COST PERFORMANCE OF
SCALING APPLICATIONS ON THE CLOUD

Rathnayake Mudiyanselage Sunimal Rathnayake
(B.Sc. Engineering (Honours), University of Moratuwa, Sri Lanka)

A THESIS SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

2020

Supervisor:
Associate Professor Teo Yong Meng

Examiners:
Associate Professor He Bingsheng
Assistant Professor Ma Tianbai, Richard
Professor Cai Wentong, Nanyang Technological University
DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Rathnayake Mudiyanseelage Sunimal Rathnayake
30th October 2020
“Science is nothing but perception.”
- Plato
Acknowledgements

First and foremost I would like to express my sincere gratitude to my thesis advisor Professor Teo Yong Meng. He was always there for me whenever I needed guidance. His critical approach to discussions guided by asking the right questions not only trained me to become a better researcher but also to face life challenges and arrive at better decisions. Professor Teo was also the first teaching mentor who I got to assist as a teaching assistant. His commitment to teaching and his teaching style inspires me to pursue a career in academia. Thank you, Professor Teo for your advice and support throughout.

I would like to take this opportunity to thank my thesis advisory committee, Professor He Bingsheng and Professor Richard Ma for their comments, feedback, and support. A special acknowledgement to Dr. Lavanya Ramapantulu and Dr. Dumitrel Loghin who were my mentors, co-authors and beloved friends from the lab for their immense support.

I was fortunate to gain invaluable teaching experience at NUS under the guidance of Dr. Cristina Carbunaru, Prof. Roland Yap, Dr. Alan Cheng, Prof. Khoo Siau Cheng, Dr. Zhou Lifeng and Prof. Djordje Jevdjic with whom I not only learned teaching and managing skills but also enjoyed the journey with smiles even under pressing deadlines and heavy workloads. I would like to express my gratitude for showing me the ropes of teaching. I would also like to thank all the administration staff at NUS and SoC, especially in the graduate office, for their kind assistance throughout my stay.

I would like to thank all my amazing friends in Singapore who made my life abroad a wonderful and colorful journey full of happiness. Suman, Teodora, Oana, Irvan, Han, Alireza and rest of my lab mates; thank you for being the best lab mates and friends one could ask for. A special appreciation goes to my Sri Lankan friends; Ashan, Dileepa, Manupa, Himeshi, Sandareka, Lihil, Tharindu, Ridwan, Ruchira, Supuli, Gayana and others who I have not mentioned here for all the
philosophical discussions, quarrels, parties, games, and most importantly your friendship. The fond memories we created together will stay with me forever.

Finally, I would like to thank my parents, sister, and family for their continuous love and encouragement despite being thousands of miles apart. There are many more to thank who I have not mentioned here and I sincerely apologize to those who have been overlooked.
# Table of Contents

Acknowledgements vii

Table of Contents ix

Abstract xiii

List of Tables xv

List of Figures xvii

List of Algorithms and Code Listings xx

1 Introduction 1
   1.1 Motivation .................................................. 1
      1.1.1 Cloud Computing ........................................ 1
      1.1.2 Scaling Cloud Resources .............................. 2
      1.1.3 Scaling Cloud Applications ............................ 2
   1.2 Proposed Fixed-Cost Scaling ................................. 4
   1.3 Challenges and Research Questions .......................... 9
   1.4 Approach and Methodology ................................ 11
   1.5 Objective and Contributions ............................... 14
   1.6 Thesis Organization ....................................... 17

2 Related Work 20
   2.1 Resource Scaling ........................................... 20
      2.1.1 On-Premise Resources .................................. 20
      2.1.2 Cloud Resources ...................................... 23
   2.2 Application Scaling ....................................... 26
      2.2.1 Amdahl’s Fixed-Workload Scaling ..................... 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.2 Gustafson’s Fixed-Time Scaling</td>
<td>29</td>
</tr>
<tr>
<td>2.3 Fixed-Cost Application Scaling</td>
<td>33</td>
</tr>
<tr>
<td>2.3.1 Cost-Time</td>
<td>33</td>
</tr>
<tr>
<td>2.3.2 Cost-Accuracy</td>
<td>35</td>
</tr>
<tr>
<td>2.4 Monetary Cost Optimization on Cloud</td>
<td>38</td>
</tr>
<tr>
<td>2.5 Summary</td>
<td>42</td>
</tr>
<tr>
<td>3 Fixed-Cost Scaling with Fixed Workload</td>
<td>44</td>
</tr>
<tr>
<td>3.1 Scaling a Fixed Workload</td>
<td>44</td>
</tr>
<tr>
<td>3.2 Determining Cost-Time Efficient Cloud Configurations</td>
<td>46</td>
</tr>
<tr>
<td>3.3 Approach</td>
<td>48</td>
</tr>
<tr>
<td>3.3.1 Overview</td>
<td>48</td>
</tr>
<tr>
<td>3.3.2 Cloud Configuration</td>
<td>50</td>
</tr>
<tr>
<td>3.3.3 Time Model</td>
<td>50</td>
</tr>
<tr>
<td>3.3.4 Cost Model</td>
<td>51</td>
</tr>
<tr>
<td>3.3.5 Configuration Selection Algorithm</td>
<td>52</td>
</tr>
<tr>
<td>3.4 Evaluation</td>
<td>52</td>
</tr>
<tr>
<td>3.4.1 Resource Demand (Workload) of Applications</td>
<td>53</td>
</tr>
<tr>
<td>3.4.2 Determining Cloud Resource Capacity</td>
<td>57</td>
</tr>
<tr>
<td>3.4.3 Optimizing Resource Characterization</td>
<td>58</td>
</tr>
<tr>
<td>3.4.4 Validation</td>
<td>59</td>
</tr>
<tr>
<td>3.4.5 Configuration Space</td>
<td>60</td>
</tr>
<tr>
<td>3.4.6 Cost of Scaling a Fixed Workload</td>
<td>65</td>
</tr>
<tr>
<td>3.4.7 Cost of Time Deadline</td>
<td>68</td>
</tr>
<tr>
<td>3.5 Summary</td>
<td>69</td>
</tr>
<tr>
<td>4 Scaling Application Problem Size</td>
<td>71</td>
</tr>
<tr>
<td>4.1 Changing Application Workload</td>
<td>71</td>
</tr>
<tr>
<td>4.2 Largest Pareto-Optimal Problem Size</td>
<td>73</td>
</tr>
<tr>
<td>4.3 Approach</td>
<td>75</td>
</tr>
<tr>
<td>4.3.1 Overview</td>
<td>76</td>
</tr>
<tr>
<td>4.3.2 Determining Application Growth Function</td>
<td>79</td>
</tr>
<tr>
<td>4.3.3 Model Derivation</td>
<td>79</td>
</tr>
<tr>
<td>4.3.4 Largest Pareto-optimal Size Algorithm</td>
<td>81</td>
</tr>
<tr>
<td>4.4 Evaluation</td>
<td>81</td>
</tr>
<tr>
<td>4.4.1 Experiment Setup</td>
<td>83</td>
</tr>
</tbody>
</table>
A Experiment Setup and Baseline Execution 159
A.1 Cloud Applications ........................................ 159
A.2 Cloud Resources ............................................. 161
A.3 Baseline Execution ......................................... 162
   A.3.1 Baseline Execution Measurements on Local Server .... 164
   A.3.2 Baseline Execution Measurements on Cloud Resources ... 166

B Knapsack Approach for Determining Largest Problem Size 168
B.1 Determining Largest Problem Size .......................... 168
   B.1.1 Case 1: Choosing a one resource .......................... 169
   B.1.2 Case 2: Choosing a subset of multiple resources ......... 169
B.2 Results ....................................................... 172
   B.2.1 detect on Homogeneous Cloud Resources ............... 172
   B.2.2 n-body and sand on Heterogeneous Cloud Resources . . . 173
Abstract

The inherent scaling capability and pay-per-use charging have led to the growth of cloud computing as a choice of compute resource infrastructure for a wide range of users from large corporations to individuals. While the inherent resource scaling on cloud has been well explored, the cost performance of scaling applications on cloud has received much less attention. Leveraging application scalability on inherently scalable cloud resources opens up new opportunities for cloud consumers to yield the maximum advantage from cloud. This thesis investigates the premise of scaling applications on cloud and its implications on cost performance.

Motivated by Amdahl’s fixed-workload scaling and Gustafson’s fixed-time scaling for high performance computing, we propose fixed-cost scaling for cloud computing and investigate the implications of the proposed fixed-cost law under fixed-workload and fixed-time. Under fixed-cost application scaling, we address three key issues: large cloud resource configuration space, scaling application problem size, and scaling accuracy.

To address the challenge presented by a large cloud resource configuration space, we propose a measurement-driven analytical modeling approach for determining cost-time Pareto-optimal cloud resource configurations. Our approach exposes the existence of multiple Pareto-optimal configurations that meet the application cost budget and time deadline constraints. We investigate the impact of fixed-workload scaling on cloud and discuss the effect of resource configuration on cost and time performance. Our results show that up to 30% cost savings can be achieved with Pareto-optimal resource configuration for our example applications.

Given a fixed-cost budget and a time deadline, we investigate the effect of scaling the problem size of an application. Through a measurement-driven analytical modeling approach, we show that cost-time-problem size Pareto-frontier exists with multiple sweet spots meeting cost budget as well as time deadline constraints. Among the Pareto-optimal problem sizes, we show that there are opportunities for scaling the problem size for relatively smaller increase in cost for applications with sub-linear resource demand growth. For example, the matching threshold of genome sequence alignment application sand could be scaled by
a factor of 1.6 with only 20% cost increase. Moreover, we show the importance of cost-performance characterization in cloud resource selection. To characterize the cost-performance of cloud resources, we introduce a new metric *Performance Cost Ratio* (PCR) and demonstrate the use of PCR to efficiently derive cost-time efficient cloud resource configurations for executing applications on cloud.

In contrast to traditional applications that produce exact results, there exists a class of applications that produce approximate results with a notion of accuracy. For such applications, accuracy can be traded-off for reduced execution cost and time. With a measurement-driven approach, we investigate the cost-time-accuracy performance for executing applications on cloud using *Convolution Neural Networks (CNN)* as an example. In contrast to studies that focus on CNN learning phase that incurs largely a one-off cost, we focus on improving the cost-performance of CNN inference. Without a significant reduction in inference accuracy, our approach focuses on determining multiple sweet-spots where cost and time could be reduced. We show that selecting the right degree of pruning reduces inference cost and time by half with a one-tenth reduction in accuracy. We expose the cost-accuracy and time-accuracy Pareto-optimal configurations spanning large cost and time range where selecting the right configuration results in halving the inference cost and time. To address the challenge of having multiple resource and application configurations that give the same accuracy but with different cost and time, we introduce *Cost Accuracy Ratio* (CAR) and *Time Accuracy Ratio* (TAR) for quantifying the performance of cloud resources with respect to accuracy. Using CAR and TAR to guide our heuristic, we present a polynomial-time algorithm to select cloud resources.
List of Tables

2.1 Amazon EC2 Cloud Resource Types .................................. 34
2.2 Comparison with other model-driven approaches .................. 43

3.1 Model Notations ......................................................... 49
3.2 Applications .......................................................... 53
3.3 Amazon EC2 Cloud Resource Types .................................. 57
3.4 Model Validation ....................................................... 59
3.5 Cloud Resource Configuration Space ................................. 63

4.1 Model Notations .......................................................... 77
4.2 Applications .......................................................... 83
4.3 Amazon EC2 Cloud Resource Types .................................. 84
4.4 Model Validation ....................................................... 87

5.1 Symbols Used .......................................................... 111
5.2 Caffenet Layers ........................................................ 115
5.3 Amazon EC2 Cloud Resource Types .................................. 115

A.1 Amazon EC2 Cloud Resource Types .................................. 162
A.2 Instruction Count - bt, ft and is .................................... 164
A.3 Instruction Count - n-body masses ................................... 164
A.4 Instruction Count - n-body steps .................................... 164
A.5 Instruction Count - sand sequences .................................. 164
A.6 Instruction Count - sand threshold ................................... 165
A.7 Instruction Count - x264 Number of Clips .......................... 165
A.8 Instruction Count - x264 compression factor ....................... 165
A.9 Execution Time - n-body 16384 masses and 1000 steps .......... 166
A.10 Execution Time - sand 16 million sequences and 0.04 threshold .. 166
A.11 Execution Time - bt .................................................... 167
A.12 Execution Time - $ft$ .................................................. 167
A.13 Execution Time - $is$ .................................................. 167

B.1 Dynamic Programming Table ........................................... 171
B.2 Applications ................................................................. 172
B.3 Amazon EC2 Cloud Resource Types ................................. 172
List of Figures

1.1 Proposed Fixed-Cost Scaling with Fixed Workload .................. 6
1.2 Proposed Fixed-Cost Scaling with Fixed Time .................... 7
1.3 General Approach ........................................... 12

2.1 Related work on resource scaling ............................... 21
2.2 Related work on application scaling ............................. 21
2.3 Virtualization on Cloud ....................................... 27
2.4 Changing Number of Masses in n-body Simulation ................. 30
2.5 Changing Number of Steps in n-body Simulation ................. 31
2.6 Resource Capacity of Cloud Resources on Amazon EC2 .......... 32

3.1 Overview of Approach ........................................ 48
3.2 Resource Demand of x264 - n ................................ 54
3.3 Resource Demand of x264 - f ................................ 54
3.4 Resource Demand of n-body - n ................................ 55
3.5 Resource Demand of n-body - s ................................ 55
3.6 Resource Demand of sand - n .................................. 56
3.7 Resource Demand of sand - t .................................. 56
3.8 Cloud Resource Characterization ................................ 58
3.9 Cloud Resource Configuration Space for 6hr Time Deadline and $50 Cost Budget - bt ................................. 60
3.10 Cloud Resource Configuration Space for 6hr Time Deadline and $50 Cost Budget - ft ................................. 61
3.11 Cloud Resource Configuration Space for 6hr Time Deadline and $50 Cost Budget - is ................................. 61
3.12 Cloud Resource Configuration Space for 24hr Time Deadline and $350 Cost Budget - x264 (32000,20) ................. 62
3.13 Cloud Resource Configuration Space for 24hr Time Deadline and $350 Cost Budget - n-body (65536,8000) ........................................... 62
3.14 Cloud Resource Configuration Space for 24hr Time Deadline and $350 Cost Budget - sand (8192 × 10^6,0.32) ................................. 63
3.15 Effect of Scaling a Fixed Workload (Problem Size) on Cost for n-body - masses (n) ................................................................. 65
3.16 Effect of Scaling a Fixed Workload (Problem Size) on Cost for sand - sequences (n) ............................................................... 66
3.17 Effect of Fixed-Workload Scaling (Accuracy) on Cost for n-body - steps (s) ................................................................. 67
3.18 Effect of Fixed-Workload Scaling (Accuracy) on Cost for sand - threshold (τ) ............................................................... 67

4.1 Approach Overview ................................................................. 76
4.2 Performance Cost Ratio of EC2 Resources Types for n-body and sand 84
4.3 Pareto-optimal Size for n-body (64k, s) 24hrs Time Deadline and $100 Cost Budget ................................................................. 88
4.4 Pareto-optimal Size for n-body (n, 4000) 24hrs Time Deadline and $100 Cost Budget ................................................................. 89
4.5 Pareto-optimal Size for sand (n, 0.16) 24hrs Time Deadline and $100 Cost Budget ................................................................. 89
4.6 Pareto-optimal Size for sand (4096m, τ) 24hrs Time Deadline and $100 Cost Budget ................................................................. 90
4.7 Impact of Time Deadline on Largest Size of n-body (64k, s) .... 93
4.8 Impact of Time Deadline on Largest Size of n-body (n, 4000) .... 93
4.9 Impact of Time Deadline on Largest Size of sand (n, 0.16) ....... 94
4.10 Impact of Time Deadline on Largest Size of sand (4096m, τ) .... 94
4.11 Impact of Cost Budget on Largest Size of n-body (64k, s) ....... 96
4.12 Impact of Cost Budget on Largest Size of n-body (n, 4000) ....... 97
4.13 Impact of Cost Budget on Largest Size of sand (n, 0.16) ....... 97
4.14 Impact of Cost Budget on Largest Size of sand (4096m, τ) ....... 98
4.15 Resource Allocation with PCR - n-body (64k, 8000) ............... 99

