this interval. Based on our assumptions, the job arrivals in each time slot are i.i.d.. Also, the workload of jobs are also i.i.d.. Thus, the random variables \( \{ \tilde{K}_j, \ j = 1, 2, \ldots \} \) are i.i.d..

**Theorem 2.5.5.** In the preemptive scenario, any work-conserving scheduler has a constant efficiency ratio

\[
B_2 + B_1^2 \leq \max \left\{ 2, \frac{1-p_0}{\lambda} \right\} \max \left\{ 1, \frac{1}{N(1-\rho)} \right\},
\]

where \( p_0 \) is the probability that no job arrives in a time slot, and \( B_1 \) and \( B_2 \) are given in Eqs. (2.5.11) and (2.5.12).

\[
B_1 = \min_{\epsilon \in (0, N - \lambda(M + R))} \left\{ 1 + \left[ \frac{(N-1)R_{max}}{\epsilon} \right] \right\}, \quad (2.5.11)
\]

\[
B_2 = \min_{\epsilon \in (0, N - \lambda(M + R))} \left\{ \left( 1 + \left[ \frac{(N-1)R_{max}}{\epsilon} \right] \right)^2 \right\}, \quad (2.5.12)
\]

where the rate function \( l(a) \) is defined as

\[
l(a) = \sup_{\theta \geq 0} \left( \theta a - \log(E[e^{\theta W_s}]) \right). \quad (2.5.13)
\]

Proof. Consider the artificial system constructed in Corollary 2.5.3. Since we have more dummy Reduce workload in the artificial system, the total flow time \( \tilde{F} \) of the artificial system is equal to or greater than the total flow time \( F \) of the work-conserving scheduling policy. (The total flow time is defined in Section 2.3.)

Observe that in each interval, all the job arrivals in this interval must finish their Map functions in this interval, and all their Reduce functions will finish before the
end of the next interval. In other words, for all the arrivals in the $j^{th}$ interval, the Map functions are finished in $K_j$ time slots, and the Reduce functions are finished in $K_j + K_{j+1}$ time slots, as shown in Fig. 2.3. (The vertical axis represents the total number of scheduled machines in each time slot.) And this is also true for the artificial system with dummy Reduce workload, i.e., all the arrivals in the $j^{th}$ interval of the artificial system can be finished in $\widetilde{K}_j + \widetilde{K}_{j+1}$ time slots.

Let $A_j$ be the number of arrivals in the $j^{th}$ interval. Since the algorithm is work-conserving, all the available Map functions which arrive before the interval must have finished before this interval. And all the Reduce function which are generated by the arrivals in this interval must finish before the end of the last time slot of the next interval. So the total flow time $F_j$ of work-conserving algorithm for all the arrivals in the $j^{th}$ interval is less than $A_j(K_j + K_{j+1})$. Similarly, the total flow time $\widetilde{F}_j$ for all the arrivals in the $j^{th}$ interval of the artificial system is less than $\widetilde{A}_j(\widetilde{K}_j + \widetilde{K}_{j+1})$, where $\widetilde{A}_j$ is the number of arrivals in the $j^{th}$ interval of the artificial system.

Figure 2.3: An example schedule of a work-conserving scheduler
\( T \triangleq \sum_{j=1}^{m} K_j \) is equal to the finishing time of all the \( n \) jobs. For any scheduling policy, the total flow time is not less than the number of time slots \( \hat{T} \) that the number of scheduled machines is not equal to 0. Thus, for the optimal scheduling method, its the total flow time \( F_j^* \) should be not less than \( \hat{T} \).

For any scheduling policy, each job needs at least 1 time slot to schedule its Map function and 1 time slot to schedule its Reduce function. Thus, the lower bound of the each job’s flow time is 2. Thus, for the optimal scheduling method, its total flow time \( F_j^* \) should not be less than \( 2A_j \).

Let \( F \) be the total flow time of any work-conserving algorithm, and \( F^* \) be the total flow time of optimal scheduling policy. Assume that there are a total of \( m \) intervals for the \( n \) arrivals. Correspondingly, there are a total of \( \tilde{m} \) intervals in the artificial system. Also, based on the description of the artificial system, \( m \geq \tilde{m} \) and \( F = \tilde{F} = \sum_{j=1}^{\tilde{m}} \tilde{F}_j \).

Since \( E(M_i + R_i) > 0, \rho < 1 \), then \( \lim_{n \to \infty} m = \infty \) and \( \lim_{n \to \infty} \tilde{m} = \infty \).

\[
\lim_{n \to \infty} \frac{F}{F^*} = \lim_{n \to \infty} \frac{\tilde{F}}{F^*} = \lim_{n \to \infty} \frac{\sum_{j=1}^{\tilde{m}} \tilde{F}_j}{\sum_{j=1}^{m} F_j^*} \leq \lim_{\tilde{m} \to \infty} \frac{\sum_{j=1}^{\tilde{m}} A_j(\tilde{K}_j + \tilde{K}_{j+1})}{\hat{T}} \leq \lim_{\tilde{m} \to \infty} \frac{\sum_{j=1}^{\tilde{m}} \frac{A_j K_j}{m}}{\hat{T}} + \lim_{\tilde{m} \to \infty} \frac{\sum_{j=1}^{\tilde{m}} A_j \tilde{K}_{j+1}}{\hat{T}}.
\]

(2.5.14)

In the artificial system, the distribution of \( \tilde{A}_j \) is the number of arrivals in \( j^{th} \) interval. Since \( \{\tilde{K}_j, \ j = 1, 2, \ldots\} \) are i.i.d. distributed random variables, then \( \{\tilde{A}_j \tilde{K}_j, \ j = 1, 2, \ldots\} \) are also i.i.d. Based on Strong Law of Large Number (SLLN), we can achieve that

\[
\lim_{\tilde{m} \to \infty} \frac{\sum_{j=1}^{\tilde{m}} \tilde{A}_j \tilde{K}_j}{\tilde{m}} \xrightarrow{w.p.1} E[\tilde{A}_j \tilde{K}_j].
\]

(2.5.15)
For \( \{ A_j \bar{K}_{j+1}, \ j = 1, 2, \ldots \} \), they are identically distributed, but not independent. However, all the odd term are independent, and all the even term are independent.

Based on SLLN, we can achieve that

\[
\lim_{\tilde{m} \to \infty} \frac{\sum_{j=1}^{\tilde{m}} A_j \bar{K}_{j+1}}{\tilde{m}} = \lim_{\tilde{m} \to \infty} \frac{\sum_{j \leq \tilde{m}, j \text{ is odd}} A_j \bar{K}_{j+1}}{\tilde{m}} + \lim_{\tilde{m} \to \infty} \frac{\sum_{j \leq \tilde{m}, j \text{ is even}} A_j \bar{K}_{j+1}}{\tilde{m}}
\]

\[
\overset{w.p.1}{=} \frac{1}{2} E[\bar{A}_j \bar{K}_{j+1}] + \frac{1}{2} E[\bar{A}_j \bar{K}_{j+1}] = E[\bar{A}_j \bar{K}_{j+1}].
\]

Based on Eqs. 2.5.15 and 2.5.16, we can achieve that

\[
\lim_{\tilde{m} \to \infty} \frac{\sum_{j=1}^{\tilde{m}} \bar{A}_j \bar{K}_j}{\tilde{m}} + \lim_{\tilde{m} \to \infty} \frac{\sum_{j=1}^{\tilde{m}} \bar{A}_j \bar{K}_{j+1}}{\tilde{m}} \overset{w.p.1}{=} E[\bar{A}_j \bar{K}_j] + E[\bar{A}_j \bar{K}_{j+1}]
\]

\[
= E[\bar{A}_j \bar{K}_j] + E[\bar{A}_j] E[\bar{K}_{j+1}] = E[E[\bar{A}_j | \bar{K}_j]] + E[\bar{A}_j E[\bar{K}_{j+1}]] \overset{2.5.17}{=}
\]

\[
= E[\bar{K}_j E[\bar{A}_j | \bar{K}_j]] + E[E[\bar{A}_j | \bar{K}_j]] E[\bar{K}_{j+1}] = E[\bar{K}_j (\lambda \bar{K}_j)] + E[\lambda \bar{K}_j E[\bar{K}_{j+1}]] = \lambda (E[\bar{K}_j^2] + E[\bar{K}_j^2]).
\]

For these \( n \) jobs, there are \( m \) developing time slots before the time \( T \), and for each developing time slot, there are at most \( N - 1 \) machines are assigned. Then, we can have

\[
N \sum_{j=1}^{m} (K_j - 1) + (N - 1)m \geq \sum_{s \in \{ \text{Arrivals before } T \}} W_s.
\]

Thus,

\[
\left( N - \frac{\sum_{s \in \{ \text{Arrivals before } T \}} W_s}{T} \right) \frac{\sum_{j=1}^{m} K_j}{m} \geq 1
\]

27
Since the workload $W_s$ of each job is i.i.d., by SLLN, we have

$$
(N - \lambda(\bar{M} + \bar{R})) \lim_{m \to \infty} \frac{\sum_{j=1}^{m} K_j}{m} \geq 1 \text{ w.p.1,} \tag{2.5.20}
$$

i.e.,

$$
\lim_{m \to \infty} \frac{\sum_{j=1}^{m} K_j}{m} \geq \frac{1}{N(1 - \rho)} \text{ w.p.1.} \tag{2.5.21}
$$

Thus,

$$
\lim_{m \to \infty} \frac{T}{m} = \lim_{m \to \infty} \frac{\sum_{j=1}^{m} K_j}{m} \geq \max \left\{ \frac{1}{N(1 - \rho)} \right\} \text{ w.p.1.} \tag{2.5.22}
$$

For any work-conserving scheduler, $T - \hat{T}$ is less then the total number of no-arrival time slots, we have

$$
\lim_{m \to \infty} \frac{\hat{T}}{m} \geq (1 - p_0) \lim_{m \to \infty} \frac{T}{m} \text{ w. p. 1,} \tag{2.5.23}
$$

At the same time, for each time slot that the number of scheduled machines is greater than 0, there are at most $N$ machines that are scheduled. Then, we can get

$$
N\hat{T} \geq \sum_{i=1}^{n} (M_i + R_i). \tag{2.5.24}
$$

Then, we have

$$
\lim_{m \to \infty} \frac{\hat{T}}{m} \geq \frac{\lambda(\bar{M} + \bar{R})}{N} \lim_{m \to \infty} \frac{T}{m} \text{ w.p.1.} = \rho \lim_{m \to \infty} \frac{T}{m} \tag{2.5.25}
$$

Thus,

$$
\lim_{n \to \infty} \frac{F}{F^*} \leq \frac{\lambda(E[\tilde{K}_j^2] + E[\tilde{K}_j^2])}{\max\{\rho, 1 - p_0\} \max\left\{ \frac{1}{N(1 - \rho)} \right\}} \text{ w.p.1.} \tag{2.5.26}
$$

\begin{align*}
& \leq \frac{\lambda(B_2 + B_1^2)}{\max\{\rho, 1 - p_0\} \max\left\{ \frac{1}{N(1 - \rho)} \right\}} \text{ w.p.1.} \\
& \tag{2.5.26}
\end{align*}
Similarly,
\[
\lim_{n \to \infty} \frac{F}{F^*} = \lim_{n \to \infty} \frac{\sum_{j=1}^{m} F_j}{\sum_{j=1}^{m} F^*_j}
\]

\[
\leq \frac{\lim_{\tilde{m} \to \infty} \sum_{j=1}^{\tilde{m}} A_j (K_j + \tilde{K}_{j+1})}{\lim_{m \to \infty} \sum_{j=1}^{m} 2A_j} \leq \frac{\lim_{\tilde{m} \to \infty} \sum_{j=1}^{\tilde{m}} 2A_j}{\lim_{m \to \infty} \sum_{j=1}^{m} 2A_j}
\]

(2.5.27)

With Strong Law of Large Number (SLLN), we can achieve that
\[
\lim_{m \to \infty} \frac{\sum_{j=1}^{m} A_j}{m} = \lim_{m \to \infty} \frac{\sum_{j=1}^{m} K_j}{m} = \lambda \frac{\sum_{j=1}^{m} K_j}{m} \text{ w. p. 1.} \geq \lambda \max \left\{ 1, \frac{1}{N(1-\rho)} \right\}
\]

(2.5.28)

Thus,
\[
\lim_{n \to \infty} \frac{F}{F^*} \leq \frac{\lambda (B_2 + B_2^2)}{2\lambda \max \left\{ 1, \frac{1}{N(1-\rho)} \right\}} \text{ w.p.1.}
\]

(2.5.29)

Thus, based on the Eqs. 2.5.26 and 2.5.29, we can achieve that the efficient ratio of any work-conserving scheduling policy is not greater than
\[
\frac{B_2 + B_2^2}{\max \left\{ 2, \frac{1-p_0}{\lambda}, \frac{g}{\lambda} \right\} \max \left\{ 1, \frac{1}{N(1-\rho)} \right\}}
\]

with probability 1, when n goes to infinity.

2.5.2 Non-preemptive Scenario

The definitions of \(B_1, B_2, p_0, F, \tilde{F}, F^*, F_j, \tilde{F}_j, K_j, \tilde{K}_j, A_j, \tilde{A}_j, W_{j,t}, \) and \(R_{j,t} \) in this section are same as Section 2.5.1.

**Lemma 2.5.6.** *Lemma 2.5.1, Corollary 2.5.2, Corollary 2.5.3 and Remark 2.5.13 in Section 2.5.1 are also valid in non-preemptive Reduce scenario.*

**Proof.** Different from migratable Reduce scenario in Section 2.5.1, the constitution of \(R_{j,t} \) are different, because it contains both two parts. First, it contains the
unavailable Reduce functions which are just released by the Map functions which are finished in the current time slot, the last time slot of each interval. Second, it also contains the workload of Reduce functions which cannot be processed in this time slot, even if there are idle machines, because the Reduce functions cannot migrate among machines.

However, we can also achieve that the remaining workload of Reduce functions from previous interval satisfies $R_{j,0} \leq (N - 1)R_{\text{max}}$, because the total number of remaining unfinished Reduce functions and new available Reduce functions in this time slot is less than $N$. All the following steps are same. ■

**Theorem 2.5.7.** If the Reduce functions are not allowed to migrate among different machines, then for any work-conserving scheduling policy, the efficient ratio is not greater than

$$ \frac{B_2 + B_1^2 + B_1(R_{\text{max}} - 1)}{\max \{2, \frac{1-p_0}{\lambda}, \sigma \} \max \{1, \frac{1}{N(1-\rho)}\}} $$

with probability 1, when $n$ goes to infinity. $B_1$ and $B_2$ can be achieved by Corollary 2.5.2, and $p_0$ is the probability that a time slot is no-arrival time slot.

**Proof.** In the non-preemptive Reduce scenario, in each interval, all the Map functions of job arrivals in this interval are also finished in this interval, and all their Reduce functions will be started before the end of the next interval. In other words, for all the arrivals in the $j^{th}$ interval, the Map functions are finished in $K_j$ time slots, and the Reduce functions are finished in $K_j + K_{j+1} + R_{\text{max}} - 1$ time slots, as shown in Fig. 2.3. (The vertical axis represents the total number of scheduled machines in each time slot.) And this is also true for the artificial system with dummy Reduce workload, i.e., all the arrivals in the $j^{th}$ interval of the artificial system can be finished in $\tilde{K}_j + \tilde{K}_{j+1} + R_{\text{max}} - 1$.

We also assume that there are $A_j$ arrivals in the $j^{th}$ interval. Since the algorithm is work-conserving, all the available Map functions which arrive before the interval are already finished before this interval. And all the Reduce function which are generated
by the arrivals in this interval can be finished before the end of the time slot next to this interval plus $R_{\text{max}} - 1$. So the total flow time $F_i$ of work-conserving algorithm in the $j^{\text{th}}$ interval is less than $A_j (K_j + K_{j+1} + R_{\text{max}} - 1)$. Similarly, the total flow time $\bar{F}_i$ in the $j^{\text{th}}$ interval of the artificial system is less than $\bar{A}_j (\bar{K}_j + \bar{K}_{j+1} + R_{\text{max}} - 1)$, where $\bar{A}_j$ is the number of arrivals in the $j^{\text{th}}$ interval of the artificial system.

Compared to Section 2.5.1, the flow time of each arrival has one more term $R_{\text{max}} - 1$ because of non-preemptive of Reduce functions in this section. Thus, we can use the same proving method in Theorem 2.5.5 to achieve that

$$\lim_{m \to \infty} \frac{F}{F^*} \leq \frac{\sum_{j=1}^{m} \bar{A}_j (\bar{K}_j + \bar{K}_{j+1}) + \sum_{i=1}^{n} (R_i - 1)}{\sum_{j=1}^{m} (K_j - S_j)}$$

$$= \lim_{m \to \infty} \frac{1}{m} \sum_{j=1}^{m} (K_j - S_j)$$

$$\leq \frac{\lambda [B_2 + B_1^2 + B_1 (\bar{R} - 1)]}{\max \{\rho, 1 - p_0\} \max \left\{1, \frac{1}{N(1-\rho)}\right\}} \text{ w.p.1, (2.5.30)}$$

and

$$\lim_{n \to \infty} \frac{F}{F^*} \leq \frac{\lambda [B_2 + B_1^2 + B_1 (\bar{R} - 1)]}{2\lambda \max \left\{1, \frac{1}{N(1-\rho)}\right\}} \text{ w.p.1. (2.5.31)}$$

Thus, based on the Eqs. 2.5.30 and 2.5.31, we can achieve that any work-conserving scheduler has the efficient ratio

$$\frac{B_2 + B_1^2 + B_1 (\bar{R} - 1)}{\max \left\{2, \frac{1 - p_0}{\lambda}, \frac{2}{\lambda}\right\} \max \left\{1, \frac{1}{N(1-\rho)}\right\}}.$$

\[\square\]

**Theorem 2.5.8.** In the non-preemptive scenario, any work-conserving scheduler has
a constant efficiency ratio \( a \frac{B_2 + B_1^2 + B_1(\overline{R} - 1)}{\max\{2, \frac{1-p_0}{\lambda}, \frac{A}{N(1-\rho)}\} \max\{1, \frac{1}{N(1-\rho)}\}} \), where \( p_0 \) is the probability that no job arrives in a time slot, and \( B_1 \) and \( B_2 \) are given in Eqs. (2.5.11) and (2.5.12).

**Proof.** In this scenario, for all the arrivals in the \( j^{th} \) interval, the Map functions are finished in \( K_j \) time slots, and the Reduce functions are finished in \( K_j + K_{j+1} + R - 1 \) time slots, as shown in Fig. 2.4. Since the number of arrivals in each interval is independent of the workload of Reduce functions, different from Eqs. 2.5.30 and 2.5.31, we can get

\[
\lim_{n \to \infty} \frac{F}{F^*} \leq \lim_{m \to \infty} \frac{\sum_{j=1}^{m} \widetilde{A}_j (\overline{K}_j + \overline{K}_{j+1} + R - 1)}{\sum_{j=1}^{m} K_j - S_j} \leq \frac{\lambda [B_2 + B_1^2 + B_1(\overline{R} - 1)]}{\max\{\rho, 1 - p_0\} \max\{1, \frac{1}{N(1-\rho)}\}} \text{ w.p.} 1, \tag{2.5.32}
\]

and

\[
\lim_{n \to \infty} \frac{F}{F^*} \leq \frac{\lambda [B_2 + B_1^2 + B_1(\overline{R} - 1)]}{2\lambda \max\{1, \frac{1}{N(1-\rho)}\}} \text{ w.p.} 1. \tag{2.5.33}
\]

Thus, we can achieve that the efficient ratio of any work-conserving scheduling policy is not greater than \( a \frac{B_2 + B_1^2 + B_1(\overline{R} - 1)}{\max\{2, \frac{1-p_0}{\lambda}, \frac{A}{N(1-\rho)}\} \max\{1, \frac{1}{N(1-\rho)}\}} \) with probability 1, when \( n \) goes to infinity. Similarly, if the workload of Reduce functions is light-tailed distributed, then the efficient ratio also holds if we use \( B_1' \) and \( B_2' \) instead of \( B_1 \) and \( B_2 \).

\[\blacksquare\]

**Remark 2.5.9.** In Theorems 2.5.5 and 2.5.8, we can relax the assumption of boundedness for each Reduce job and allow them to follow a light-tailed distribution, i.e., a
distribution on $R_i$ such that $\exists r_0$, such that $P(R_i \geq r) \leq \alpha \exp(-\beta r)$, $\forall r \geq r_0$, $\forall i$, where $\alpha, \beta > 0$ are two constants. We obtain similar results to Theorem 2.5.5 and 2.5.8 with different expressions.

**Lemma 2.5.10.** If the scheduling algorithm is work-conserving, then for any given number $H$, there exists a constant $B_H$, such that $E[K_j^H] < B_H$, $\forall j$.

**Proof.** Similarly to Eqs. 2.5.1 and 2.5.2, we can achieve

$$P(K_j = k) \leq P \left( \sum_{t=1}^{k-1} W_{j,t} + R_{j,0} \geq (k - 1)N \right). \quad (2.5.34)$$
Let $R'$ be the maximal remaining workload of jobs in the previous interval, then

$$P(K_j = k) \leq P \left( \sum_{t=1}^{k-1} W_{j,t} + (N - 1)R' \geq (k - 1)N \right)$$

$$= \sum_{r=0}^{\infty} \left[ P \left( \sum_{t=1}^{k-1} W_{j,t} + (N - 1)R' \geq (k - 1)N \middle| R' = r \right) \right] P(R' = r)$$

$$= \sum_{r=0}^{\infty} \left[ P \left( \sum_{t=1}^{k-1} W_{j,t} \geq N - \frac{(N - 1)r}{k - 1} \right) P(R' = r) \right].$$

(2.5.35)

Thus, we can achieve

$$E[K_j^H] = \sum_{k=0}^{\infty} k^H P(K_j = k)$$

$$= \sum_{k=0}^{\infty} \left\{ \sum_{r=0}^{\infty} \left[ P \left( \sum_{t=1}^{k-1} W_{j,t} \geq N - \frac{(N - 1)r}{k - 1} \right) P(R' = r) \right] \right\}$$

$$= \sum_{r=0}^{\infty} \left\{ P(R' = r) \sum_{k=0}^{\infty} \left[ k^H P \left( \sum_{t=1}^{k-1} W_{j,t} \geq N - \frac{(N - 1)r}{k - 1} \right) \right] \right\}.$$

The next steps are similar to the proof of Lemma 2.5.1, we skip the details and directly show the following result. For any $0 < \epsilon < N - \lambda(M + R)$, we know that $l(N - \epsilon) > 0$. Thus, we can achieve
\[ E[K^H_j] \leq \sum_{r=0}^{\infty} \left\{ P(R' = r) \left[ (1 + \left\lfloor \frac{(N-1)r}{\epsilon} \right\rfloor)^{H} + \sum_{k=2}^{\infty} k^H e^{-kl(N-\epsilon)} \right] \right\} \]

\[ = \sum_{k=2}^{\infty} k^H e^{-kl(N-\epsilon)} + \sum_{r=0}^{\infty} \left\{ P(R' = r) \left[ (1 + \left\lfloor \frac{(N-1)r}{\epsilon} \right\rfloor)^{H} \right] \right\} \]

Since we know that \( l(N-\epsilon) > 0 \) and \( \beta > 0 \) in this scenario, \( \sum_{k=2}^{\infty} k^H e^{-kl(N-\epsilon)} \),
\[ \sum_{r=0}^{\infty} \left\{ P(R' = r) \left[ (1 + \left\lfloor \frac{(N-1)r}{\epsilon} \right\rfloor)^{H} \right] \right\} \) and \( \sum_{r=0}^{r_0-1} \left\{ (1 + \left\lfloor \frac{(N-1)r}{\epsilon} \right\rfloor)^{H} \right\} \) are bounded for given \( H \).

Thus, given \( H \), \( E[K^H_j] \) is bounded by a constant \( B_H \), for any \( j \), where \( B_H \) is given as below.

\[ B_H \triangleq \min_{\epsilon \in (0, N-\lambda(M+R))} \left\{ \sum_{k=2}^{\infty} k^H e^{-kl(N-\epsilon)} + \sum_{r=0}^{r_0-1} \left\{ (1 + \left\lfloor \frac{(N-1)r}{\epsilon} \right\rfloor)^{H} \right\} \right\} \]

\[ \sum_{r=r_0}^{\infty} \alpha e^{-\beta r} \left(1 + \left\lfloor \frac{(N-1)r}{\epsilon} \right\rfloor\right)^{H} \right\} \right\} \right\} \right\}. \quad (2.5.36) \]

**Corollary 2.5.11.** \( E[K^H_j] \) is bounded by a constant \( B'_1 \), \( E[K^2_j] \) is bounded by a constant \( B'_2 \), for any \( j \). The expressions of \( B'_1 \) and \( B'_2 \) are shown as below, where the rate function \( l(a) \) is defined in Eq. (2.5.13).
Proof. By substituting $H = 2$ and $H = 1$ in Eq. (2.5.36), we directly achieve Eqs. (2.5.39) and (2.5.40).

\[
B_1' = \min_{\epsilon \in (0,N-\lambda(\mathcal{M}+\mathcal{R}))} \left\{ 2r_0 + \frac{(N-1)r_0(r_0-1)}{2\epsilon} + \frac{2\alpha}{e^\beta - 1} + \frac{\alpha(N-1)}{4\epsilon} \right. \\
+ \left. \frac{2e^{l(N-\epsilon)} - 1}{e^{l(N-\epsilon)}(e^{l(N-\epsilon)} - 1)^2} \right\}, \quad (2.5.37)
\]

\[
B_2' = \min_{\epsilon \in (0,N-\lambda(\mathcal{M}+\mathcal{R}))} \left\{ 4r_0 + \frac{2(N-1)r_0(r_0-1)}{\epsilon} + \frac{(N-1)^2(r_0-1)r_0(2r_0-1)}{6\epsilon^2} + \frac{\alpha(N-1)^2}{8\epsilon^2} \sinh(\beta) \cosh^4\left(\frac{\beta}{2}\right) \right. \\
+ \left. \frac{4\alpha}{e^\beta - 1} + \frac{\alpha(N-1)}{\epsilon} \cosh^2\left(\frac{\beta}{2}\right) \right. \\
+ \left. \frac{4e^{2l(N-\epsilon)} - 3e^{l(N-\epsilon)} + 1}{e^{l(N-\epsilon)}(e^{l(N-\epsilon)} - 1)^3} \right\}. \quad (2.5.38)
\]

To get a simple expression without summation format, we relax the right hand side of Eq. 2.5.39 as below:
\[
B_1 < \min_{\epsilon \in (0, N-\lambda(M+R))} \left\{ \sum_{r=0}^{r_0-1} \left( 2 + \frac{(N-1)r}{\epsilon} \right) + \sum_{r=0}^{\infty} \alpha e^{-\beta r} \left( 2 + \frac{(N-1)r}{\epsilon} \right) + \frac{2e^{(N-\epsilon)} - 1}{e^{(N-\epsilon)}(e^{(N-\epsilon)} - 1)^2} \right\} 
\]

\[
= \min_{\epsilon \in (0, N-\lambda(M+R))} \left\{ 2r_0 + \frac{(N-1)r_0(r_0-1)}{\epsilon} + \frac{2\alpha}{e^{\beta} - 1} + \frac{\alpha(N-1)}{4\epsilon} \cosh^2 \left( \frac{\beta}{2} \right) + \frac{2e^{(N-\epsilon)} - 1}{e^{(N-\epsilon)}(e^{(N-\epsilon)} - 1)^2} \right\} \triangleq B'_1. \tag{2.5.41}
\]

Similarly,

\[
B_2 < \min_{\epsilon \in (0, N-\lambda(M+R))} \left\{ 4r_0 + \frac{2(N-1)r_0(r_0-1)}{\epsilon} + 4\alpha + \frac{(N-1)^2}{6\epsilon^2} \sinh (\beta) \cosh^4 \left( \frac{\beta}{2} \right) + \frac{4\alpha}{e^{\beta} - 1} + \frac{\alpha(N-1)}{\epsilon} \cosh^2 \left( \frac{\beta}{2} \right) + \frac{4e^{2(N-\epsilon)} - 3e^{(N-\epsilon)} + 1}{e^{(N-\epsilon)}(e^{(N-\epsilon)} - 1)^3} \right\} \triangleq B'_2. \tag{2.5.42}
\]

\[\blacksquare\]

**Remark 2.5.12.** Following the same proof steps, we can get the Corollary 2.5.3, Remark 2.5.4, Theorem 2.5.5 and Remark 2.5.13 as Section 2.5.1. The difference is that we need use \(B'_1\) and \(B'_2\) instead of \(B_1\) and \(B_2\) in this part.

**Remark 2.5.13.** Although any work-conserving scheduler has a constant efficiency ratio, the constant efficiency ratio may be large (because the result is true for “any” work-conserving scheduler). We further discuss algorithms to tighten the constant efficiency ratio in Section 2.6.
2.6 Available-Shortest-Remaining-Processing-Time (ASRP-T) Algorithm and Analysis

In the previous sections we have shown that any arbitrary work-conserving algorithm has a constant efficiency ratio, but the constant can be large as it is related to the size of the jobs. In this section, we design an algorithm with much tighter bounds that does not depend on the size of jobs. Although, the tight bound is provided in the case of preemptive jobs, we also show via numerical results that our algorithm works well in the non-preemptive case.

Before presenting our solution, we first describe a known algorithm called SRPT (Shortest Remaining Processing Time) [13]. SRPT assumes that Map and Reduce tasks from the same job can be scheduled simultaneously in the same slot. In each slot, SRPT picks up the job with the minimum total remaining workload, i.e., including Map and Reduce tasks, to schedule. Observe that the SRPT scheduler may come up with an infeasible solution as it ignores the dependency between Map and Reduce tasks.

Lemma 2.6.1. Without considering the requirement that Reduce tasks can be processed only if the corresponding Map tasks are finished, the total flow-time $F_S$ of Shortest-Remaining-Processing-Time (SRPT) algorithm is a lower bound on the total flow-time of MapReduce framework.

Proof. Without the requirement that Reduce tasks can be processed only if the corresponding Map tasks are finished, the optimization problem in the preemptive scenario
will be as follows:

$$\min_{m_{i,t}, r_{i,t}} \sum_{i=1}^{n} \left( f_{i}^{(r)} - a_{i} + 1 \right)$$

s.t. $\sum_{i=1}^{n} (m_{i,t} + r_{i,t}) \leq N$, $r_{i,t} \geq 0$, $m_{i,t} \geq 0$, $\forall t$, \hspace{1cm} (2.6.1)

$$\sum_{t=a_{i}}^{f_{i}^{(r)}} (m_{i,t} + r_{i,t}) = M_{i} + R_{i}, \forall i \in \{1, ..., n\}.$$ 

The readers can easily check that

$$\bigcup_{i=1}^{n} \left\{ \sum_{t=a_{i}}^{f_{i}^{(r)}} m_{i,t} + r_{i,t} = M_{i} + R_{i} \right\} \supset$$

$$\bigcup_{i=1}^{n} \left\{ \sum_{t=a_{i}}^{f_{i}^{(m)}} m_{i,t} = M_{i} \right\} \cap \left\{ \sum_{t=f_{i}^{(m)}+1}^{f_{i}^{(r)}} r_{i,t} = R_{i} \right\}. \hspace{1cm} (2.6.2)$$

Thus, the constraints in Eq. (2.6.1) are weaker than constraints in Eq. (2.3.1).

Hence, the optimal solution of Eq. (2.6.1) is less than the optimal solution of Eq. (2.3.1). Since we know that the SRPT algorithm can achieve the optimal solution of Eq. (2.6.1) [13], then its total flow-time $F_{S}$ is a lower bound of any scheduling method in the MapReduce framework. Since the optimization problem in the non-preemptive scenario has more constraints, the lower bound also holds for the non-preemptive scenario. □

### 2.6.1 ASRPT Algorithm

Based on the SRPT scheduler, we present our greedy scheme called ASRPT. We base our design on a very simple intuition that by ensuring that the Map tasks of ASRPT finish as early as SRPT, and assigning Reduce tasks with smallest amount of remaining workload, we can hope to reduce the overall flow-time. However, care must be taken to ensure that if Map tasks are scheduled in this slot, then the Reduce tasks are scheduled after this slot.
ASRPT works as follows. ASRPT uses the schedule computed by SRPT to determine its schedule. In other words, it runs SRPT in a virtual fashion and keeps track of how SRPT would have scheduled jobs in any given slot. In the beginning of each slot, the list of unfinished jobs \( J \), and the scheduling list \( S \) of the SRPT scheduler are updated. The scheduled jobs in the previous time slot are updated and the new arriving jobs are added to the list \( J \) in non-decreasing order of available workload in this slot. In the list \( J \), we also keep the number of schedulable tasks in this time-slot. For a job that has unscheduled Map tasks, its available workload in this slot is the units of unfinished workload of both Map and Reduce tasks, while its schedulable tasks are the unfinished Map tasks. Otherwise, its available workload is the unfinished Reduce workload, while its schedulable tasks are the unfinished workload of the Reduce tasks (preemptive scenario) or the number of unfinished Reduce tasks (non-preemptive scenario), respectively. Then, the algorithm assigns machines to the tasks in the following priority order (from high to low): the previously scheduled Reduce tasks which are not finished yet (only in the non-preemptive scenario), the Map tasks which are scheduled in the list \( S \), the available Reduce tasks, and the available Map tasks. For each priority group, the algorithm greedily assign machines to the corresponding tasks through the sorted available workload list \( J \). The pseudo-code of the algorithm is shown in Algorithm 1.

2.6.2 Efficiency Ratio Analysis of ASRPT Algorithm

We first prove a lower bound on its performance. The lower bound is based on the SRPT. For the SRPT algorithm, we assume that in time slot \( t \), the number of machines which are scheduled to Map and Reduce tasks are \( M_t^S \) and \( R_t^S \), respectively.

Now we construct a scheduling method called Delayed-Shortest-Remaining-Processing-Time (DSRPT) for MapReduce framework based on the SRPT algorithm. We keep
Algorithm 1: Available-Shortest-Remaining-Processing-Time (ASRPT) Algorithm for MapReduce Framework

**Require:** List of unfinished jobs (including new arrivals in this slot) $J$

**Ensure:** Scheduled machines for Map tasks, scheduled machines for Reduce tasks, updated remaining jobs $J$

1: Update the scheduling list $S$ of SRPT algorithm without task dependency. For job $i$, $S(i).MapLoad$ is the corresponding scheduling of the Map tasks in SRPT.

2: Update the list of jobs with new arrivals and machine assignments in the previous slot to maintain a non-decreasing order of the available total workload $J(i).AvailableWorkload$. Also, keep the number of schedulable tasks $J(i).SchedulableTasksNum$ for each job in the list.

3: $d \leftarrow N$;

4: if The scheduler is a non-preemptive scheduler then

5: Keep the assignment of machines which are already scheduled to the Reduce tasks previously and not finished yet;

6: Update $d$, the idle number of machines;

7: end if

8: for $i = 1 \rightarrow |J|$ do

9: if $i$ has Map tasks which is scheduled in $S$ then

10: if $J(i).SchedulableTasksNum \geq S(i).MapLoad$ then

11: if $S(i).MapLoad \leq d$ then

12: Assign $S(i).MapLoad$ machines to the Map tasks of job $i$;

13: else

14: Assign $d$ machines to the Map tasks of job $i$;

15: end if

end if
else
    if \( J(i) \).SchedulableTasksNum \leq d \) then
        Assign \( J(i) \).SchedulableTasksNum machines to the Map tasks of job \( i \);
    else
        Assign \( d \) machines to the Map tasks of job \( i \);
    end if
end if

Update status of job \( i \) in the list of jobs \( J \);
Update \( d \), the idle number of machines;
end if
end for
for \( i = 1 \rightarrow |J| \) do
    if \( i^{th} \) job still has unfinished Reduce tasks then
        if \( J(i) \).SchedulableTasksNum \leq d \) then
            Assign \( J(i) \).SchedulableTasksNum machines to the Reduce tasks of job \( i \);
        else
            Assign \( d \) machines to the Reduce tasks of job \( i \);
        end if
        Update \( d \), the idle number of machines;
    end if
end for
if \( d = 0 \) then
    RETURN;
end if
for $i = 1 \rightarrow |J|$ do

if $i^{th}$ job still has unfinished Map tasks then

if $J(i).SchedulableTasksNum \leq d$ then

Assign $J(i).SchedulableTasksNum$ machines to the Map tasks of job $i$;

else

Assign $d$ machines to the Map tasks of job $i$;

end if

end if

Update $d$, the idle number of machines;

end if

if $d = 0$ then

RETURN;

end if

end for

Figure 2.5: The construction of schedule for ASRPT and DSRPT based on the schedule for SRPT. Observe that the Map tasks are scheduled in the same way. But, roughly speaking, the Reduce tasks are delayed by one slot.
the scheduling method for the Map tasks exactly same as in SRPT. For the Reduce
tasks, they are scheduled in the same order from the next time slot compared to S-
RPT scheduling. An example showing the operation of SRPT, DSRPT and ASRPT
is shown in Fig. 2.5.

**Theorem 2.6.2.** In the preemptive scenario, DSRPT has an efficiency ratio 2.

**Proof.** We construct a queueing system to represent the DSRPT algorithm. In each
time slot \( t \geq 2 \), there is an arrival with workload \( R_{t-1}^S \), which is the total workload
of the delayed Reduce tasks in time slot \( t - 1 \). The service capacity of the Reduce
tasks is \( N - M_t^S \), and the service policy of the Reduce tasks is First-Come-First-
Served (FCFS). The Reduce tasks which are scheduled in previous time slots by the
SRPT algorithm are delayed in the DSRPT algorithm up to the time of finishing
the remaining workload of delayed tasks. Also, the remaining workload \( W_t \) in the
DSRPT algorithm will be processed first, because the scheduling policy is FCFS. Let
\( D_t \) be the largest delay time of the Reduce tasks which are scheduled in the time slot
\( t - 1 \) in SRPT.

In the construction of the DSRPT algorithm, the remaining workload \( W_t \) is from
the delayed tasks which are scheduled in previous time slots by the SRPT algorithm,
and \( M_{s \geq t}^S \) is the workload of Map tasks which are scheduled in the future time slots
by the SRPT algorithm. Hence, they are independent. We assume that \( D_t' \) is the
first time such that \( W_t \leq N - M_s \), where \( s \geq t \). Then \( D_t \leq D_t' \).

If a time slot \( t_0 - 1 \) has no workload (except the \( R_{t_0-1} \) which is delayed to the
next time slot) left to the future, we call \( t_0 \) the first time slot of the interval it belongs
to. (Note that the definition of interval is different from Section 2.5.) Assume that
\( P(W_{t_0} \leq N - M_s) = P(R_{t_0-1} \leq N - M_s) = p \), where \( s \geq t_0 \). Since \( R_s \leq N - M_s \),
then \( p \geq P(R_{t_0-1} \leq R_s) \geq 1/2 \). Thus, we can get that
\[ E[D_{t_0}] \leq E[D'_t] = \sum_{k=1}^{\infty} kp(1-p)^{k-1} = \frac{1}{p} \leq 2. \tag{2.6.3} \]

Note that \( N - M_s \geq R_s \) for all \( s \), then the current remaining workload \( W_t \leq W_{t_0} \).

Then, \( E[D_t] \leq E[D_{t_0}] \leq 2. \)

Then, for all the Reduce tasks, the expectation of delay compared to the SRPT algorithm is not greater than 2. Let \( D \) has the same distribution with \( D_t \). Thus, the total flow-time \( F_D \) of DSRPT algorithm satisfies

\[ \lim_{T \to \infty} \frac{F_D}{F_S} \leq \frac{\lim_{n \to \infty} \frac{E[n]}{n} + E[D]}{\lim_{n \to \infty} \frac{E[n]}{n}} \text{ w.p.1} = 1 + \frac{E[D]}{\lim_{n \to \infty} \frac{E[n]}{n}} \leq 1 + E[D] \leq 3. \tag{2.6.4} \]

For the flow time \( F \) of any feasible scheduler in the MapReduce framework, we have

\[ \lim_{T \to \infty} \frac{F_D}{F} \leq \frac{\lim_{n \to \infty} \frac{E[n]}{n} + E[D]}{\lim_{n \to \infty} \frac{E[n]}{n}} \text{ w.p.1} \leq 1 + \frac{E[D]}{\lim_{n \to \infty} \frac{E[n]}{n}} \leq 1 + \frac{E[D]}{2} \leq 2. \tag{2.6.5} \]

\[ \blacksquare \]

**Corollary 2.6.3.** From the proof of Theorem 2.6.2, the total flow-time of DSRPT is not greater than 3 times the lower bound given by Lemma 2.6.1 with probability 1, when \( n \) goes to infinity.

**Corollary 2.6.4.** In the preemptive scenario, the ASRPT scheduler has an efficiency ratio 2.

**Proof.** For each time slot, all the Map tasks finished by DSRPT are also finished by ASRPT. For the Reduce tasks of each job, the jobs with smaller remaining workload will be finished earlier than the jobs with larger remaining workload. Hence, based on the optimality of SRPT, ASRPT can be viewed as an improvement of DSRPT. Thus,
the total flow-time of ASRPT will not be greater than DSRPT. So, the efficiency ratio of DSRPT also holds for ASRPT.

Note that, the total flow-time of ASRPT is not greater than 3 times of the lower bound given by Lemma 2.6.1 with probability 1, when \( n \) goes to infinity. We will show this performance in the next section. Also, the performance of ASRPT is analyzed in the preemptive scenario in Corollary 2.6.4. We will show that ASRPT performs better than other schedulers in both preemptive and non-preemptive scenarios via simulations in the next section.

2.7 Simulation Results

2.7.1 Simulation Setting

We evaluate the efficacy of our algorithm ASRPT for both preemptive and non-preemptive scenarios. We consider a data center with \( N = 100 \) machines, and choose Poisson process with arrival rate \( \lambda = 2 \) jobs per time slot as the job arrival process. We choose uniform distribution and exponential distribution as examples of bounded workload and light-tailed distributed workload, respectively. For short, we use \( \text{Exp}(\mu) \) to represent an exponential distribution with mean \( \mu \), and use \( U[a, b] \) to represent a uniform distribution on \( \{a, a+1, \ldots, b-1, b\} \). We choose the total time slots to be \( T = 800 \), and the number of tasks in each job is up to 10.

We compare 3 typical schedulers to the ASRPT scheduler:

The **FIFO** scheduler: It is the default scheduler in Hadoop. All the jobs are scheduled in their order of arrival.

The **Fair** scheduler: It is a widely used scheduler in Hadoop. The assignment of machines are scheduled to all the waiting jobs in a fair manner. However, if some jobs need fewer machines than others in each time slot, then the remaining machines
are scheduled to the other jobs, to avoid resource wastage and to keep the scheduler work-conserving.

The LRPT scheduler: Jobs with larger unfinished workload are always scheduled first. Roughly speaking, the performance of this scheduler represents in a sense how poorly even some work-conserving schedulers can perform.

2.7.2 Efficiency Ratio

In the simulations, the efficiency ratio of a scheduler is obtained by the total flow-time of the scheduler over the lower bound of the total flow-time in $T$ time slots. Thus, the real efficiency ratio should be smaller than the efficiency ratio given in the simulations. However, the proportion of all schedulers would remain the same.

First, we evaluate the exponentially distributed workload. We choose the workload distribution of Map for each job as $\text{Exp}(5)$ and the workload distribution of Reduce for each job as $\text{Exp}(40)$. The efficiency ratios of different schedulers are shown in Fig. 2.6. For different workload, we choose workload distribution of Map as $\text{Exp}(30)$ and the workload distribution of Reduce as $\text{Exp}(15)$. The efficiency ratios of schedulers are shown in Fig. 2.7.

Then, we evaluate the uniformly distributed workload. We choose the workload distribution of Map for each job as $U[1, 9]$ and the workload distribution of Reduce for each job as $U[10, 70]$. The efficiency ratios of different schedulers are shown in Fig. 2.8. To evaluate for a smaller Reduce workload, we choose workload distribution of Map as $U[10, 50]$ and the workload distribution of Reduce as $U[10, 20]$. The efficiency ratios of different schedulers are shown in Fig. 2.9.

As an example, we show the convergence of efficiency ratios in Figs. 2.10-2.13, for the 4 different scenarios.

From Figures. 2.6-2.9, we can see that the total flow-time of ASRPT is much
Figure 2.6: Efficiency Ratio (Exponential Distribution, Large Reduce)

Figure 2.7: Efficiency Ratio (Exponential Distribution, Small Reduce)
Figure 2.8: Efficiency Ratio (Uniform Distribution, Large Reduce)

Figure 2.9: Efficiency Ratio (Uniform Distribution, Small Reduce)
Figure 2.10: Convergence of Efficiency Ratios (Exponential Distribution, Large Reduce)

Figure 2.11: Convergence of Efficiency Ratios (Exponential Distribution, Small Reduce)
Figure 2.12: Convergence of Efficiency Ratios (Uniform Distribution, Large Reduce)

Figure 2.13: Convergence of Efficiency Ratios (Uniform Distribution, Small Reduce)
smaller than all the other schedulers. Also, as a “bad” work-conserving, the LRPT scheduler also has a constant (maybe relative large) efficiency ratio, from Figures 2.13

2.7.3 Cumulative Distribution Function (CDF)

For the same setting and parameters, the CDFs of flow-times are shown in Fig. 2.14-2.17. We plot the CDF only for flow-time up to 100 units. From these figures, we can see that the ASRPT scheduler has a very light tail in the CDF of flow-time, compared to the FIFO and Fair schedulers. In other words, the fairness of the ASRPT scheduler is similar to the FIFO and Fair schedulers. However, the LRPT scheduler has a long tail in the CDF of flow time. In the other words, the fairness of the LRPT scheduler is not as good as the other schedulers.

2.8 Conclusion

In this chapter, we study the problem of minimizing the total flow-time of a sequence of jobs in the MapReduce framework, where the jobs arrive over time and need to
Figure 2.15: CDF of Schedulers (Exponential Distribution, Small Reduce)

Figure 2.16: CDF of Schedulers (Uniform Distribution, Large Reduce)
be processed through Map and Reduce procedures before leaving the system. We show that no on-line algorithm can achieve a constant competitive ratio. We define a weaker metric of performance called the efficiency ratio and propose a corresponding technique to analyze on-line schedulers. Under some weak assumptions, we then show a surprising property that for the flow-time problem any work-conserving scheduler has a constant efficiency ratio in both preemptive and non-preemptive scenarios. More importantly, we are able to find an online scheduler ASRPT with a very small efficiency ratio. The simulation results show that the efficiency ratio of ASRPT is much smaller than the other schedulers, while the fairness of ASRPT is as good as other schedulers.

Figure 2.17: CDF of Schedulers (Uniform Distribution, Small Reduce)
CHAPTER 3
EXPLOITING LARGE SYSTEM DYNAMICS FOR
DESIGNING SIMPLE DATA CENTER SCHEDULERS

3.1 Introduction

The number of large-scale data centers (e.g., those with tens of thousands of machines) is rapidly increasing. MapReduce is a framework designed to process massive amounts of data in data centers [1]. Although it was first proposed by Google [1], today, many other companies including Microsoft, Yahoo, and Facebook also use this framework. This framework is being widely used for applications such as search indexing, distributed searching, web statistics generation, and data mining.

In MapReduce, each arriving job consists of a set of Map tasks and Reduce tasks. The scheduler is centralized and responsible for making decisions on which task will be executed at what time and on which machine. The key metric considered in this chapter is the total delay in the system per job, which is the time it takes a job, since its arrival into the system, to be processed fully. This includes both the delay in waiting before the first task in the job begins to be processed, and the time for processing all tasks in the job.

A critical consideration for the design of the scheduler is the dependence between the Map and Reduce tasks. For each job, the Map tasks need to be finished before
starting any of its Reduce tasks\(^1\) [1, 6]. Various scheduling solutions have been proposed within the MapReduce framework [7, 8, 6, 5, 9]. However, under some weak assumptions it was proven [3] that there is no upper bound on the competitive ratio of any causal scheduler for total delay (flow time) minimization problem. In an attempt to minimize the total delay, a new metric to analyze the performance of schedulers called *efficiency ratio* was introduced in [2, 3], where loose bounds were provided on the performance of general work-conserving schedulers, and under the special case of preemptive and parallelizable tasks, it was shown that a scheduler called ASRPT can guarantee an efficiency ratio of two. However, ASPRT requires a sorting operation whose complexity grows rapidly with the number of machines in the data center. What we show in this chapter is that the scheduler design can be significantly simplified by taking advantage of scale. Moreover, this scale also allows us to design simple schedulers that not only provide a bounded efficiency ratio as in [2, 3], but in fact approach optimality (i.e., the efficiency ratio goes to 1) as the number of machines grows large. Thus, by exploiting the scale inherent in current data centers, we can improve both the complexity of scheduler design in MapReduce-style multiple-phases system and the performance guarantees that we can provide.

We briefly summarize the main contributions of our work:

- Under certain weak assumptions, we show that for the flow-time minimization problem any work-conserving scheduler is asymptotically optimal in two classes of scenarios. (i) Preemptive and Parallelizeable: in which the Reduce tasks can be preempted and served in parallel over different machines (Section 3.3). (ii) Non-Preemptive and Non-Parallelizeable: when each Reduce task has to be served only on one machine and cannot be preempted (Section 3.4). These

\(^1\)Here, we consider the most popular case in reality without the Shuffle phase. For discussion about the Shuffle phase, see [5].
scenarios can be thought to capture two extreme cases of data locality, the first when the cost of data migration is negligible, and the second when the cost of data migration is prohibitive.

- For long running applications, we study the relationship between the number of machines $N$ and total running time $T$. We provide sufficient conditions to guarantee the asymptotic optimality of work-conserving schedulers, when $T$ is function of $N$ (Section 3.5).

- We verify via simulations (Section 3.6) that while state-of-the-art work-conserving schedulers have different delay performance for a small number of machines $N$, these differences rapidly vanish as $N$ becomes large, and the performance is virtually indistinguishable and close to optimal for a few hundred machines.

Our results provide the following two surprising properties: First, in a large system, it is not necessary to implement complex schedulers, as long as they honor the work-conserving principle, thus ensuring both high performance and scalability. Second, under appropriate and general assumptions, work-conserving schedulers can guarantee asymptotic optimality under both the Noah Effect [4] (a large amount of workload arrives into the system in the preemptive and parallelizable scenario) and Joseph Effect [4] (a large number of cumulative running jobs remain in the system in the non-preemptive and non-parallelizable scenario).

3.2 System Model and Asymptotic Optimality

3.2.1 System Model under MapReduce Framework

As introduce in Chapter 2, MapReduce has two elemental processes: Map and Reduce. For the Map tasks, the inputs are divided into several small sets, and processed
by different machines in parallel. The Reduce tasks then operate on the intermediate 
data, possibly running the operation on multiple machines in parallel to generate the 
final result. Note that for a given job, the Reduce tasks have to start executing after 
all (or a subset) of Map tasks for that job have been completed.

Consider a data center with $N$ machines and $n$ jobs that arrive during a time 
period $T$. We assume that each machine can only process one job at a time. A 
machine could represent a processor, a core in a multi-core processor or a virtual 
machine. We assume the scheduler periodically collects the information on the state 
of jobs running on the machines, which is used to make scheduling decisions.

We assume that each job $i$ brings $M_i$ units of workload for its Map tasks and 
$R_i$ units of workload for its Reduce tasks. Here, the $R_i$ workload belongs to Reduce 
operations. Each Map task has 1 unit of workload\textsuperscript{2}, however, each Reduce task can 
have multiple units of workload. Time is slotted and each machine can run one unit 
of workload in each time slot.

We assume that the number of job arrivals in each time slot is i.i.d., and the 
arrival rate is $\lambda$. Assume that $\{M_i\}$ are i.i.d. with expectation $\overline{M}$, and $\{R_i\}$ are i.i.d. 
with expectation $\overline{R}$. In time slot $t$ for job $i$, $m_{i,t}$ and $r_{i,t}$ machines are scheduled for 
Map and Reduce tasks, respectively. As we know, each job contains several tasks.