5.1 Approach for Determining Accuracy Performance .................. 108
5.2 Caffenet CNN Architecture [53] .............................................. 115
5.3 Googlenet CNN Architecture [96] ............................................ 116
5.4 Caffenet Execution Time Distribution of CNN Layers . . . . . . . 117
5.5 Googlenet Execution Time Distribution of CNN Layers . . . . . 118
5.6 Time for a Single Inference . . . . . . . . . . . . . . . . . . . . . . 119
5.7 Parallel Inference on a GPU . . . . . . . . . . . . . . . . . . . . . 120
5.8 Caffenet Effect of Changing Accuracy with Individual Layer Pruning 121
5.9 Googlenet Effect of Changing Accuracy with Individual Layer Pruning 122
5.10 Caffenet Effect of Changing Accuracy with Multi-Layer Pruning . . 125
5.11 Impact of Accuracy on Cloud Execution Time . . . . . . . . . . . . 126
5.12 Impact of Accuracy on Cloud Cost . . . . . . . . . . . . . . . . . 128
5.13 Time-Accuracy of Degrees of Pruning with TAR . . . . . . . . . . 132
5.14 Caffenet CAR Across Resource Types . . . . . . . . . . . . . . . 132

A.1 Baseline Execution . . . . . . . . . . . . . . . . . . . . . . . . . 163

B.1 Growth of Problem Size with Time - detect . . . . . . . . . . . . 173
B.2 Largest Possible Problem Size - n-body . . . . . . . . . . . . . . 174
B.3 Largest Possible Problem Size - sand . . . . . . . . . . . . . . . 174
List of Algorithms and Code Listings

3.1 Resource Configuration Selection .................................................. 52
4.1 Pareto Optimal Size (input: G, C, T, f(S) output: \( S^{\text{max}} \), \( C' \), \( T' \), \( G_{\text{optimal}} \)) .......................................................... 82
5.1 Resource Allocation with TAR and CAR ................................. 133
B.1 Knapsack solution ................................................................. 171
Chapter 1

Introduction

1.1 Motivation

1.1.1 Cloud Computing

Cloud computing has achieved significant progress during the past decade and continues to gain traction. According to Gartner Inc., a leading research and advisory firm on IT and finance, worldwide public cloud service revenue in 2018 was 182 billion dollars and it is forecast to grow to 331 billion dollars by 2022 [31]. Due to its attractive service delivery models and flexible pay-per-use charging, cloud providers are able to cater to a wide spectrum of consumers ranging from small-scale startups to large corporations. Given its potential as a computing platform, tech giants such as Amazon [10], Google [36], Microsoft [68] and Alibaba [4], among others, have joined the cloud computing market. Rapid commercialization and competition among cloud providers have led to a versatile cloud ecosystem and affordable cloud services for consumers. Among the various benefits of cloud computing, scalability is considered as one of the greatest promises. We categorize scalability on cloud as resource scaling and application scaling.
1.1.2 Scaling Cloud Resources

Cloud provides an array of resources and services with different performance and cost with the ability to change the resource configuration dynamically. Thus, a consumer has an opportunity to select the most appropriate cloud resource configuration to adapt to fluctuating resource demands over time. Scaling resources has a direct cost impact on the cloud consumer because different cloud configurations result in different costs.

Traditionally, research on scaling [19, 35, 51, 57, 62, 63, 89, 108] in computer systems research focused on on-premise compute resources. Amdahl [12] proposed the famous fixed-workload speedup law which states that, given a fixed workload, the maximum parallel speedup achievable is limited by the non-parallel portion (sequential fraction) of the workload. Building upon Amdahl’s law, Gustafson [39] demonstrated that when workload scales up with the increase in compute resources - which is a more realistic scenario - a near-linear speedup is achievable. Thus, it is possible to meet a predefined execution time deadline by allocating more compute resources when the sequential fraction grows slower than the increase in the problem size. During the pre-cloud era, adding more compute resources arbitrarily was challenging due to high capital costs. With the advent of cloud, compute resources can be dynamically allocated and deallocated on demand from a large (theoretically unlimited) pool of resources. Moreover, due to pay-per-usage pricing model, the resources are only charged when used thus eliminating the high capital costs.

1.1.3 Scaling Cloud Applications

In contrast to resource scaling, application scaling refers to scaling the application to fit available resources. Often, application scaling comes with a performance
trade-off. For instance, an application’s output accuracy may be lowered to speed-up the computation so as to meet the time deadline and execution cost budget. Compared to resource scaling, application scaling has not received much attention in the cloud context.

When it comes to computer applications, recent advancements in big data, machine learning and high performance computing have resulted in large applications that process a large amount of data. Often, these applications are tightly coupled with special business requirements such as the need to produce outputs in near real-time. Among these applications, there is a set of applications in which the accuracy of the output produced could be traded-off for reduced execution time, for example, machine learning applications. Due to the scalability of resources on cloud and flexibility in adapting to fluctuating application resource demand, executing such scalable applications on cloud opens up new opportunities to trade-off application accuracy for execution time. Since execution time on cloud directly translates into cost, the consumer is able to trade-off accuracy to meet cost budget constrains while obtaining an acceptable output from the application. While the concept of trading off the accuracy of results for execution time and memory performance is a well-explored topic in approximate computing [69], those works focus on applications that run on resource constrained environments such as embedded systems. In contrast, this thesis investigates the fixed-cost scaling of application with changing accuracy in the context of cloud where the focus is on reducing execution cost by leveraging scalability on cloud.
1.2 Proposed Fixed-Cost Scaling

Fixed-Cost Scaling on Cloud

Fixed-cost scaling of applications refers to improving application performance while the cost incurred is kept fixed. Fixed-cost scaling is an entirely novel concept that emerged due to the advent of utility computing model [78]. Before cloud, the costs of application execution were incurred due to indirect expenses such as capital expenditure for computer systems, electricity for running the equipment, and other related costs such as cooling cost in datacenters. With the advent of utility computing model through cloud computing, the cost of executing an application is a reflection of the application performance.

Proposed Fixed-Cost Scaling Law for the Cloud

Gustafson’s law states that the increase of problem size for large machines is near-linear with respect to the number of processors in the machine [39]. Let us revisit this premise in the context of cloud. In contrast to scaling up of resources in one machine as explored by Gustafson, we consider scaling out of resources across cloud which is more apt in the cloud context. Scaling up refers to improving the performance by making a component faster whereas scaling out refers to adding more components to spread out the load. Moreover, for simplicity, let us assume that there is only one type of cloud resources available on the cloud.

Assume that the workload is scaled out from a single compute resource to \( n \) compute resources. Then the scaled out workload \( W' \) is

\[
W' = \alpha W + (1 - \alpha)nW
\]
where $W$ is the original workload processed by a single resource and $\alpha \in [0, 1]$ is the fraction of sequential work. The speedup of $W'$ is

$$S = \frac{\text{execution time on one resource}}{\text{execution time on } n \text{ resources}}$$

(1.2)

$$S = \frac{\alpha W + (1-\alpha)nW}{\alpha W + \frac{(1-\alpha)nW}{n}}$$

(1.3)

Simplifying equation 1.3 we can get

$$S = \alpha + (1 - \alpha)n$$

(1.4)

However, on cloud, $n$ is a function of cloud consumer’s cost budget.

$$n = \frac{C}{\theta}$$

(1.5)

where $C$ is consumer’s cost budget and $\theta$ is the cost using of one cloud resource on the cloud. Therefore $S$ is in fact a function of $C$. From equation 1.4

$$S_C = \alpha + \frac{(1 - \alpha)C}{\theta}$$

(1.6)

According to equation 1.6, for a fixed workload, $S_C$ increases with $C$. At the same time, when the workload changes, if $\alpha$ grows slower than the parallel portion of the workload, we can achieve a near-linear increase in workload while keeping $S_C$ unchanged for a larger $C$.

Now we relax our earlier assumption that there is only one resource type and focus on the realistic setting of a commercial cloud where a number of different cloud resource types are available. Suppose there are $k$ available resource types.
From equation 1.4,

\[ S = \alpha + \beta_1 n_1 + \beta_2 n_2 + \ldots + \beta_i n_i + \ldots + \beta_k n_k \] (1.7)

and

\[ \beta_1 + \beta_2 + \ldots + \beta_i + \ldots + \beta_k = 1 - \alpha \] (1.8)

where \( \beta_i \) denotes the portion of workload assigned for resource type \( i \) and \( n_i \) denotes the number of instances taken from resource type \( i \).

Thus, equation 1.6 becomes

\[ S_C = \alpha + \left\{ \frac{\beta_1 n_1 C_1}{\theta_1} + \frac{\beta_2 n_2 C_2}{\theta_2} + \ldots + \frac{\beta_i n_i C_i}{\theta_i} + \ldots + \frac{\beta_k n_k C_k}{\theta_k} \right\} \] (1.9)

and

\[ C_1 + C_2 + \ldots + C_i + \ldots + C_k = C \] (1.10)

where \( \theta_i \) denotes the cost of using one instance from resource type \( i \).

---

The cost of executing applications on cloud is affected by the cost-performance trade-offs represented by the Pareto frontier in Figure 1.1: Proposed Fixed-Cost Scaling with Fixed Workload.
of the cloud resource configuration. Across different cloud resources, the cost-performance ratio is non-linear. Thus, there are multiple cloud resource configurations to execute the fixed workload within the fixed cost budget. Figure 1.1 illustrates the existence of multiple cloud resource configurations and Pareto-frontier that have different cost and time performance for executing a fixed workload. Given that the cost budget is fixed at $c_1$, and time deadline is fixed at $t_1$, as marked with the darkened areas on the plot, there exist multiple cloud configurations that can execute the workload within $c_1$ and $t_1$.

Secondly, the application’s problem size or the expected accuracy of the results produced by the application affects the resource demand of the application. Thus, changing the problem size and accuracy of the application results in a different cost and time. As shown in Figure 1.2, for a fixed cost budget and a time deadline, there are multiple largest Pareto-optimal problem sizes executable on the cloud. Each of these problem sizes is associated with multiple cloud resource configurations as shown in Figure 1.1.
Chapter 1. Introduction

Pareto-optimality

An allocation is Pareto-optimal if there is no alternative allocation where at least one of the participating elements of the allocation is improved without degrading any other participating element. Thus, a Pareto-optimal configuration implies that all the participating dimensions are reaching the best possible value at the Pareto-optimal configuration. A Pareto frontier is the set of such Pareto-optimal points that yield the best values for one of the participating dimensions.

To investigate the fixed-cost scaling on cloud, in this thesis, we focus on two main directions:

1. Impact of cloud resource configuration on cloud cost.
   
   We explore the large cloud resource configuration space and determine cost time Pareto-optimal cloud resource configurations in Chapter 3. With predictions from our measurement-driven analytical models, we analyze the dynamics of mixing cloud resources and how they impact the execution cost and time on cloud.

2. Impact of application problem size and accuracy on cloud cost
   
   In Chapter 4, We explore the three-dimensional cost-time-problem size space and determine “largest” Pareto-optimal application problem sizes that can be executed on a given set of cloud resources. With the insights we gain from our model predictions, we analyze how far can an application scale in terms of problem size (workload) and how it impacts the execution cost and time. To understand the impact of accuracy on cloud cost, we conduct a measurement-based study on the fixed-cost scaling with changing accuracy on cloud in Chapter 5. We select widely used Convolution Neural Networks (CNN) as the example application and investigate the role of scalable applications towards achieving fixed-cost scaling on cloud.
Chapter 1. Introduction

1.3 Challenges and Research Questions

The new cost-based cloud computing introduces a multitude of research problems and challenges. Cloud consumers face the challenge of utilizing a very large pool of heterogeneous cloud resources and services efficiently while minimizing the cost of computation. Before cloud computing, among the key focuses was building time-efficient [12,39,43,46,66,95] and energy-efficient systems [22,23,60,75,76,77, 100,102]. With cloud computing, achieving cost-efficiency is a new challenge in addition to meeting the execution time deadline.

Today’s commercial cloud providers offer hundreds of different resource types with different cost-performance that are characterized by their compute performance, memory performance, network performance, among others. For example, Amazon’s Elastic Compute Cloud (EC2) offers 167 resource types [88] categorized into general purpose, compute optimized, memory optimized, storage optimized, and accelerated computing categories as of June 2019. From each of these resource types, a consumer may obtain multiple instances. Resource types can be mixed together to form different heterogeneous cloud configurations with different cost-performance giving rise to an extremely large resource configuration space. For example, to choose a configuration to run an application, if we select from nine resource types, each with a maximum of five instances, we have a configuration space of over ten million configurations. The multi-dimensional, multi-level configuration space offered by cloud poses a challenge for cloud consumers to determine the best cost-performance configuration for executing an application and for understanding the cost-time performance of the large configuration space. Investigating fixed-cost application scaling in such a large configuration space poses many research questions:

1. Given an application with a cost budget and an execution time deadline,
what is a cost-efficient cloud resource configuration?

2. For a given cloud application, what is the impact of fixed-cost scaling with a fixed time deadline?

3. Given an application with a cost budget, a time deadline, and a set of cloud resources, what is the largest problem size that could be executed?

4. For a given cloud application, how can cloud resources be characterized with respect to their cost-performance?

The relationship between accuracy and resource demand of application varies from application to application and is non-trivial. Even within a single application, accuracy can scale in different ways with respect to different application parameters. The fixed-cost application scaling with changing accuracy poses a set of research questions:

1. Given an application, what is the impact of fixed-cost scaling with changing accuracy on resource configuration?

2. Given an application with an execution cost budget, what is the highest accuracy that can be achieved on cloud?

3. For a given application and a set of cloud resources, what are the metrics that can be used to evaluate the cost-accuracy performance of different cloud resource configurations?

Answering these research questions help cloud consumers to make decisions on cloud configuration selection for applications with time deadline and cost budget constraints. Moreover, the insights on how application problem size and accuracy scales with cost help cloud consumers yield the best possible outcome from the application while meeting the business deadlines. Furthermore, the metrics
that represent the cost-performance of application execution on cloud will help characterize cloud resources in a way that suits cloud computing in contrast to traditional characterization metrics that do not take cost into account.

This thesis investigates from the user’s perspective the opportunity for scaling applications on public cloud platforms. Thus, we rely on the services, tools, and information made publicly available by commercial cloud platforms. We focus on cloud resources (virtual machines) with on-demand pricing. Moreover, to keep the communication overhead at a minimum, we select resources within one cloud geographical area or cloud region.

1.4 Approach and Methodology

To answer these research questions we propose a general approach that combines analytical modeling and measurements. In this section, we briefly explain the main components of our approach and how they are organized in this thesis.

Inputs

The main inputs to our approach are the application and user constraints. User constraints include the time deadline for completing application execution, cost budget which constraints the cloud resource selection and application accuracy requirements. For cloud resources, we derive cloud resources available, and their pricing information directly from respective cloud vendors.

Baseline Execution

The baseline execution phase serves two main purposes. Firstly, we use baseline measurements to characterize applications and determine their scaling patterns. Secondly, we use it for characterizing cloud resources. Application and resource
Chapter 1. Introduction

Figure 1.3: General Approach
characterization is used as an input to our models. Baseline measurements are recorded on selected cloud resources as well as on-premise resources. Appendix A describes the baseline execution phase in detail.

Models and Measurements

To predict the cost and time for executing applications on cloud, we develop measurement-driven analytical models. As presented in subsequent chapters, our approach uses models for determining the relationship between (i) cost and time, (ii) cost, time, and problem size of cloud applications, and (iii) cost, time, and accuracy of cloud applications. In addition to analytical models, we utilize direct measurements on cloud to determine the impact of scaling applications on cloud.

Outputs

Depending on the type of analysis, our approach outputs cost-time Pareto-efficient cloud configurations, Pareto-optimal problem sizes, and optimal accuracy of applications. Moreover, using the model results and measurements on cloud resources, we derive useful insights that help a cloud consumer to make decisions on selecting cloud resources and setting application parameters while satisfying time deadline and cost budget constraints.

Pareto Optimization

In multi-objective optimization, a Pareto optimal solution is a solution where none of the involved objective functions can be further optimized without degrading some objective functions. In our work, we achieve two-dimensional Pareto optimality between cost and time, and three-dimensional Pareto optimality between cost, time, and problem size. By determining all Pareto optimal solutions,
we expose the Pareto frontier consisting of diverse resource and application configurations.

In summary, we model the cost and time of executing application on cloud under consumer’s cost budget and time deadline constraints with a set of analytical models. To improve the accuracy of analytical model predictions, we incorporate measurements from on-premise and cloud resources into the models. Since our approach is simpler and easily applicable on any public cloud platform, it is better equipped to help the consumer to make scaling decisions than a purely theoretical model.

1.5 Objective and Contributions

Using the above mentioned approach, this thesis investigates the fixed-cost application scaling on cloud and makes the following main contributions.

1. Cost-Time Efficient Cloud Configurations [79]

(a) Measurement-Driven Analytical Models

We propose analytical models for determining executing cost and time efficient cloud resource configurations for a given cloud application on cloud resources with different cost and performance characteristics. Existing approaches that focus on cloud resource optimization focus on execution time. In contrast, we model the cost of executing applications which is more important in the context of cloud. To maintain high accuracy in model prediction, the proposed models take input from baseline execution of cloud application on cloud resources.