We assume that job $i$ contains $K$ tasks, and the workload of the Reduce task $k$ 
of job $i$ is $R^{(k)}_i$. Thus, for any job $i$, $\sum_{k=1}^{K} R^{(k)}_i = R_i$. In time slot $t$ for job $i$, $r_{i,t}^{(k)}$ 
machines are scheduled for the Reduce task $k$. As each Reduce task may consist of 
multiple units of workload, it can be processed in either preemptive and parallelizable 
or non-preemptive and non-parallelizable fashion based on the type of scheduler \cite{3}.

**Definition 3.** A scheduler is called **preemptive and parallelizable** if Reduce tasks

\textsuperscript{2}Because the Map tasks are independent and have small workload \cite{8}, such assumption is valid.
belonging to the same job can run in parallel on multiple machines, can be interrupted by other tasks, and can be rescheduled to different machines in different time slots.

A scheduler is called non-preemptive and non-parallelizable if each Reduce task can only be scheduled on one machine and, once started, it must keep running without any interruption.

The schedulers are classified into these two different scenarios based on implementation consideration. In data centers, there is additional cost of data migration and initialization for each job. Such a cost could be caused by limited I/O speed, limited network transmission rate, and large amount of initialization work needed before a job. When the cost of data migration and initialization is not significant, then the schedulers can interrupt and migrate data from one machine to another to avoid unnecessary delay, which corresponds to the preemptive and parallelizable scenario. Alternatively, when such a cost is large, then the schedulers should avoid any action of interruption and migration, which corresponds to the non-preemptive and non-parallelizable scenario. Which scenario is more appropriate depends on the application, network topology and transmission rate, and the I/O speed.

Let the arrival time of job $i$ be $a_i$, the time slot in which the last Map task finishes execution be $f_i^{(m)}$, and the time slot in which all Reduce tasks are completed be $f_i^{(r)}$. For the preemptive and parallelizable scenario, we restate the problem definition as follows:

$$\min_{m_{i,t}, r_{i,t}} \sum_{i=1}^{n} \left( f_i^{(r)} - a_i + 1 \right)$$

s.t.

$$\sum_{i=1}^{n} (m_{i,t} + r_{i,t}) \leq N, \quad r_{i,t} \geq 0, \quad m_{i,t} \geq 0, \quad \forall t,$$

$$\sum_{t=a_i}^{f_i^{(m)}} m_{i,t} = M_i, \quad \sum_{t=f_i^{(m)}+1}^{f_i^{(r)}} r_{i,t} = R_i, \quad \forall i \in \{1, ..., n\}. \quad (3.2.1)$$
In the non-preemptive and non-parallelizable scenario, the optimization problem in this scenario is as in Eq. (3.2.1), with additional constraints representing the non-preemptive and non-parallelizable nature as follows:

\[
\forall k, i, t, r_{i,t}^{(k)} = 0 \text{ or } 1, \quad r_{i,t}^{(k)} = 1 \text{ if } 0 < \sum_{s=0}^{t-1} r_{i,s}^{(k)} < R_{i}^{(k)}. \tag{3.2.2}
\]

The scheduling problems (both preemptive and parallelizable and non-preemptive and non-parallelizable) are NP-hard in the strong sense [3].

### 3.2.2 Asymptotic Optimality of Schedulers

**Definition 4.** For a data center with \(N\) machines, let \(T\) represent the total time slots. Define \(F^S(N,T)\) as the total flow time of a scheduler \(S\), and \(F^*(N,T)\) as the minimum total flow time over all feasible schedulers. Then, a scheduler \(S\) is called **asymptotically optimal** if

\[
\lim_{N \to \infty} \frac{F^S(N,T)}{F^*(N,T)} = 1 \text{ w.p.1 for any given } T. \tag{3.2.3}
\]

**Definition 5.** In a data center with \(N\) machines, **traffic intensity** \(\rho_N\) is the ratio of expected arrival workload over the processing ability of \(N\) machines. When \(\rho_N\) is a constant for any value of \(N\), it is written as \(\rho\) for short.

Note that when we let \(N\) goes to infinity, we also let the corresponding arrival load go to infinity, such that \(\rho_N > 0\) (and also \(\rho_N \to 1\)), to avoid trivialities.

Also, we define work-conserving schedulers as follows.

**Definition 6.** A scheduler is called **work-conserving scheduler** if jobs are always scheduled when there are idle machines without violating the dependencies among multiple phases.

The definition contains two parts. First, jobs are scheduled without unnecessary waiting, i.e., there is no a time slot that jobs are waiting and machines are idle.
Second, the dependence between multiple phases (called **Phase Precedence** [2]) must hold. In MapReduce framework, specially, the phase dependence between Map and Reduce tasks has to always hold. That is for a job, Reduce tasks can only be processed if all (or in general a subset) of Map tasks are completed for that job. In the following sections, we will show the asymptotic optimality of the work-conserving schedulers that are the most popular and widely used in data centers.

### 3.2.3 Assumptions

In this part, we list the assumptions which will be used in the following sections. It does not mean that all the assumptions are needed in each theorem. We will point out which assumption is required when we state the theorems.

(A1) \( \{M_i, i = 1, 2, \ldots\} \) are i.i.d. with finite expectation \( 0 < \mathbb{E}[M] < \infty \), \( \{R_i, i = 1, 2, \ldots\} \) are i.i.d. with finite expectation \( 0 < \mathbb{E}[R] < \infty \).

(A2) \( \{M_i, i = 1, 2, \ldots\} \) are i.i.d. with finite expectation \( 0 < \mathbb{E}[M] < \infty \) and finite variance \( \sigma_m^2 < \infty \), \( \{R_i, i = 1, 2, \ldots\} \) are i.i.d. with finite expectation \( 0 < \mathbb{E}[R] < \infty \) and finite variance \( \sigma_r^2 < \infty \).

(A3) The workload distribution of each job does not change with \( T \) and \( N \).

(A4) For each \( N \), the number of arrival jobs \( A_t^{(N)} \) in each time slot \( t \) are i.i.d., and are independent of \( M_i \) and \( R_i \).

(A5) For all \( t \), \( A_t^{(N)} \) and \( \frac{A_t^{(N)}}{N} \) converges to infinity and a constant w.p.1, respectively, as \( N \) goes to infinity.

(A6) The traffic intensity \( \rho \) is a constant, where \( 0 < \rho < 1 \).
(A7) The traffic intensity $\rho_N \to 1$ as $N \to \infty$, and
\[
\liminf_{N \to \infty} \frac{(1 - \rho_N)\sqrt{N}}{\sqrt{\log \log N}} > \frac{\sqrt{2(\sigma_m + \sigma_r)}}{\sqrt{M + R}}. \tag{3.2.4}
\]

(A8) When the traffic intensity $\rho_N \to 1$, if $\frac{A^{(N)}_t}{N} \to \frac{1}{M + R}$ w.p.1 as $N \to \infty$, then
\[
\liminf_{N \to \infty} \frac{\left(\mathbb{E}\left[\frac{A^{(N)}_t}{N}\right] - \frac{A^{(N)}_t}{N}\right)\sqrt{N}}{\sqrt{\log \log N}} \geq 0 \text{ w.p.1}. \tag{3.2.5}
\]

(A9) In non-preemptive and non-parallelizable scenario, the number of Reduce tasks in each job are i.i.d., and the workload of each Reduce task are also i.i.d.. Also, the largest number of Reduce tasks in each job are bounded.

(A10) The traffic intensity $\rho_N \to 1$ as $N \to \infty$, and
\[
\liminf_{N \to \infty} \frac{(1 - \rho_N)\sqrt{N}}{\sqrt{\log \log N}} = \infty. \tag{3.2.6}
\]

Remark 3.2.1. Note that the assumptions of the arrival jobs are weak. In Sections 3.3 and 3.4, we will see that, when $\rho < 1$, we allow for heavy tailed distributions and only requires finite mean (3.2.3). Similarly, for appropriate heavy traffic region of $\rho_N$ goes to 1, we allow for heavy tailed distributions and only requires finite mean and variance (3.2.3).

Lemma 3.2.2. Assumption 3.2.3 implies that $\lim_{N \to \infty} \frac{A^{(N)}_t}{N} = c$ w.p.1, where $c$ is a constant. By the definition of traffic intensity $\rho$, assumptions 3.2.3 and 3.2.3 imply that
\[
\rho = \frac{1}{N} \mathbb{E}\left[\sum_{j=1}^{A^{(N)}_t} \frac{M}{M + R}\right] = \frac{1}{N} \mathbb{E}\left[A^{(N)}_t\right] (M + R). \tag{3.2.7}
\]
and
\[
c \leq \lim_{N \to \infty} \mathbb{E}\left[A^{(N)}_t\right] / N = \frac{\rho}{M + R}. \tag{3.2.8}
\]
Similarly, assumptions 3.2.3 and 3.2.3 or 3.2.3 imply that

$$c \leq \lim_{N \to \infty} E \left[ A_t^{(N)} \right] / N = \frac{1}{M + R}. \quad (3.2.9)$$

**Remark 3.2.3.** Roughly speaking, 3.2.3 implies that

$$1 - \rho_N > \frac{\sqrt{2(\sigma_m + \sigma_r)}}{\sqrt{(M + R)}} \frac{\log \log N}{N}, \quad (3.2.10)$$

when $N$ is large enough.

Similarly, 3.2.3 implies that $1 - \rho_N \gg \frac{\log \log N}{\sqrt{N}}$, when $N$ is large enough.

In other words, assumptions 3.2.3 and 3.2.3 define the appropriate heavy traffic regions, which show the convergence rate of $\rho_N$ to 1 corresponding to $N$.

### 3.3 Asymptotic Optimality of Work-conserving Schedulers in Preemptive and Paralleled Scenario

In this section, we will show that all the work-conserving schedulers are asymptotically optimal. For simplicity, we let $M_j^{(t)}$ and $R_j^{(t)}$ be the Map and Reduce workload of $j^{th}$ arrival job in time slot $t$, respectively, in the following sections.

**Theorem 3.3.1.** Under assumptions 3.2.3 and 3.2.3-3.2.3, any work-conserving scheduler is asymptotically optimal.

**Proof.** To prove Theorem 3.3.1, we first show that if the machines only serve the Map tasks of jobs arriving in the current time slot and the Reduce tasks of jobs arrived in the previous time slot, then the probability that all these task can be served immediately goes 1 as $N$ goes to infinity. In the following part, “no unnecessary delay” or “be served immediately” means that all the available phases can be served. Obviously, the phases, whose dependent phases are running, are unavailable and cannot be served for any schedulers.
For the first time slot, there is no Reduce tasks corresponding to previous time slot. When time slot \( t > 1 \), there could be both new arrival workload of Map and available workload of Reduce. Thus, if we can prove no unnecessary delay for each job when \( t > 1 \), then it is obvious that there will be no delay when \( t = 1 \).

For any other time slot \( t > 1 \), we have

\[
\lim_{N \to \infty} \frac{A^{(N)}_t}{N} = \frac{1}{N} \sum_{j=1}^{A^{(N)}_{t-1}} R_j^{(t-1)} + \sum_{j=1}^{A^{(N)}_t} M_j^{(t)} - N
\]

\[
= \lim_{N \to \infty} \frac{1}{A^{(N)}_{t-1}} \sum_{j=1}^{A^{(N)}_{t-1}} R_j^{(t-1)} - \frac{1}{N} + \lim_{N \to \infty} \frac{N}{A^{(N)}_t} \lim_{N \to \infty} \frac{1}{N} - 1
\]

\[
= (R + M) c - 1 \leq \rho - 1 \text{ w.p.1.}
\]

Since \( \rho < 1 \), then Eq. (3.3.1) is less than 0 w.p.1. Then, for any time slot \( t \), the Map tasks of new arrival jobs and the Reduce tasks of jobs which arrive in the previous time slot can be finished at time slot \( t \) w.p.1, as \( N \) goes to infinity.

Let \( F^{\text{wc}}(N,T) \) be the total flow time of any work-conserving scheduler, and let \( F^*(N,T) \) is the optimal total flow time of all feasible schedulers.

We define the set \( \Xi_t \) as follows:

\[
\Xi_t \triangleq \left\{ \left\{ \epsilon : \frac{A^{(N)}_t}{N} \geq 1 \text{ as } N \to \infty \right\} \right\}, \quad t = 1
\]

\[
\left\{ \left\{ \epsilon : \frac{1}{N} \sum_{j=1}^{A^{(N)}_{t-1}} R_j^{(t-1)} + \sum_{j=1}^{A^{(N)}_t} M_j^{(t)} \geq 1 \text{ as } N \to \infty \right\} \right\}, \quad t \geq 2
\]

Then, set \( \Xi_t \) is a null set for any \( t \) by the previous proof. \( \bigcup_{t=1}^{T} \Xi_t \), which is a finite
union of null sets, is also a null set. Thus, 

\[ P \left( \bigcap_{t=1}^{T} (\Omega \setminus \Xi_t) \right) = P \left( \Omega \setminus \bigcup_{t=1}^{T} \Xi_t \right) = 1, \]

i.e., for any \( T \), when \( N \) goes to infinity, all the Map and Reduce workload will be finished either in the time slot they arrive or the very next time slot w.p.1. In the set \( \Omega \setminus \bigcup_{t=1}^{T} \Xi_t \), any work-conserving scheduler has the same total flow time as the infinite number of machines scenario w.p.1, as \( N \) goes to infinity. If there are infinite number of machines, then the total flow time is a lower bound of the finite number of machines scenario. In other words, \( \left( \bigcap_{t=1}^{T} \Xi_t \right) \subseteq \left\{ \lim_{N \to \infty} \frac{F_{wc}(N,T)}{F^*(N,T)} = 1 \right\} \). Thus, 

\[ P \left( \lim_{N \to \infty} \frac{F_{wc}(N,T)}{F^*(N,T)} = 1 \right) = 1. \] Eq. (3.2.3) is achieved.

**■**

**Corollary 3.3.2.** In Theorem, we can also get that

\[ \lim_{T \to \infty} \lim_{N \to \infty} \frac{F_S(N,T)}{F^*(N,T)} = 1 \quad \text{w.p.1}. \] \hspace{1cm} (3.3.3)

**Proof.** Similar to the proof of Theorem 3.3.1, \( \bigcup_{t=1}^{\infty} \Xi_t \), which is a countable union of null sets, is also a null set. Hence, using the same method to prove Eq. (3.2.3), we can get Eq. (3.3.3).

**■**

In previous results, we fixed traffic intensity \( \rho \) as a constant which is less than 1. In the next theorem, we will discuss heavy traffic scenario, and show that work-conserving scheduler is still asymptotically optimal in appropriate heavy traffic regions.

**Theorem 3.3.3.** In the MapReduce framework, assume 3.2.3-3.2.3 and 3.2.3-3.2.3 hold. Then, any work-conserving scheduler is asymptotically optimal.

**Proof.** Similarly to the proof of Theorem 3.3.1, we focus on the workload of Map tasks which corresponds to jobs arrive in the current time slot, and the workload of Reduce tasks which corresponds to jobs arrived in the previous time slot. If such
workload can be immediately served by any work-conserving scheduler, then by the same proof of Theorem 3.3.1, we can directly achieve Theorem 3.3.3.

For time slot \( t > 1 \), there could be both Map arrival tasks and remaining Reduce tasks. For time slot 1, there are only Map tasks are available in the framework, i.e., there is a lighter workload in time slot 1 than other time slots. Thus, we only need to focus on time slot \( t > 1 \). If the workload can be immediately served at \( t > 1 \), then it is also true when \( t = 1 \).

For time slot \( t \geq 2 \), we have

\[
\lim_{N \to \infty} \sup_{\tau \to \infty} \frac{\sum_{j=1}^{A_{\tau}^{(N)}} R_j^{(t-1)} + \sum_{j=1}^{A_{\tau}^{(N)}} M_j^{(t)} - N}{\sqrt{N \log \log N}} \\
= \lim_{\tau_{t-1} \to \infty} \sup_{\tau_{t-1}^{(N)}} \frac{A_{\tau_{t-1}}^{(N)}}{A_{\tau_{t-1}}^{(N)} \log \log A_{\tau_{t-1}}^{(N)}} \lim_{N \to \infty} \frac{A_{\tau_{t-1}}^{(N)} \log \log A_{\tau_{t-1}}^{(N)}}{\sqrt{N \log \log N}} \\
+ \lim_{\tau_{t} \to \infty} \sup_{\tau_{t}^{(N)}} \frac{\sum_{j=1}^{A_{\tau}^{(N)}} (M_j^{(t)} - \overline{M})}{\sqrt{A_{\tau}^{(N)} \log \log A_{\tau}^{(N)}}} \lim_{N \to \infty} \frac{A_{\tau}^{(N)} \log \log A_{\tau}^{(N)}}{\sqrt{N \log \log N}} \\
- \lim_{N \to \infty} \inf_{N} \frac{N - A_{\tau}^{(N)} \overline{M} - A_{\tau_{t-1}}^{(N)} \overline{R}}{\sqrt{N \log \log N}}.
\]

By Eq. (3.2.9) in Lemma 3.2.2, we can get that

\[
\lim_{N \to \infty} \frac{1}{\sqrt{N \log \log N}} \sqrt{A_{\tau}^{(N)} \log \log A_{\tau}^{(N)}} \\
= \lim_{N \to \infty} \sqrt{\frac{A_{\tau}^{(N)}}{N}} \log \left( \frac{\log A_{\tau}^{(N)}}{\log \log N} + \log N \right) = \sqrt{c} \leq \frac{1}{\sqrt{M + R}} \text{ w.p.1.}
\]

By the Law of the Iterated Logarithm (LIL) [14], we can get that
\[
\limsup_{N \to \infty} \frac{\sum_{j=1}^{A_t(N)} (R_j^{(t-1)} - \overline{R})}{\sqrt{A_t(N) \log \log A_t(N)}} = \sqrt{2} \sigma_r \text{ w.p.1 (3.3.6)}
\]

and
\[
\limsup_{N \to \infty} \frac{\sum_{j=1}^{A_t(N)} (M_j^{(t)} - \overline{M})}{\sqrt{A_t(N) \log \log A_t(N)}} = \sqrt{2} \sigma_m \text{ w.p.1. (3.3.7)}
\]

Based on Eq. 3.2.8 in Lemma 3.2.2, there are two cases: \( c < \frac{1}{M + R} \) or \( c = \frac{1}{M + R} \).

a. Case 1: \( c < \frac{1}{M + R} \). We get
\[
\liminf_{N \to \infty} \frac{\sqrt{N}}{\sqrt{\log \log N}} \left( 1 - \frac{A_t(N)}{N} \right) (M + R) \sqrt{\log \log N} = \frac{1}{M + R} \left( M + R - c \right) \lim_{N \to \infty} \frac{\sqrt{N}}{\sqrt{\log \log N}} \text{ w.p.1. (3.3.8)}
\]

Since \( c < \frac{1}{M + R} \) and \( \lim_{N \to \infty} \frac{\sqrt{N}}{\sqrt{\log \log N}} = \infty \), we can get that
\[
\liminf_{N \to \infty} \frac{\sqrt{N}}{\sqrt{\log \log N}} (1 - \frac{A_t(N)}{N} (M + R)) \sqrt{\log \log N} = \infty \text{ w.p.1. (3.3.9)}
\]

b. Case 2: \( c = \frac{1}{M + R} \). In this case, by assumption 3.2.3, we can get that
\[
\liminf_{N \to \infty} \frac{N - A_t(N) \overline{M} - A_t(N) \overline{R}}{\sqrt{N \log \log N}} = \liminf_{N \to \infty} \frac{\sqrt{N} (1 - \rho_N)}{\sqrt{\log \log N}} + \liminf_{N \to \infty} \frac{\sqrt{N} (E [\frac{A_t(N)}{N}] - \frac{A_t(N)}{N}) (M + R)}{\sqrt{\log \log N}} \geq \liminf_{N \to \infty} \frac{\sqrt{N} (1 - \rho_N)}{\sqrt{\log \log N}} > \frac{\sqrt{2} (\sigma_m + \sigma_r)}{\sqrt{M + R}} \text{ w.p.1. (3.3.10)}
\]
For both cases, combined with Eqs. (3.3.6) and (3.3.7), we can get that

$$\limsup_{N \to \infty} \frac{\sum_{j=1}^{A_{t-1}^{(N)}} R_j^{(t-1)} + \sum_{j=1}^{A_{t}^{(N)}} M_j^{(t)} - N}{\sqrt{N \log \log N}} < 0 \text{ w.p.1.}$$

(3.3.11)

The following proof is similar to the proof of Theorem 3.3.1.

Remark 3.3.4. If we change $\rho_N$ to $\frac{A_{t}^{(N)}}{N} (M + P)$ in assumption 3.2.3 and the assumption also holds w.p.1, then the same applies for Theorem 3.3.3 without assumption 3.2.3.

Remark 3.3.5. Although the proof is given under MapReduce framework (2 phases), the results in Theorem 3.3.1 and Theorem 3.3.3 can be directly generalized to multiple-phases scenarios.

Remark 3.3.6. There is no assumption on the dependence between the workload of Map and Reduce for each job, thus the results in Theorem 3.3.1 and Theorem 3.3.3 can be applied in the scenario, where there could be arbitrary dependencies between the workload of Map and Reduce for each job.

3.4 Asymptotic Optimality of Work-conserving Schedulers in Non-preemptive and Non-parallelizable Scenario

In the previous section, we show the asymptotic optimality of work-conserving schedulers in the preemptive and parallelizable scenario. In this section, we will show the performance of work-conserving scheduler in non-preemptive and non-parallelizable scenario.

Theorem 3.4.1. In the non-preemptive and non-parallelizable scenario of MapReduce framework, under assumptions 3.2.3, 3.2.3-3.2.3, and 3.2.3, any work-conserving scheduler is asymptotically optimal.
To prove Theorem 3.4.1, we first consider the Reduce only jobs scenario and show Lemma 3.4.2.

**Lemma 3.4.2.** In the non-preemptive and non-parallelizable scenario, if the jobs are Reduce only and each job only has one task. Then, under assumptions 3.2.3 and 3.2.3-3.2.3, any work-conserving scheduler is asymptotically optimal.

**Proof.** For the queue length of the one-phase jobs $Q_t^{(N)}$ at time slot $t$ and given number of machines $N$, if the queue length satisfies that

$$\limsup_{N \to \infty} \frac{Q_t^{(N)}}{N} < 1 \text{ w.p.1.} \tag{3.4.1}$$

for any time slot $t$, then Lemma 3.4.2 can be directly proven by the same proof process of Theorem 3.3.1. To prove this result, we use mathematical induction, which includes the following two steps:

1) For $t = 1$, we prove that $\limsup_{N \to \infty} \frac{Q_1^{(N)}}{N} < 1$, w.p.1.

When $t = 1$, there is no remaining workload from the previous time slots. If each job which arrives in time slot 1 only has 1 unit of workload, then there is no difference between preemptive and parallelizable and non-preemptive and non-parallelizable. In this case, $\limsup_{N \to \infty} \frac{W_1^{(N)}}{N} < 1$ w.p.1. holds based on Theorem 3.3.1 and the fact that $W_1^{(N)}$ is equal to the queue length in this case. If not all the jobs have 1 unit of workload, then the total queue length of jobs $Q_1^{(N)}$ is less than the total workload of all the jobs. Thus, $\limsup_{N \to \infty} \frac{Q_1^{(N)}}{N} \leq \limsup_{N \to \infty} \frac{W_1^{(N)}}{N} < 1$ w.p.1.

2) Assume that for all $s = 1, 2, \ldots t - 1$, $\limsup_{N \to \infty} \frac{Q_s^{(N)}}{N} < 1$ w.p.1. Under this assumption, prove that for time slot $t$, $\limsup_{N \to \infty} \frac{Q_t^{(N)}}{N} < 1$ w.p.1 is also true.

Let’s consider time slot $t$. $Q_t^{(N)}$ contains all the remaining jobs which are not finished from all the previous $t - 1$ time slots, i.e.
\[ Q_{t}^{(N)} = \sum_{s=1}^{t} \text{Number of jobs which arrive at time slot } s \text{ and are remained at time slot } t. \] (3.4.2)

By the induction hypothesis, for all the previous \( t - 1 \) time slots before time slot \( t \), there is no delay caused because there are not enough machines to accommodate the load. Thus, the only reason that makes some jobs are not finished yet is that these jobs do not have long enough time to finish all of their workload. Thus,

\[ \frac{Q_{t}^{(N)}}{N} = \frac{1}{N} \sum_{s=1}^{t} A^{(N)}(s, t - s + 1), \] (3.4.3)

where \( A^{(N)}(t, w) \) is the number of jobs, which arrives at time slot \( t \) and have at least \( w \) units of workload.

When \( N \) goes to infinity, it can be simplified as follows:

\[
\lim_{N \to \infty} \frac{Q_{t}^{(N)}}{N} = \sum_{s=1}^{t} \left( \lim_{N \to \infty} \frac{A^{(N)}_{s}}{N} \lim_{s \to \infty} \frac{A^{(N)}(s, t - s + 1)}{A^{(N)}_{s}} \right) \\
= \sum_{s=1}^{t} \left( \lim_{N \to \infty} \frac{A^{(N)}_{s}}{N} \right) P(R_{i} \geq t - s + 1) \text{ w.p.1} \] (3.4.4)

By mathematical induction, we can get that for any given time slot \( t \), the queue length \( Q_{t}^{(N)} \) satisfies \( \lim_{N \to \infty} \frac{Q_{t}^{(N)}}{N} < 1 \) w.p.1. Thus, Lemma 3.4.2 can be directly achieved by a similar proving process as Theorem 3.3.1.

If the jobs are not Reduce-only applications, then there are both Map and Reduce phases. Based on Lemma 3.4.2, we can prove Theorem 3.4.1 as follows.

**Proof of Theorem 3.4.1.** To prove the theorem, we first introduce another type of scheduler called **Scheduler with Proper Threshold (SPT)** if it schedules the Map and Reduce tasks as follows: All the Map tasks are scheduled within a pool of

70
\[ N_m = N - \sum_{k=0}^{K} N_r(k) \] machines, where \( K \) is the maximum number of tasks in Reduce jobs. Since each task are also i.i.d., let \( \overline{R_k} \) be the expectation of \( k^{th} \) task, where \( 1 \leq k \leq K \). For the Reduce task \( k \) of each job, it should be scheduled within a pool of \( N_r(k) \) machines. The \( K \) pools of \( \{N_r(1), \cdots N_r(K)\} \) machines, which are reserved for Reduce tasks, and the pool of \( N_m \), which are reserved for Map tasks, does not have intersection. Within each pool of machines, the corresponding tasks are scheduled by any work-conserving method. We choose the threshold \( N_r(k) \) as \( \left\lceil \frac{\overline{R_k}}{M + R} N \right\rceil \).

Given a total of \( N \) machines in the data center, let \( Q_{0,t}^{(N)} \) be the queue length of the Map jobs at time slot \( t \), and \( Q_{k,t}^{(N)} \) be the queue length of the \( k^{th} \) Reduce task of each job. Then, if the scheduler only schedules the Map tasks of new arriving jobs and Reduce tasks which arrived in the previous time slot, we can get \( \frac{Q_{0,t}^{(N)}}{N_m} < 1 \) w.p.1 by the proof of Theorem 3.3.1. Also, based on Lemma 3.4.2, we can get \( \frac{Q_{k,t}^{(N)}}{N_r(k)} < 1 \) w.p.1 for all \( 1 \leq k \leq K \). Thus, following the same proof of Theorem 3.3.1, we can directly get that any scheduler in SPT is asymptotically optimal.

In fact, the threshold between Map and Reduce tasks are unnecessary when \( N \) goes to infinity. Since each pool of machines can guarantee enough space for corresponding jobs when \( N \) goes to infinity, then without these thresholds, there is still enough space for all the jobs. In other words, for any work-conserving scheduler in non-preemptive and non-parallelizable scenario, the queue length \( Q_{t}^{(N)} = \sum_{k=0}^{K} Q_{k,t}^{(N)} \), then

\[
\lim_{N \to \infty} \frac{Q_{t}^{(N)}}{N} = \lim_{N \to \infty} \frac{\sum_{k=0}^{K} Q_{k,t}^{(N)}}{N_m + \sum_{k=1}^{K} N_r(k, t)} \\
\leq \lim_{N \to \infty} \max_{k_1, \ldots, k_K} \left\{ \frac{Q_{0,t}^{(N)}}{N_m}, \frac{Q_{k,t}^{(N)}}{N_r(k, t)} \right\} \\
\leq \max_{k_1, \ldots, k_K} \left\{ \lim_{N \to \infty} \frac{Q_{0,t}^{(N)}}{N_m}, \lim_{N \to \infty} \frac{Q_{k,t}^{(N)}}{N_r(k, t)} \right\} < 1 \text{ w.p.1.}
\] (3.4.5)
Thus, by the same proof method of Theorem 3.3.1, we can directly get that any work-conserving scheduler is asymptotically optimal.

**Theorem 3.4.3.** In the non-preemptive and non-parallelizable scenario of MapReduce framework, under assumptions 3.2.3-3.2.3, and 3.2.3-3.2.3, any work-conserving scheduler is asymptotically optimal.

**Proof.** In the Reduce only scenario, assume that each job only has one non-preemptive and non-parallelizable Reduce task. What we need to prove now is that

$$
\limsup_{N \to \infty} \frac{Q_t^{(N)} - N}{\sqrt{N \log \log N}} < 0 \text{ w.p.1.} \quad (3.4.6)
$$

where $Q_t^{(N)}$ is the queue length of Reduce jobs in time slot $t$ when the system has $N$ machines. If Eq. (3.4.6) holds for all $t$, Theorem 3.4.3 can be achieved by the same proof of Theorem 3.4.1.

Similar to the proof of Lemma 3.4.2, we use mathematical induction to prove Eq. (3.4.6). With $N$ machines, let $A^{(N)}(t, w)$ be the number Reduce jobs, which arrive at time slot $t$ and have at least $w$ units of workload.

First, when $t = 1$, $Q_1^{(N)} = A^{(N)}(1, 1)$, then we can get that

$$
\limsup_{N \to \infty} \frac{Q_1^{(N)} - N}{\sqrt{N \log \log N}} = \limsup_{N \to \infty} \frac{A^{(N)}(1, 1) - N}{\sqrt{N \log \log N}}
$$

$$
= \limsup_{N \to \infty} \frac{P(R_j^{(t)} \geq 1)\frac{A^{(N)}(1, 1)}{N} - E[A^{(N)}(1, 1)]\sqrt{N}}{\sqrt{\log \log N}} + \limsup_{N \to \infty} \frac{\sqrt{N}}{\sqrt{\log \log N}} \left( \frac{P(R_j^{(t)} \geq 1)\rho_N}{R} - 1 \right)
$$

By Markov’s inequality, $P(R_j^{(t)} \geq 1) \leq \bar{R}$. Thus, by assumptions 3.2.3 and 3.2.3, we can directly get that Eq. (3.4.6) holds for $t = 1$.

Assume that Eq. (3.4.6) holds for all $s = 1, 2, \ldots, t - 1$. If we can prove that Eq. (3.4.6) also holds for $t$, then by mathematical induction, we can show that Eq. (3.4.6) holds for any given $t$. Since Eq. (3.4.6) holds for all $s = 1, 2, \ldots, t - 1$, then
no Reduce jobs are delayed because there are not enough machines to accommodate the workload. In other words, if a job arriving at time slot \( s \) is still running in time slot \( t \), then the workload of that job is greater than \( t - s \) units.

For simplicity, we introduce \( Y_j^{(t)}(w) \) as follows:

\[
Y_j^{(t)}(w) \triangleq 1(R_j^{(t)} \geq w) = \begin{cases} 
1, & R_j^{(t)} \geq w \\
0, & R_j^{(t)} < w
\end{cases},
\] (3.4.8)

where \( 1(\cdot) \) is a indicator function.

Then,

\[
\limsup_{N \to \infty} \frac{Q_t^{(N)} - N}{\sqrt{N \log \log N}} = \limsup_{N \to \infty} \frac{\sum_{s=1}^{t} A(N)(s, t - s + 1) - N}{\sqrt{N \log \log N}}
\]

\[
= \limsup_{N \to \infty} \frac{\sum_{s=1}^{t} \sum_{j=1}^{A(N)} Y_j^{(t)}(t - s + 1) - N}{\sqrt{N \log \log N}}
\]

\[
= \liminf_{N \to \infty} \frac{(1 - \rho_N)\sqrt{N}}{\sqrt{\log \log N}} + \limsup_{N \to \infty} \frac{\sqrt{A(N) \log \log A(N)}}{\sqrt{N \log \log N}} \cdot \sum_{s=1}^{t} \limsup_{A(N) \to \infty} \frac{A(N) \log \log A(N)}{\sqrt{N}}
\]

\[
+ \limsup_{N \to \infty} \frac{\left( \sum_{s=1}^{t} E[Y_j^{(t)}(t - s + 1) \frac{A(N)}{N} - \bar{R}E[\frac{A(N)}{N}]] \right) \sqrt{N}}{\sqrt{\log \log N}}.
\] (3.4.9)

For the expectation \( E \left[ Y_j^{(t)}(w) \right] \) and the standard derivation \( \sigma_{Y_j^{(t)}(w)} \) of \( Y_j^{(t)}(w) \), we have

\[
E \left[ Y_j^{(t)}(w) \right] = E \left[ 1(R_j^{(t)} \geq w) \right] = P(R_j^{(t)} \geq w)
\] (3.4.10)

and

\[
\sigma_{Y_j^{(t)}(w)} \leq \sqrt{E \left[ \left( 1(R_j^{(t)} \geq w) \right)^2 \right]} \leq 1.
\] (3.4.11)

By LIL and assumption 3.2.3, we get
\[
\limsup_{N \to \infty} \frac{Q_i^{(N)} - N}{\sqrt{N \log \log N}} \leq \sqrt{2ct} - \liminf_{N \to \infty} \frac{(1 - \rho_N)\sqrt{N}}{\sqrt{\log \log N}} \\
+ \limsup_{N \to \infty} \frac{\sum_{s=1}^{t} E[Y_j^{(t)}(t-s+1)]A_i^{(N)} N - \bar{R}E[A_i^{(N)}]}{\sqrt{\log \log N}} \sqrt{N} \tag{3.4.12}
\]

where \( c \) is constant. Since \( \sum_{s=1}^{t} E[Y_j^{(t)}(t-s+1)] = \sum_{s=1}^{t} P(R_j^{(t)}(t-s+1) \geq (t-s+1)) \leq \bar{R} \), by assumption 3.2.3 we can get that

\[
\limsup_{N \to \infty} \frac{Q_i^{(N)} - N}{\sqrt{N \log \log N}} \leq \sqrt{2ct} - \liminf_{N \to \infty} \frac{(1 - \rho_N)\sqrt{N}}{\sqrt{\log \log N}} \tag{3.4.13}
\]

Thus, by assumption 3.2.3, we can directly get that Eq. (3.4.6) holds for \( t \). By mathematical induction, for any given \( t \), Eq. 3.4.6 holds. Then, the remaining proof is same as the proof of Theorem 3.4.1. \( \square \)

**Remark 3.4.4.** If we change \( \rho_N \) to \( \frac{A_i^{(N)}}{N(M+R)} \) in assumption 3.2.3 and the assumption also holds w.p.1, then the same applies for Theorem 3.4.3 without assumption 3.2.3.

### 3.5 The Relationship between \( T \) and \( N \)

In the previous sections, we have studied the behavior of asymptotic optimality of work-conserving schedulers when \( N \) goes to infinity. However, in long running system, the total number of time slots can also be very large. To show when the asymptotic optimality holds when both \( T \) and \( N \) are large, we need to consider the relationship between \( T \) and \( N \), which is shown in this section.

Let \( T(N) \) be a function of \( N \). We show the sufficient conditions to guarantee asymptotic optimality of work-conserving schedulers in the following theorem.
Theorem 3.5.1. In the preemptive and parallelizable scenario, assume 3.2.3, 3.2.3-3.2.3 hold. Additionally, if there exists a constant $\alpha > 0$, such that

$$\lim_{N \to \infty} T(N) \exp \left( -k N \right) N^{1+\alpha} \leq 1,$$  \hspace{1cm} (3.5.1)

where $k = c I_W \left( \frac{W}{\rho} \right)$. $W$ is sum of two independent random variables $M$ and $R$, which have the same distribution as $\{M_i\}$ and $\{R_i\}$, respectively. $W = M + R$ and $I_W(\cdot)$ are expectation and rate function of workload $W$, respectively. Assume that the moment generating function of $W$ has finite value in some neighborhood of 0. Then

$$\lim_{N \to \infty} \frac{F^{wc}(N, T(N))}{F^*(N, T(N))} = 1 \text{ w.p.1}.$$  \hspace{1cm} (3.5.2)

Proof. Let

$$E_N = \left\{ \sum_{i=1}^{A_i(N)} M_i(t) + \sum_{i=1}^{A_{i-1}(N)} R_i(t) > N \right\},$$

such that $t = 1, 2, \ldots T(N)$, there exists a $t_0$.

Since

$$E_n = \bigcup_{t=1}^{T(N)} \left\{ \sum_{i=1}^{A_i(N)} M_i(t) + \sum_{i=1}^{A_{i-1}(N)} R_i(t) > N \right\},$$

by i.i.d. assumption, we get

$$P(E_n) \leq \sum_{t=1}^{T(N)} P \left( \sum_{i=1}^{A_i(N)} M_i(t) + \sum_{i=1}^{A_{i-1}(N)} R_i(t) > N \right) = \sum_{t=1}^{T(N)} P \left( \sum_{i=1}^{A_i(N)} W_i(t) > N \right),$$  \hspace{1cm} (3.5.5)

where $\{W_i(t)\}$ is a sequence of i.i.d random variables that have the same distribution as $W$.

By Chernoff’s inequality, we have

$$P(E_N) \leq \sum_{t=1}^{T(N)} \exp \left( -A_i(N) I_W \left( \frac{W}{\rho} \right) \right) = T(N) \exp \left( -k_i(N) N \right),$$  \hspace{1cm} (3.5.6)
where $I_W(\cdot)$ is the rate function of $W$ and $k_i^{(N)} = \frac{A_i^{(N)}}{N} I_W(\overline{W})$. When $N$ goes to infinity, $k_i^{(N)}$ converges to $k = eI_W(\overline{W}/\rho)$ for all $t$. Then, when $N$ goes to infinity, we have

$$\lim_{N \to \infty} P(E_N) N^{1+\alpha} \leq \lim_{N \to \infty} T(N) \exp (-kN) N^{1+\alpha} \leq 1.$$  

(3.5.7)

Thus, $\sum_{n=1}^{N} P(E_n) < \infty$. Based on Borel-Cantelli Lemma [14], we can get that

$$P\left(\limsup_{N \to \infty} E_N\right) = 0,$$

(3.5.8)

which means that $P(E_N$ infinitely often $)= 0$. Thus, $P(E_N$ only happens for finite $N$) $= 1$. Then, there exists a $N_0$, such that for any $N > N_0$, $P(E_N^c) = 1$.

Now let us see what is the meaning of $E_N^c$. Based on Eq. (3.5.4), we have $E_N^c = T(N) \cap \left\{ \sum_{i=1}^{A_i^{(N)}} M_i^{(t)} + \sum_{i=1}^{A_i^{(N-1)}} R_i^{(t)} > N \right\}$. In other words, for any event in $E_N^c$, there are always enough machines to accommodate the workload in each time slot. Thus, $E_N^c \subseteq \left\{ \frac{F_{wc}(N,T)}{F^*(N,T)} = 1 \right\}$. Since $P(E_N^c$, where $N > N_0) = 1$, then $P\left(\frac{F_{wc}(N,T)}{F^*(N,T)} = 1, where N > N_0\right) = 1$. Then we can have that

$$\lim_{N \to \infty} \frac{F_{wc}(N,T(N))}{F^*(N,T(N))} = 1 \text{ w.p.1.}$$

(3.5.9)

**Remark 3.5.2.** For a constant $T$, we can easily check that Eq. (3.5.1) is satisfied.

### 3.6 Simulation Results

#### 3.6.1 Simulation Setting

We evaluate the efficacy of different schedulers for both preemptive and parallelizable and non-preemptive and non-parallelizable scenarios for different number of machines.
We choose Poisson process as the job arrival process [8][3]. To show the performance of work-conserving schedulers under heavy tailed distributions, we choose Pareto distribution as the workload distributions of Map and Reduce [15]. The cumulative distribution function of a random variable $X$ with Pareto distribution is shown as follows:

$$P(X > x) = \begin{cases} 
1 - \left(\frac{x_m}{x}\right)^\alpha & x \geq x_m \\
0 & x < x_m \end{cases}, \quad (3.6.1)$$

where $x_m$ is a scale parameter and $\alpha$ is a shape parameter.

For the workload distribution of Map phase of jobs, we choose the scale parameter $x_m$ as 20, and the shape parameter $\alpha$ as 3. For the workload distribution of Reduce phase of jobs, we choose the scale parameter $x_m$ as 10, and the shape parameter $\alpha$ as 3.

We compare 4 typical work-conserving schedulers:

The **FIFO** scheduler: It is the default scheduler in Hadoop. All the jobs are scheduled in their order of arrival.

The **Fair** scheduler: It is a widely used scheduler in Hadoop. The assignment of machines are scheduled to all the waiting jobs in a fair manner. If some jobs need fewer machines, then the remaining machines are scheduled to other jobs, to avoid resource wastage and to keep the scheduler work-conserving.

The **ASRPT** scheduler: ASRPT (Available Shortest Remaining Precessing Time) is given in [3], which is based on SRPT [13] and provides good performance guarantee for any given number of machines $N$.

The **LRPT** scheduler: Jobs with larger unfinished workload are always scheduled first. Roughly speaking, the performance of this scheduler represents in a sense how poorly even some work-conserving schedulers can perform.
In the simulations, the ratio of a scheduler is obtained by the total flow-time of the scheduler over the lower bound of the total flow-time in 100 time slots. Here, we choose the total flow time when all the jobs can be immediately served as the lower bound. The lower bound is just the total flow time of jobs when there is infinity number of machines. Obviously, there are better (larger) lower bound estimations [3]. However, when $N$ is large enough, as we will see, the performance of all work-conserving schedulers converge to this lower bound of the flow time, that showing asymptotic optimality.

### 3.6.2 Fixed Traffic Intensity

In this part, we choose traffic intensity $\rho$ as 0.5. The ratios to the lower bound of different schedulers are shown in Fig. 3.1 for independent Map and Reduce workload for each job. Also, we choose the correlation coefficient is 1 and $-1$ between the workload of Map and Reduce for each job as positive and negative correlation scenarios. The ratios are shown in Fig. 3.2 and Fig. 3.3, respectively.

From Fig. 3.1 to Fig. 3.3, we can directly get that the ratio is very close to 1 in all the figures, when $N$ is large (e.g., $N = 500$, the difference from the lower bound
Figure 3.2: Ratio to the Lower Bound (Pareto Distribution, Fixed Traffic Intensity, Map and Reduce are Positively Correlated)

Figure 3.3: Ratio to the Lower Bound (Pareto Distribution,Fixed Traffic Intensity, Map and Reduce are Negatively Correlated)
is less than 0.3%). Thus, all the four work-conserving schedulers are asymptotically optimal in the fixed traffic intensity scenario.

3.6.3 Heavy Traffic Scenario

In the heavy traffic scenario, we retain all the parameters as with the fixed traffic intensity scenario, except that the traffic intensity changes corresponding to different number of machines. We choose \((1 - \rho N)N^{1/3} = 1\), which satisfies assumptions 3.2.3 and 3.2.3. Then, the ratios to the lower bound of different schedulers are shown in Fig. 3.4 (independent), Fig. 3.5 (positively correlated), and Fig. 3.6 (negatively correlated).

Similarly, from Fig. 3.4 to Fig. 3.6, we can see that all the four work-conserving schedulers are asymptotically optimal in the heavy traffic scenario (the traffic intensity is about 0.93 when total number of machines is 10000).

3.6.4 Exponentially Distributed Workload

We choose the workload distribution of Map and Reduce for each job as exponential distribution with mean 30 and 15, respectively. The efficiency ratios of different
Figure 3.5: Ratio to the Lower Bound (Pareto Distribution, Heavy Traffic, Map and Reduce are Positively Correlated)

Figure 3.6: Ratio to the Lower Bound (Pareto Distribution, Heavy Traffic, Map and Reduce are Negatively Correlated)
schedulers are shown in Fig. 3.7 (independent Map and Reduce), Fig. 3.8 (positively correlated) and Fig. 3.9 (negatively correlated).

For heavy traffic scenario, the efficiency ratios of different schedulers are shown in Fig. 3.10 (independent Map and Reduce), Fig. 3.11 (positively correlated) and Fig. 3.12 (negatively correlated).

### 3.6.5 Uniformly Distributed Workload

Then, we evaluate the uniformly distributed workload. We choose the workload distribution of Map for each job as \( U[10, 50] \) and the workload distribution of Reduce
Figure 3.9: Ratio to the Lower Bound (Exponential Distribution, Fixed Traffic Intensity, Map and Reduce are Negatively Correlated)

Figure 3.10: Ratio to the Lower Bound (Exponential Distribution, Heavy Traffic, Map and Reduce are Independent)

Figure 3.11: Ratio to the Lower Bound (Exponential Distribution, Heavy Traffic, Map and Reduce are Positively Correlated)
for each job as $U[10, 20]$. The efficiency ratios of different schedulers are shown in Fig. 3.13.

For heavy traffic scenario, the efficiency ratios of different schedulers are shown in Fig. 3.16 (independent Map and Reduce), Fig. 3.17 (positively correlated) and Fig. 3.18 (negatively correlated).

From the figures, for both exponential and uniform distributions, the four work-conserving schedulers are asymptotically optimal.
Figure 3.14: Ratio to the Lower Bound (Uniform Distribution, Fixed Traffic Intensity, Map and Reduce are Positively Correlated)

Figure 3.15: Ratio to the Lower Bound (Uniform Distribution, Fixed Traffic Intensity, Map and Reduce are Negatively Correlated)

Figure 3.16: Ratio to the Lower Bound (Uniform Distribution, Heavy Traffic, Map and Reduce are Independent)

85
Figure 3.17: Ratio to the Lower Bound (Uniform Distribution, Heavy Traffic, Map and Reduce are Positively Correlated)

Figure 3.18: Ratio to the Lower Bound (Uniform Distribution, Heavy Traffic, Map and Reduce are Negatively Correlated)
3.7 Conclusion

This work is motivated by growth in the size and number of data centers. We prove that any work-conserving scheduler is asymptotically optimal under a wide range of traffic loads, including the heavy traffic limit. These results are shown for scenarios where data migration is costly (non-preemptive and non-parallelizable) and cheap (preemptive and parallelizable). We also verify these analytical results via extensive simulations, whereby state-of-the-art work-conserving schedulers are shown to have similar and close-to-optimal delay performance when the number of machines is large.

Our results suggest that for large data centers, there is little to be gained by designing schedulers that optimize beyond ensuring the work conserving principle. Hence, providing a clear guideline on developing simple and scalable schedulers for large data centers.
CHAPTER 4
DESIGN OF A POWER EFFICIENT CLOUD COMPUTING ENVIRONMENT: HEAVY TRAFFIC LIMITS AND QOS

4.1 Introduction

Many large queueing systems, such as call centers and data centers, contain thousands of servers. For call centers, it is common to have 500 servers in one call center [16]. For data centers, Google has more than 45 data centers as of 2009, and each of them contains more than 1000 machines [17]. The traffic intensity $\rho_n$ for a queueing system with $n$ servers can be thought of as the rate of job arrivals divided by the rate at which jobs are served. At the same time, as the number of servers and requests becomes large, due to statistical multiplexing, the queueing system should work efficiently, which means that $\rho_n$ could approach 1 in the limit, i.e., $\lim_{n \to \infty} \rho_n = 1$. This regime of operation is called the heavy traffic regime. This work focuses on establishing new heavy traffic limits, and using these limits to design a power efficient data center based on different QoS requirements.

In this chapter, we consider four different classes of QoS, from the most stringent to the weakest. As will be explained later, for two of these QoS classes, we develop new heavy traffic limits. What we will show is that the number of servers required
to satisfy the QoS of each of these classes scales differently with respect to traffic intensity.

In our model, we assume that the arrival process is general, which allows it to capture a wide variety of traffic classes. Further, we assume that the service distribution is hyper-exponential, which also captures the significant variability in the service requirements of practical systems (e.g., data centers could have a broad mix of job types). For example, the hyper-exponential distribution can characterize any coefficient of variation (standard deviation divided by the mean) greater than 1.

The main contributions of this chapter are:

- This chapter makes new contributions to heavy traffic analysis, in that it derives new heavy traffic limits for two important QoS classes (to be defined in Section II) for queueing systems with general arrival processes and hyper-exponential service time distribution.

- Using the heavy traffic limits results, this chapter answers the important question for enabling a power efficient data center as an application: How many machines should a data center have to sustain a specific system load and a certain level of QoS, or equivalently how many machines should be kept “awake” at any given time?

The chapter is organized as follows. In Section 4.2, we present the system model and list four different practical QoS requirements, from the most to the least stringent. In Section 4.3, we describe related work, and discuss how heavy traffic limits can directly be obtained from existing literature under two of these four QoS metrics, but not for the other two practically important classes. In Sections 4.4 and 4.5, we develop new heavy traffic limits for these other two QoS metrics. Using these heavy traffic limits results, in Section 4.6 we consider cloud computing as an application
and compute the operational number of machines needed for different classes of QoS. Simulation results are provided in Section 4.7. Finally, we conclude this chapter in Section 4.8.

4.2 System Model and QoS Classes

4.2.1 System Model and Preliminaries

In our model, we assume that the queueing system contains \( n \) active/operational servers. A larger \( n \) will result in better QoS at the expense of higher operational costs. For example, a large number of servers means a large amount of energy needed for run, which corresponds to a substantial increase in the operational cost of the data center. Similarly, a large \( n \) implies a greater cost in staffing a call center. We assume that the job arrivals to the system are independent with rate \( \lambda_n \) and coefficient of variation \( c \). Assume that the service time \( v \) of the system satisfies a hyper-exponential distribution with \( k \) phases, as given by

\[
P(v > t) = \sum_{i=1}^{k} P_i e^{-\mu_i t}.
\]  

(4.2.1)

Without loss of generality, we assume that

\[
0 < \mu_1 < \mu_2 < ... < \mu_k < \infty;
\]

\[
P_i > 0, \forall i \in 1,...,k; \sum_{i=1}^{k} P_i = 1.
\]  

(4.2.2)

The buffer that holds the jobs that are yet to be scheduled is assumed to be unbounded. The service priority obeys a first-come-first-serve (FCFS) rule. In this chapter we consider a service model where each job is served by one server. All servers are considered to have similar capability.
4.2.2 Definition of QoS Classes

We study the performance of the queueing system for four different types of QoS requirements: Zero-Waiting-Time (ZWT), Minimal-Waiting-Time (MWT), Bounded-Waiting-Time (BWT) and Probabilistic-Waiting-Time (PWT).

Before we give the definition of the different QoS classes, we first provide some notations that will be used throughout this section. Here, we let $n$ denote the total number of servers. For a given $n$, we let $Q_n$ denote the total number of jobs in the system, $W_n$ denote the time that the job waits in the system before being processed. For two functions $f(n)$ and $g(n)$ of $n$, $g(n) = o(f(n))$ if and only if $\lim_{n \to \infty} g(n)/f(n) = 0$. Also, we use $\sim$ as equivalent asymptotics, i.e., $f(n) \sim g(n)$ means that $\lim_{n \to \infty} f(n)/g(n) = 1$. We also let $\phi(\cdot)$ and $\Phi(\cdot)$ denote the probability density function and cumulative distribution function of normal distribution respectively, and let $\varphi_X(\cdot)$ denote the characteristic function of the random variable $X$.

We now provide precise definitions for the four classes of interest. Since we are interested in studying the performance of the system in the heavy traffic limit, we let the traffic intensity $\rho_n \to 1$ as $n \to \infty$ in the case of each QoS class we study.

Zero-Waiting-Time (ZWT) Class

A system of the ZWT class is one for which

$$\lim_{n \to \infty} P\{Q_n \geq n\} = 0.$$  

The ZWT class corresponds to the class that provides the strictest of the QoS requirements we consider here. For such systems, the requirement is that a job is immediately served upon arrival (i.e., zero waiting time in queue).
Minimal-Waiting-Time (MWT) Class

For this class, the QoS requirement is

$$\lim_{n \to \infty} P\{Q_n \geq n\} = \alpha,$$

where $\alpha$ is a constant such that $0 < \alpha < 1$. For this class, there is a nonvanishing probability that the jobs queue of the system is not empty, which implies that the waiting time $P\{W_n > \epsilon\} \to 0$, for any $\epsilon > 0$. In other words, the waiting time for an arriving job is minimal.