(b) Cost-Time Pareto-Optimization

Given the large cloud resource configuration space consisting of millions
of configurations with different cost and performance, it is non-trivial for a cloud consumer to decide on the best cloud resource configuration for a given application. We show the existence of cost-time Pareto frontier that spans a large range of cost and time. These Pareto-optimal configurations are cost-time efficient as they require the minimum cost for a given execution time, and represents the fastest execution time for a given cost budget.

2. Cost-Time Efficient Application Problem Sizes [80]

(a) Measurement-Driven Analytical Models

To understand the relationship between the application problem size, execution cost, and time, we propose analytical models for determining the largest problem size executable on a set of cloud resources constrained by a time deadline and cost budget. Although applications that allow changing the problem size leading to different application demands are commonplace in today’s computer science context, there is no existing research that looks into the impact of problem size on cloud cost and time. Our work fills in this gap and opens up avenues to advance the understanding of the trade-offs involved.

(b) Cost-Time-Size Pareto Optimization

We determine the existence of a three-dimensional Pareto optimization between the cost, time, and the largest problem size executable for a given cloud application. Illustration of this Pareto optimization extends our understanding of the trade-offs related to application problem size on cloud that has not been explored before. Among Pareto-optimal problem sizes, we investigate the impact of scaling cost budget and time deadline. Contrary to intuition, we show that relaxing the time
and cost constraints does not always result in executing a larger problem size.

(c) Performance Cost Ratio (PCR)

To determine near-optimal cloud resource configurations for executing a given application, we propose the usage of PCR metric. PCR characterizes and quantifies the capacity of cloud resources with respect to cost and the application. Thus, allocating resources greedily using PCR as a heuristic, cost efficient cloud resource configurations can be determined.

3. Cost-Time Efficient Application Accuracy [81]

(a) Measurement-Driven Analysis

We present a measurement-driven approach to investigate the relationship between the cost and accuracy of cloud applications. Using Convolution Neural Networks (CNN) as an example, we show that there is an opportunity to reduce the cost of executing cloud applications with minimal loss of accuracy.

(b) Cost-Accuracy “Sweet Spots”

We show the existence of sweet spots among many application configurations where the cost and time of executing the application could be reduced with zero loss in accuracy.

(c) Cost-Accuracy and Time-Accuracy Pareto Optimization

We show the existence of cost-accuracy and time-accuracy Pareto-optimal configurations spanning considerably large time and cost range, and opportunity for reducing cost and time of CNN inference by selecting the right degree of pruning and resource configuration.
(d) Time Accuracy Ratio (TAR) and Cost Accuracy Ratio (CAR)

We introduce TAR and CAR as metrics to characterize the impact of accuracy on time and cost on cloud, respectively. Using CAR and TAR a cloud consumer can differentiate between different application configurations and cloud resource configurations, and select configurations to minimize time and cost with minimal impact to accuracy. Moreover, we show the usage of these metrics as heuristics for efficient cloud resource determination.

1.6 Thesis Organization

This thesis is organized into six chapters as follows.

Chapter 1 provides a general introduction including the motivation and research questions addressed in the thesis. Firstly, we explain the motivation and importance of investigating scaling on cloud. Secondly, we propose fixed-cost scaling on cloud and theoretically derive the fixed-cost law equation. Thirdly, we present the challenges and research questions in the context of fixed-cost scaling followed by our approach for addressing the research questions. Finally, we define our objectives and state our contributions.

Chapter 2 presents a review of related literature categorized into four main sections. Firstly, we review work on resource scaling covering both on-premise and cloud resources. Secondly, we discuss application scaling divided into two subsections as fixed-workload and fixed-time scaling of applications. Thirdly, the current state of fixed-cost application scaling is discussed with emphasis on cost-time and cost-accuracy scaling. Finally, we review some of the approaches to monetary cost optimization on cloud.

Chapter 3 investigates fixed-cost scaling with a fixed workload and presents a
measurement-driven analytical modeling approach for determining Pareto-optimal cloud resource configurations for a given application. We demonstrate the application of our approach for determining cost-efficient cloud resource configuration and discuss the trade-offs between execution cost and time in scaling cloud resources. We derive insights on the implications of the configuration space, time deadline, and cost budget towards scaling resources.

Chapter 4 investigates fixed-cost scaling with a fixed time deadline but changing workload, and analyzes the impact of scaling problem size of an application on the cost, time, and the cloud resource configuration. To estimate the resource demand of applications when scaling, we derive application growth functions for each application in terms of problem size. Through our approach, we demonstrate the three-dimensional Pareto optimization between cost, time and problem size with many Pareto-optimal problem sizes present and discuss the trade-offs. We present insights on the impact of the cost budget and time deadline on the largest Pareto-optimal problem size. To assist with determining near-optimal cloud resource configurations we introduce Performance Cost Ratio (PCR) metric.

Chapter 5 investigates the fixed-cost scaling with changing application accuracy and discusses the cost-accuracy performance on the cloud. Taking Convolution Neural Networks (CNN) inference as an example for applications with variable accuracy, we apply our measurement-driven approach and determine cost and time efficient application accuracy. We explore the impact of the configuration space combined with applications with variable accuracy and derive insights on the trade-offs involved and Pareto-optimization of cost-accuracy and time-accuracy. To determine cost and time efficient application accuracy and resource configurations, we introduce Cost Accuracy Ratio (CAR) and Time Accuracy Ratio (TAR) metrics. Using CAR and TAR to guide the heuristic allows a greedy resource allocation approach with polynomial-time time complexity.
Chapter 1. Introduction

Finally, Chapter 6 concludes the thesis with a summary of contributions and limitations. Several directions for further extending our work are proposed.
Chapter 2

Related Work

We categorize related work into four main sections. Firstly we present work on resource scaling where we discuss on-premise resource scaling that belong to pre-cloud era and cloud resource scaling. Secondly, we present existing work on application scaling divided into two categories as fixed-workload and fixed-time scaling. Thirdly, we discuss the current status of fixed-cost application scaling categorized under cost-time and cost-accuracy scaling. Finally, we discuss the current state of monetary cost optimization on cloud. A high level categorization of main related works are shown in Figures 2.1 and 2.2.

2.1 Resource Scaling

2.1.1 On-Premise Resources

With speculations regarding technology limitations and thermal dissipation constraining the improvement of processor clock frequency [65], computer scientists began research on parallelizing computation across scaled-out resources many decades ago in 1960s. The idea was to partition the workload of an application workload into multiple portions that could be processed in parallel and dispatch
Chapter 2. Related Work

Figure 2.1: Related work on resource scaling

Figure 2.2: Related work on application scaling
multiple such parallel workload portions into independent processing elements that operate simultaneously. This was a promising computing paradigm that fueled the development of advanced parallel and distributing processing technologies we have today. However, due to the nature of computation, oftentimes a given application workload cannot be entirely parallelized. There may be a portion of workload that must be processed sequentially. The impact of scaling out on application performance was first studied by Amdahl [12] in 1967. Amdahl stated, as widely known today as Amdahl’s law, that the maximum improvement in performance of a parallel application is determined and constrained by its non-parallel portion. Amdahl’s law projected a grim picture on the parallel computing paradigm. However, Gustafson [39] in 1988, observed that one of the major assumptions in Amdahl’s law, which assumes a fixed-workload, is not practical and can be relaxed. He explained that in a realistic scenario, the application scales when resources scale out, thus if the growth of the sequential portion of the workload is smaller than the growth of the parallel portion, a near-linear speedup could be achieved. Later in 1990, Sun and Ni [95] extended the speedup model proposed by Amdahl and Gustafson to factor-in memory. Since memory access speeds could not keep up with the exponential improvement in processing speeds, memory became the limiting factor in maximizing parallel performance. Much later in 2011, Hwang [46] developed a uniform framework combining the prior work on modeling parallel performance of systems to evaluate the performance of parallel systems with respect to a given application. Hwang represents the workload of an application in terms of the number of instructions required by the application to execute. To represent the processing capacity of a resource, Hwang’s framework use the instruction execution rate of the application on a given compute resource. We draw inspiration from Hwang’s framework and use the same metrics to represent application workload and capacity of compute resources.
Chapter 2. Related Work

In more recent works such as Hill [43], the focus has been shifted towards improving the parallel performance by balancing the workload between the cores of multicore CPU architectures. Rise of asymmetric and heterogeneous parallel computer architectures has forced the computer scientists to think about better ways of workload distribution and balancing rather than merely dividing a workload into multiple portion and dispatching into parallel processors. Specially with increased popularity of Graphic Processing Units (GPUs), such workload balancing considerations have become even more important [66].

2.1.2 Cloud Resources

Arrival of cloud computing marked an important milestone in parallel computing and fueled the utility computing paradigm. In contrast to traditional on-premise computing paradigm that demands ownership of physical resources, cloud enabled consumers to own computation capability (and compute time) from a wide resource pool without investing for physical resources. According to National Institute of Standards and Technology (NIST) definition on cloud computing [67], cloud possesses five main characteristics; (i) on-demand measured service, (ii) self-service (iii) resource pooling (iv) rapid elasticity and (v) broad network access. On-demand services through a large resource pool with self service capability and the ability to instantly scale up and scale out resources results in a large configurations space. The capability to measure the usage of these services based on their utilization and usage time gives rise to pay-per-use charging. Below we discuss few existing approaches that leverage the ability to scale resources on cloud to improve performance.
Auto-scaling and Resource Migration

Many major cloud providers facilitate the dynamic scale up and scale out resources on their respective cloud platforms. This is achieved through the use of commercially available APIs such as AWS Autoscaling [7]. Autoscaling is a widely explored research domain in cloud computing where many different cloud resource autoscaling techniques have been proposed [72]. We do not investigate all of them in detail in this thesis as they are beyond our scope. Mao et al. [62, 63] proposes leveraging autoscaling for cost optimization for cloud workflows using a performance model to estimate workload demand and resource capacity in terms of number of jobs. This approach relies on the user to estimate job processing times on each type of cloud resource. Kokkinos et al. [51] and Sharma et al. [89] formulate cost optimization as an integer linear programming problem and determine a suitable cloud configuration to migrate an already running application. They utilize performance data retrieved for the current cloud configuration as inputs to the integer linear programming algorithm. Our analytical modeling approach is complementary to autoscaling and resource migration solutions as it could be used to determine the cost-time optimal cloud configuration to execute the application.

Scheduling and Model-driven Resource Optimization

Zhang et al. [57], Kllapi et al. [50] and Zhou et al. [108] propose scheduling frameworks that optimize time and cost of execution for cloud workflows consisting of inter-dependent tasks. Similar to our work, Zhang’s approach considers multiple cloud resource types and selects a combination of resource types as the cloud configuration. Closer to our approach, Huang et al. [44] build an analytical model to predict the performance of resources and to map resources with workload demand estimated using TAU profiler [90]. Since resource demand estimation is
solely based on the profiling on cloud by TAU profiler, the usage of their approach is limited on commercial cloud platforms due to the restrictions imposed due to virtualization. In comparison, our measurement-driven analytical modeling approach uses a combination of time measurements on selected cloud resources and performance counter readings using Linux `perf` utility on a local server to estimate application demand and cloud resource capacity, thus is applicable on commercial clouds. Model-driven resource allocation solution by Bicer et al. [19] divides the workload into equal sized chunks of data or jobs and predicts the time to execute data-intensive jobs concurrently on hybrid cloud environments. A limitation of this method is that when application resource demand is not a uniform function of time, workload execution time estimation would be inaccurate. In contrast to other existing approaches, our application profiling determines the growth function of an application with respect to different application problem size scaling parameters. Determining the growth function helps predict the application resource demand accurately.

**Checkpoints with Spot-Pricing**

Spot-pricing is an innovative pricing strategy first adopted by Amazon EC2 cloud where the cloud consumers are given the opportunity to bid for compute resources. The actual compute resource price fluctuates based on demand [57]. Whenever the actual price drops below the bid price, the consumer is able to run their application. Scheduling execution based on resource pricing is an optimization strategy applied in clouds that support spot pricing such as Amazon EC2. Marathe et al. [64] present an algorithm to find the best checkpointing strategy for executing an application on spot resources such that time and cost of execution are optimized by making use of historical spot pricing data. Due to price fluctuations, executing application on spot resources risks abrupt termination, thus, is difficult to guar-
antee time deadline satisfaction. Fault-tolerant mechanisms could be employed to eliminate termination risks. Gong et al. [35] address this issue by replicating execution on on-demand resources in addition to spot resources to guarantee time deadline satisfaction. However, this approach may exceed cost budget constraints. Predicting the price fluctuations is the largest component in leveraging spot pricing advantage. Novel ideas such as usage of machine learning is being explored to improve the accuracy of spot price predictions [18]. In this thesis, we limit our scope to scaling on on-demand cloud resources.

2.2 Application Scaling

Historically, much of the research focus on scaling in computer systems has been focused towards scaling resources. Intuitively, scaling up or scaling out resources is a straightforward means of improving performance of a given application. However, with the emergence of large scale applications and rise of cloud computing where computing power is charged as a utility, scaling applications on top of scalable resources opens up new opportunities. Especially, for a cloud consumer ability to scale applications provides additional leverage to better optimize their applications to suite business needs while satisfying time deadline and cost budget constraints. In this section, we discuss the different types of application scaling in combination of resource scaling techniques that are relevant to each type of application scaling. We discuss this topic under two categories: (i) fixed-workload and (ii) fixed-time and fixed cost scaling.

2.2.1 Amdahl’s Fixed-Workload Scaling

Fixed-workload scaling is the most straightforward means of scaling performance of an application where the application’s workload is kept intact while changing the
resource allocations. This is in fact the basis for Amdahl’s law. In that sense, fixed-workload scaling of applications is more related to resource scaling. However, for coherence, we discuss fixed-workload scaling as one type of application scaling as well. We discussed the historical background of fixed-workload scaling with regard to Amdahl’s law in section 2.1.1 where we presented the work that exploited fixed-workload scaling on on-premise resources. Thus, in this section, we focus more on fixed-workload scaling in the context of cloud.

**Cloud Configurations**

To understand fixed-workload scaling on cloud, we first need to understand how cloud facilitates changing resource allocation for a given cloud application. A cloud resource is a virtual computing resource running on top of a hypervisor that mediates between the physical resource pool and the virtual cloud resources. Figure 2.3 shows a high level sketch of how physical resources and virtual resources are organized to implement cloud using a hypervisor. The hypervisor is transparent to the cloud user. When a cloud consumer needs to acquire some compute resource, a new virtual machine is created on top of the hypervisor by connecting to one or more processor, memory and storage elements. Hypervisor facilitates mapping multiple virtual machines to the same physical resource. For example, there can be two virtual machines that are mapped to the same physical CPU.
Thus, when both virtual machines are operational, they use the service of physical shared CPU in a time-sharing bases. There are many advantages as well as disadvantages in virtualization, however the discussion on those are beyond the scope of this thesis.

A cloud configuration is an arrangement of one or more cloud virtual machines to facilitate the execution of an application. The performance characteristics of virtual machines in a given cloud configuration may or may not be similar. Today, public cloud providers such as Amazon, Google and Microsoft have made available a multitude of predefined types of cloud virtual machines as well as facilitates for building customized virtual machines as per the wishes of the consumers [17,37,88]. Such flexibility in acquiring multiple different types of virtual machines has given rise to a large configuration space. Since different cloud configurations consist of different virtual-machines with different performance characteristics, their usage cost is different too. For example, Amazon cloud has more than 150 types of cloud resources with the cost per hour varying from $0.025 to $26 [87]. This large variation in cost adds up to the complexity of cloud configuration space.

**Changing Cloud Configurations**

To apply fixed-workload scaling for an application executing on cloud, we need to change the cloud resource configuration assigned for the application. There are a many work proposed leveraging fixed-workload scaling. Most of these work overlap with the work we cited in section 2.1.2 under resource migration. Resource migration involves migrating an already running application from one cloud resource configuration to another cloud configuration. Kokkinos [51] and Sharma et al. [89] propose model driven approaches to predict the application demand based on metrics collected on cloud and migrate the application to an new cloud resource configuration. One of the major limitations in current resource migration work
on cloud is that their migration happens only in one of the following ways; (i) by replicating similar type of cloud resource instances or (ii) moving the application to a resource type which is different in performance. They do not exploit the ability to mix and match heterogeneous cloud resource types together to create new configuration. Moreover, since these approaches rely entirely on execution time feedback retrieved through cloud performance monitoring utilities, it is not clear how to determine the initial cloud configuration to start the application. In this thesis we address the challenge of determining the initial cloud configuration as well as determining the optimum cloud configurations that minimize execution cost and time. Moreover, rather than limiting to a specific virtual machine type (cloud resource type), we mix and match heterogeneous cloud resource types to build complex cloud configurations that better optimize the execution cost and time.

2.2.2 Gustafson’s Fixed-Time Scaling

Fixed-time scaling of applications refers to achieving application performance improvements while keeping the execution time unchanged. Fixed-time scaling is the same phenomenon explained by Gustafson based on the observations made about Amdahl’s law as we discussed in section 2.1.1. Although fixed-time scaling has been explored thoroughly in the context of on-premise resources [39, 43, 46, 66, 95], given the large resource configuration space with different performance that cloud offers it is more applicable for cloud applications.