Bounded-Waiting-Time (BWT) Class

For this class,

$$\lim_{n \to \infty} P\{Q_n \geq n\} = 1,$$

$$P\{W_n > t_1\} \sim \delta_n,$$

where

$$\lim_{n \to \infty} \delta_n = 0.$$

The BWT class corresponds to the class for which the probability of the waiting time $W_n$ exceeding a constant threshold $t_1$ decreases to 0 as $n$ goes to infinity. Here $t_1$ can be thought of as a tunable design parameter related to the needs of the applications.

Probabilistic-Waiting-Time (PWT) Class

For this class,

$$\lim_{n \to \infty} P\{Q_n \geq n\} = 1,$$

$$\lim_{n \to \infty} P\{W_n > t_2\} = \delta,$$

where $\delta$ is a given constant and satisfies $0 < \delta < 1$.

The PWT class corresponds to the class that provides the least strict QoS requirements of the four types of systems considered here. Hence, the probability that the
waiting time $W_n$ is greater than some constant threshold $t_2$ is non-zero, for large enough $n$. This means that the QoS requirement for this system is such that the waiting time $W_n$ is between 0 and $t_2$ with probability $1 - \delta$, as $n$ goes to infinity.

In applications of large queueing systems (e.g., data and call centers), the different QoS classes will result in different heavy traffic limits, and will thus be governed by different design rules for the number of operational servers $n$ and traffic intensity $\rho_n$. Our aim in this chapter is to characterize these heavy traffic limits. We can then directly use these heavy traffic limits to develop design rules for the data and call center environments based on different QoS requirements. For example, we can compute the number of machines needed for a given value of the offered load or calculate the maximum value of the supportable offered load for the available number of servers.

4.3 Related Work

It turns out that heavy traffic limits for the ZWT and PWT classes can be directly derived from the current literature. Hence, we will focus on developing new heavy traffic limits for the MWT and BWT classes. Late in the Numerical Evaluation section (Section 4.7), we will again refer to these classes to compare the heavy traffic limits against each other.

For the MWT class, some classical results on heavy traffic limits are given by Iglehart in [18], Halfin and Whitt in [19], and summarized by Whitt in Chapter 5 of his recent book [4]. This heavy traffic limit $((1 - \rho_n)\sqrt{n}$ goes to a constant as $n$ goes to infinity) is now called the Halfin-Whitt regime. Recently, the behavior of the normalized queue length in this regime has been studied by A. A. Puhalskii and M. I. Reiman [20], J. Reed [21], D. Gamarnik and P. Momeilovic [22], and Ward Whitt.
Based on these studies, some design and control policies have been proposed in [25, 26, 27, 28].

Our analysis of MWT class differs from prior work in two key aspects. First, the literature on heavy traffic limits assumes a Poisson arrival process and exponential service time [25, 26, 27, 28]. These models are perhaps appropriate for some simple systems, but need to be generalized for today’s queueing systems such as increasingly complex call-centers and cloud computing environments. The arrival process in such complex and large systems may be independent, but more general. More importantly as mentioned in the Introduction, the service times of jobs are quite diverse, hence, unlikely to be accurately modeled by an exponential service time distribution. In [23], Whitt also considers the hyper-exponential distributed service time, but only with two stages and where one of them always has zero mean. Moreover, while there are studies that give heavy traffic solutions for more general scenarios [20, 21], these solutions can only be described by complex stochastic differential equations, which are quite cumbersome to use and provide little insight.

In the literature, ZWT and PWT classes of heavy traffic have been studied [25, 29, 26]. However, to the best of our knowledge, MWT and BWT classes have not been analyzed. In Section 4.4 and Section 4.5, we now provide new heavy traffic limits for MWT and BWT classes (described in the Introduction), respectively.

### 4.4 Heavy Traffic Limit for the MWT class

Proposition 4.4.1 shows us how the number of servers must scale in the heavy traffic limit for the MWT class.

**Proposition 4.4.1.** Assume

\[
\lim_{n \to \infty} \rho_n = 1,
\]

(4.4.1)
\[ \lim_{n \to \infty} P\{Q_n \geq n\} = \alpha, \quad \alpha \in [0, 1], \quad (4.4.2) \]

then

\[ L \leq \lim_{n \to \infty} (1 - \rho_n) \sqrt{n} \leq U, \quad (4.4.3) \]

\[ U = \sum_{i=1}^{k} \left(1 + \frac{c^2 - 1}{2} P_i \right) \frac{P_i}{\mu_i} \psi_U \sqrt{\mu_i}, \quad (4.4.4) \]

\[ L = \max_{i \in \{1,...,k\}} \left\{\left(1 + \frac{c^2 - 1}{2} P_i \right) \frac{P_i}{\mu_i} \right\} \psi_L \sqrt{\mu}, \quad (4.4.5) \]

where

\[ \mu = \left(\sum_{i=0}^{k} \frac{P_i}{\mu_i}\right)^{-1}, \quad \rho_n = \frac{\lambda_n}{n \mu}, \quad (4.4.6) \]

\[ \psi_U \text{ is given by } \frac{\alpha}{k} = \left[1 + \sqrt{2 \pi \psi_U \Phi(\psi_U) \exp(\psi_U^2/2)}\right]^{-1}, \quad (4.4.7) \]

\[ \psi_L \text{ is given by } \alpha = \left[1 + \sqrt{2 \pi \psi_L \Phi(\psi_L) \exp(\psi_L^2/2)}\right]^{-1}, \quad (4.4.8) \]

From Proposition 4.4.1, we can see that, to satisfy the QoS of MWT, the limit of \((1 - \rho_n) \sqrt{n}\) is bounded by two constants \(L\) and \(U\).

To prove Proposition 4.4.1, we first construct an artificial system structure, in which the arrival process and the capacity of each server are same as the original system. In the artificial system, we assume that there are \(k\) types of jobs. For each arrival, we know that the probability of the \(i^{th}\) type job is \(P_i\), and the service time of each \(i^{th}\) type of job is exponentially distributed with mean \(1/\mu_i\). Thus, the service time \(v\) of the system can be viewed as a hyper-exponential distribution which
satisfies Eq. (4.2.1). Assume that there is an omniscient scheduler for the artificial system. This scheduler can recognize the type of arriving jobs, and send them to the corresponding queue. For arrivals of type $i$, the scheduler sends them to the $i^{th}$ queue, which contains $n_i$ servers. Then the arrival rate of the $i^{th}$ queue is $P_i \lambda_n$. Also, the priority of each separated queue obeys the FCFS rule. The artificial system is shown in Fig. 4.1(b).

Before the proof of this proposition, we first show Lemmas 4.4.2 and 4.4.3.

**Lemma 4.4.2.** The inter-arrival time $\{Y_j^{(i)}, j = 1, 2, \ldots\}$ between jobs headed to the
Figure 4.2: The inter-arrival time of the separated queues

$i^{th}$ separated queue in the artificial system is is i.i.d., with coefficient of variation $c^{(i)} = \sqrt{1 + (c^2 - 1)P_i}$.

Proof. For the $i^{th}$ separated queue in Fig. 4.1(b), the inter-arrival time $Y^{(i)}$ is shown in Fig. 4.4. Formally,

$$Y^{(i)}_j = \sum_{t=\sum_{s=1}^{j-1}k_s+1}^{\sum_{s=1}^j k_s} X_t,$$

where $k^{(i)}_j$ is equal to the number of original arrivals between $(j - 1)^{th}$ and $j^{th}$ arrivals in the $i^{th}$ separated queue, and $\{X_t\}$ are the inter-arrival times in the original queueing system.

Based on the structure of the artificial system, $k^{(i)}_j$ is an independent random variable with geometric distribution with parameter $P_i$. Note that $\{X_t\}$ is also independent of $k^{(i)}_j$, because $k^{(i)}_j$ is only dependent on the distribution of the service time.

Then, for each $i$, the inter-arrival time $\{Y^{(i)}_j, j = 1, 2, \ldots\}$ is i.i.d.

By Wald’s Equation [30], we can obtain that

$$\mathbb{E}(Y^{(i)}_j) = \mathbb{E}(k^{(i)}_j)\mathbb{E}(X_t).$$

(4.4.9)
Similarly,

\[ \text{Var}(Y_j^{(i)}) = \mathbb{E} \left( (Y_j^{(i)})^2 \right) - \left( \mathbb{E}(Y_j^{(i)}) \right)^2 \]  

(4.4.10)

\[ = \text{Var}(k_j^{(i)}) \mathbb{E}(X_t)^2 + \mathbb{E}(k_j^{(i)}) \text{Var}(X_t). \]

Thus, the coefficient of variation \( c^{(i)} \) for all the separated queues is given as follows:

\[
c^{(i)} = \sqrt{\frac{\text{Var}(Y_j^{(i)})}{\mathbb{E}(Y_j^{(i)})^2}} = \sqrt{\frac{\text{Var}(k_j^{(i)}) (\mathbb{E}(X_t))^2 + \mathbb{E}(k_j^{(i)}) \text{Var}(X_t)}{\mathbb{E}(k_j^{(i)})^2 \mathbb{E}(X_t)^2}}
\]

(4.4.11)

\[ = \sqrt{\frac{1 - P_i^2 \mathbb{E}(X_t)^2}{\mathbb{E}(X_t)^2}} \sqrt{\frac{1}{P_i}} \text{Var}(X_t) = \sqrt{1 + (c^2 - 1) P_i}. \]

Note that, if the arrival process is Poisson, then \( c = 1 \), and \( c^{(i)} = 1, \forall i = 1, 2, \ldots k. \)

**Lemma 4.4.3.** For a GI/M/n queue,

\[ \lim_{n \to \infty} P\{Q_n \geq n\} = \alpha_c \]  

(4.4.12)

if and only if

\[ \lim_{n \to \infty} (1 - \rho_n) \sqrt{n} = \beta, \]  

(4.4.13)

under the following conditions:

\[ \beta = \frac{(1 + c^2) \psi}{2}, \quad \alpha_c = [1 + \sqrt{2 \pi \psi} \Phi(\psi) \exp(\psi^2/2)]^{-1}. \]  

(4.4.14)

**Proof.** This result and the corresponding proof is given by S. Halfin and W. Whitt as Theorem 4 in [19].

**Proof of Proposition 4.4.1.** To prove the upper bound, we consider Artificial System I. In this system, we choose \( n_i \) such that

\[ (1 - \rho_n) \sqrt{n_i} = \beta^{(i)}_U, \]  

(4.4.15)
where
\[ \rho_{ni} = \frac{P_i \lambda_i}{n_i \mu_i}, \quad \beta^{(i)}_U = \frac{(1 + (e^{(i)})^2) \psi_U}{2} = (1 + \frac{c^2 - 1}{2} P_i) \psi_U, \]
(4.4.16)
and
\[ \frac{\alpha}{k} = [1 + \sqrt{2} \pi \psi_U \Phi(\psi_U) \exp(\psi_U^2/2)]^{-1}. \]
(4.4.17)

By applying Lemma 4.4.3 into Artificial System I, for each individual queue, we have
\[ \lim_{n_i \to \infty} P\{Q^{(i)}_{n_i} \geq n_i\} = \frac{1}{k}, \quad \forall i \in \{1, \ldots, k\}, \]
(4.4.18)
where \( Q^{(i)}_{n_i} \) is the length of the \( i^{th} \) separated queue.

Let \( n_U = \sum_{i=1}^{k} n_i \), \( Q_{n_U} = \sum_{i=1}^{k} Q^{(i)}_{n_i} \). Then, for Artificial System I, we have
\[ P\{Q_{n_U} \geq n_U\} = P\left\{ \sum_{i=1}^{k} Q^{(i)}_{n_i} \geq \sum_{i=1}^{k} n_i \right\} \leq P\left( \bigcup_{i=1}^{k} \{Q^{(i)}_{n_i} \geq n_i\} \right) \leq \sum_{i=1}^{k} P\{Q^{(i)}_{n_i} \geq n_i\}. \]
(4.4.19)

By taking the limit on both sides,
\[ \lim_{n_i \to \infty} P\{Q_{n_i} \geq n_i\} \leq \lim_{n_i \to \infty} \left( \sum_{i=1}^{k} P\{Q^{(i)}_{n_i} \geq n_i\} \right) = \left( \sum_{i=1}^{k} \lim_{n_i \to \infty} P\{Q^{(i)}_{n_i} \geq n_i\} \right) = \alpha \]
(4.4.20)

From Eq. (4.4.20), we know that when Artificial System I has \( n_U \) servers, the probability that queue length \( Q_{n_U} \) is greater than or equal to \( n_U \) is asymptotically less than or equal to \( \alpha \). (Observe that the original system needs no more servers than Artificial System I since there may be some idle servers in Artificial System I, even when the other job queues are not empty. To guarantee the statement, we assume that there is a sequence of phase-type distributions \( F_l \), which weakly converges to the distribution \( F \) of the inter-arrival time as \( l \to \infty \) [31]. For each \( l \), the state space \((Q^l_1, \ldots, Q^l_k)\) of each separate queue length in the Artificial System I and in the original system are
both $k$ dimensional spaces with Markovian property. We can observe that the transition rates from $(Q^l_1, ..., Q^l_{i-1}, Q^l_i + 1, Q^l_{i+1}, ..., Q^l_k)$ to $(Q^l_1, Q^l_2, ..., Q^l_{i-1}, Q^l_i, Q^l_{i+1}, ..., Q^l_k)$ in the Artificial System I is less than the transmission rates in the original system, when $Q^l_i \geq n_i$ and $\sum_{i=1}^{k} Q^l_i \leq n$; otherwise, the transmission rates remain the same. Thus, the total queue length $Q^l_U$ of Artificial System I is greater than or equal to the queue length $Q_l$ of the original system in the stochastic order [32], for each $l$. In other words, $Q^l_U \geq Q^l$. Also, as $l \to \infty$, $Q^l_U$ and $Q^l$ will weakly converge to $Q_U$ and $Q$ respectively [4]. Hence, $Q_U \geq Q$. Thus, to satisfy the same requirement, the original system does not need more servers than Artificial System I.

By using Eqs. (4.4.15) and (4.4.16), we can solve for $n_i$. That is, $n_i = \frac{P_i \lambda_n}{\mu_i} + \beta_U^{(i)} \sqrt{\frac{P_i \lambda_n}{\mu_i}} \sqrt{\frac{1}{\rho_i}}$. Since we consider the scenario with large $n$ and $n_i$, and $\lim_{n_i \to \infty} \rho_i = 1$, we ignore the factor $\sqrt{\frac{1}{\rho_i}}$ and obtain Eq. (4.4.21).

$$n \leq n_U = \sum_{i=1}^{k} n_i = \sum_{i=1}^{k} \left( \frac{P_i \lambda_n}{\mu_i} + \beta_U^{(i)} \sqrt{\frac{P_i \lambda_n}{\mu_i}} \right)$$

By taking Eq. (4.4.21) into the definition of $\rho_n$ in Eq. (4.4.6), we can directly obtain the upper bound Eq. (4.4.4) of Eq. (4.4.3).

For the lower bound, we consider Artificial System II shown in Fig. 4.3, which has similar structure as Artificial System I and Fig. 4.1(b). Now in this system, we choose $n_i$ such that it satisfies the following conditions:

$$n_i = \begin{cases} \frac{P_i \lambda_n}{\mu_i}, & i \in \{1, \ldots, k\}, \ i \neq m \\ \frac{P_m \lambda_n}{\mu_m} + \beta_L^{(m)} \sqrt{\frac{P_m \lambda_n}{\mu_m}}, & i = m \end{cases}$$

(4.4.22)
Figure 4.3: Artificial System II
where
\[ \beta_L^{(i)} = \frac{(1 + (c^{(i)})^2)\psi}{2} = (1 + \frac{e^2 - 1}{2}P_i)\psi, \]
\[ m = \inf \arg\max_{i \in \{1, \ldots, k\}} \left( \beta_L^{(i)} \sqrt{\frac{P_i}{\mu_i}} \right), \] (4.4.23)
and
\[ \alpha = [1 + \sqrt{2\pi} \psi_L \Phi(\psi_L) \exp(\psi_L^2/2)]^{-1}. \] (4.4.24)

Then,
\[ \lim_{n_m \to \infty} (1 - \rho_{n_m})\sqrt{n_m} = \beta_L^{(m)}, \] (4.4.25)
where
\[ \rho_{n_m} = \frac{P_m\lambda_n}{n_m\mu}. \] (4.4.26)

By substituting Eqs. (4.4.12-4.4.14) into Eqs. (4.4.22-4.4.24), the reader can verify the following result for Artificial System II as shown in Fig. 4.3.

\[ \lim_{n_i \to \infty} P\{Q_{n_i}^{(i)} \geq n_i\} = \begin{cases} 1, & i \in \{1, \ldots, k\}, i \neq m \\ \alpha, & i = m \end{cases} \] (4.4.27)

Define \( n_L = \sum_{i=1}^{k} n_i \). If the original system has \( n_L \) servers, then we can construct a scheduler based on Artificial System II. This scheduler can make QoS of the arrivals satisfy Eq. (4.4.27). The original system, needs more servers than Artificial System II to satisfy Eq. (4.4.2). Therefore, \( n \) should be greater than or equal to \( n_L \), i.e.,

\[ n \geq n_L = \sum_{i=1}^{k} n_i = \sum_{i=1}^{k} \left( \frac{P_i\lambda_n}{\mu_i} \right) + \beta_L^{(m)} \sqrt{\frac{P_m\lambda_n}{\mu_m}} \]
\[ = \lambda_n \frac{\mu}{\mu} + \sqrt{\frac{\lambda_n}{\mu} \max_{i \in \{1, \ldots, k\}} \left( \beta_L^{(i)} \sqrt{\frac{P_i}{\mu_i}} \right)} \sqrt{\mu}. \] (4.4.28)

Since \( \rho \triangleq \frac{\lambda_n}{n\mu} \), we can substitute \( \rho_n \) with Eq. (4.4.28) to obtain the lower bound Eq. (4.4.5) of Eq. (4.4.3).
Corollary 4.4.4. For a Poisson arrival process, we can obtain the following upper bound $\hat{U}$:

$$\hat{U} = \left( \sum_{i=1}^{k} \sqrt{\frac{P_i}{\mu_i}} \right) \sqrt{\mu \psi_U} \leq U, \quad (4.4.29)$$

where

$$\mu = \left( \sum_{i=0}^{k} \frac{P_i}{\mu_i} \right)^{-1}, \quad \rho_n = \frac{\lambda_n}{n\mu}, \quad (4.4.30)$$

and

$$1 - (1 - \alpha)^\frac{1}{k} = [1 + \sqrt{2\pi \psi_U \Phi(\psi_U) \exp(\psi_U^2 / 2)}]^{-1}. \quad (4.4.31)$$

Proof. For a Poisson arrival process, $c = 1$ and $c^{(i)} = 1$, $\forall i \in \{1, 2, ..., k\}$. We consider a similar Artificial System III, which has same structure as Artificial System I. Let Artificial System III satisfy the following conditions.

$$\lim_{n_i \to \infty} (1 - \rho_{n_i}) \sqrt{n_i} = \hat{\psi}_U, \quad (4.4.32)$$

where

$$\rho_{n_i} = \frac{P_i \lambda_n}{n_i \mu_i}, \quad (4.4.33)$$

and

$$1 - (1 - \alpha)^\frac{1}{k} = [1 + \sqrt{2\pi \psi_U \Phi(\psi_U) \exp(\psi_U^2 / 2)}]^{-1}. \quad (4.4.34)$$

Similar to Artificial System I, for each individual queue, we have

$$\lim_{n_i \to \infty} \mathbb{P}\{Q^{(i)}_{n_i} \geq n_i\} = 1 - (1 - \alpha)^\frac{1}{k}, \quad \forall i \in \{1, ...k\}, \quad (4.4.35)$$

where $Q^{(i)}_{n_i}$ is the length of the $i^{th}$ separated queue.

Let $n_U = \sum_{i=1}^{k} n_i$. Since arrival process is a Poisson process, by the Colouring Theorem [33], the arrival process in each separated queue is also a Poisson process. Then, for Artificial System III, we have
\[ P\{Q_{n_U} \geq n_U\} = 1 - P\{Q_{n_U} < n_U\} \leq 1 - \prod_{i=1}^{k} \left(1 - P\{Q_{n_i}^{(i)} \geq n_i\}\right) \quad (4.4.36) \]

where
\[ Q_{n_U} = \sum_{i=1}^{k} Q_{n_i}^{(i)}. \quad (4.4.37) \]

By taking limits on each side, we obtain
\[
\lim_{n_i \to \infty} \frac{P\{Q_{n_U} \geq n_U\}}{P\{Q_{n_i}^{(i)} \geq n_i\}} \leq \lim_{n_i \to \infty} \left(1 - \prod_{i=1}^{k} \left(1 - P\{Q_{n_i}^{(i)} \geq n_i\}\right)\right) = 1 - \prod_{i=1}^{k} \left(1 - \lim_{n_i \to \infty} P\{Q_{n_i}^{(i)} \geq n_i\}\right) = \alpha \quad (4.4.38)
\]

From Eq. (4.4.38), we know that when artificial system III has \( n_U \) servers, \( \lim_{n_U \to \infty} P\{Q_{n_U} \geq n_U\} \leq \alpha \). To satisfy the same requirement, the original system does not need more servers than Artificial System III. By using Eqs. (4.4.32) and (4.4.33), we can get an expression of \( n_i \). That is,
\[
n \leq n_U = \sum_{i=1}^{k} n_i = \sum_{i=1}^{k} \left(\frac{P_i \lambda_n}{\mu_i} + \tilde{\psi}_U \sqrt{\frac{P_i \lambda_n}{\mu_i}}\right) = \frac{\lambda_n}{\mu} + \sqrt{\frac{\lambda_n}{\mu} \left(\sum_{i=1}^{k} \frac{P_i}{\mu_i}\right) \sqrt{\mu \tilde{\psi}_U}}. \quad (4.4.39)
\]

By substituting Eq. (4.4.39) into the definition of \( \rho_n \) in Eq. (4.4.30), we directly obtain the upper bound Eq. (4.4.29).

Since for a Poisson arrival process, \( c = 1 \) and \( c^{(i)} = 1 \), \( \forall i \in \{1, 2, \ldots, k\} \), \( \beta^{(i)}_{U} = \psi_U \) in Eq. (4.4.7). Further, since \( (1 - \frac{\alpha}{k})^k \) is an increasing function, \( (1 - \frac{\alpha}{k})^k \geq 1 - \alpha \). Thus, \( 1 - (1 - \alpha)^{\frac{k}{2}} \geq \frac{\alpha}{k} \). Since the right hand side of Eq. (4.4.34) is a decreasing function of \( \tilde{\psi}_U \), it directly follows that \( \psi_U \geq \tilde{\psi}_U \), i.e. \( \tilde{U} \leq U \) and thus Eq. (4.4.29) is a tighter upper bound than Eq. (4.4.4) for Poisson arrival process.

\[ \blacksquare \]
Corollary 4.4.5. When k = 1, the service time reduces to an exponential distribution. Based on the Proposition 4.4.1, we can see that U = L = \beta U = \beta L \triangleq \beta in this scenario, i.e., \( \lim_{n \to \infty} (1 - \rho_n)\sqrt{n} = \beta \). Hence, Proposition 4.4.1 in this work is consistent with Proposition 1 and Theorem 4 in [19].

Next, we provide a way of further sharpening the upper bounds by solving an optimization problem. Proposition 4.4.6 provides a better upper bound for the general arrival case, i.e. it improves upon Eq. (4.4.4), while Corollary 4.4.7 focuses on the Poisson arrival process and improves the bound on Eq. (4.4.29). The disadvantage of these bounds are that they are not in a closed form.

Proposition 4.4.6. The solution \( \tilde{U} \) of the following optimization problem results in an upper bound for the Eq. (4.4.4).

\[
\min_{\alpha_1, \ldots, \alpha_k} \frac{\sum_{j=1}^{k} \beta_j \sqrt{\frac{P_j}{\mu_j}}}{\sqrt{\sum_{j=1}^{k} \frac{P_j}{\mu_j},}}
\]

s.t. \[ \sum_{j=1}^{k} \alpha_j \leq \alpha, \]

where

\[
\beta_j = (1 + \frac{c^2 - 1}{2} P_j) \psi_j,
\]

\[
\alpha_j = [1 + \sqrt{2\pi} \psi_j \Phi(\psi_j) \exp(\psi_j^2/2)]^{-1},
\]

\[ 0 \leq \alpha_j \leq 1, \quad 0 \leq \beta_j \leq \infty, \quad \forall j. \]

Proof. It is not necessary to choose all the \( \alpha_j \) equally. Once Eq. (4.4.19) is satisfied, it is sufficient to find an upper bound. Thus, the minimum of all the upper bounds is a new tighter upper bound for Proposition 4.4.1. 

Note that the corresponding objective value of every \( \{\alpha_j, \ j = 1, \ldots, k\} \) in the feasible set of the optimization problem (4.4.40-4.4.41) is an upper bound of the limit in (4.4.3). Hence, if we choose \( \tilde{\alpha}_j = \frac{\alpha}{k}, \forall j = 1, \ldots, k \), it is easy to check that the
value of \( \{\alpha_j, j = 1,\ldots,k\} \) is in the feasible set, and the objective value is same as the upper bound in Eq. (4.4.4).

**Corollary 4.4.7.** The solution \( \widetilde{U} \) of the following optimization problem results in an upper bound for Poisson arrival process.

\[
\min_{\alpha_1,\ldots,\alpha_k} \frac{\sum_{j=1}^k \beta_j \sqrt{\frac{P_j}{\mu_j}}}{\sqrt{\sum_{j=1}^k \frac{P_j}{\mu_j}}},
\]

\[\text{s.t. ~} \lim_{s \to \infty} \int_{-\infty}^{\infty} \left[ \prod_{j=1}^k \phi_{Q_j} \left( \sqrt{\frac{P_j}{\mu_j}} \frac{1 - \exp(-its)}{it} \right) \right] dt \leq 2\pi \alpha, \tag{4.4.44}\]

where

\[
\beta_j = \frac{(1 + c^2) \psi_j}{2},
\]

\[
\alpha_j = \left[ 1 + \sqrt{2\pi \psi_j \Phi(\psi_j)} \exp(\psi_j^2/2) \right]^{-1},
\]

\[0 \leq \alpha_j \leq 1, \quad 0 \leq \beta_j \leq \infty, \quad \forall j,
\]

and the probability density function of \( \hat{Q}_j \) is

\[
f_j(x) = \begin{cases} 
\alpha_j \beta_j \exp(-\beta_j x), & \text{when } x > 0 \\
(1 - \alpha_j) \frac{\phi(x + \beta_j)}{\Phi(\beta_j)}, & \text{when } x < 0
\end{cases}
\]  \tag{4.4.46}

*Proof.* We construct a new comparable system with a similar structure as in Fig. 4.1(b).

For sub-queue \( j \), let the probability that queue length \( Q_j \) is greater than or equal to \( n_j \) be \( \alpha_j \). Then, the total number of servers \( n \) is

\[
n = \sum_{j=1}^k n_j = \left( \sum_{j=1}^k \frac{P_j}{\mu_j} \right) \lambda + \left( \sum_{j=1}^k \beta_j \sqrt{\frac{P_j}{\mu_j}} \right) \sqrt{\lambda}
\]

\[
= \frac{\lambda}{\mu} + \sum_{j=1}^k \beta_j \sqrt{\frac{P_j}{\mu_j}} \sqrt{\lambda} \sqrt{\mu}, \tag{4.4.47}
\]

where \( \mu \) is same as Eq. (4.4.6).
Note that, for the same number $n$, from the proof of Proposition 4.4.1, $P(\tilde{Q} \geq n) \geq P(Q \geq n)$. In other words, if we assume that the QoS of the artificial system can satisfy $P(\tilde{Q} \geq n) \leq \alpha$, then, to obtain the same QoS, the original system needs no more than $n$ servers. This means that there is room for improvement on the upper bound.

Now, consider the artificial system with the same QoS. Let $\hat{Q}_j = \frac{\tilde{Q}_j - n_j}{\sqrt{n_j}}$ be called the normalized queue length. Note that

$$\alpha \geq P\left(\sum_{j=1}^{k} \tilde{Q}_j \geq n\right) = P\left(\sum_{j=1}^{k} (n_j + \sqrt{n_j} \tilde{Q}_j) \geq n\right) = P\left(\sum_{j=1}^{k} \sqrt{n_j} \tilde{Q}_j \geq 0\right) = P\left(\sum_{j=1}^{k} \sqrt{\frac{P_j}{\mu_j}} \tilde{Q}_j \geq 0\right)$$

(4.4.48)

From Theorems 1 and 4 in [19], the pdf (probability density function) of the normalized queue length $\tilde{Q}_j$ is $f_j(x)$ given by Eq. (4.4.46). Then, the characteristic function of $\sum_{j=1}^{k} \sqrt{\frac{P_j}{\mu_j}} \tilde{Q}_j$ in Eq. (4.4.48) is

$$\varphi_{\sum_{j=1}^{k} \sqrt{\frac{P_j}{\mu_j}} \tilde{Q}_j}(t) = \prod_{j=1}^{k} \varphi_{\frac{P_j}{\mu_j} \tilde{Q}_j}(\sqrt{\frac{P_j}{\mu_j}} t) = \prod_{j=1}^{k} \varphi_{\tilde{Q}_j}(\sqrt{\frac{P_j}{\mu_j}} t).$$

(4.4.49)

By Levy’s inversion theorem [34], Eq. (4.4.48) can be written as

$$\alpha \geq P\left(\sum_{j=1}^{k} \sqrt{\frac{P_j}{\mu_j}} \tilde{Q}_j \geq 0\right) \geq \frac{1}{2\pi} \lim_{s \to \infty} \int_{-\infty}^{\infty} \left[\prod_{j=1}^{k} \varphi_{\tilde{Q}_j} \left(\sqrt{\frac{P_j}{\mu_i}} t\right) \frac{1 - \exp(-its)}{it}\right] dt$$

(4.4.50)

Thus, from Eq. (4.4.47) and (4.4.50), the solution of optimization problem (4.4.43-4.4.44) is an upper bound of the limit in Eq. (4.4.3) for the artificial system. Then, for the original system, no more servers are needed under the same value of traffic intensity, i.e., the solution of optimization problem (4.4.43-4.4.44) is also an upper bound of the limit in Eq. (4.4.3) for the original system.
Note that, if we choose any \( \{\alpha_j, j = 1, ..., k\} \) in the feasible set of the optimization problem (4.4.43-4.4.44), then the corresponding objective value is an upper bound for Poisson arrivals. If we choose \( \tilde{\alpha}_j = 1 - (1 - \alpha_j)^{\frac{1}{k}} \), \( \forall j = 1, ..., k \), it is easy to check that the value of \( \{\tilde{\alpha}_j, j = 1, ..., k\} \) is in the feasible set, and the objective value is same as the upper bound in Eq. (4.4.29).

### 4.5 Heavy Traffic Limit for the BWT Class

Proposition 4.5.1 provides conditions under which the waiting time of a job is bounded by a constant \( t_1 \) but the probability that new arrivals need to wait approaches one in the heavy traffic scenario.

**Proposition 4.5.1. Assume**

\[
\lim_{n \to \infty} \delta_n = 0, \tag{4.5.1}
\]

then

\[
\lim_{n \to \infty} \rho_n = 1 \tag{4.5.2}
\]

\[
\lim_{n \to \infty} P\{Q_n \geq n\} = 1 \tag{4.5.3}
\]

\[
P\{W_n > t_1\} \sim \delta_n \tag{4.5.4}
\]

if and only if

\[
\lim_{n \to \infty} \frac{(1 - \rho_n)n}{- \ln \delta_n} = \tau \tag{4.5.5}
\]

\[
\lim_{n \to \infty} \delta_n \exp(k \sqrt{n}) = \infty, \quad \forall k > 0 \tag{4.5.6}
\]
where

\[
\tau = \frac{\mu^2 \sigma^2 + c^2}{2\mu t_1}, \quad \rho_n = \frac{\lambda_n}{n\mu},
\]

\[\mu = \left(\sum_{i=1}^{k} \frac{P_i}{\mu_i}\right)^{-1}, \quad \sigma^2 = 2 \sum_{i=1}^{k} \left(\frac{P_i}{\mu_i^2}\right) - \left(\sum_{i=1}^{k} \frac{P_i}{\mu_i}\right)^2.\]

From Proposition 4.5.1, we can see that, to satisfy the QoS of BWT, \(\frac{(1 - \rho_n)n}{-\ln \delta_n}\) should converge to a constant, and this constant can be computed by the given parameters using this proposition.

\textbf{Proof of Proposition 4.5.1.} To prove Proposition 4.5.1, we must prove both the necessary and sufficient conditions.

\textbf{Necessary Condition:} From the heavy traffic results given by Kingman [35] and Kollerstrom [36, 37], the equilibrium waiting time in our system can be shown to asymptotically follow an exponential distribution, i.e.,

\[P(W_n \geq t_1) \sim \exp\left(\frac{2(\mathbb{E}(v_n) - \frac{\mathbb{E}(s_n)}{n})}{\text{Var}(\frac{s_n}{n}) + \text{Var}(v_n)}t_1\right).\]

(4.5.9)

In Eq. (4.5.9), \(s_n\) is the service time, and \(v_n\) is the inter-arrival time. Assume the mean and variance of service time is \(\mu^{-1}\) and \(\sigma^2\). Then, we get

\[P(W_n \geq t_1) \sim \exp\left(-\frac{2(1 - \frac{1}{n\mu} - \frac{1}{\lambda_n})}{\frac{\sigma^2}{n^2} + \frac{c^2}{\lambda_n}}t_1\right) = \exp\left(-\frac{2\mu(1 - \rho_n)n}{\mu^2\sigma^2 + c^2n^2}t_1\right)\]

(4.5.10)

Since \(c_n = c\) and for this class the equilibrium waiting time satisfies that \(P(W_n \geq t_1) \sim \delta_n\), it implies that

\[\lim_{n \to \infty} \frac{(1 - \rho_n)n}{-\ln \delta_n} = \tau,\]

(4.5.11)
\[ \tau \triangleq \frac{\mu^2 \sigma^2 + c^2}{2 \mu t_1}, \quad \mu = \left( \sum_{i=1}^{k} \frac{P_i}{\mu_i} \right)^{-1}, \quad \sigma^2 = 2 \sum_{i=1}^{k} \left( \frac{P_i}{\mu_i^2} \right) - \left( \sum_{i=1}^{k} \frac{P_i}{\mu_i} \right)^2. \]

Based on Proposition 4.4.1, from \( \lim_{n \to \infty} P\{Q_n \geq n\} = 1 \), we get \( \lim_{n \to \infty} (1 - \rho_n) \sqrt{n} = 0 \), i.e., \( \lim_{n \to \infty} \frac{\ln \frac{1}{\delta_n}}{\sqrt{n}} = 0 \). This means that \( \frac{1}{\delta_n} = o(\sqrt{n}) \). Hence, \( \lim_{n \to \infty} \delta_n \exp(k \sqrt{n}) = \infty, \forall k > 0 \). Thus, Eq. (4.5.6) is achieved.

**Sufficient Condition:** When Eq. (4.5.6) is satisfied, we get \( \frac{1}{\delta_n} = o(n) \), i.e., \( \lim_{n \to \infty} \frac{\ln \frac{1}{\delta_n}}{n} = 0 \), which is equivalent to \( \lim_{n \to \infty} \rho_n = 1 \) based on Eq. (4.5.5). Hence, Eq. (4.5.2) is achieved.

Now, based on Eqs. (4.5.2) and (4.5.6), and using the heavy traffic limit result Eqs. (4.4.3), the lower bound in Proposition 4.4.1 should satisfy that \( L = 0 \). By applying \( L = 0 \) in Eqs. (4.4.2), (4.4.5) and (4.4.8), we can directly obtain \( \lim_{n \to \infty} P\{Q_n \geq n\} = 1 \). Hence, Eq. (4.5.3) is satisfied.

Based on Eq. (4.5.5), it can be readily shown that
\[
\lim_{n \to \infty} \frac{\exp[-n(1 - \rho_n)/\tau]}{\delta_n} = 1. \tag{4.5.12}
\]

Based on Eq. (4.5.10), we get \( \lim_{n \to \infty} \frac{P\{W_n > t_1\}}{\delta_n} = 1 \). That is \( P\{W_n > t_1\} \sim \delta_n \). Eq. (4.5.4) is achieved. ■

**Corollary 4.5.2.** Let \( k = 1 \), then \( \mu_1 = \mu \) and \( P_1 = 1 \). We can directly obtain the scenario with exponentially distributed service time from Proposition 4.5.1. In the case of exponential distributed service time, Proposition 4.5.1 still holds, and \( \tau \) can be simplified to \( \hat{\tau} \triangleq \frac{1 + c^2}{2 \mu t_1} \).

**Corollary 4.5.3.** Comparing the two cases in Proposition 4.5.1 and Corollary 4.5.2, assume that they have the same parameters \( (t_1 \text{ and } \mu) \) and functions \( (\rho_n \text{ and } \delta_n) \), which satisfies Eqs. (4.5.1-4.5.4). Then, as should be expected, the number of servers
needed in the case of hyper-exponential distributed service time \( n \) is larger than the number of servers needed in the case of exponential distributed service time \( \hat{n} \), i.e., \( n \geq \hat{n} \).

**Proof.** Using Eqs. (4.5.7) and (4.5.8), we obtain

\[
\tau = \frac{\mu^2 \sigma^2 + c^2}{2\mu t_1} = \frac{2 \sum_{i=1}^{k} \left( \frac{P_i}{\mu_i^2} \right) + (c^2 - 1) \left( \sum_{i=1}^{k} \frac{P_i}{\mu_i} \right)^2}{2 \left( \sum_{i=1}^{k} \frac{P_i}{\mu_i} \right) t_1}.
\]

(4.5.13)

Based on Jensen’s Inequality, we can get that

\[
\sum_{i=1}^{k} \left( \frac{P_i}{\mu_i^2} \right) \geq \left( \sum_{i=1}^{k} \frac{P_i}{\mu_i} \right)^2.
\]

(4.5.14)

Then,

\[
\tau \geq \frac{(c^2 + 1) \left( \sum_{i=1}^{k} \frac{P_i}{\mu_i} \right)}{2t_1} = \frac{c^2 + 1}{2\mu t_1} = \hat{\tau}.
\]

(4.5.15)

Then, in Eq. (4.5.5), \( \tau \geq \hat{\tau} \). Thus, for same \( \rho_n \) and \( \delta_n \), hyper-exponential distributed service time needs more servers than exponential distributed service time, i.e., \( n \geq \hat{n} \).

Note that Eq. (4.2.2) defines the hyper-exponential service time, we can also get that Eq. (4.5.15) achieves equality if and only if \( k = 1 \), i.e., \( n = \hat{n} \) if and only if \( k = 1 \).

Next, we consider a data center application where our results will be used to provide guidelines on how many machines to keep active when the arriving traffic requires QoS corresponding to one of the four classes mentioned in the Introduction.

### 4.6 Applications in Cloud Computing/Data Centers

The concept of cloud computing can be traced back to the 1960s, when John McCarthy claimed that “computation may someday be organized as a public utility” [38].
In recent years, cloud computing has received increased attention from the industry [39]. Many applications of cloud computing, such as utility computing [40], Web 2.0 [41], Google app engine [42], Amazon web services [43, 44] and Microsoft’s Azure services platform [45], are widely used today. Some future application opportunities are also discussed by Michael Armbrust et al. in [39]. With the rapid growth of cloud based applications, many definitions, concepts, and properties of cloud computing have emerged [39, 46, 47, 48, 49]. Cloud computing is an attractive alternative to the traditional dedicated computing model, since it makes such services available at a lower cost to the end users [39, 50]. In order to provide services at a low cost, the cost of operating the data centers themselves, needs to be kept low. In [51], based on detailed cost analysis of the data center, 30% of the ongoing cost is electrical utility costs, and more than 70% of the ongoing cost is power-related cost which also includes power distribution and cooling costs. Some typical companies, like Google, have already claimed that their annual energy costs exceed their server costs, and the power consumption of Google is 260 million watts [52]. So, power related cost, which is directly dependent on the number of operational machines in the data center, is a significant fraction of the total cost of operating a data center.

In [48], P. McFedries points out that data centers are typically housed in massive buildings and may contain thousands of machines. This claim is consistent with the fact that large data centers today often have thousands of machines [39]. The service system of a data center can be viewed as a queueing system. Based on the stability and efficiency discussions in Section 4.1, we focus on the behavior of a data center in the heavy traffic scenarios. Figure 4.4 shows the basic architecture. Using the new set of heavy traffic limit results developed in Section 4.4 and 4.5, we can obtain the design criteria of power efficient data center [53, 54], which allows for general and independent arrival processes and hyper-exponential distributed service times.
Figure 4.4: Cloud Computing Architecture
4.6.1 Heavy Traffic Limits for Different Classes Summarized

As discussed earlier, it is important that the data center operates stably, which means that the traffic intensity \( \rho_n \) should be less than 1. Further, the data center also needs to work efficiently, which means that the traffic intensity \( \rho_n \) should be as close to 1 as possible and should approach 1 as \( n \to \infty \). The different classes of data centers will result in different heavy traffic limits, and will thus be governed by different design rules for the number of operational machines \( n \) and traffic intensity \( \rho_n \). In this work, we have derived new heavy traffic limits for the BWT and MWT traffic classes. From the known literature [25, 26, 55, 56, 35, 36, 37], one can easily derive the heavy traffic limits for the ZWT and PWT classes. The derivation is also explicitly shown in our technical report [53], and so, here, to save space, we simply state how \( n \) and \( \rho_n \) should scale to satisfy the QoS requirements of various data centers.

**ZWT Class**

For a data center of ZWT Class, using Proposition 4.4.1, we observe that

\[
(1 - \rho_n) \sqrt{n} \to \infty,
\]  

(4.6.1)

from Eqs. (4.4.12)-(4.4.14). If we define \( f(n) \) as \( 1 - \rho_n \), then

\[
\lim_{n \to \infty} f(n) = 0,
\]

\[
\lim_{n \to \infty} f(n) \sqrt{n} = \infty.
\]

(4.6.2)

**MWT Class**

Applying the result of Proposition 4.4.1, we can show that the QoS of a data center of MWT Class can be satisfied if

\[
L \leq \lim_{n \to \infty} (1 - \rho_n) \sqrt{n} \leq U.
\]

(4.6.3)

\( U \) and \( L \) can be computed from Eq. (4.4.4)–Eq. (4.4.8) in Proposition 4.4.1.
BWT Class

We can satisfy the QoS requirement of this class by applying Proposition 4.5.1 to obtain
\[ \lim_{n \to \infty} \frac{(1 - \rho_n)n}{-\ln \delta_n} = \tau, \tag{4.6.4} \]
where \( \tau \) can be computed by Eq. (4.5.7) and Eq. (4.5.8).

For a data center of BWT Class, not all functions \( \delta_n \), which decrease to 0, as \( n \) goes to infinity, can satisfy the condition. An appropriate \( \delta_n \) that can be used to satisfy the QoS of BWT Class should satisfy the condition Eq. (4.5.6) given in Proposition 4.5.1. Then, the waiting time of jobs for BWT Class is between 0 and \( t \) almost surely as \( n \to \infty \).

PWT Class

The QoS requirement of a data center of PWT Class based on Eq. (4.5.10) satisfies
\[ P\{W_n \geq t_2\} \sim e^{-\frac{2n(1-\rho)t_2}{\mu^2+\sigma^2}}. \]

For a data center of PWT Class, to satisfy its QoS requirement, the traffic intensity must scale as
\[ \lim_{n \to \infty} (1 - \rho_n)n = \gamma, \tag{4.6.5} \]
where
\[ \gamma = \frac{-(\mu^2\sigma^2 + c^2)\ln \delta}{2\mu t_2}. \]

Here, \( \mu \) and \( \sigma \) are same as Eq. (4.5.8).

4.6.2 Number of Operational Machines for Different Classes

As discussed in Section 4.1, an important motivation of cloud computing is to maximize the workload that the data center can support and at the same time satisfy the QoS requirements of the users. Based on the heavy traffic limits shown in Sections
4.4 and 4.5, we have different heavy traffic limits for different data center classes (The details of the ZWT and PWT classes are shown in our technical report [53]). Thus, in order for the data center to work efficiently and economically, we need to compute the least number of machines that the data center needs to continue operating for a given QoS requirement.

When $\rho$ is close to 1 and $n$ is large, the heavy traffic limit is a good methodology to approximate the relationship between $\rho$ and $n$. Based on the heavy traffic limits, we list the minimum number of machines that the data center needs to provide under four classes of data centers, as below.

- **The ZWT class:** The $\rho_n$ and $n$ satisfy that $1 - \rho_n \sim f(n)$. Then, the number of operational machines $n$ is $\lceil f^{-1}(1 - \rho) \rceil$.

- **The MWT class:** The $\rho_n$ and $n$ satisfy that $L \leq (1 - \rho_n)\sqrt{n} \leq U$. Then, for the number of optimal machines $n$, the lower bound is $\lceil \left(\frac{L}{1 - \rho} \right)^2 \rceil$, and the upper bound is $\lceil \left(\frac{U}{1 - \rho} \right)^2 \rceil$.

- **The BWT class:** The $\rho_n$ and $n$ satisfy that $(1 - \rho_n)n - \ln \delta_n = \tau$. Then, the number of operational machines $n$ is $\lceil \frac{\tau \ln \delta_n}{\rho - 1} \rceil$.

- **The PWT class:** The $\rho_n$ and $n$ satisfy that $(1 - \rho_n)n = \gamma$. Then, the number of operational machines $n$ is $\lceil \frac{\gamma}{1 - \rho} \rceil$.

Since there are many advanced techniques that can be used to estimate the parameter $\rho$ and this is not the main focus of this chapter, we assume that the parameter $\rho$ can be estimated from the data. The number of machines can then be determined by the QoS requirements and the estimated $\rho$, as shown above.
4.7 Numerical Analysis

4.7.1 Evaluation Setup

We assume that the queueing system, e.g., the data center described in Section 4.6, can accommodate at most $N$ machines. Clearly, to reduce power consumption, we want to keep the number of powered servers to a minimum while at the same time satisfying the corresponding QoS requirements. The parameters for the four classes are as follows:

a. For the ZWT class, we choose $f(n) = n^{-k_1}$, where $k_1 = 0.25$.

b. For the MWT class, we choose the waiting probability $\alpha = 0.005$.

c. For the BWT class, we choose $\delta_n = \exp(-n^{1/4})$, which satisfies Eq. (4.5.6), and $t_1 = 0.5$.

d. For the PWT class, we choose the probability threshold $\delta = 0.1$ and $t_2 = 1$.

4.7.2 Necessity of Class-based Design

We begin with the arrival process being a Poisson process and service time being exponentially distributed with parameter $\mu = 0.3$.

The results characterizing the relationship between the number $n$ of requested machines and the traffic intensity $\rho$ are shown in Fig. 4.5 for $N = 10000$. The figure shows that with a larger pool of machines, not only a large number of jobs, but also a higher intensity of the offered load, can be sustained, which is especially true for systems with more stringent QoS requirements.

From Fig. 4.5, we can also see that the number of machines needed for a given value of $\rho$ is quite different for different QoS classes. Classes with higher QoS require several times more machines than classes with lower QoS under the same traffic intensity $\rho$, 117
which implies that different number of operational machines are necessary for different QoS classes.

In Fig. 4.6, we have a hyper-exponential distribution for the service time, with \( \mu = [1 \ 8 \ 20] \) and \( P = [0.6 \ 0.25 \ 0.15] \). The results characterizing the relationship between the number \( n \) of requested machines and the traffic intensity \( \rho \) are shown in Fig. 4.6 for \( N = 10000 \). The figure is similar to that of the exponential distributed service time case shown in Fig. 4.5. The main difference is that for the analytical result we only have upper and lower bounds for the MWT class in this scenario.

Note that Figs. 4.5 and 4.6 can also be used to find the maximal traffic intensity a data center can support while satisfying a given QoS requirement for a given number of machines in the data center.

Given an arrival rate \( \lambda \), the minimum number of machines needed is equal to \( \frac{\lambda}{\mu} \). However, this is not enough to satisfy the different QoS requirements. For different
Figure 4.6: Operational Number of Machines for Hyper-exponential Distributed Service Time

Figure 4.7: Additional Operational Machines for Exponential Distributed Service Time
QoS requirements, the corresponding number of machines are shown in Figs. 4.7 and 4.8. Figs. 4.7 and 4.8 are under the same scenarios as Figs. 4.5 and 4.6 respectively. From these two figures, we can see that, for the same arrival rate, different classes need different additional number of machines to satisfy different QoS requirements. Similarly, given an arrival rate $\lambda$, the heaviest traffic intensity the system can support under a given QoS requirement is shown in Figs. 4.9 and 4.10.

### 4.7.3 Evaluation for the MWT and BWT Classes

For the MWT class, we also choose the same hyper-exponential distribution of service time distribution as before (i.e., $\mu = [1 8 20]$ and $P = [0.6 0.25 0.15]$). The performance of the MWT class is shown in Fig. 4.11.

Now we define a ratio $r$ to evaluate the tightness of the upper and lower bounds
Figure 4.9: Traffic Intensity for Exponential Distributed Service Time

Figure 4.10: Traffic Intensity for Hyper-exponential Distributed Service Time
Figure 4.11: Simulation results for the queueing systems of the MWT Class (Log Y-Axis)

for the MWT class:

\[ r \triangleq \frac{U}{L} = r_1 r_2, \quad (4.7.1) \]

where

\[ r_1 = \frac{\psi_U}{\psi_L}, \]

\[ r_2 = \left\{ \begin{array}{c}
\sum_{i=1}^{k} \left( 1 + \frac{c^2 - 1}{2} P_i \right) \sqrt{\frac{P_i}{\mu_i}} \\
\right. \left. \max_{i \in \{1, \ldots, k\}} \left( 1 + \frac{c^2 - 1}{2} P_i \right) \sqrt{\frac{P_i}{\mu_i}} \right\}. \quad (4.7.2) \]

For a given \( k \), \( r_1 \) and \( r_2 \) are independent. \( r_2 \) is determined by how the sum \( \sum_{i=1}^{k} \left( 1 + \frac{c^2 - 1}{2} P_i \right) \sqrt{\frac{P_i}{\mu_i}} \) dominates the largest term \( \left( 1 + \frac{c^2 - 1}{2} P_i \right) \sqrt{\frac{P_i}{\mu_i}} \) max \( i \in \{1, \ldots, k\} \) \( \left( 1 + \frac{c^2 - 1}{2} P_i \right) \sqrt{\frac{P_i}{\mu_i}} \). Its domain is interval \([1, k]\). \( r_1 \) is determined by parameters \( k \) and \( \alpha \), and is independent of \( P \) and \( \mu \). For different values of \( k \) and \( \alpha \), the corresponding ratio \( r_1 \) is shown in Fig. 4.12. From Fig. 4.12, we see that \( r_1 \) is typically a small constant, even when \( \alpha \) and \( k \) are large (e.g., if \( \alpha = 0.15 \) and \( k = 20 \), then \( r_1 < 2 \)).
The performance of the BWT class is shown in Fig. 4.13, as one can observe, the simulation and the heavy traffic limits match well across a range of values of the traffic intensity $\rho$.

We next consider non-Poisson arrival processes. Hence we select a 2-state Erlang distribution and deterministic distribution as examples. The simulation results for the MWT and BWT classes are shown in Figs. 4.14 and 4.15, respectively.

We have used the heavy traffic limit results to design the data center for finite values of $n$ in Figs. 4.11, 4.13, 4.14 and 4.15.