Fixed-time scaling on cloud is affected by two main factors: (i) change in resource demand of the application with the problem size and (ii) the resource capacity of cloud resource configurations.
Application Resource Demand

The behaviour of resource demand with problem size of an application depends on many factors such as the algorithm, memory access patterns, communication patterns, among others [105]. As a result there are applications with a wide variety of resource demand growth functions. Moreover, if there are multiple ways of scaling the problem size of the application, each of those different ways can result in a different growth function. For example, the classic n-body simulation application has two ways of increasing the problem size (i) the number of masses in the simulation and (ii) the number of simulation steps. As shown in Figure 2.4,

![Graph showing changing number of masses in n-body simulation](image)

Figure 2.4: Changing Number of Masses in n-body Simulation

with the number of masses, the resource demand of n-body simulation application grows exponentially whereas as shown in Figure 2.5, with the number of simulation steps (s), the resource demand of the simulation grows linearly. Determining the application resource demand is not a trivial task. There may be known applications where the resource demand function could be estimated if the computation is known, such as matrix multiplication. However, in most occasions, such estimate
is non-feasible. To overcome this challenge, some work [62, 63] rely on the user to provide input on the resource demand. Such user reliant approaches to estimate the resource demand may not be suitable for cloud as clouds are designed to cater a wide range of consumers from different backgrounds, including non-technical users that may not be aware of the internals of the application. Another approach is to have a feedback system where the application is first deployed in an arbitrary manner until being migrated to a better resource configuration based on the feedback collected during execution [51, 52]. To retrieve application execution information such as utilization information, they use in-cloud tools such as Amazon CloudWatch [9]. In this thesis we rely on static benchmarking of the application for determining the resource demand function of the application.

**Resource Capacity of Cloud Configurations**

As discussed in Section 2.2.1, the heterogeneous pool of resource types clouds offer have different performance, in other words, resource capacities. Figure 2.6 shows a performance comparison between nine resource types belonging to three
resource categories on Amazon EC2. We observe that the performance varies across resource categories. However, despite having different pricing, performance among resource types within a given resource category seem to be similar. It is important to note that the plot is based on instruction execution performance and other performance aspects such as memory or storage may demonstrate a different relationship. Having such cloud resources lead to cloud resource configurations with different cloud capacity with non-linear relationship between them.

Although, cloud is a suitable platform for fixed-time and fixed-cost application scaling, there has been not much focus among research community towards investigating this new opportunity. We identify the contributions made by Han [41] and Guo [38] to be of great value for extending the state-of-the-art of application scaling on the cloud. They investigate a time(or cost)-adaptive set of algorithms called elastic algorithms where the objective is to enable time and cost sensitive
application scaling on cloud. Their formalization of such applications specify a set of key properties that should be held by elastic algorithms; (i) measurable quality, (ii) meaningful results, (iii) quality monotonicity, and (iv) accumulative computation. Their approach is to convert existing algorithms to be adaptable as elastic algorithms or to develop new application from the scratch to comply with their formalization of elastic algorithm. In this thesis, based on our experiments, we show that there exist a wide range of applications already implemented with properties that support application scaling. Thus, our focus is more towards developing techniques and approaches for supporting existing application to scale on cloud rather than re-implementing them from the scratch.

2.3 Fixed-Cost Application Scaling

In this thesis, we are interested in (i) cost-time and (ii) cost-accuracy performance under fixed-cost scaling in the context of cloud.

2.3.1 Cost-Time

Given a single cloud resource type, the cost of executing an application on that particular cloud resource is a linear function of time.

\[ \text{cost} = \text{cost\_per\_unit\_time} \times \text{execution\_time} \]  

(2.1)

However, as we highlighted in section 2.2.1, clouds offer a large variety of cloud resource instances with different cost and performance. Table 2.1 provides a snapshot of cloud pricing for a subset of Amazon EC2 cloud resource instance along with their performance characteristics. As shown in the table, the relationship between the cost and performance is not always linear. When it comes to more
complex cloud resource types such as GPU instances and FPGA instances, the relationship between cost and performance becomes even more complex. This non-linear relationship between cost and performance among cloud resources can be leveraged by a cloud consumer to trade-off execution time for reduction in cost.

In performance engineering terminology, vertical scaling refers to adding more compute power to an existing compute resource and horizontal scaling refers to adding in more compute resources in parallel to the existing resource. Clouds inherently can support both vertical and horizontal scaling [30]. However, in commercial cloud platforms, it is more convenient to scale horizontally rather than vertically as the vendors provide prepackaged set of cloud resource types. Moreover, cloud can support heterogeneous horizontal scaling where resources of different performance specifications are deployed in parallel.

Ciavotta et al. [25], Kokkinos et al. [51] and Sharma et al. [89] formulate the cost optimization as a linear and mixed integer programming problems and solve for an optimal solution that minimizes cost. Ciavotta’s objective is to determine the best configuration for a given application considering the requires service rates of the application and the service capacities of cloud resources. Kokkinos and Sharma determine a suitable cloud configuration to migrate an already running
application. They utilize performance data retrieved for the current cloud configuration as inputs to the integer linear programming algorithm. Ciavotta’s approach considers cloud configurations across multiple cloud providers and determines one configuration that fits best. Kokkinos and Sharma only considers single cloud provider and determines the optimal configuration. In contrast to these work, our objective is to investigate the cost-performance and provide the consumer with a set of choices that satisfy cost budget and time deadline constraints. Instead of returning one optimal configuration, we output a set of Pareto-optimal configurations. Zhang et al. [57], Kllapi et al. [50] and Zhou et al. [108] propose scheduling frameworks that optimize time and cost of execution for cloud workflows consisting of inter-dependent tasks. Similar to our work, Zhang’s approach considers multiple cloud resource types and selects a combination of resource types as the cloud configuration. In contrast, in this thesis we rely on measurement-driven analytical models and Pareto-optimization for determining cloud resource configurations for executing a given application.

2.3.2 Cost-Accuracy

Traditional computer applications usually produce exact results. If the application executes successfully, there would a result, and no result otherwise. However, there are classes of applications that do not demand exact results. For example, a weather prediction application may produce satisfactory output with a notion of probability. For such applications that do not demand exact answers, there exist an opportunity to trade-off the accuracy of the application for gains in application performance. In this section we discuss the cost-accuracy performance of such applications and the work that leverage the cost-accuracy trade-off.

Changing the accuracy of applications in resource constrained environments is a well-explored resource topic. Especially, in the embedded computing domain
where applications are required to deliver results under tight resource constraints (ie. power, storage, memory), trading off the accuracy of applications is a popular approach. Such approaches include changing the precision of variable representation [13, 24, 28, 73, 85, 97], lossy computations and compression [83, 104], data and task sampling [14, 33, 84], among others [69]. However, outside embedded computing domain, changing accuracy of applications have received comparatively much less attention. The primary reason is that the traditional applications operated in on-premise resources with accesses to large processing capacity, memory, and storage. With the advent of machine learning applications, specially CNNs that produce imprecise results associated with an accuracy percentage, trading off accuracy has become relevant to applications outside embedded systems as well. Li et al. [55] and Luo et al [61] proposes pruning to reduce the number of parameters, resulting in sparse matrices. Later in this thesis, we use pruning as a technique to change inference accuracy. Quantization [34, 107] is used to change the length of variables that hold CNN parameters. For example, a parameter data type which is usually represented by 64-bits will be changed to be represented by 32-bits. This has a direct impact on the memory usage of the application. Quantization improves the execution time if there is hardware support for higher speed computations with shorter bit representation. Like Quantization, weight sharing [3] is a technique to cluster parameters in CNNs together based on a “closeness” measure. Multiple parameters that have values close to each other would be reduced to one parameter. This also has a direct impact on the memory and storage usage of the CNN rather than the execution time. In the context of the cloud where the major concern is the cost, we are more interested in the execution time because the execution time directly translates into the cost due to pay-per-use charging. Unlike resource constrained environments such as embedded systems, the cloud has ample storage and memory capacity for a relatively inexpensive price. Thus, in
this thesis, we select pruning as the technique for changing the accuracy of CNNs. When it comes to performance of CNNs, the major focus has been on the training phase of the application [49, 56, 93]. Li et al. [56] present a measurement based analysis on training CNNs. They compare different implementations of popular CNN models on on-premise GPUs and investigate performance bottlenecks. In contrast, we focus on CNN inference and conduct our study on cloud GPU instances. It is important to note that existing studies on CNNs have been focusing on the time-performance of CNN hosted on on-premise systems and we focus on the cost-performance of CNN hosted on cloud resources.

Cloud Context

The concept of accuracy or quality of results in cloud computing has traditionally been interpreted as the Quality-of-Service (QoS). QoS refers to the levels of performance, reliability and availability of cloud services and serves as a business metric within cloud stakeholders. QoS standards are enforced using service level agreements (SLA) that specify economic penalties for QoS violations. Thus, the trade-off between QoS and operational cost has been explored [16]. As QoS usually does not represent the quality or accuracy of the output produced by a cloud application, it is desirable to have a metric that quantifies the quality or accuracy of application output in consumer’s terms and that allows consumer to set execution targets to achieve a desired quality of results. He et al. [42] work on Cost-Quality-of-Experience (QoE) trade-off is conceptually close to our idea of trading-off the accuracy of an application for cost savings. They build a theoretical model to exploit the cost-QoE trade-off for cloud based video content streaming providers. They characterize cloud resources in terms of video streaming performance and cost, and provide guidance for procuring the best resource type to satisfy consumers’ desired QoE. While this approaches the problem from a
cloud provider’s perspective, our work approaches the problem from a consumer’s perspective where the consumer executes an application with time deadline and cost budget constraints.

Han [41] and Guo et al. [38] investigate a class of algorithms they call elastic algorithms where iterative algorithms produce approximate results with monotonic accuracy at each step. These algorithms produce results of different quality depending on how long they execute. They propose to exploit this property to trade off cost of execution for compromises in accuracy. Also, they propose a framework to transform existing traditional algorithms into elastic algorithms.

2.4 Monetary Cost Optimization on Cloud

Monetary cost optimization has been well studied in the context of cloud [18, 32, 44, 48, 51, 58, 62, 63, 64, 98, 103]. Although the main focus of this thesis is not cost optimization, different approaches that have been taken for optimizing cost are worth exploring and can be helpful for better understanding the cost-performance trade-offs we discuss in this thesis. In this section, we discuss few recent work on monetary cost optimization on cloud.

Decomposing cloud applications into smaller tasks or fine grained workflows and executing them in a schedule such that cost is minimized is one of the popular approaches for cost optimization on cloud. To come up with the schedule, such approaches require an estimation of the workload and the resource utilization, which is determined by application profiling or user input. Once the schedule is determined, the resource allocation rules and timings can be setup on the cloud platform. Scheduling is often paired with resource migration [94] where the cloud resource configuration is dynamically changed based on a certain criteria. Public cloud platforms provide autoscaling capability, an automated resource migration
system, to simplify this process whereas the scaling rules according to the schedule can be input through the autoscaling API. For example, Mao et al.’s work [62,63] generates Directed Acyclic Graphs to represent cloud workflows based on user’s input on the workload, and schedule the workflow execution using autoscaling. To pick a resource types, they estimate processing time of tasks.

Formulating the cost optimization problem into a an integer programming problem is another popular approach. In this approach an objective function for cost is formulated considering various parameters such as application requirements, system properties and user constraints. The solution to the integer problem reveals the cloud resource configuration which gives the optimal cost under given constraints. A downside of this approach is that the user is given the optimal resource configuration, but no alternative strategies and their trade offs are provided that may be useful for making an informed business decision. For integer linear optimization, various methods are proposed to retrieve performance data from cloud platform such as benchmarking applications on cloud and using in built performance monitors. Kokkinos et al. [51] present a technique to determine cloud configurations to optimize cost and resource utilization by formulating an integer programming problem. They rely on Amazon CloudWatch API [1] to retrieve performance data form the cloud. Thus, the application needs to be deployed on some cloud configuration for initial performance data to be collected. Lin et al. [58] proposes a mixed integer linear problem formulation for solving the cost optimization problem for geographically distributed cloud virtual machines considering communication costs whereas Mohammadi et al. [70] proposes integer linear optimization for executing scientific workflows with deadline constraints in multi-cloud platforms. Appendix B of this thesis demonstrate the usage of Knapsack optimization, a variant of integer programming, for determining the optimal cloud resource configuration.
Predicting prices of spot instances [6] and deciding on bidding decisions is a well explored strategy in cloud computing. Given the highly dynamic nature of spot prices, careful derivation of a bidding strategy can save cost of acquiring resources. The most common approach used for deriving a bidding strategy is to utilize historical price fluctuation data. To analyze the historical data, methods ranging from simple statistical analysis to deep neural networks are used [18, 32, 44, 48, 64, 103]. One issue with spot instance is that the application is preemptively terminated whenever the resource price rises above the bid price. Such unplanned termination can be catastrophic for businesses. Thus, checkpointing is used as a technique to overcome this issue. The basic idea of checkpointing is that the resources’ price changes are checked at certain check points and a decision is made whether to continue executing the application on the current cloud resource configuration, move into a difference resource configuration or to halt execution based on price fluctuations. Gong et al. [35] and Marathe et al. [64] model the cost optimization problem as a constrained optimization problem to determine cloud configurations in AWS spot market and use checkpoints to control execution. Marathe et al. develop algorithms to decide on bid prices based on historical data and schedule checkpoints at which the system status is checked to take further action on application execution. There is a growing number of research focusing on better leveraging spot instances with minimum overhead [5]. In this thesis, we do not focus on spot instances as they are only available on very limited number of cloud platforms and also due to their fluctuating resource prices which introduces additional complexity on our study about implications of scaling applications on cloud.

Analytical model based prediction for optimizations on cloud has been used as a supporting technique for many works, specially for predicting the workload demand and performance of cloud resources without entirely relying on user in-
put. Oftentimes, these analytical models take into account the benchmark profile data, user constraints as well as other performance metrics provided by in-cloud performance monitoring tools. Huang et al. [44] build an analytical model to predict the performance of resources and to map resources with workload demand estimated using TAU profiler [90]. Since resource demand estimation is solely based on the profiling on cloud by TAU profiler, the usage of their approach is limited on commercial cloud platforms due to the restrictions imposed due to virtualization. In comparison, our measurement-driven analytical modeling approach uses a combination of time measurements on selected cloud resources and performance counter readings using Linux perf utility on a local server to estimate application demand and cloud resource capacity, thus is applicable on commercial clouds. Model-driven resource allocation solution by Bicer et al. [19] divides the workload into equal sized chunks of data or jobs and predicts the time to execute data-intensive jobs concurrently on hybrid cloud environments. A limitation of this method is that when application resource demand is not a uniform function of time, workload execution time estimation would be inaccurate. In contrast, we first profile the application to determine its resource demand function with respect to problem size and accuracy, and thus, provide a better estimation of resource demand, execution time and cost.

Machine learning is gaining traction as a highly effective tool for decision support systems and optimization specially when a large dataset is involved. Even in cloud cost optimization, increasingly many research are trying to leverage machine learning techniques. MLscale [98] proposes an application-agnostic autoscaler that requires minimal application knowledge and manual tuning. They use neural networks for building an online performance model and then use linear regression to predict the post-scaling state of the system. Based on these predictions, scaling decisions are made. Spot pricing bid prediction is a popular application for ma-
chine learning techniques, specially due to the availability of massive historical price fluctuation dataset. Baughman et al. [18] considers spot price as a time series and use a hybrid long/short term memory (LSTM) dense neural network architecture for predicting spot prices in the future. Xu et al. [103] proposes iSpot, a cost-effective transient server provisioning framework for achieving predictable performance in the cloud, by focusing on stream processing as a representative Directed Acyclic Graph style big data analytics workload and predict prices using a Long Short-Term Memory (LSTM) model.

2.5 Summary

Performance scaling is a thoroughly sought after and well explored research area in computer science of which the inception dates back to 1960s. Historically, performance scaling research has been mostly revolving around fixed-workload and fixed-time scaling of cloud resources where on-premise resources were scaled up or scaled out to achieve improvements in application performance. With the advent of new utility computing paradigm, and cloud computing which implements utility computing for common users, new scaling opportunities has emerged to scale compute resources. Clouds offering a theoretically unlimited pool of compute resources with pay-pay-per use charging has elevated the state of resources scaling allowing fixed-workload, fixed-time and fixed-cost scaling of resources.

In contrast to resource scaling, application scaling has been a comparatively less-explored area of research despite early work in the context of application scaling dating back to 1980s. Application scaling is better suited for cloud given the opportunities it opens up for a consumer to optimize application execution by trading-off application performance for cost. This leads to a new era of research where the major focus of performance scaling moving from time-performance to
cost-performance of applications. While combining the advantages of resource scaling on cloud with scalable applications, this thesis investigates the fixed-workload, fixed-time and fixed-cost scaling opportunities, and, cost-time and cost-accuracy performance on cloud using a measurement-driven analytical modeling approach. A summarized comparison between common model driven approaches and the approach proposed in this thesis is shown in Table 2.2.