From these figures, we observe that the simulation results closely follow the results obtained from the heavy traffic analysis even when the number of machines is not very large (e.g., only 100) and traffic is not very heavy (e.g., $\rho = 0.85$).
Figure 4.13: Simulation results for the queueing systems of the BWT Class

Figure 4.14: Simulation Results for the MWT class with Other Inter-Arrival Processes (Log Y-Axis)
4.8 Conclusion

In this chapter, we have studied new heavy traffic limits for GI/H/n queues that depend on practical QoS requirements. This work significantly enhances the state-of-the-art over previously known heavy traffic results on very specific Hyper-exponential service time distributions developed in [23]. The approach is to develop artificial counterparts to the original queueing systems, whereby we are able to obtain upper and lower bounds on the number of servers needed in the heavy traffic limits to satisfy the QoS requirements. For the MWT class of traffic, we characterize the gap between the lower and upper bound, while for the BWT type of traffic class we show that the upper and lower bounds coincide. These results can be used to derive rules of thumb for computing the number of operational machines in a variety of different applications involving large-scale queueing systems, such as data centers and call centers. In the context of the data center application, we provide numerical results to illustrate the accuracy of the results.
CHAPTER 5

FORGET THE DEADLINE: SCHEDULING
INTERACTIVE APPLICATIONS IN DATA CENTERS

5.1 Introduction

Most interactive services such as web-search, social networks, online gaming, and financial services now are heavily dependent on computations at data centers because their demands for computing resource are both high and variable. Interactive services are time-sensitive as users expect to receive a complete or possibly partial response within a short period of time. The exact nature of the quality of the response and the allowed time-limit for completing a job may vary across individual users and applications, but interactive jobs typically have the following characteristics:

a. **Partial Execution:** Although fully completed jobs are preferred, partially completed jobs are also acceptable to the end user. For example, in a web search, users mostly care about the top few search results which can be obtained without completing the full job.

b. **Time-Sensitive:** Each job has its corresponding strict deadline, whose exact value may be difficult to determine a priori. For example, in a web search, users may not want to wait beyond a few seconds.
c. **Heterogeneity:** Various characteristics of the jobs such as their utility functions and their deadlines may all be heterogeneous. Such heterogeneity is inherent in data centers as they support a variety of users (e.g., VIP customers vs. free users) and applications.

d. **Concavity:** Based on the law of diminishing marginal utility (Gossen’s laws), most utility functions are concave. Studies on the Bing search engine [57, 58] have mapped this utility function and it has been found to be concave on average. We have also made similar observations in our own experiments (see Section 5.6). However, the utility functions are not necessarily concave for all the jobs.

The comparison between the state-of-the-art schedulers and our proposed ISPEED (Interactive Services with Partial ExEcution and Deadlines) scheduler [59] is shown in Table 5.1. In this table, “Yes” means that related issues are considered in the literature, although the exact definitions and approaches could be different from this work. More detailed discussion about the related work is provided in Section 5.8.

In this chapter, we study the problem of scheduling interactive jobs in a data center with the goal of maximizing the total utility of all jobs. This problem is particularly challenging because future arrivals of jobs and their requirements are unknown, which renders it difficult to make the right scheduling decisions for jobs currently in the system. We assume that service preemption is allowed, and summarize our contributions as follows:

- First, we propose a deadline agnostic scheduler, called ISPEED (Section 5.2), and prove that ISPEED maximizes the total utility when the jobs have homogeneous deadlines and their utility functions are non-decreasing and concave (Section 5.3).
• Second, when the jobs have heterogeneous deadlines, we prove that IPSEED achieves a competitive ratio of $2 + \alpha$, where $\alpha$ is a shape parameter for a large class of non-decreasing utility functions. In the special case of $\alpha = 0$, i.e., the utility functions are non-decreasing and concave, the competitive ratio of ISPEED can be improved to 2 (Section 5.4).

• We discuss various challenges in practical implementations, and show the robustness of our proposed solution in dealing with these issues (Section 5.5). Finally, we conduct extensive trace-driven simulations, and show that ISPEED outperforms three other widely used schedulers (Section 5.6).

The design of our solution is grounded on a rigorous mathematical foundation. Since our solution does not require the knowledge of deadlines, it is suitable for real data centers. Indeed, in practice it is non-trivial to obtain the exact information about individual user deadlines. The performance gap between our solution and the
best causal scheduler is small in terms of the competitive ratio (2 vs. $\frac{\sqrt{5} + 1}{2}$), which shows that there is a limited space for further improvement upon our solution. We also show how this basic solution based on theoretical foundations can be enhanced to work well in real settings when a number of practical issues need to be addressed.

The remainder of the chapter is organized as follows. In Section 5.2, we describe the system model that we consider for our analytical results, and propose our ISPEED scheduler. In Sections 5.3 and 5.4, we analyze the performance of ISPEED in the scenarios where the job deadlines are homogeneous and heterogeneous, respectively. Performance evaluations based on simulations are provided in Section 5.6. Finally, we discuss the related work in Section 5.8, and conclude this chapter in Section 5.7.

## 5.2 System Model and ISPEED Scheduler

In this section, we describe the system model that we consider for our analytical results, and propose our scheduler ISPEED. In Section 5.5, we will discuss various practical issues and how our proposed solution can be adapted to deal with these issues.

### 5.2.1 System Model

Consider a data center with $N$ machines. Let there be $n$ jobs arriving into the system within a specific period of time. Let $W_i$ denote the total workload of job $i$, and let $U_i(x)$ denote the utility function for job $i$, where the argument $x$ denotes the amount of completed workload for job $i$. We assume that each machine can process one unit of workload in each time slot. We define the marginal utility gain as the increase in the total utility when one additional unit of workload is served. In Fig. 5.1, we provide an example of the utility function and the marginal utility gain for each workload. Let
\( w_{i,t} \) denote the amount of workload scheduled for job \( i \) in time slot \( t \). Clearly, \( w_{i,t} \) is also the number of machines allocated to job \( i \). Therefore, we must have \( \sum_{i=1}^{n} w_{i,t} \leq N \) and \( w_{i,t} \geq 0 \) for all \( t \). For each job \( i \), there is an associated arrival time \( a_i \) and a deadline \( d_i \). The workload of the job cannot be scheduled after the deadline. In this chapter, we focus on the preemptive scenario, where a job that is being served can be interrupted by other jobs, and its service can be resumed later.

Let \( w_i \triangleq \sum_{t=a_i}^{d_i} w_{i,t} \) be the amount of workload that job \( i \) completes before its deadline. Clearly, we have \( w_i \leq W_i \) for all \( i \in \{1,2,\ldots,n\} \). Then, the utility maximization problem we consider in this chapter can be formulated as follows.

\[
\begin{align*}
\max_{w_{i,t}} & \quad \sum_{i=1}^{n} U_i (w_i) \\
\text{s.t.} & \quad \sum_{i=1}^{n} w_{i,t} \leq N, \ w_{i,t} \geq 0, \ \forall t, \\
& \quad \sum_{t=a_i}^{d_i} w_{i,t} = w_i \leq W_i, \ \forall i \in \{1,\ldots,n\} \\
& \quad w_{i,t} = 0 \text{ for } t < a_i \text{ and } t > d_i, \ \forall i \in \{1,\ldots,n\}.
\end{align*}
\]

In the next section we consider a simplified version of the above model in which jobs are constrained by homogeneous deadlines, i.e., all the deadlines are the same. In Section 5.4, we study the above problem (with heterogeneous deadlines for each job).

### 5.2.2 ISPEED scheduler

We present a deadline agnostic scheduler, called **ISPEED Scheduler**: In each time-slot, repeatedly schedule the first unit of workload from the list of waiting jobs that have the highest marginal utility gain in this time slot, until there is no machine available to schedule, or no waiting job. Observe that multiple units of workload of
a single job can be scheduled within a time slot. Upon arrival, a job is added to the queue of waiting jobs and is deleted when it is completely served or its deadline expires. The detailed operations of ISPEED are described in Algorithm 2.

5.3 Performance Analysis of ISPEED with Homogeneous Deadlines

We show that ISPEED is an optimal solution in the scenario when the utility functions are non-decreasing and concave, and there are homogeneous deadlines, i.e., deadlines of all jobs are same. Note that if there is no deadline for jobs, but a stopping time for the system, then it is equivalent to homogeneous deadlines for all the jobs. Later in the next section, we show that ISPEED can achieve a competitive ratio of $2 + \alpha$ for non-decreasing utility functions while no other causal scheduler can achieve a competitive ratio less than $\frac{\sqrt{5} + 1}{2}$. Also note that our proposed ISPEED scheduler
Algorithm 2 Interactive Services with Partial ExEcution and Deadlines (ISPEED)

**Require:** List of unfinished jobs (including new arrivals in the current time slot) $J$ whose deadlines have not expired in each time slot.

**Ensure:** Scheduled machines for jobs, and updated remaining jobs list $J$ in each time slot.

1: Sort $J$ in a non-increasing order of the marginal utility gains;
2: $d \leftarrow N; // d$ is the number of available machines; $N$ is the total number of machines.
3: **while** $J$ is not empty and $d \neq 0 /*$ There are available machines **do**
4: $i^*$ is the first job in $J$;
5: Allocate a machine to the first unit of workload of job $i^*$;
6: Delete the allocated unit of workload from $i^*$;
7: **if** All the workload of Job $i^*$ is scheduled **then**
8: Delete Job $i^*$ from $J$;
9: **else**
10: Update Job $i^*$ in $J$ corresponding to its new marginal utility gain;
11: **end if**
12: $d \leftarrow d - 1$;
13: **end while**
14: **for** Each Job $i$ in $J$ **do**
15: **if** Job $i$ will expire by the end of this time slot **then**
16: Delete Job $i$ from $J$;
17: **end if**
18: **end for**
is deadline agnostic, i.e., it makes scheduling decisions without having the information of job deadlines.

For a schedule $S$, let $S(t)$ be the schedule in time slot $t$ (a list of units of workload), and let $S(t, p)$ be the unit of workload in time slot $t$ and position $p$ within the list.

The proof proceeds as follows. We start with an optimal (and feasible) schedule and at each step the schedule is altered to another feasible schedule which matches one more element of the ISPEED schedule without reducing the total utility gain. By repeating the above operations, we end up with a schedule that matches with the ISPEED schedule and has a total utility gain that is no smaller than the total utility gain of the optimal schedule. This implies that ISPEED also produces an optimal solution.

**Theorem 5.3.1.** Under the assumptions that the utility functions are non-decreasing and concave, and that the jobs have homogeneous deadlines, ISPEED achieves the optimal total utility.

*Proof.* Let $O$ be an optimal schedule with total utility $U(O)$ and let $G$ be the schedule computed by ISPEED with total utility $U(G)$. Let $\tau$ be the first time-slot when the two schedules differ. For both $O$ and $G$, within each time-slot we sort the units of workload in a non-decreasing order according to their utility gains, respectively. While going through the two sorted lists in time-slot $\tau$ in both schedules, we arrive at the first position (say $p$) where there is a difference. It implies that the first $p - 1$ positions have the exact same units of workload in both schedules, and at least one of the two schedules ($O$ and $G$) has more than $p - 1$ units. Now, there are three cases:

a. **Both $O(\tau)$ and $G(\tau)$ have more than $p - 1$ units:** Let $o = O(\tau, p)$ and $g = G(\tau, p)$ be the workload scheduled in position $p$ in time slot $\tau$ under schedules $O$ and $G$, respectively. We know that $o \neq g$. Also, it is easy to see that the
utility gain of $g$ is no smaller than that of $o$, by the concavity of the utility functions and the greedy operations of ISPEED. Suppose that there exists a position $(\tau', p')$ beyond $(\tau, p)$ (a later position in slot $\tau$ or a position in a later slot) in which the same unit $g$ is scheduled in $O$. Then, we obtain another schedule $O'$ from $O$ by swapping $o$ and $g$. This swapping operation is feasible due to the following reason: 1) clearly it is feasible to move $o$ to a later position than $(\tau, p)$ in $O'$, because all the jobs have the same deadlines; 2) it is feasible to schedule $g$ in position $(\tau, p)$ in $O'$ as $O$ and $G$ have the same schedule by position $(\tau, p - 1)$ and $g$ is scheduled in position $(\tau, p)$ in $G$. As a result, $O'$ matches one more position with $G$ than $O$ does, while keeping the total utility unchanged.

Alternately, if $g$ is not scheduled in $O$ beyond position $(\tau, p)$, then we obtain $O'$ from $O$ by replacing the unit in position $(\tau, p)$ with $g$. This operation is feasible as $O$ and $G$ have the same schedule by position $(\tau, p - 1)$ and $g$ is scheduled in position $(\tau, p)$ in $G$ (by ISPEED). Further, this operation does not reduce the total utility as the utility of $g$ is no smaller than that of $o$. So $O'$ will have a total utility no smaller than that of $O$ and will match with $G$ in one more position.

b. $O(\tau)$ has $p - 1$ units and $G(\tau)$ has more than $p - 1$ units: Consider job $g = G(\tau, p)$. The number of machines in the system is at least $p$. We obtain $O'$ from $O$ by scheduling unit $g$ in position $p$, i.e., $O'(\tau, p) = g$. If $g$ is not scheduled in slots after $\tau$ in $O$, then this operation does not reduce the total utility of $O$. Hence, we have $U(O') \geq U(O)$.

If $g$ is scheduled in a slot after $\tau$ in $O$, say $g = O(\tau', p')$, then we obtain $O'$ from $O$ by switching unit $g$ from position $(\tau', p')$ to position $(\tau, p)$. This operation
is feasible as $O$ and $G$ have the same schedule by position $(\tau, p - 1)$ and $g$ is scheduled in position $(\tau, p)$ in $G$ (by ISPEED). It is also easy to see that this operation keeps the total utility unchanged, i.e., $U(O') = U(O)$, and that the new schedule $O'$ matches with $G$ in one more position.

c. $O(\tau)$ has more than $p - 1$ units and $G(\tau)$ has $p - 1$ units: We want to show that this case does not occur. Consider the workload $o = O(\tau, p)$. Then, $o$ must also be available for scheduling under ISPEED in time slot $\tau$, and thus in position $(\tau, p)$ ISPEED must schedule workload $o$ or another workload with a larger marginal utility gain than $o$. This contradicts with the assumption that $G(\tau)$ has $p - 1$ units, which further implies that case 3) does not occur.

By repeating the above steps, we reach a final schedule that exactly matches with $G$. Since the total utility does not decrease at any step, we end up with a schedule whose total utility is no smaller than that of $O$. This implies that $U(G) \geq U(O)$.

Due to the assumed optimality of $O$, we have $U(G) = U(O)$, i.e., $G$ is also an optimal solution for Eq. (5.2.1) with the assumptions of homogeneous deadlines and non-decreasing and concave utility functions.

\[ \blacksquare \]

5.4 Performance Analysis of ISPEED with Heterogeneous Deadlines

In this section, we consider a more general scenario of heterogeneous job deadlines. We first show that no causal scheduler can achieve a competitive ratio less than $\frac{\sqrt{5} + 1}{2}$. Then, we prove that ISPEED has a competitive ratio of $2 + \alpha$, where $\alpha$ is a shape parameter for a large class of non-decreasing utility functions, and this competitive ratio can be improved to 2 when the utility functions are non-decreasing and concave.
We now give the definition of the competitive ratio, which is quite standard. A scheduler $S$ has a competitive ratio $c$, if for any arrival pattern, the total utility $U(S)$ of $S$ satisfies that

$$\frac{U^*}{U(S)} \leq c,$$  \hspace{1cm} (5.4.1)

where $U^*$ is the maximal total utility over all possible schedulers (including non-causal schedulers) for the same sample path.

5.4.1 Lower Bound on the Competitive Ratio for All Schedulers: $\sqrt{\frac{5 + 1}{2}}$

It has been shown in [67] that the lower bound on the competitive ratio for all causal schedulers is $\sqrt{\frac{5 + 1}{2}} \approx 1.618$. The main idea of the proof is as follows. For any given causal scheduler, one can always construct a special job arrival pattern such that the scheduler cannot achieve a total utility that is larger than $\frac{\sqrt{5} - 1}{2}$-fraction of the maximal achievable total utility.

To illustrate the main idea to get the lower bound, in the following we present a simple example and show how to obtain a lower bound of $\sqrt{2}$ for the competitive ratio. By applying a similar argument, one can prove a tighter lower bound of $\sqrt{\frac{5 + 1}{2}}$.

Consider Jobs I and II that arrive at a single machine system in time slot 1. Assume that each job has one unit of workload. Also, assume that Job I has a utility of $\sqrt{2} - 1$ and its deadline is time slot 1, and Job II has a utility of 1 and its deadline is time slot 2.

We consider the following two cases.

Case 1): Suppose that the scheduler $S$ schedules Job I in the first time slot. Then, we construct the following future arrival pattern: Job III arrives into the system in time slot 2. Job III also has only 1 unit of workload with a utility of 1, and its deadline is time slot 2. In this case, scheduler $S$ achieves a total utility of $\sqrt{2}$ no matter whether it schedules Job II or Job III in time slot 2. However, if we consider
another scheduler $S'$, which schedules Job II in time slot 1, and schedules Job III in time slot 2, then a total utility of 2 can be achieved.

Case 2): Suppose that the scheduler $S$ schedules Job II in the first time slot. Then, we construct the following future arrival pattern: no job arrives in time slot 2 or later. In this case, the total utility of the scheduler $S$ is 1. However, if we consider another competitive scheduler $S'$, which schedules Job I in time slot 1, and schedules Job II in time slot 2, then a total utility of $\sqrt{2}$ can be achieved.

Combining the two cases, we conclude that for any given deterministic causal scheduler, the competitive ratio is no less than $\sqrt{2}$. This example focuses on how to schedule a job in the first time slot. If the scheduler faces a similar situation in a sequence of consecutive time slots, then the lower bound of competitive ratio can be improved to $\frac{\sqrt{5} + 1}{2}$. More details can be found in [67].

5.4.2 Competitive Ratio of ISPEED

Next, we prove an achievable competitive ratio of ISPEED for a large class of non-decreasing utility functions.

**Theorem 5.4.1.** Assume that the utility functions are non-decreasing for all jobs. Suppose that there exists a constant $\alpha \geq 0$, such that the utility function $U_i$ of any job $i$ satisfies

$$\frac{U_i(m) - U_i(k)}{m - k} \leq (1 + \alpha) \nabla U_i(k), \forall k, \forall m > k,$$

where $\nabla U_i(k) = U_i(k) - U_i(k - 1)$ is the marginal utility gain of the $k^{th}$ unit of workload of job $i$. Then ISPEED has a competitive ratio of $2 + \alpha$.

**Proof.** We can describe a scheduler $S$ as follows. For each time slot $t$, the schedule can be represented as a set $\tilde{U}_t(S) = \{(i, j)\}$, if the $j^{th}$ unit of workload of job $i$ is scheduled in time slot $t$. The marginal utility gain of $(i, j)$ is $U_{i,j}$. Then, the schedule
can be represented by a set $U(S) = \bigcup U_t(S)$. Let $U(G)$ be the representation of the schedule under the same arrival pattern for ISPEED. We make a copy of $U(G)$ and denote it by $\overline{U(G)}$. We want to show that the summed utility gain of both $U(G)$ and $\overline{U(G)}$ times $1 + \alpha$ is greater than or equal to the summed utility gain of $U(S)$ for an arbitrary scheduler $S$. Our proof has two steps.

In the first step, we compare $U(G)$ with $U(S)$, and remove the units of workload with the same index $(i, j)$. In other words, if some units of jobs are scheduled by both scheduler $S$ and ISPEED, we pick them out and include them in a set called $E(S)$. We erase all such elements from both $U(G)$ and $U(S)$. This erasing process is illustrated in Fig. 5.2(a). Since $E(S) \subseteq U(G)$, we get

$$\sum_{(i,j) \in E(S)} U_{i,j} \leq \sum_{(i,j) \in U(G)} U_{i,j}. \quad (5.4.3)$$

In the second step, we consider all the remaining elements in $U(S)$ after the
erasing process. Let \( R(S) \) denote the set of the remaining elements, and we have \( R(S) = U(S) \setminus E(S) \). For the purpose of illustration, we provide an example of set \( E(S) \) and \( R(S) \) in Fig. 5.2. Due to the way we construct set \( R(S) \), we have \( R(S) \cap \overline{U(G)} = \emptyset \).

To compare \( R(S) \) with \( \overline{U(G)} \), we first show that the scheduled number of units in \( \overline{U(G)} \) is no less than that in \( R(S) \) in each time slot. Then, we group all the units scheduled in \( R(S) \) into disjoint pools, each of which corresponds to a different job. For each pool in \( R(S) \), we also construct a corresponding pool in \( \overline{U(G)} \). For each pair of such pools, we show that the marginal utility gain of each unit from the pool in \( \overline{U(G)} \) is no less than that of the first unit from the pool in \( R(S) \). Finally, we obtain our results by summing the utility gain over all the pools for \( U(G) \) and \( R(S) \), respectively, and comparing their summed utility.

First, we want to show that in each time slot, the number of scheduled units in \( \overline{U(G)} \) is no less than that in \( R(S) \). Let \( N_G(t) \) and \( N_R(t) \) be the units of workload in \( \overline{U(G)} \) and \( R(S) \) in time slot \( t \), respectively. If \( N_G(t) = N \), then clearly we have \( N_R(t) \leq N = N_G(t) \). If \( N_G(t) < N \), then we must have \( N_R(t) = 0 \). The reason is as follows. Suppose \( N_R(t) > 0 \). This means that there exists some units of a job, say \( u \), which belongs to \( R(S) \). Then, the first unit of \( u \) in \( R(S) \) must also belong to \( \overline{U(G)} \) because the unit of job \( u \) is feasible for ISPEED to schedule and ISPEED has idle machines (i.e., \( N_G(t) < N \)). However, this contradicts with the fact that \( R(S) \cap \overline{U(G)} = \emptyset \).

Let’s assume that the workload in \( R(S) \) belongs to \( K \) jobs. Among these \( K \) jobs, each job has \( N_1, \ldots, N_K \) units of workload scheduled, respectively. We consider the first scheduled unit of each job in \( R(S) \), whose utility gain is denoted by \( F_1, \ldots, F_K \).
Then, we obtain the following:

\[
\sum_{(i,j)\in R(S)} U_{i,j} = \sum_{k=1}^{K} \text{Utility gain of the } k^{th} \text{ job in } R(S) \\
\leq (1 + \alpha) \sum_{k=1}^{K} N_k F_k,
\]

(5.4.4)

where the inequality is from our assumption of Eq. (5.4.2).

Now, we construct \( K \) disjoint pools of units in \( \overline{U(G)} \) corresponding to the location of jobs in \( R(S) \). In time slot \( t \), if there are some units of job \( k \) that are scheduled in \( R(S) \), then we choose the same number of units in \( \overline{U(G)} \), and put them into the \( k^{th} \) pool of \( \overline{U(G)} \). We provide an example in Fig. 5.2(b), where 2 units and 1 unit of Job 1 are scheduled in \( R(S) \) in time slot 1 and 2, respectively. Then, we arbitrarily choose 2 units and 1 unit in \( \overline{U(G)} \) in time slot 1 and 2, respectively, and put them in the first pool of \( \overline{U(G)} \). Since \( N_G(t) \geq N_R(t) \) holds for any time slot \( t \), it is feasible to construct the \( K \) disjoint pools.

Let \( H_k \) be the minimal marginal utility gain in the \( k^{th} \) pool of \( \overline{U(G)} \). Then, we have \( H_k \geq F_k, \forall k \in 1,2,...K \). This is because 1) ISPEED always chooses the job whose first unscheduled unit has the largest utility gain, and 2) the first unit of each job in \( R(S) \) is feasible but does not belong to \( \overline{U(G)} \). Then, from Eq. (5.4.4) we have

\[
\sum_{(i,j)\in R(S)} U_{i,j} \leq (1 + \alpha) \sum_{k=1}^{K} N_k F_k \\
\leq (1 + \alpha) \sum_{k=1}^{K} N_k H_k \leq (1 + \alpha) \sum_{(i,j)\in \overline{U(G)}} U_{i,j}.
\]

(5.4.5)

By summing Eq. (5.4.3) and Eq. (5.4.5), we obtain

\[
\sum_{(i,j)\in U(S)} U_{i,j} \leq (2 + \alpha) \sum_{(i,j)\in \overline{U(G)}} U_{i,j},
\]

(5.4.6)

which is equivalent to \( U(S) \leq (2 + \alpha) U(G) \).  

\[ \blacksquare \]
In Theorem 5.4.1, $\alpha$ can be viewed as a shape parameter, which is determined by the shape of the utility functions. In particular, we have $\alpha = 0$ when the utility functions are non-decreasing and concave. Then, we can have the following theorem.

**Theorem 5.4.2.** If all the utility functions are non-decreasing and concave, then $2$ is a competitive ratio of ISPEED, and no competitive ratio smaller than $2$ can be achieved by ISPEED.

*Proof.* Theorem 5.4.2 contains two parts: I) ISPEED has a competitive ratio $2$; II) Any constant which is less than $2$ is not a competitive ratio for ISPEED.

Since all the utility functions are non-decreasing and concave, Eq. (5.4.2) is satisfied when $\alpha = 0$. By Theorem 5.4.1, we can directly get that $2$ is a competitive ratio of ISPEED. The first part is proven.

To prove the second part, we only need to give a counter example to show that the competitive ratio cannot be less than $2$. We construct an arrival pattern to show that the competitive ratio of ISPEED cannot be less than $2$. Let us consider two arrivals in a data center with 1 machine. Let Job I arrive in time slot 1 with utility gain $c$ ($0 \leq c < 1$) and deadline the time slot 1. Let Job II also arrive in time slot 1 with utility gain 1 and deadline 2.

It is easy to see that ISPEED will schedule Job II in the first time slot, and no job will be available in the following time slots. Then, the totaly utility of ISPEED is 1. However, consider a scheduler $S$ that schedules Job I in the first time slot, and Job II in the second time slot, then the total utility of $S$ is $1 + c$. Thus, the competitive ratio of ISPEED is no less than $1 + c$. Since $c$ can be arbitrarily close to 1, the competitive ratio of ISPEED is no less than $2$. \qed

141
5.5 Practical Considerations

So far, our discussion has focused on the model that supports partial execution, heterogeneous customers (e.g., VIP vs. free customers), and heterogeneous job deadlines. In this section, we discuss a number of other critical practical implementation issues and how our proposed solution can be adapted to deal with these issues.