Table 2.2: Comparison with other model-driven approaches

<table>
<thead>
<tr>
<th>Other model-driven approaches</th>
<th>This thesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determines one best option</td>
<td>Presents alternatives (trade-offs)</td>
</tr>
<tr>
<td>Requires user input for application characterization</td>
<td>Relies on performance measurements</td>
</tr>
<tr>
<td>Do not cater for applications with variable resource demand</td>
<td>Caters for variable resource demand</td>
</tr>
<tr>
<td>Rely on performance data from cloud tools (eg. AWS Cloudwatch)</td>
<td>Applicable on any cloud platform</td>
</tr>
</tbody>
</table>
Chapter 3

Fixed-Cost Scaling with Fixed Workload

This chapter investigates fixed-cost scaling for an application with a fixed workload and the impact of resource configuration on the cost. Given an application with a fixed workload, time deadline and a cost budget, this chapter presents a measurement-driven analytical modeling approach for determining cost-time Pareto-optimal cloud resource configurations for executing the application. Firstly, we introduce fixed-cost scaling for an application with a fixed workload. Secondly, we present our approach for determining configurations including models and algorithm. Thirdly, we evaluate our approach on Amazon EC2 cloud and discuss the implications of cloud configurations and fixed-workload scaling on cost and time using model predictions.

3.1 Scaling a Fixed Workload

The traditional view of scaling applications was to make the application run faster by putting in more resources. For example, a computer simulation that runs for
1000 steps in one minute can be run in less than one minute without reducing the number of simulation steps, by increasing the compute power of the system. Such scaling of performance where the workload stays intact while the performance changes fixed-workload scaling. In fixed-workload scaling, the improvement in speedup is achieved by increasing the processing capability of the system. With the rise of parallel computing, fixed-workload scaling received much attention as the full workloads were able to be completed in much shorter time with no compromise in the amount of work done by parallelizing the execution. The limitations of parallelizing application execution has been widely explored over the past half-century. The most well-known law on this regard is Amdahl’s law [12].

\[
S_{max} = \frac{1}{f + \frac{(1-f)}{P}} \tag{3.1}
\]

where \(S_{max}\) is the maximum achievable speedup, \(f\) is the non-parallel portion of the application and \(P\) is the number of parallel processors. Amdhal’s law shows that the parallel speedup of application execution is limited by its sequential fraction when the size of the application is fixed. As a result the performance improvement plateaus when we keep increasing the amount of parallel compute resources.

**Impact of Resource Configuration On Fixed-Cost Scaling on Cloud**

In traditional parallel computing systems, parallel processing resources were limited and the performance improvement was limited due to this constraint. In contrast, cloud computing offers a theoretically unlimited amount of resources. Even in commercial cloud platforms such as Amazon AWS, Google Cloud and Microsoft Azure, there are hundreds of resources instances ready to be deployed instantly. Thus, the improvement of performance of application is constrained
mainly due to the application constraints and user’s cost budget.

A given cloud application is executed on one or more cloud resource instances that belong to one or more cloud resources categories. The set of such cloud resources that a particular application is run on is called the application’s cloud resource configuration. Since cloud resources have different cost, a cloud configuration’s cost and performance depends on the type of cloud resources instances in the configuration. When the workload is fixed, we change the amount of parallel resources allocated for the application by changing the cloud configuration on which the application is executed, thus impacts the performance and cost of execution. In this chapter, we investigate this through determining different cloud configurations that can execute a given application with a fixed workload.

3.2 Determining Cost-Time Efficient Cloud Configurations

For different fixed workloads of a given application, we determine the cloud configurations to execute them on cloud. We represent the size of the parallel platform in terms of the cloud configuration. To investigate fixed-workload scaling, depending on the characteristics of the application, we fix different workloads in two ways. (i) fixing both time deadline and accuracy of an application while fixing different problem sizes or (ii) fixing both time deadline and problem size and fixing the accuracy at different values. For each of these fixed workload, we determine cloud configuration, cost and time to execute them.

We propose a measurement-driven analytical modeling approach to determine cost-time optimal cloud configurations for executing a given application with a time deadline and a cost budget. To match application resource demand with cloud resource capacity, we use the number of instructions executed as the proxy
Chapter 3. Fixed-Cost Scaling with Fixed Workload

for resource demand and the instruction execution rate as the proxy for resource capacity. To measure application resource demand and cloud resource capacity, we conduct baseline executions of applications on a local server and on selected cloud resources.

In the context of exploring fixed-cost scaling on cloud, this chapter makes the following key contributions:

1. We introduce a measurement-driven analytical modeling approach to determine cost-time optimal cloud configurations that execute a given application with a time deadline and cost budget. We evaluate our approach on more than ten million configurations exposed by nine Amazon EC2 resource types, using representative applications covering domains such as scientific solvers, video compression, n-body simulations, and bioinformatics.

2. Using our approach, we show that among feasible configurations that execute an application within a time deadline and cost budget, the cloud exposes multiple cost-time Pareto-optimal configurations that minimize time and cost. We show that selecting a Pareto-optimal configuration among feasible configurations can save up to 30% of the cost for n-body application.

3. We investigate the cost of fixed-workload scaling of applications’ resource demand and observe that cost grows faster than resource demand when multiple cloud resources with different cost efficiency are used.

4. We show that when the workload is fixed, the relative increase in cost when tightening the execution time deadline is smaller than the relative decrease of time deadline. For example, if the time deadline is tightened by two-thirds, the cost increases with just 40% for n-body application.
Chapter 3. Fixed-Cost Scaling with Fixed Workload

3.3 Approach

In this section, we present our measurement-driven analytical modeling approach to determine cost-time optimal cloud configurations.

3.3.1 Overview

Given an application with a time deadline and a cost budget, and a set of cloud resources, our approach determines cost-time Pareto-optimal cloud configurations that satisfy deadline and budget constraints, as illustrated in Figure 3.1. Our approach consists of three parts. Firstly, baseline runs of a scale-down versions of the application are executed on both a local server and on cloud-based nodes to characterize application resource demand and cloud resource capacity. Secondly, our time and cost models take as input application and cloud resources
characterizations to determine the time and cost of executing the application on a set of cloud resources. Thirdly, our configuration selection algorithm finds cloud configurations that satisfy the time deadline and cost budget using our time and cost models. These configurations are filtered to determine Pareto-optimal configurations that minimize time and cost.

Table 3.1: Model Notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{n,a}$</td>
<td>an application with problem size $n$ and accuracy $a$</td>
</tr>
<tr>
<td>$P_{n',a'}$</td>
<td>a scale-down application with problem size $n'$ and accuracy $a'$</td>
</tr>
<tr>
<td>$D_{P_{n,a}}$</td>
<td>resource demand of application $P_{n,a}$</td>
</tr>
<tr>
<td>$I$</td>
<td>set of all resource types</td>
</tr>
<tr>
<td>$M$</td>
<td>total number of resource types in $I$</td>
</tr>
<tr>
<td>$m_{i,max}$</td>
<td>maximum number of nodes available from resource type $i$</td>
</tr>
<tr>
<td>$c_i$</td>
<td>cost per unit time for resource type $i$</td>
</tr>
<tr>
<td>$v_i$</td>
<td>number of virtual processors in resource type $i$</td>
</tr>
<tr>
<td>$W_{i,vCPU}$</td>
<td>instruction execution rate of resource type $i$ per virtual CPU</td>
</tr>
<tr>
<td>$W_i$</td>
<td>resource capacity of cloud resource type $i$</td>
</tr>
<tr>
<td>$S$</td>
<td>total number of cloud configurations</td>
</tr>
<tr>
<td>$G$</td>
<td>set of all cloud configurations</td>
</tr>
<tr>
<td>$G_j$</td>
<td>a cloud configuration $j$ consisting of one or more resources</td>
</tr>
<tr>
<td>$U_j$</td>
<td>total resource capacity of configuration $j$</td>
</tr>
<tr>
<td>$C_{j,u}$</td>
<td>total cost per unit time of configuration $j$</td>
</tr>
<tr>
<td>$T'$</td>
<td>time deadline to execute $P_{n,a}$</td>
</tr>
<tr>
<td>$T$</td>
<td>predicted execution time for $P_{n,a}$</td>
</tr>
<tr>
<td>$C'$</td>
<td>cost budget to execute $P_{n,a}$</td>
</tr>
<tr>
<td>$C$</td>
<td>predicted execution cost for $P_{n,a}$</td>
</tr>
</tbody>
</table>

We explain in detail all three parts of the approach using the notations in Table 3.1. For each application, we determine its resource demand by conducting baseline executions of different scale-down values of problem size, $n'$, and accuracy, $a'$, and by measuring the instruction count on a local server that has the same
architecture as cloud resources.

The number of instructions executed is used as a proxy to match resource demand of the application with the capacity of cloud resources. This matching enables us to determine cloud configurations capable of executing the given application with time deadline and cost budget constraints. Among these configurations, we select cost-time Pareto-optimal configurations that either minimize cost for a given deadline or minimize time for a given budget. The set of Pareto-optimal configurations represents the Pareto frontier in cost-time space.

### 3.3.2 Cloud Configuration

A cloud configuration is a combination of nodes from one or more types of cloud resources. For example, if there are three different resource types, the tuple \( <2,3,5> \) represents a configuration that uses two nodes from first resource type, three nodes from the second resource type and five nodes from the third resource type. Generalizing for \( M \) cloud resource types, a configuration \( G_j \) is a tuple \( <m_{j,1}, m_{j,2}, ..., m_{j,M}> \), where \( m_{j,i} \) is the number of nodes of type \( i \in [1, M] \). This number of nodes can take integer values between 0 and \( m_{i,max} \), hence, the total number of configurations exposed by cloud resources is

\[
S = \left( \prod_{i=1}^{M} (m_{i,max} + 1) \right) - 1
\]  

(3.2)

considering that the configuration with no nodes is not useful.

### 3.3.3 Time Model

Given an application \( P_{n,a} \) with problem size \( n \) and accuracy \( a \), \( D_{P_{n,a}} \) denotes its resource demand in terms of total number of instructions as a function of \( n \) and \( a \). The time to execute this application on a cloud configuration \( G_j =< \)
where $U_j$ is the total capacity of configuration $G_j$ expressed in instructions per second. We characterize both the resource demand of real applications and the resource capacity of real cloud instances in Section 3.4.

The total resource capacity of configuration $G_j$ is the summation of resource capacities of all nodes in this configuration,

$$U_j = \sum_{i=1}^{M} (m_{j,i} \times W_i) \quad (3.4)$$

We derive the computational capacity of a cloud resource using the instruction execution performance (MIPS) per virtual processor, $W_{i,vCPU}$, as derived from the baseline measurements on both local and cloud-based nodes. We assume that all cloud resources are multicore systems and that a virtual processor core in a cloud resource, $vCPU$, is a hyper-thread of the underlying physical processor. Thus, the resource capacity of cloud resource $i$ comprised of $v_i$ virtual processing cores is

$$W_i = W_{i,vCPU} \times v_i \quad (3.5)$$

In this work, we focus on highly-parallelizable, compute-intensive applications where communication overhead is negligible, thus, we do not model communication overhead.

### 3.3.4 Cost Model

We derive the execution cost of an application $P_{n,a}$ on a cloud configuration $G_j$ by multiplying the execution time $T$ by the total cost per unit time of configuration’s
resources,

\[ C = T \times C_{j,u} \]  

(3.6)

where

\[ C_{j,u} = \sum_{i=1}^{M} (m_{j,i} \times c_i) \]  

(3.7)

Cloud pricing data is taken from the cloud vendor website.

### 3.3.5 Configuration Selection Algorithm

**Algorithm 3.1 Resource Configuration Selection**

1. compute \( G \) using \( I \) and \( m_{i,\text{max}} \)
2. for each \( G_j \) in \( G \) do
3. compute \( U_j, C_{j,u}, T, C \)
4. if \( T < T' \) and \( C < C' \) then
5. output \( G_j, T, C \)
6. end if
7. end for

Our configuration selection algorithm generates a list of configurations that meet the execution time and budget constraints. As shown in the pseudocode listed in Algorithm 3.1, the execution time and cost for every configuration \( G_j \) are computed using our models. The predicted values are compared against deadline and budget constraints. Finally, the configuration list is passed through a Pareto-optimal filter to get the optimal configurations, as shown in Figure 3.1. As our approach explores the entire configuration space, it guarantees to find all optimal configurations for a given resource demand with a time deadline and a cost budget.

### 3.4 Evaluation

Evaluation is divided into five main sections. Firstly, applications with different execution profiles are characterized to determine the relationship between appli-
Table 3.2: Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th>Problem Size (Input)</th>
<th>Source</th>
</tr>
</thead>
</table>
| bt          | Block tri-diagonal solver | Grid: 162 x 162 x 162x
Iterations: 200 x 1024
Time step: 0.0001 | NPB [27] |
| ft          | Fourier transform | Grid: 512 x 512 x 512
Iterations: 20 x 1024 | NPB [27] |
| is          | Integer Sort | Number of keys: $2^{27}$
Max value: $2^{28}$
Iterations: 20 x $2^{14}$ | NPB [27] |
| x264        | Video compression | number of videos (n)
size of video compression rate (f) | Parsec [20] |
| n-body      | Simulation of masses (eg. space particles) | number of masses (n)
simulation steps (s) | PetaKit [54] |
| sand        | Genome sequencing | number of sequences (n)
threshold ($\tau$) | CCTools [71] |

cation parameters and application resource demand (workload). Secondly, cloud resources with different cost-performance profiles are characterized to estimate their computation capacities. Thirdly, we present a method to optimize cloud resource characterization. Fourthly, our cost-time model is validated against cloud measurements. Finally, we present an analysis of the results from our model-driven approach for the selected applications on target Amazon EC2 resources.

3.4.1 Resource Demand (Workload) of Applications

As shown in Table 3.2, we use six applications in our evaluation. To determine the relationship between application parameters, such as problem size and accuracy, and application resource demand, we run each application with different problem sizes and accuracy levels on a local Intel Xeon E5-2630 v4 server and measure the instruction count using Linux `perf` utility. bt, ft, and is applications are executed for one fixed problem size as shown in Table 3.2 and x264, n-body and sand are executed for multiple fixed problem sizes. For x264, we range the problem size from 2 to 32 and $f$ from 10 to 50. For n-body, $n$ ranges from 8,192 to 65,536...
Chapter 3. Fixed-Cost Scaling with Fixed Workload

Figure 3.2: Resource Demand of x264 - n

Figure 3.3: Resource Demand of x264 - f
Chapter 3. Fixed-Cost Scaling with Fixed Workload

Figure 3.4: Resource Demand of n-body - n

Figure 3.5: Resource Demand of n-body - s
Chapter 3. Fixed-Cost Scaling with Fixed Workload

Figure 3.6: Resource Demand of sand - n

Figure 3.7: Resource Demand of sand - t
bodies and $s$ ranges from 1,000 to 8,000 steps. For sand, $n$ range is from 1 million to 64 million sequences, and $\tau$ range is from 0.01 to 1.

As shown in Figures 3.2 and 3.3, x264 exhibits a linear relationship between $n$ and resource demand, while the relationship between $f$ and resource demand is quadratic. As shown in Figures 3.4 and 3.5 n-body, $n$ is quadratically related to resource demand, while $s$ affects resource demand in a linear way. As shown in Figures 3.6 and 3.7 sand, resource demand grows linearly with $n$ and logarithmically with $\tau$. These results show that selected applications exhibit linear, quadratic and logarithmic resource demands as function of problem size and accuracy.

### 3.4.2 Determining Cloud Resource Capacity

<table>
<thead>
<tr>
<th>Type</th>
<th>vCPUs</th>
<th>Frequency (GHz)</th>
<th>Memory (GB)</th>
<th>Storage</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>2</td>
<td>2.9</td>
<td>3.75</td>
<td>EBS</td>
<td>0.105</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>2.9</td>
<td>7.5</td>
<td>EBS</td>
<td>0.209</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>2.9</td>
<td>15</td>
<td>EBS</td>
<td>0.419</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
<td>2.3</td>
<td>8</td>
<td>EBS</td>
<td>0.133</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>2.3</td>
<td>16</td>
<td>EBS</td>
<td>0.266</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>2.3</td>
<td>32</td>
<td>EBS</td>
<td>0.532</td>
</tr>
<tr>
<td>r3.large</td>
<td>2</td>
<td>2.5</td>
<td>15</td>
<td>32</td>
<td>0.166</td>
</tr>
<tr>
<td>r3.xlarge</td>
<td>4</td>
<td>2.5</td>
<td>30.5</td>
<td>80</td>
<td>0.333</td>
</tr>
<tr>
<td>r3.2xlarge</td>
<td>8</td>
<td>2.5</td>
<td>61</td>
<td>160</td>
<td>0.664</td>
</tr>
</tbody>
</table>

$^1$ EBS = Amazon Elastic Block Storage

To apply our modeling approach we select nine types of cloud resources from Amazon EC2 cloud as shown in Table 3.3. We determine cloud resource capacities in terms of instruction execution rate. One way to estimate this rate is to use the base CPU frequency obtained from the specification, and to derive an upper-bound of the performance. However, different applications have different execution profiles and different instruction execution rates, as we show in the previous section.
Therefore, to determine the instruction execution rate more accurately, we profile the execution of each application with a small problem size on all resource types. Since cloud virtualization restricts the usage of hardware counters to get instruction count, we execute the application on our local Xeon server to get this value. This local Xeon server and selected cloud resources have the same instruction set architecture (ISA) and the same micro-architecture, hence, the instruction count should be similar. We run the same application on the cloud resources and measure execution time. Measured instruction count is divided by measured execution time to determine the instruction execution rate for each resource type.

3.4.3 Optimizing Resource Characterization

![Figure 3.8: Cloud Resource Characterization](image)

In this section, we show that the user should profile only one resource type
for each resource category to get the instruction execution rate per unit cost for one vCPU. Analyzing the ratio between instruction execution rate and cost for each application on all resource types, we observe a large gap between resource categories, as shown in Figure 3.8. For all applications, \( c_4 \) resources have two times better performance per cost compared to \( r_3 \) resources, while \( m_4 \) resources have 1.5 times better performance per cost compared to \( r_3 \) resources. Moreover, resource types within the same resource category have similar performance in terms of instructions executed per unit cost. For example, in n-body, normalized performances for \( c_4.\text{large} \), \( c_4.\text{xlarge} \) and \( c_4.2\text{xlarge} \) are 26.27 billion, 26.21 billion and 26.01 billion instructions per second per dollar, respectively. Thus, we safely assume that instruction execution rate per unit cost is similar for all resource types within a resource category. This optimization allows a more practical characterization of a large number of cloud resource types.

### 3.4.4 Validation

To validate our models, we compare the predicted execution time and cost against measured values from Amazon EC2 cloud. Due to research budget constraints, we validate only three cases for each application and show the results in Table 3.4. Maximum prediction errors are 9.5%, 13.1% and 16.7% respectively.

<table>
<thead>
<tr>
<th>Application</th>
<th>Configuration</th>
<th>Time (hr)</th>
<th>Cost ($)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Predicted</td>
<td>Actual</td>
<td>Predicted</td>
</tr>
<tr>
<td>x264(8000,20)</td>
<td>( {2.1,0,0,0,0,0,0,0,0} )</td>
<td>21</td>
<td>23</td>
<td>9.5</td>
</tr>
<tr>
<td>x264(16000,20)</td>
<td>( {5.1,1,0,0,0,0,0,0,0} )</td>
<td>20</td>
<td>46</td>
<td>5.0</td>
</tr>
<tr>
<td>x264(32000,20)</td>
<td>( {5.5,5,1,0,0,0,0,0,0} )</td>
<td>23</td>
<td>99</td>
<td>8.7</td>
</tr>
<tr>
<td>n-body(65536,4000)</td>
<td>( {5.5,0,0,0,0,0,0,0,0} )</td>
<td>18</td>
<td>53</td>
<td>11.1</td>
</tr>
<tr>
<td>n-body(65536,6000)</td>
<td>( {5.5,5,0,0,0,0,0,0,0} )</td>
<td>23</td>
<td>85</td>
<td>13.0</td>
</tr>
<tr>
<td>n-body(65536,8000)</td>
<td>( {5.5,5,3,0,0,0,0,0,0} )</td>
<td>24</td>
<td>126</td>
<td>8.3</td>
</tr>
<tr>
<td>sand(1024m,0.32)</td>
<td>( {5.4,1,0,0,0,0,0,0,0} )</td>
<td>6</td>
<td>18</td>
<td>16.7</td>
</tr>
<tr>
<td>sand(2048m,0.32)</td>
<td>( {5.5,0,0,0,0,0,0,0,0} )</td>
<td>23</td>
<td>72</td>
<td>8.6</td>
</tr>
<tr>
<td>sand(4096m,0.32)</td>
<td>( {5.3,1,0,0,0,0,0,0,0} )</td>
<td>51</td>
<td>144</td>
<td>13.7</td>
</tr>
</tbody>
</table>
for x264, n-body, and sand. To a large extent, the variation in prediction accuracy could be attributed to the processor sharing approach implemented by cloud providers such as Amazon [99]. The underlying physical processor is shared among multiple cloud instances simultaneously, such that vCPUs are hyper threads and not physical CPU cores. On the other hand, n-body and sand have higher prediction error compared to x264 due to inter-node communication. In contrast, x264 processes execute standalone on each node with no inter-node communication.

### 3.4.5 Configuration Space

![Figure 3.9: Cloud Resource Configuration Space for 6hr Time Deadline and $50 Cost Budget - bt](image)

To understand the implications of large configuration space on cost, we look at the distribution of feasible configurations in the time-cost plane for one problem size. For relatively small workloads bt, ft, lu and sp, we set the time deadline at 6 hours and the cost budget at $50. For larger workloads x264, n-body and sand, the time deadline and cost budget are set at 24 hours and $350, respectively.
Figure 3.10: Cloud Resource Configuration Space for 6hr Time Deadline and $50 Cost Budget - ft

Figure 3.11: Cloud Resource Configuration Space for 6hr Time Deadline and $50 Cost Budget - is
Figure 3.12: Cloud Resource Configuration Space for 24hr Time Deadline and $350 Cost Budget - x264 (32000, 20)

Figure 3.13: Cloud Resource Configuration Space for 24hr Time Deadline and $350 Cost Budget - n-body (65536, 8000)
Chapter 3. Fixed-Cost Scaling with Fixed Workload

Figure 3.14: Cloud Resource Configuration Space for 24hr Time Deadline and $350 Cost Budget - sand (8192 \times 10^6,0.32)

Table 3.5: Cloud Resource Configuration Space

<table>
<thead>
<tr>
<th>Application</th>
<th>Config Space (million)</th>
<th># PO Config</th>
<th>PO Cost Range ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bt</td>
<td>7.1</td>
<td>28</td>
<td>27 - 47</td>
</tr>
<tr>
<td>ft</td>
<td>10.0</td>
<td>40</td>
<td>7 - 13</td>
</tr>
<tr>
<td>is</td>
<td>9.9</td>
<td>55</td>
<td>11 - 27</td>
</tr>
<tr>
<td>x264 (32000,20)</td>
<td>7.6</td>
<td>26</td>
<td>98 - 135</td>
</tr>
<tr>
<td>n-body (65536,8000)</td>
<td>5.8</td>
<td>23</td>
<td>126 - 167</td>
</tr>
<tr>
<td>sand (8192 \times 10^5,0.32)</td>
<td>2.0</td>
<td>58</td>
<td>180 - 210</td>
</tr>
</tbody>
</table>
Summary of observations on the configuration spaces for all applications is shown in Table 3.5.

Observation 1:

*Among the large configuration space exposed by cloud resources, there exists a Pareto frontier of multiple Pareto-optimal configurations where cost can be reduced by relaxing the time deadline.*

We observe a large number of feasible configurations spread-out across the time-cost plane, as shown in Figures 3.9 - 3.14. For example, considering the three largest workloads, there are around 7.6 million, 5.8 million and 2 million feasible configurations for x264, n-body and sand, respectively. Moreover, there are multiple Pareto-optimal configurations that satisfy both time deadline and cost budget constraints. For x264, there are 26 Pareto-optimal configurations spanning the cost range $98 - $135. For n-body, there are 23 Pareto-optimal configurations spanning the cost range $126 - $167. For sand, there are 58 Pareto-optimal configurations with costs spanning from $180 to $210. Among these optimal configurations, there is a considerable span of cost where the highest cost is about 1.4 times, 1.3 times and 1.2 times higher compared to the lowest cost, for x264, n-body and sand, respectively. Existing cost-aware cloud resource allocation approaches that use model predictions \[25,51,89\] propose integer linear programming to determine the best cloud configuration for a given set of constraints. In comparison, we expose a range of Pareto-optimal cloud resource configurations that enable consumers to select the best suited one. Moreover, having access to such different alternatives gives freedom to the consumer to change the configuration dynamically if required. Although our approach is complementary to dynamic resource migration, it is out of the scope for this thesis. CAP3 \[44\] proposes a similar analytical model based approach based on application profiling with TAU profiler. Their cloud configura-
tion is limited to one resource type whereas in this thesis we explore configurations made up of a combination of multiple resource types.

3.4.6 Cost of Scaling a Fixed Workload

![Figure 3.15: Effect of Scaling a Fixed Workload (Problem Size) on Cost for n-body - masses (n)](image)

To investigate the implications of fixed-cost scaling of a fixed workload on cost, we fix the workload at different levels in terms of (i) application problem size and (ii) accuracy and determine the minimum cost for executing each workload.

First, we fix the time deadline and accuracy, and determine the minimum cost for different problem sizes. As shown in Section 3.4.1, the relationship between problem size and resource demand is quadratic for n-body and linear for sand. As shown in Figures 3.15 and 3.16, the cost follows approximately the same trend as the resource demand: it grows quadratically with problem size for n-body and linearly with problem size for sand. However, at some points, curve’s gradient is
changing and the cost curve deviates from the expected path. We are analyzing this behavior below.

Secondly, we fix the time deadline and the problem size, and determine the minimum execution cost for different accuracy. As shown in Figures 3.17 and 3.18, similar to the previous case where the workload was set by problem size, growth of cost with accuracy corresponds to the growth of resource demand. For n-body where the resource demand grows linearly with accuracy, we observe a linear growth of cost while for sand where the resource demand grows logarithmically with accuracy, we observe a logarithmic growth of cost. For applications with sub-linear relationship between cost and accuracy, such as sand, significant accuracy improvements can be achieved for a small increase in cost. As shown in Figure 3.18 for sand, to improve the accuracy of sand 1.6 times, from 0.64 to 1, we need only a 20% increase of the cost.

Figure 3.16: Effect of Scaling a Fixed Workload (Problem Size) on Cost for sand - sequences (n)
Chapter 3. Fixed-Cost Scaling with Fixed Workload

Figure 3.17: Effect of Fixed-Workload Scaling (Accuracy) on Cost for n-body -steps (s)

Figure 3.18: Effect of Fixed-Workload Scaling (Accuracy) on Cost for sand -threshold (τ)
Observation 2:

For applications running on cloud, cost grows faster than resource demand when multiple resources having different cost efficiency are used.

We observe that cost curves exhibit sudden changes of gradient at certain points when problem size and accuracy is scaled. As shown in Figure 3.17, the cost curve of n-body for the 24 hours deadline grows linearly till \( s = 6,000 \) and then it suddenly increases the gradient. To investigate this unexpected behavior, we analyze cloud configurations, as annotated in Figure 3.17 for each accuracy along the cost curve. In each configuration, first three values represent the numbers of nodes from \( c4 \) category, second three values represent the numbers of nodes from \( m4 \) category and the last three values represent the number of nodes from \( r3 \) category. We observe that whenever the cost curve deviates, the configuration uses a new resource category. In n-body, when \( s = 6,000 \) the configuration is \( \{5,5,5,0,0,0,0,0,0\} \), which indicates that maximum number of nodes have been taken from \( c4 \) category. When \( s = 8,000 \), the configuration \( \{5,5,5,3,0,0,0,0,0\} \) includes additional three nodes from \( m4 \) category. The difference of normalized instruction execution rate across categories, shown in Figure 3.8, explains this behavior. Since the normalized instruction execution rate of \( m4 \) is higher than the rate of \( c4 \), the cost gradient increases when the configuration spills into \( m4 \).

\[ \]

3.4.7 Cost of Time Deadline

Observation 3:

The increase in cost is always smaller than the reduction of execution time when the deadline of an application is tightened.

Resource elasticity makes cloud a suitable platform for time critical computations. To understand the impact of time deadline on cost, we fix the prob-
lem size and accuracy for n-body and sand, and observe the variation of minimum cost while tightening time deadline. We observe that the cost increase and the time deadline decrease are not proportional. As shown in Figure 3.15 for n-body(262,144, 1000), tightening the time deadline from 72 hours to 24 hours incurs only $95 of cost increase. Similarly for sand(8192, 0.32), as shown in Figure 3.16, tightening the time deadline from 48 hours to 24 hours incurs only $35 more. Thus, tightening the deadline by 67% results in 40% increase in cost for n-body, and tightening the deadline by 50% results in 25% increase in cost for sand.

3.5 Summary

With the availability of many heterogeneous resource types, clouds are providing a myriad of elastic resource configurations. Thus, choosing the right configuration to execute an application on cloud is a daunting task. This chapter introduces a measurement-driven analytical modeling approach to determine cloud resource configurations to execute an application with a given time deadline and cost budget. Our execution time and cost models use measured values from baseline runs to estimate application resource demand and cloud resource capacity in terms of executed instructions. We evaluate our approach on Amazon EC2 cloud with a resource configuration space of approximately ten million configurations and six representative applications. Validation against time and cost measurements from Amazon EC2 shows that the prediction error of our models is less than 17%.

We perform a model-based analysis and show that multiple cost-time Pareto-optimal configurations exist for executing applications on the cloud wherein selection of the optimal cloud resource configuration could reduce the execution cost by up to 30%. We investigate the cost of scaling a fixed workload by fixing the
application workload in two different ways; the problem size and accuracy. Our results show that there is potential to trade-off the accuracy of applications for execution cost on cloud. For example, approximately 1.5 times increase in accuracy can be achieved with approximately 1.2 times increase in cost. This will be explored in detail in later chapters of this thesis. Moreover, we observe that cost grows faster than application resource demand when multiple resources with different cost efficiency are used. Lastly, we show that the cost of tightening the time deadline is always not proportional to the increase in execution cost. For example, tightening the time deadline by two-thirds increases the cost by just 40%.
Chapter 4

Scaling Application Problem Size

This chapter investigates fixed-cost scaling of applications with changing problem size. Given an application with a changing workload, a fixed time deadline and a cost budget, this chapter presents a measurement-driven analytical modeling approach to determine the largest Pareto-optimal problem size that can be executed on a given set of cloud resources and the cloud resource configuration to achieve that. Firstly, we define fixed-cost scaling with a fixed time deadline and how changing application workload leads to fixed-cost scaling. Secondly, we define a three-dimensional Pareto-optimality in our context. Thirdly, we present our approach including the models, algorithm and details on determining the application growth function. Lastly, we evaluate our approach on Amazon EC2 cloud and discuss the implications of cost and time on the largest executable problem size.

4.1 Changing Application Workload

Building upon Amdahl’s law, Gustafson [39] presented an observation concerning the limits of parallel speedup of application. He showed that although the parallel
speedup plateaus with fixed-workload scaling, in reality, the workload is scaled with the amount of compute resources thrown into the computation. Thus, if the non-parallel portion of the application stays intact or grows slower than the growth of the parallel section, a near linear speedup can be achieved. Gustafson’s law can be illustrated as follows:

$$S(P) = P - f(P - 1)$$  

where $S$ is the maximum achievable speedup, $f$ is the non-parallel portion of the application and $P$ is the number of parallel processors.

Gustafson’s observation is relevant today more than ever with the advent of cloud computing. As highlighted earlier, due to the availability of a large configuration space, we are equipped with a great opportunity to scale the application workload and still achieve speedup by switching into a different cloud resource configuration if the cloud consumer’s cost budget permits. Thus, it is important to understand the impact of changing the application workload on the cloud configuration, and the execution cost.

**Fixed-Cost Scaling of an Application with a Fixed Time Deadline**

The ability to change the configuration while the application workload grows enables cloud consumers to accomplish executing larger workloads within the same time deadline by changing the cloud configuration. With increasingly many time-sensitive applications moving towards cloud, adhering to time deadlines is a critical factor in achieving consumers’ business deadlines. However, the changing to a higher performing configuration likely incurs a higher cost. Thus, it is imperative
for cloud consumers to understand the impact of fixed-cost application scaling with a fixed time deadline.

4.2 Largest Pareto-Optimal Problem Size

In the previous chapter we emphasized the importance of understanding the cloud configuration space and the trade-off between the cost and time for scaling a fixed workload. As a result of recent advancements in computer science such as big data analytics, machine learning, advanced scientific simulations, among others, the size of cloud applications is increasingly becoming larger. Thus, it is important for a cloud consumer to understand the trade-offs involved in executing larger workloads, especially what is the maximum workload or the size of the application that could be executed within the given resource and business constraints. In this chapter we further investigate the scaling of cloud applications with a fixed time deadline, specially focusing on determining the largest application problem size executable on cloud and its cost.