5.5.1 Initialization Cost

Some applications need a non-negligible amount of preparation time before a job can begin execution. The preparation time is needed for various activities including building the virtual machine, collecting necessary data from remote machines and waking up physical devices from sleep. During the setup time, the job may yield minimal or zero utility gain. So, the utility function may have a shape as in Fig. 5.3. Since the utility gain in the setup period is small, then the shape parameter $\alpha$ in Section 5.4 will be large, which renders the theoretical performance guarantees shown in Theorem 5.4.1 loose. However, if the initialization time is a small fraction of the total processing time, we still expect our solution to perform well.
5.5.2 Multiple Tasks Per Job

Each job may contain multiple tasks and the utility of the job may increase in a discrete fashion when a task is completed. For example, MapReduce jobs typically have multiple tasks. The utility function may become a step function in such cases (Fig. 5.4). As the step size becomes small, the function is approximately continuous and possibly concave.

5.5.3 Cost of Parallelization and Interruption

When a job is divided into multiple units which are executed in parallel, there is an additional cost of parallelization. In addition, when a job is interrupted, the state needs to be stored and when it is resumed, the state needs to be setup once again. The costs for such actions have not been considered in our model. Alternatively, we investigate the effect of such costs in detail in our simulations (Section 5.6). If the cost of parallelization and interruption is extremely large, then a job may need to run in a non-preemptive fashion.
5.5.4 Robustness to Incomplete Information

For a job arriving to the data center, there are various important information associated with the job, such as the total workload, the deadline, the utility function, and so on. In general, it is very difficult, if not impossible, to obtain these information upon the arrival of the job. However, these information are crucial to making scheduling decisions. Next, we discuss the robustness of our proposed solution when the aforementioned information are not completely known.

- **Workload of each job is unknown**: In some scenarios, the total workload may be unknown or difficult to obtain from the applications. However, our solution does not require knowledge of the total workload. Only the utility gain of the next unit of workload is needed by the algorithm.

- **Deadline of each job is unknown**: In many interactive applications, the deadline is difficult to know in advance. For example, the waiting time before a customer leaves the system depends on several aspects, such as the type of the customer, the importance of the service to the customer, the charges for the service and the claimed quality of service given by the system. For an individual customer, it is difficult to define a deadline for a specific service. Being deadline agnostic, our solution does not need the deadline information of the jobs. It only requires the knowledge of the currently available jobs and their utility gains.

- **Utility Function is unknown**: The exact utility function may be unknown for individual users in many applications. However, the general utility function for a specific class of customers can be achieved via stochastic averaging over a sample pool of customers. We will present our methodology for collecting data from customers to obtain the utility function in Section 5.6.
In summary, our solution only requires the knowledge of the currently available jobs and their utility gains in the next time slot, and is thus inherently robust to lack of the knowledge of various information of the jobs. This is a unique strength of our solution.

5.5.5 System Overloading

When the traffic intensity exceeds the capacity of the data center, some jobs have to be abandoned or partially completed. Allowing partial execution has already been explicitly considered in our model described in Section 5.2. As a result, our proposed solution still works in an overloaded system.

5.6 Simulation Result

5.6.1 Understanding the Nature of Utility Function

Before evaluating the performance of different schedulers, we need to understand the utility function by collecting data from users. We ask 10 users to submit 3 search jobs in the Google search engine. For each such job, the user is required to give a score to show the utility gain of each search result given by Google. Each user gives scores to at least 30 results for each job to indicate the utility gain for this job. The search keywords are chosen by the users. In total we have about 1000 collected scores from the users. Based on these scores, we create the average utility function as shown in Fig. 5.5(a). For the utility functions, several jobs have non-concave utility functions. However, most of the jobs are close to concave functions.

Microsoft Research (MSR) has also investigated the utility function for Microsoft’s Bing search engine in [57, 58], which is shown in Fig. 5.5(b). Both curves are close
to concave, which means that the performance guarantee is tight as $\alpha$ is small. Our evaluation is based on both datasets.

### 5.6.2 Evaluation Setup

Next, we describe the default setting of our experimental evaluations. We consider a data center with $N = 100$ machines. There are 400 jobs arriving into the data center and their arrival time slots are uniformly distributed within the first 20 time slots. Their deadlines are arrival time slot plus an integer which is either 1 (tight) or 100 (loose). The workload of each job is 50 units. There are two types of users, VIP users and free users. For the 20 VIP users, each have 100 times larger utility than a free user, for the same amount of completed workload. To model the cost of interruption and parallelization, we introduce a parameter $\epsilon$ to represent the time cost (whose unit is time slot) of such operations. We assume that $\epsilon$-fraction of the total workload is used to account for the cost of interruption and parallelization. The default value of $\epsilon$ is 0.05. Also, there are several units of workload used to build up the environment for the job before the job can be executed. These units of workloads have zero utility gain. The default initialization cost is 5% of the total workload. Unless
otherwise mentioned, these default parameters are used throughout our evaluation. These parameters are also summarized in Table 5.2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Machines N</td>
<td>100</td>
</tr>
<tr>
<td>Number of Jobs</td>
<td>400</td>
</tr>
<tr>
<td>Number of VIP Jobs</td>
<td>20</td>
</tr>
<tr>
<td>Workload per Job</td>
<td>50</td>
</tr>
<tr>
<td>Tight Deadline from Arrivals</td>
<td>1 time slot</td>
</tr>
<tr>
<td>Loose Deadline from Arrivals</td>
<td>100 time slots</td>
</tr>
<tr>
<td>Cost Parameter $\epsilon$</td>
<td>5%</td>
</tr>
<tr>
<td>Initialization Cost</td>
<td>5%</td>
</tr>
</tbody>
</table>

Table 5.2: Default parameters in our evaluations

We consider three other widely used schedulers and compare their performance with our proposed ISPEED: FIFO (First In First Out), EDF (Earliest Deadline First) and EP (Equal Partitioning, or called Fair Scheduler). The definitions of these schedulers are as follows:

a. **FIFO (First In First Out):** Scheduler schedules the jobs corresponding to their arrival times. The jobs with earlier arrival times have a higher priority to schedule.

b. **EDF (Earliest Deadline First):** Scheduler schedules the jobs corresponding to their deadlines. The jobs with earlier deadlines have a higher priority to schedule.
c. **EP (Equal Partitioning):** Scheduler schedules the jobs with equal opportunity. All the jobs will obtain equal resources, except when the jobs do not have any available units to schedule. The EP Scheduler is a totally fair scheduler.

Different from [57], we assume that the job can be preempted while incurring some additional cost.

### 5.6.3 Homogeneous Deadlines

We let the jobs uniformly arrive to the system from time slot 1 to time slot $T_1$. All the jobs have homogeneous deadlines, which is set to be $T_1$. To keep homogeneity among all the jobs, all users in the homogeneous scenario are free users with the same utility function. Also, to verify the optimality of ISPEED, we ignore the cost of migration and initialization in this part. Then, the performance of schedulers is summarized in Fig. 5.6(a) (based on the collected data from Google search) and Fig. 5.6(b) (based on the data from Bing from MSR).

From these two figures, we observe that the performance of utility gain of ISPEED is consistently better than that of the others in the scenario of homogeneous
job deadlines, without consideration of additional cost. For example, when the deadline is 50 time slots, the performance of ISPEED is 172% better than EDF and FIFO, and is 22% better than EP in Google search engine. From Fig. 5.6, we can see that FIFO and EDF perform similarly, and their total utility gains increase almost linearly as the deadline $T_1$ increases. This is because both FIFO and EDF schedule the jobs one by one and execute the jobs completely if their deadlines have not expired yet. When the deadline increases and hence there is more time available for scheduling the jobs, the number of jobs executed by FIFO and EDF increases and thus the obtained total utility gain both increases proportionally.

We observe that the performance of ISPEED is better than EP, and both of them are much better than FIFO and EDF. This is consistent with the results given in [57]. However, we will show later that the performance of ISPEED is much better than EP, FIFO and EDF in the scenario of heterogeneous deadlines. This shows that ISPEED is more robust to uncertainty in job deadlines.

5.6.4 Heterogeneous Deadlines

We now study the performance of different schedulers under heterogeneous deadlines. For simplicity, we introduce a parameter $T_0$. We let the arrival times of jobs to be uniformly distributed among the first $T_0$ time slots, and let their heterogeneous deadlines to be their arrival times plus a random value, which shows how urgent the job is. We choose two types of jobs: one is very urgent, whose deadline is the very next time slot after the jobs arrive; the other is not urgent, whose deadline is $T_0$ time slots after the jobs arrive. We introduce different types of customers (VIP vs. free users) and choose $T_0$ from 1 to 100. The performance of different schedulers with heterogeneous deadlines is summarized in Fig. 5.7(a) (Google) and Fig. 5.7(b) (Bing). From these two figures, we can see that the performance of ISPEED is better than the
other three schedulers, especially when the deadlines are not extremely tight or loose. For example, when the parameter of deadline $T_0$ is 40 time slots, the performance of ISPEED is 80% better than EP, 195% better than EDF, and 135% better than FIFO in Google search engine.

From Fig. 5.7, we observe that EP performs much worse than ISPEED, especially when there is a large variation in the deadlines among different jobs. This is because when the deadline is very tight for some jobs, and very loose for others, EP will lose the opportunity of scheduling the newly arrived jobs, which may have an urgent deadline but large utility gains. For example, suppose that there is a job with a loose deadline, which arrived at the system before the current slot and has a small marginal utility gain in this slot. Also, there is a new arrived job, whose deadline is very tight. Then serving the latter one will produce more utility gain than serving the former one. EP gives them same weight to share the resource, while ISPEED prefers giving the resource to the jobs with more utility gain, leading to better performance.

The performance of different schedulers with heterogeneous deadlines when all users are free users is shown in Fig. 5.8(a) (Google) and Fig. 5.8(b) (Bing). From these two figures, we can see that the performance of utility gain of ISPEED is better.
than the other three schedulers, especially when the deadlines are not extremely tight or loose. For example, when the parameter of deadline $T_0$ is 40 time slots, the performance of ISPEED is 30% better than EP, 63% better than EDF, and 73% better than FIFO in Google search engine.

5.6.5 Fairness of Different Schedulers

Next, we discuss the fairness issue for different schedulers, by studying the cumulative distribution function (CDF) of the completed percentage of each job. Based on the CDFs for FIFO and EDF schedulers shown in Fig. 5.9, we can see that most of the jobs are in the two extreme points of the CDF curve: 0 or 1. It means that one job is either totally completed, or not scheduled at all. This is because once FIFO or EDF decides to work on a job, it will stick to that job until it is finished or the deadline expires. However, EDF finishes more jobs than FIFO, because EDF gives a higher priority to jobs with tight deadlines. For EP, we can see that about half of the jobs have only completed a small part of the workload, while the other half completes almost all their workload. This is because the jobs with tight deadlines have the
same opportunity to share the resources with the jobs with loose deadlines. In other words, the jobs with small utility have the same chance to share the resources with the jobs with larger utility.

ISPEED prefers jobs with higher utility gains, such as VIP users. Also, eventually the jobs with loose deadlines can obtain more resources when the jobs with tight deadlines are finished or their deadlines expire. ISPEED is much more fair to achieve utility gain than all the others (including EP, which is fair to share the resources) in the heterogeneous scenario.

5.6.6 Different Cost Parameters

We also study the impact of $\epsilon$ (cost parameter) when it changes from 0 (equivalent to no parallelization and interruption cost) to 1. When $\epsilon$ is 1, it means that the entire slot is used to prepare the machine for execution. The total utility gain of schedulers with different $\epsilon$ is shown in Fig. 5.10. Since FIFO and EDF greedily parallelize all the workload of a job, the total utility gain of these two schedulers are very close when $\epsilon$ is going to 1. As $\epsilon$ approaches 1, FIFO and EDF waste almost all of the resources in parallelization and interruption cost, which leads to very small utility.
On the other hand, ISPEED and EP work well in the scenario with higher $\epsilon$. EP gives equal chances to all jobs, which leads to a stable number of allocation of machines for each job. Roughly speaking, the system behaves in a non-preemptive fashion, and the effect of the cost remains low. ISPEED prefers jobs with larger marginal utility gain. From Fig. 5.10, we see that the performance of ISPEED is better than EP in most cases. When $\epsilon$ is close to 1, the performance of ISPEED is very close to EP. Also, the performance of ISPEED is consistently better than FIFO and EDF, for all values of $\epsilon$. For example, when $\epsilon$ is equal to 0.3, the utility gain of ISPEED is 53% better than EP, and 29% better than FIFO and EDF for the Google dataset.

5.6.7 Job Initialization Cost

As we discussed in Section 5.5, some applications may need a non-trivial amount of time to build up the environment (e.g., virtual machine) before the job can begin execution. For such scenarios, an example utility function is shown in Fig. 5.3. We change the percentage of initialization cost (as a percentage of the total workload) from 0% to 99%. Fig. 5.11 shows that the performance of ISPEED is much better than the other schedulers in most cases. As many jobs fail to be completed due to
the deadline, when the initialization cost is high the utility gain is low as there is no utility gain during initialization. When the initialization cost is not high, the performance of schedulers is relatively flat, because the total utility gain does not change much, if the completion percentage of job is not very small, which is true for most jobs.

When the traffic in the system is not heavy, the total utility of each job does not change corresponding to different setup percentage, then the whole utility gain of schedulers should be roughly keep in a constant level. However, when the initialization cost is high, a job can achieve its utility gain only if all of its workload is finished before its deadlines. If the system does not always have idle machines, then most of the jobs can only do partial execution, which will cause the performance of all these schedulers quickly decrease to 0.

5.6.8 Multiple Tasks Per Job

Certain types of jobs, such as MapReduce, have multiple tasks per job. An example utility function for such a scenario is shown in Fig. 5.4. For the evaluation, each
job consists of 10 tasks. Partial execution of task does not lead to increase in the utility. The performance of the schedulers is shown in Fig. 5.12. ISPEED outperforms all the other schedulers. Roughly speaking, the trend and relationship between the schedulers remains the same when the deadline is relatively loose. For example, when the parameter of individual deadline is 60 time slot, ISPEED is 40% better than FIFO, 63% better than EDF, and 35% better than EP in Bing search engine. However, the introduction of tasks makes the utility gain less than the previous evaluations when deadline is tight. This is because many tasks cannot be completed when deadline is tight. Also, since the utility gain can be achieved only after the whole task is finished, the curves are not as smooth as before.

5.7 Conclusion

This chapter has explored the problem of scheduling interactive jobs in data centers which have four distinct properties: partial execution, deadlines, heterogeneity and concavity. We have developed a greedy scheduler called ISPEED, and proven

Figure 5.12: Multiple Tasks
that ISPEED has guaranteed performance in terms of total utility gain, which is characterized by the competitive ratio. Further, we evaluated the performance of ISPEED in various practical scenarios via trace-driven simulations, which show that ISPEED achieves a larger utility gain compared to the state-of-the-art schedulers. For future works, it would be interesting to consider a more general model that potentially accounts for multiple tasks, multiple resources, initialization cost, and parallelization and interruption cost. The key challenge will be to design an efficient and simple scheduler with provable performance guarantees.

5.8 Related Work

In [6, 5, 2, 3, 11], different schedulers for data centers under the MapReduce framework are proposed. In [6], the authors design a scheduler for minimizing the weighted sum of the job completion times, and propose approximation algorithms that work within a factor of 3 of the optimal. Based on the work of [6], the authors in [5] consider the effect of the Shuffle phase between the Map and Reduce tasks. In [2, 3, 68], the authors consider the problem of minimizing total flow time, where no constant competitive ratio can be achieved under some general assumptions. The authors proposed a new metric called efficiency ratio, and under this new metric proposed schedulers with provable performance guarantees. However, these schedulers consider the scenario where the entire workload needs to be completed. Partial execution is not considered in these works.

In [60, 61, 62], the problem is analyzed based on classification of QoS. QoS-adaptive systems are proposed for supporting different QoS levels in [60, 61, 62]. However, they assume homogeneous utility gain in the same QoS level or assume that the utility gain is directly determined by the current allocation regardless of previously allocated workload. Moreover, they do not consider job deadlines.
In [57, 58], both deadlines and concavity of the utility functions are studied. However, their analytical results are based on the assumption of homogeneity and strict concavity. For example, homogeneity of all the deadlines and utility functions is necessary in the proof of Lemma 1 in [57]. For heterogeneous or non-concave scenarios, no performance guarantee is given.

In [63, 64], different schedulers and corresponding performance guarantees are studied with the partial execution property. However, these works are limited to the (weighted) linear utility functions assumption. In our work, we study both concave and non-concave utility functions. Thus, the linear scenario can be viewed as a subcase of our study.

Authors in [65, 66] propose another mechanism to explore the utility maximization problem, although the original problems in their work are to study how to provide QoS to maximize utility for wireless clients. In their work, the utility is determined by stochastic parameters, e.g., the delivery ratio guarantee with probability 1. Different from the stochastic study, our work focuses on the worst-case guarantee, which is stronger and can be characterized by the competitive ratio.
In this thesis, we developed efficient scheduling algorithms for cloud computing systems. We first investigated the design of efficient schedulers under the MapReduce framework (Chapter 2). We then proposed asymptotically optimal schedulers in large scale cloud computing systems (Chapter 3). Based on different QoS requirements, we investigated how many machines need to be operating in these systems (Chapter 4). For interactive services, which are very popular in cloud computing environments, we developed efficient schedulers to improve the total utility of customers (Chapter 5). We summarize the contributions of this thesis as follows.

6.1 Summary of Contributions

6.1.1 Efficient Schedulers for MapReduce Framework

In Chapter 2, we investigated the scheduling problem in the MapReduce framework. We first showed that no on-line algorithm can achieve a constant competitive ratio in the non-preemptive scenario for the problem of minimizing the total flow time (delay) in MapReduce-based Cloud Computing systems. Then, we proposed a new metric to measure the performance of schedulers, called the efficiency ratio. Under some weak assumptions, we then showed a surprising property that for the flow-time problem any work-conserving scheduler has a constant efficiency ratio in both preemptive and
non-preemptive scenarios. We also presented an online scheduling algorithm called ASRPT (Available Shortest Remaining Processing Time) with a very small (less than 2) efficiency ratio, and showed that it outperforms state-of-the-art schedulers through simulations.

6.1.2 Scheduler Design for Large Scale Cloud Computing Systems

In Chapter 3, we explored the asymptotically optimal schedulers in large scale cloud computing systems. As the first step, we showed that for the flow-time minimization problem any work-conserving scheduler is asymptotically optimal in both preemptive and parallelizable, and non-preemptive and non-parallelizable scenarios. Second, for long-running applications, we studied the relationship between the number of machines $N$ and the total running time $T$. We provided sufficient conditions to guarantee the asymptotic optimality of work-conserving schedulers, when $T$ is a function of $N$. We also verified via simulations that while state-of-the-art work-conserving schedulers have different delay performance for a small number of machines $N$, these differences rapidly vanish as $N$ becomes large. Our results provided the following two surprising properties: First, in a large system, it is not necessary to implement complex schedulers, as long as they honor the work-conserving principle, thus ensuring both high performance and scalability. Second, under appropriate and general assumptions, work-conserving schedulers can guarantee asymptotic optimality under both the *Noah Effect* [4] (a large amount of workload arrives into the system in the preemptive and parallelizable scenario) and *Joseph Effect* [4] (a large number of cumulative running jobs remain in the system in the non-preemptive and non-parallelizable scenario).
6.1.3 Design of Cloud Computing Systems based on QoS

In Chapter 4, we studied the design of cloud computing systems based on QoS. First, we made new contributions to heavy traffic analysis, in that we derived new heavy traffic limits for two important QoS classes for queueing systems with general arrival processes and hyper-exponential service time distribution. Then, using the heavy traffic limits results, we answered the important question for enabling a power efficient data center as an application: How many machines should a data center have to sustain a specific system load and a certain level of QoS, or equivalently how many machines should be kept “awake” at any given time?

6.1.4 Maximizing Utility of Interactive Services in Cloud Computing Systems

In Chapter 5, we focused on the scheduling algorithms of interactive services in cloud computing systems. First, we proposed a deadline agnostic scheduler, called ISPEED, and proved that ISPEED maximizes the total utility when the jobs have homogeneous deadlines and their utility functions are non-decreasing and concave. Second, when the jobs have heterogeneous deadlines, we proved that IPSEED achieves a competitive ratio of $2 + \alpha$, where $\alpha$ is a shape parameter for a large class of non-decreasing utility functions. In the special case of $\alpha = 0$, i.e., the utility functions are non-decreasing and concave, the competitive ratio of ISPEED can be improved to 2. We also discussed various challenges in practical implementations, and showed the robustness of our proposed solution in dealing with these issues. Finally, we conducted extensive trace-driven simulations and show that ISPEED outperforms three other widely used schedulers.
6.2 Future Directions

In this thesis, we discussed how to design efficient scheduling algorithms in cloud computing systems. In the remaining section we show what we learnt from the thesis and present some open problems for future research.

6.2.1 Applications of Efficiency Ratio

In Chapter 2, we focused on the delay minimization problem under the MapReduce paradigm. We firstly did the worst case analysis, and found that there is no finite competitive ratio in the non-preemptive scenario. Then, we introduced a new metric called efficiency ratio, which is a little weaker in terms of the provided guarantee compared to the competitive ratio. The efficiency ratio represents the performance guarantee with probability one instead of the traditional guarantee which holds for all the sample paths. Although the efficiency ratio cannot guarantee all the sample paths, the probability of such corner cases will decrease to zero with increase in the scale of cloud computing systems. In other words, it is very unlikely to meet a sample path beyond the efficiency ratio guarantee in the current data centers, whose scale is very large.

The concept of the efficiency ratio was introduced for the delay minimization problem in the MapReduce framework, but it can be applied to many different scenarios. For example, for the non-preemptive problem, which is similar to the preemptive utility maximization problem in Chapter 5, it is impossible to find a finite competitive ratio for any causal scheduler in the non-preemptive scenario. The proving method is similar to Chapter 2: For any causal scheduler $S$, we only need to pick up a future arrival pattern, such that the causal scheduler $S$ cannot achieve the bound of the competitive ratio $c$. Thus, the efficiency ratio, which represents the performance guarantee with high probability needs to be applied to evaluate whether a scheduler
is efficient or not. Simple schedulers which can schedule the jobs with guaranteed efficiency ratio in the non-preemptive scenarios of Chapter 5, can be designed based on a direct application of the techniques designed for the efficiency ratio.

The non-preemptive scenario of utility maximization in Chapter 5 is just a quick and direct example of applications of efficiency ratio. In fact, for the problems for which we cannot find finite performance guarantee among all sample paths, the efficiency ratio with high probability guarantee is a potential way to analyze them.

### 6.2.2 Algorithm Design with Incomplete Information

In many applications, the controller may not a priori know some information of the jobs and systems. Even when the exact information of each job at each moment can be obtained, the overhead to achieve such information is too large to implement in online algorithms. In Chapter 2, ASRPT only needs the workload information of each job, and such information is necessary for the design of a simple scheduler in large scale cloud computing systems (Chapter 3). Similarly, ISPEED in Chapter 5 can work well with the absence of deadline information. Both of these schedulers have performance guarantees compared to the optima.

When we design algorithms in a system, we need some information as input to such algorithms. However, many systems, especially the cloud computing systems, are rapidly increasing in size. If the amount of requested information in algorithms increase much faster than the scale of the system, then such algorithms may work well in some small systems, but not in the large ones. In other words, some algorithms which worked well previously with small systems, may become slower and slower because of the rapidly increasing amount of overhead. New algorithms that are simple with smaller overhead should be designed for large scale systems.
6.2.3 Joint Design of Scheduling and Networking

Many large cloud-based companies own multiple data centers [69]. There is an imminent need for an efficient scheduling that considers the limit of the local resource and network capacity among multiple data centers. Additionally, many middle-size companies have their own data center. These companies may need to subscribe to some additional resources from the public cloud service provider. It creates a unique scenario of hybrid data center environment that includes both private and public data center. A future research direction is to provide the best quality of service to the jobs considering the local resources available in a data center and network condition among different data centers.

Since many big companies now own more than one data center, it is required to schedule the jobs among these different data centers. In the previous chapters, the scheduling within a single data center is described, and how to design simple and efficient schedulers among multiple data centers is not included. As a special case of multiple data centers scenario, hybrid cloud is a potential solution for many middle size companies [70]. Such companies have a small size private cloud, which is an old solution created before, or a necessary part for some privacy consideration. However, the local resource located in a private cloud is very limited, such as limitations in number of machines, space, and IT management. A potential solution is to use a public cloud computing system to maintain a hybrid data center environment. A potential future research direction is to design simple and efficient schedulers among multiple data centers, including hybrid data centers environment.

The problem among multiple data center is challenging for the following reasons. First, the overhead of Network and I/O cannot be ignored anymore as one data center scenario. For each request, a local but resource-limited data center or a rich resources but remote data center makes a tradeoff between local resource and network
transmission ability. Second, this is a cross-area design problem, as scheduling and networking should be studied at the same time. Otherwise, the performance will degenerate significantly. For example, the large remote data center with rich resource but poor network transmission to it will result in large delays for the jobs for the resource-greedy algorithms.

Similarly, our model in Chapters 2 and 5 has considered homogenous resources. But in practice each job may have different demands for different types of resources such as CPU, memory, storage and also bandwidth. If only one type is the bottleneck then our model applies to that scenario as well. If not, then a different approach is needed to address the scheduling problem. Scheduling in presence of multiple resources has been explored before [71, 72], however these solutions do not directly apply to scheduling interactive applications or in case of a multi-phase framework like MapReduce.

### 6.2.4 Scheduling with Multiple Complex Phases Precedence

In many data center frameworks like MapReduce, there are multiple dependent phases, which need to be processed in a specific precedence order. However, the frameworks implemented in cloud computing systems becomes more complex [5, 45], and we need more complex precedence model to describe the system as compared to the two-phase precedence model used in MapReduce. Also, each job in such a framework may contain multiple phases, which will make the performance guarantee of ISPEED proposed in Chapter 5 loose, because the guarantee depends on the shape parameter of the utility function, which is very large when there are multiple phases in each job. The challenge in the design of multi-phase schedulers is that the first few phases cannot bring large utility gain to the customers, because such phases are preparation or computation of intermediate results, which are used in the last few phases to bring
the real utility to the customers. Then, if the jobs do not get an opportunity to enter to the final few phases, then all the previous processing is a waste of resources, because they only bring negligible utility to the customers. A potential future direction is to design simple schedulers, which can also work well for a multiple-phase framework, and can guarantee the performance of such schedulers.
BIBLIOGRAPHY


167