Investigating the largest executable problem size leads to a multi-objective optimization between the time, cost, and the application problem size. Cloud applications are often tied to a time deadline and a cost budget. Thus, unlike the traditional on-premise systems, the application execution and scaling on cloud is constrained by these time and cost constraints. For example, a video compression in a streaming application needs to complete its workload within a time period acceptable for the streaming customer. Given the large configuration space on cloud and each configuration having different cost and performance, each of these configurations are associated with an application problem size, cost, and time. Thus, in this context, we need to define the Pareto-optimality with respect to three objective functions size, cost, and time.
An allocation is Pareto-optimal if there is no alternative allocation where at least one of the participating elements of the allocation is improved without degrading any other participating element. In our context, we need to maximize the size while minimizing cost and time. Thus, we define that a tuple \(<size, cost, time>\) consists Pareto-optimal size if size cannot be increased without increasing either/both cost and time.

### Determining the Largest Pareto-Optimal Problem Size

Given an application with a time deadline and a cost budget, we propose a measurement-driven analytical modeling approach to determine cost-time Pareto-optimal problem sizes of the application and cloud resource configurations for executing them. In addition to determining Pareto-optimal problem size and configuration, it is important to derive useful insights into scalability of an application on cloud resources with respect to cost. To address this we introduce a new metric “Performance Cost Ratio (PCR)” to characterize cloud resources. To match the resource demand of application with the resource capacity of cloud resources, we use the instruction count as the proxy. To estimate application resource demand and cloud resource capacity, we conduct baseline executions of cloud applications on a non-virtualized server and selected cloud resource instances. As applications exhibit different resource demand growth for each scaling parameter, it is necessary to determine the scaling function for each application scaling parameter. To determine scaling functions, we utilize baseline measurements. We validate our approach on Amazon EC2 cloud with more than ten million cloud resource configurations and representative applications that cover a range of scaling functions.

This chapter focuses on cost-time Pareto-optimal scaling of application problem size and makes the following key contributions:

1. We propose a measurement-driven analytical modeling approach to deter-
Chapter 4. Scaling Application Problem Size

1. To determine Pareto-optimal problem sizes for a given application that can be executed on cost-time optimal set of cloud resources within a given time-deadline and cost budget.

2. We show that there exist multiple Pareto-optimal problem sizes with configurations with different cost-time performance, thus there is an opportunity to trade-off problem size to reduce cost and satisfy time deadline for a cloud consumer.

3. Contrary to intuition, we aid cloud consumers with the insight that increasing the cost budget and/or relaxing the time deadline does not always result in executing a larger problem size.

4. We show that the trade-off between cost, time, and problem size is non-trivial and non-linear as it depends not only on the resource demands of the application but also on the cost-time performance of cloud resources. For example in n-body simulation two-fold tightening of time deadline results in just one-fifth reduction of problem size.

5. To characterize the cost-performance of cloud resources, we introduce the “Performance Cost Ratio (PCR)” metric. PCR quantifies the resource capacity with respect to cost such that the configurations to execute cost-time efficient near-optimal Problem sizes can be easily determined by choosing resources with the largest PCR.

4.3 Approach

This section presents our measurement-driven analytical modeling approach for determining Pareto-optimal problem sizes for a given application and cloud resource configurations to execute them. Firstly, an overview of the approach is presented. Secondly, determining application resource demand growth function
is discussed. Finally, model derivation is presented followed by the optimization algorithm to determine Pareto-optimal sizes and cloud configurations for a given cost budget and a time deadline.

4.3.1 Overview

Given an application $P$ with a cost budget $C$, a time deadline $T$, and a set of cloud resources, our approach determines Pareto-optimal problem sizes $S$ of $P$ executable on the cloud. In addition, we determine Pareto-optimal cloud resource configurations to execute these sizes $S$ of $P$. To obtain an accurate matching between cloud resources and $P$’s resource demand for $S$, our approach uses a measurement-driven model. Figure 4.1 shows the outline of the proposed approach consisting of two phases (i) baseline execution to obtain the measurements, and, (ii) the model and optimization phase to determine the Pareto-optimal sizes and cloud resource configurations.

To determine Pareto-optimal problem sizes and the cloud resource configurations to execute them, our approach requires (i) characterizing the application and determining the resource demand growth function, and, (ii) characterizing the set of cloud resources and determining their execution rates. We use baseline measurements from a non-virtualized server and cloud resource instances to characterize the application and cloud resources. For characterizing the application, we measure the number instructions executed on the non-virtualized server while
### Table 4.1: Model Notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td></td>
</tr>
<tr>
<td>$P$</td>
<td>an application</td>
</tr>
<tr>
<td>$S$</td>
<td>a problem size of $P$</td>
</tr>
<tr>
<td>$W$</td>
<td>resource demand of $P$</td>
</tr>
<tr>
<td>$f(S)$</td>
<td>resource demand growth function of $P$ in terms of $S$</td>
</tr>
<tr>
<td><strong>Resources</strong></td>
<td></td>
</tr>
<tr>
<td>$R$</td>
<td>set of all cloud resources</td>
</tr>
<tr>
<td>$i$</td>
<td>a cloud resource type in $R$</td>
</tr>
<tr>
<td>$v_i$</td>
<td>number of vCPUs in cloud resource type $i$</td>
</tr>
<tr>
<td>$G$</td>
<td>set of all cloud configurations</td>
</tr>
<tr>
<td>$G_j$</td>
<td>a configuration $j$ having one or more resources</td>
</tr>
<tr>
<td>$r_{i,j}$</td>
<td>set of instances of type $i$ in configuration $j$</td>
</tr>
<tr>
<td>$c_i$</td>
<td>cost per unit time for an instance of resource type $i$</td>
</tr>
<tr>
<td>$c_{G_j}$</td>
<td>cost per unit time for configuration $G_j$</td>
</tr>
<tr>
<td>$\Delta_{vCPU}$</td>
<td>per vCPU computational capacity of a resource type $i$</td>
</tr>
<tr>
<td>$\Delta_i$</td>
<td>total computational capacity of a resource type $i$</td>
</tr>
<tr>
<td>$\Delta_{G_j}$</td>
<td>computational capacity of configuration $G_j$</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td></td>
</tr>
<tr>
<td>$T$</td>
<td>time deadline to execute $P$</td>
</tr>
<tr>
<td>$C$</td>
<td>cost budget to execute $P$</td>
</tr>
<tr>
<td>$t$</td>
<td>time to execute $W$</td>
</tr>
<tr>
<td>$T'$</td>
<td>predicted execution time for executing $W$</td>
</tr>
<tr>
<td>$C'$</td>
<td>predicted cost for executing $W$</td>
</tr>
</tbody>
</table>
running the application for different problem sizes. These measurements are utilized to derive the application resource demand growth function as presented in Section 4.3.2. For characterizing cloud resources, we execute the application on cloud resource instances and record the execution time for each problem size. The execution rate for each cloud resource instance is computed by dividing the number of instructions measured on the non-virtualized server for each problem size of the application by the corresponding execution time recording on the respective cloud resource instance.

To minimize the measurement overhead, we measure the instructions executed on the non-virtualized server using non-intrusive hardware performance counters. A more accurate means of determining the instruction execution rate of cloud resources would be measuring the instruction count directly on the cloud resource itself. However, as commercial cloud vendors restrict access to physical layer of the cloud resources due to virtualization and security reasons, we are constrained to use a non-virtualized server similar to cloud resources with the same Instruction Set Architecture (ISA) for measuring the number of instructions executed.

As presented in Section 4.3.3, we formulate analytical models to compute the instruction execution rate of a cloud resource configuration made up of a combination of one or more cloud resource types, compute the problem size of the application given the execution time and the cloud configuration, and, determine the execution cost for running a cloud configuration for a given time duration. The optimization algorithm presented in Section 4.3.4, takes as input $C$, $T$, the set of cloud configurations $G$ and the growth function of $P$, $f(S)$ and determines largest Pareto-optimal sizes of $P$, $S_{\text{max}}$, optimal cost $C'$, optimal time $T'$ and, the corresponding cloud resource configuration $\text{config}$. 

78
4.3.2 Determining Application Growth Function

An application with multiple scaling parameters may exhibit different growth in resource demand for each scaling parameter. Our approach requires determining resource demand growth functions, hereafter referred to as scaling functions, for each scaling parameter.

Suppose recorded instruction count measurements for problem sizes $S_1, S_2, ..., S_k$ of $P$ are $W_1, W_2, ..., W_k$ respectively, and, $W = f(S)$ is the generic function that expresses the resource demand growth of $P$ with respect to the scaling parameter. We find the function $f(S)$ that best fits the distribution of values $(S_1, W_1), (S_2, W_2), ..., (S_k, W_k)$ using a curve fitting technique.

4.3.3 Model Derivation

A cloud configuration is a combination of instances from one or more cloud resource types. For example, the configuration $\{1, 2, 5\}$ means that there are three resource types with one node from the first resource type, two nodes from second resource type and five nodes from the third resource type. To generalize, we use the notation $\{|r_{0,j}|, |r_{1,j}|, ..., |r_{i,j}|, ..., |r_{|R|-1,j}|\}$ to denote $G_j$, the $j$th cloud configuration in $G$, where $r_{i,j} \in R$ is the set of cloud resource instances from resource type $i \in R$.

The largest $S$ of $P$ executable on configuration $G_j$ depends on the number of instructions $G_j$ can execute while running $P$. Thus the largest $S$ of $P$ executable on $G_j$ is

$$S = f^{-1}(W) \quad (4.2)$$

where $f^{-1}$ is the inverse of the scaling function of $P$, and, $W$ is the total number of instructions executed on $G_j$. 

Suppose, $G_j$ has to execute $P$ for a time duration $t$ to execute $W$ instructions, thus, $W$ is computed as

$$W = \Delta_{G_j} \times t \quad (4.3)$$

where $\Delta_{G_j}$ is the resource capacity of $G_j$ in terms of instruction execution rate. Resource capacity of a cloud resource configuration is defined as the summation of resource capacities of all cloud resource instances in the configuration. Thus, the resource capacity of $G_j$ is

$$\Delta_{G_j} = \sum_{i=0}^{\mid R \mid - 1} (|r_{i,j}| \times \Delta_i) \quad (4.4)$$

where $|r_{i,j}|$ is the number of cloud resource instances from resource type $i$ in $G_j$, and, $\Delta_i$ is the resource capacity of cloud resource type $i$.

A cloud resource instance consists of one or more vCPUs, thus, the total resource capacity of a cloud resource instance depends on the number of vCPUs it consists and the per-vCPU resource capacity. The resource capacity of an instance from cloud resource type $i$ is

$$\Delta_i = \Delta_{\text{vCPU}} \times v_i \quad (4.5)$$

where $\Delta_{\text{vCPU}}$ is the resource capacity of one vCPU of resource type $i$, and, $v_i$ is the number of vCPUs in one instance of $i$.

The total cost to execute an application on a cloud resource configuration depends on the cost per unit time of the configuration and the time duration that the application is run on the configuration. Given that it takes time $t$ to execute size $S$ of $P$ on configuration $G_j$, the total cost is
\[ C' = t \times C_{G_j} \]  

(4.6)

where \( C_{G_j} \) is the cost per unit time for configuration \( G_j \).

\( C_{G_j} \) is determined based on the cost for unit time of each cloud resource instance in the \( G_j \) and is defined as

\[ C_{G_j} = \sum_{i=0}^{\left| R \right| - 1} (|r_{i,j}| \times c_i) \]  

(4.7)

where \( c_i \) is the cost per unit time for cloud resource type \( i \).

### 4.3.4 Largest Pareto-optimal Size Algorithm

We use the Pareto Optimal Size algorithm listed in Algorithm 4.1 to determine the list of Pareto-optimal sizes. The algorithm takes as input \( G, C, T, f(S) \) and outputs a list of tuples where each tuple contains the Pareto-optimal size \( S^{max} \), optimal cost \( C' \), optimal time \( T' \), and, the cloud resource configuration \( G_{optimal} \).

Firstly, the algorithm traverses all cloud configurations and builds a list of tuples of which each tuple contains \( S^{max} \) for each configuration, cost, and, time. Secondly, tuples with same \( \text{time} \) are clustered together. Thirdly, from each cluster with same \( \text{time} \), the tuples with minimum \( \text{cost} \) are retained and others are discarded. From the remainder, the tuples with largest \( \text{size} \) are selected as Pareto-optimal sizes. This algorithm has exponential computation complexity due to expensive Cartesian product involved in generating all cloud resource configurations.

### 4.4 Evaluation

Our approach is evaluated on Amazon EC2 cloud using a representative subset of applications to cover a range of scaling functions with respect to the problem size.
Algorithm 4.1 Pareto Optimal Size (input: \( G, C, T, f(S) \) output: \( S^{\text{max}}, C', T' \), \( G_{\text{optimal}} \))

1: \( \text{list} \leftarrow \emptyset \)
2: \( \# \text{config} \) is a resource configuration
3: \textbf{for} each \text{config} in \( G \) \textbf{do}
4: \( t = 0 \)
5: \( c = 0 \)
6: \textbf{repeat}
7: \( t = t + 1 \)
8: \( c = c_{\text{config}} \times t \)
9: \( W_{\text{config}} = \Delta_{\text{config}} \times t \)
10: \( \# \) max size is determined using inverse of \( f(S) \)
11: \( S^{\text{max}}_{\text{config}} = f^{-1}(\Delta_{\text{config}}) \)
12: \( \text{list} \leftarrow \text{list} \cup [ S^{\text{max}}_{\text{config}}, c, t, \text{config} ] \)
13: \textbf{until} \( c \leq C \) and \( t \leq T \)
14: \textbf{end for}
15: \( \# \) \text{list.t} contains tuples with same time
16: \textbf{for} each \text{list.t} \in \text{list} with same \( t \) \textbf{do}
17: \( \text{list.c} \leftarrow \emptyset \)
18: \( \# \text{tuple.c} \) is a tuple from \text{list.t} with minimum cost
19: \textbf{for} each \text{tuple.c} with \( c = \text{min}(c) \) in \text{list.t} \textbf{do}
20: \( \text{list.c} \leftarrow \text{list.c} \cup \text{tuple.c} \)
21: \textbf{end for}
22: \( \# \text{list.S}^{\text{max}} \) contains tuples with Pareto optimal size
23: \( \# \) add tuples in \text{list.c} with largest size to \text{list.S}^{\text{max}}
24: \( \text{list.S}^{\text{max}} \leftarrow \{ \text{tuple.S}^{\text{max}} : \text{tuple.S}^{\text{max}} \in \text{list.c} \text{ with } S^{\text{max}} = \text{max} (S^{\text{max}}) \} \)
25: \( \# G_{\text{optimal}} \) is the configuration for executing \( S^{\text{max}} \)
26: \( \# \) Output PO size, cost, time and configuration
27: \textbf{for} each tuple in \text{list.S}^{\text{max}} \textbf{do}
28: \( \text{output } S^{\text{max}}, C', T', G_{\text{optimal}} \)
29: \textbf{end for}
30: \textbf{end for}
Thus, we first show that the chosen applications exhibit a range of scaling functions through workload characterization. Secondly, we introduce a novel metric, “Performance Cost Ratio”, to capture the impact of resource capacity with respect to the cost of resource. Thirdly, we address the challenge of having a large cloud resource configuration space and show the application of our approach to determine Pareto-optimal problem sizes to execute the application within given cost budget and time deadline. Lastly, we present an analysis on scaling applications on cloud using predictions from our approach where we discuss the Pareto-optimal problem sizes, effect of cloud resource PCR, and, impact of time deadline and cost budget on Pareto-optimal problem sizes.

4.4.1 Experiment Setup

4.4.1.1 Applications

To evaluate our approach we selected two applications that exhibit linear, quadratic and logarithmic growth of resource demand with respect to problem size as shown in Table 4.2. More information about these applications are provided in Appendix A.

Table 4.2: Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Scaling Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-body</td>
<td>$W = 10^6n^2 + 5 \times 10^9n - 7 \times 10^9$</td>
</tr>
<tr>
<td></td>
<td>$W = 10^{12}s + 2 \times 10^{13}$</td>
</tr>
<tr>
<td>sand</td>
<td>$W = 3 \times 10^{12}n$</td>
</tr>
<tr>
<td></td>
<td>$W = 8 \times 10^{15}\ln(\tau) + 4 \times 10^{16}$</td>
</tr>
</tbody>
</table>

As depicted in Table 4.2, in $n$-body, $n$ scales quadratically and $s$ scales linearly while in sand, $n$ scales linearly and $\tau$ scales logarithmically. To determine scaling functions for each input parameter, we run baseline executions of $n$-body and sand on a non-virtualized Intel Xeon E5-2630 v4 server and measure the instruction
Figure 4.2: Performance Cost Ratio of EC2 Resources Types for \textit{n-body} and \textit{sand}

Table 4.3: Amazon EC2 Cloud Resource Types

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>vCPUs</th>
<th>Freq. (GHz)</th>
<th>Mem. (GB)</th>
<th>Cost ($/hr)</th>
<th>Execution Rate (billion instructions/s/vCPU)</th>
<th>PCR (billion instructions/$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>2</td>
<td>2.9</td>
<td>3.75</td>
<td>0.105</td>
<td>1.38, 4.53</td>
<td>94556, 310629</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>2.9</td>
<td>7.5</td>
<td>0.21</td>
<td>1.38, 4.54</td>
<td>94338, 312804</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>2.9</td>
<td>15</td>
<td>0.42</td>
<td>1.37, 4.56</td>
<td>93640, 313248</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
<td>2.3</td>
<td>8</td>
<td>0.133</td>
<td>1.16, 4.83</td>
<td>62764, 261474</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>2.3</td>
<td>16</td>
<td>0.266</td>
<td>1.24, 4.85</td>
<td>62331, 262556</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>2.3</td>
<td>32</td>
<td>0.532</td>
<td>1.23, 4.88</td>
<td>66617, 264010</td>
</tr>
<tr>
<td>r3.large</td>
<td>2</td>
<td>2.5</td>
<td>15</td>
<td>0.166</td>
<td>1.14, 3.69</td>
<td>49500, 169048</td>
</tr>
<tr>
<td>r3.xlarge</td>
<td>4</td>
<td>2.5</td>
<td>30.5</td>
<td>0.332</td>
<td>1.12, 3.69</td>
<td>48391, 159568</td>
</tr>
<tr>
<td>r3.2xlarge</td>
<td>8</td>
<td>2.5</td>
<td>61</td>
<td>0.664</td>
<td>1.11, 3.71</td>
<td>48194, 160779</td>
</tr>
</tbody>
</table>
count using Linux *perf* utility. For *n-body* we used *n* range from 8,192 to 65,536 masses and *s* range from 1000 to 16,000 steps. For sand, *n* range was from 1 million to 64 million sequences and *τ* range was from 0.01 to 1.

### 4.4.1.2 Cloud Resources and Performance Cost Ratio

As cloud resources are becoming more varied in terms of compute capacity, memory, and other performance factors, it is imperative to have a uniform metric for evaluating cost-performance across the wide spectrum of cloud resources. To aid this we introduce Performance Cost Ratio (PCR) as a novel metric to represent cost efficiency of cloud resources. PCR is defined as

\[
PCR = \frac{\text{instruction execution rate}}{\text{cost per unit time}}
\]

Amazon EC2 cloud resources are divided into five main categories, compute intensive (*c4*), general purpose (*m4*), storage optimized (*i3*), memory optimized (*r3*) and accelerated computing units (GPUs). For the scope of this chapter, we consider only non-accelerated resource categories and applications that do not require GPUs. Resource categories in the Amazon EC2 cloud are further divided into resource types according to the number of virtual CPUs (vCPU) they possess. For example, *large*, *xlarge* and *2xlarge* resource types possess two, four and eight vCPUs respectively.

PCR of four non-accelerated resource categories on Amazon EC2 cloud for *n-body* and *sand* applications are shown in Figure 4.2. There is a large gap in PCR across resource categories and resource types within a resource category have almost similar PCRs. For example, PCRs of *c4* resources are two times higher compared to PCRs for *r3* resources while *c4.large*, *c4.xlarge*, *c4.2large* resource types which belong in the same resource category have PCRs of 94556,
94338, 93640 billion instructions per dollar and 310629, 312804, 313248 billion instructions per dollar for n-body and sand respectively. Moreover, we observe that the cost efficiency in terms of PCR has a non-linear relationship across different Amazon EC2 resource categories.

For evaluation of our approach, while preserving the non-linear relation in PCR across resource categories, we selected \( c_4, m_4 \) and \( r_3 \) resource categories. Table B.3 summarizes the characterization of selected resource types. From each category, we selected large, xlarge, 2xlarge resource types. We motivate the need and use of proposed model by applying to a large cloud resource configuration space consisting of over ten million\(^1\) configurations.

### 4.4.1.3 Validation

We present the validation results of our model in Table 4.4 where the predicted time and cost are compared against measured values from Amazon EC2 cloud. Due to limited research budget, we validate only the maximum and minimum problem sizes for each application shown in Figures 4.3, 4.4, 4.5 and 4.6. Out of the validated cases, the prediction accuracy varies in the range 81% - 82% and 84% - 87%, for n-body and sand respectively. There are few sources of inaccuracy that may have contributed to prediction error. Firstly, the predictions are based on the scaling function determined by a curve fitting approach as described in Section 4.3.2. We use measurements on a single non-virtualized server to determine the growth function, thus, may introduce some error when problem size is predicted for a distributed execution. Secondly, in commercial clouds such as Amazon EC2, the underlying physical processor is shared among multiple cloud virtual machines (cloud instances) simultaneously, thus, the vCPUs are hyper threads and not physical CPU cores [99]. Thirdly, in our approach we assume that the system

\(^1\)Nine resource types with five nodes from each type = \(6^9 - 1 = 10,077,695\)
specification is the same for every cloud resource instance within a single cloud resource type. However, in Amazon EC2, there could be virtual machines with different system specifications mixed up within the same resource type. For example, $m4$ resource instances are delivered with 2.3 GHz Intel Xeon E5-2686 v4 (Broadwell) processors or 2.4 GHz Intel Xeon E5-2676 v3 (Haswell) processors [8]. Lastly, the inter-node communication in $n$-body and $sand$ also contributes to the prediction error as we do not model communication overhead in this work.

### Table 4.4: Model Validation

<table>
<thead>
<tr>
<th>Application</th>
<th>Scaling Parameter</th>
<th>Problem Size</th>
<th>Configuration†</th>
<th>Time (min)</th>
<th>Cost ($)</th>
<th>Model Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$-body</td>
<td>$n$</td>
<td>80445</td>
<td>${5.5,5.0,0.3,0.0,0.1}$</td>
<td>1415</td>
<td>1154</td>
<td>99.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>65536</td>
<td>${3.5,5.0,0.2,0.0,0.1}$</td>
<td>1218</td>
<td>1003</td>
<td>99.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6896</td>
<td>${4.5,5.0,0.0,0.0,0.1}$</td>
<td>1415</td>
<td>1149</td>
<td>99.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4000</td>
<td>${4.5,5.0,0.0,0.0,0.1}$</td>
<td>1061</td>
<td>862</td>
<td>115.18</td>
</tr>
<tr>
<td>$sand$</td>
<td>$n$</td>
<td>9976m</td>
<td>${5.5,5.0,0.3,0.0,0.1}$</td>
<td>1413</td>
<td>1630</td>
<td>99.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4996m</td>
<td>${5.5,4.0,0.0,0.0,0.1}$</td>
<td>651</td>
<td>738</td>
<td>99.99</td>
</tr>
<tr>
<td></td>
<td>$\tau$</td>
<td>9.5441</td>
<td>${5.5,5.0,0.3,0.0,0.1}$</td>
<td>1415</td>
<td>1614</td>
<td>99.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1600</td>
<td>${4.5,5.1,4.5,5.0,0.0,0.1}$</td>
<td>752</td>
<td>876</td>
<td>71.21</td>
</tr>
</tbody>
</table>

† Configuration = \{c4.2xlarge, c4.xlarge, c4.large, m4.2xlarge, m4.xlarge, m4.large, r3.2xlarge, r3.xlarge, r3.large\}

### Model Analysis

We present an analysis based on model predictions for scaling applications on the Amazon EC2 cloud. We show that the proposed approach determines Pareto-optimal configurations maximizing the problem size, and minimizing the cost and time for a given time deadline and a cost budget. In section 4.4.2 where we investigate the Pareto-optimal problem sizes, for simplicity and due to research budget constraints, we set the time deadline to 24 hrs and cost budget to $100. In sections 4.4.3 and 4.4.4 where we discuss the impact of time deadline and cost budget on Pareto-optimal problem size, we set the time deadline to 7 days (168 hrs) and cost budget to $1000, hence emulating long-running applications.
4.4.2 Pareto-optimal Problem Size

To understand the relationship between the application problem size and the cost-time performance of cloud resources, we investigate how Pareto-optimal problem sizes are distributed in the cost-time-size space. In Figures 4.3,4.4,4.5 and 4.6, we plot Pareto-optimal sizes for \( n \)-body and sand applications. For \( n \)-body, to scale \( n \), we fix \( s \) at 4000 steps and to scale \( s \), we fix \( n \) at 64k masses. For sand, to scale \( n \), we fix \( \tau \) at 0.16 and to scale \( \tau \) we fix \( n \) at 4096 million. Only Pareto-optimal sizes greater than \( n \)-body (64k, 4000) and sand (4096m, 0.16) are shown in the plot.

Figure 4.3: Pareto-optimal Size for \( n \)-body (64k, \( s \)) 24hrs Time Deadline and $100 Cost Budget

**Observation 1:**

*Given an application with a cost budget and time deadline there exist one or more Pareto-optimal problem sizes of which one consists of the largest problem size.*

As shown in In Figures 4.3,4.4,4.5 and 4.6, there is a large number of Pareto-optimal sizes for both applications. Since scaling functions are different for each
Chapter 4. Scaling Application Problem Size

Figure 4.4: Pareto-optimal Size for \( n\text{-}body \ (n, 4000) \) 24hrs Time Deadline and $100 Cost Budget

Figure 4.5: Pareto-optimal Size for \( \text{sand} \ (n, 0.16) \) 24hrs Time Deadline and $100 Cost Budget
scaling parameter as listed in the Table 4.2, the plots take distinct shapes. Given our total cloud resource configuration space of about 10 million configurations, 24 hrs time deadline, and, time granularity of one minute, there are approximately 14 billion feasible problem sizes\(^2\). From this large number of feasible problem sizes, our approach filters out 33470 and 26811 Pareto optimal sizes for \(n\)-body \(s\) and \(n\) parameters respectively. The number of Pareto-optimal sizes are 47089 and 25240 for \(sand\) \(n\) and \(\tau\) parameters respectively. Among Pareto-optimal sizes, there is a large span in the largest Pareto-optimal size of application within the time deadline and cost budget. As shown in Figure 4.3 and Figure 4.4, among Pareto-optimal sizes of \(n\)-body, \(s\) spans from 4000 to 6896 while \(n\) spans from 65536 to 80445. Likewise, as shown in Figure 4.5 and Figure 4.6, in \(sand\), \(s\) spans from 4096m to 9976m whereas \(\tau\) spans from 0.16 to 0.53. Cost of smallest and largest Pareto-optimal sizes ranges between $57 to $100 and $66 to $100 for \(n\)-body \(s\) and \(n\) respectively, whereas cost ranges between $40.5 and $71.2 for \(sand\) \(n\) and \(\tau\) respectively.

\(^2\)Assuming per-minute charging: \(10,077,695\text{ configurations} \times 60 \times 24\text{ minutes} = 14,511,880,800\)
Having a set of Pareto-optimal problem sizes within the predefined cost and time constraints provide opportunity for cloud consumer to further tighten these constraints at the expense of the problem size of the application.

Observation 2:

*Increasing the cost budget and relaxing the time deadline does not always result in obtaining a larger problem size.*

Surprisingly, as shown in Figures 4.3, 4.4, 4.5 and 4.6, we observe that the smallest Pareto-optimal size do not correspond to smallest time and cost. For example, in $n$-body $s$, the smallest Pareto-optimal size is at 1357 min and $57$ whereas for $n$, the minimum Pareto-optimal size is at 1106 min and $66$. For sand $n$ and $\tau$, the minimum Pareto-optimal sizes are at 651 min and $40.5$, and, 752 min and $71.2$ respectively.

Blindly selecting the cloud resource configuration may result in a smaller problem size even with larger execution time and higher cost. Thus, when scaling decisions are made, the cost-performance of different resource combinations have to be considered without blindly relaxing cost budget and time deadline constraints.

Observation 3:

*For a given Pareto-optimal problem size with multiple resource configurations, resource demand is allocated to different resource types in order of higher PCR.*

When there are multiple resource types with different PCR, it is always better to go for resources types with highest PCR. If there is no upper bound on the number of nodes available from a resource category, having a homogeneous configuration with resources with highest PCR yields the optimal cost-time performance. To study the effect of PCR on scaling, we consider resource configurations
Chapter 4. Scaling Application Problem Size

of Pareto-optimal problem sizes. We observe that in configurations associated with Pareto-optimal sizes, there is a bias towards certain resource categories. For example, out of three resource categories we selected from Amazon EC2 cloud, c4 category nodes tend to fill in first followed by m4 and r3. This behavior could be explained using PCR of resource categories. As shown in Figure 4.2, c4 is the resource category with highest PCR followed by m4 and r3. Our approach considers all possible resource configurations in order to determine the Pareto-optimal sizes, thus it is safe to say that resource configurations associated with Pareto-optimal sizes consist of instances of high-PCR resource types. Therefore, selecting the resources in the order of the highest PCR would obtain a configuration with a near-optimal problem size.

4.4.3 Impact of Time Deadline on Scaling

The change in largest problem size when the time deadline constraint is relaxed relates to the scaling function of the application. To investigate the impact of time deadline on scaling, as shown in Figures 4.7 - 4.10, we determine the largest size of the application executable for n-body and sand while relaxing the time deadline from 1 day to 7 days for different fixed cost values of $250, $500, $750, $1000.

Observation 4:

Among Pareto-optimal problem sizes, tightening of time deadline results in a proportionately smaller reduction in largest problem size.

The largest problem sizes of both n-body s and sand n exhibit a similar growth as shown in Figures 4.7 and 4.9. Both of these parameters have a linear relationship with the resource demand of the application. For parameter n in n-body of which the relationship with resource demand is of quadratic in nature, we observe a sublinear growth in largest problem size when the time deadline is relaxed as
Chapter 4. Scaling Application Problem Size

Figure 4.7: Impact of Time Deadline on Largest Size of $n$-body ($64k, s$)

Figure 4.8: Impact of Time Deadline on Largest Size of $n$-body ($n, 4000$)
Figure 4.9: Impact of Time Deadline on Largest Size of $sand\ (n, 0.16)$

Figure 4.10: Impact of Time Deadline on Largest Size of $sand\ (4096m, \tau)$
shown in Figure 4.8. When the time deadline is relaxed, the capability of resources increase linearly, but the resource demand increases quadratically. Hence, results in a sublinear growth in largest problem size. The feasible region for $\tau$ in sand is between 0 to 1. Thus, the curves in Figure 4.10 for $500, $750 and $1000 reaches the maximum value 1 at 2 days.

To investigate the cause for the changes of gradient along the curve, we take a closer look at the cloud resource configurations at points where the gradient change occurs. For simplicity, we consider n-body application’s $s$ parameter which has a linear relationship with workload. We expect $s$ to grow linearly when time deadline is relaxed, but, as shown in Figure 4.7, the gradient changes along the curve. If we consider $500$ curve, there are visible changes of gradient at 2 days and 6 days. Thus we take a closer look at the configurations before and after 48hrs and 144 hrs. The resource configurations at 2 days and 3 days are $\{5,5,5,5,5,4,4,0\}$ and $\{5,5,5,3,5,4,0,0,0\}$ respectively. (In a configuration, the first three types belong to $c4$ category, second three values belong to $m4$ category, and, last three types belong to $r3$ category.) Thus, at 2 days, the configuration utilizes all three categories, but, when the deadline increases to 3 days, the configuration is filled only with resources from only two categories ($c4$ and $m4$), losing all $r3$ nodes. Similarly, when going from 120 hrs to 144 hrs, the configuration changes from $\{5,5,5,4,0,0,0,0\}$ to $\{5,5,5,0,0,0,0,0\}$ losing $m4$ nodes in the configuration. As illustrated in Figure 4.2 there is a considerably large performance gap between resource categories. This change of resource categories in the configuration causes the overall performance capacity growth trend of the configuration to change. As a result, when the time deadline is tightened, the reduction of largest problem size is relatively smaller. For example, for n-body $s$ parameter when the cost budget is fixed at $750$, tightening of time deadline by half results in less than one-fifth reduction of the largest Pareto-optimal problem size.
This sub-linear relationship between the largest Pareto-optimal size and the time
deadline could be leveraged by the cloud consumer to further tighten the time
deadline with minimal trade-off of the application problem size.

4.4.4 Impact of Cost Budget on Scaling

To understand scaling the problem size when the time deadline is fixed in the
context of cloud, we investigate the impact on the largest executable problem size
when the cost budget is relaxed while keeping the time deadline fixed. As shown in
Figures 4.11, 4.12, 4.13 and 4.14, we determine the largest Pareto-optimal problem
size of the application for n-body and sand while increasing the cost budget from
$250 to $1000 for fixed-time deadlines of 1 day, 2 days, 3 days, 4 days, 5 days, 6
days and 7 days.
Chapter 4. Scaling Application Problem Size

Figure 4.12: Impact of Cost Budget on Largest Size of \textit{n-body} ($n$, 4000)

Figure 4.13: Impact of Cost Budget on Largest Size of \textit{sand} ($n$, 0.16)
Observation 5:

Among Pareto-optimal problem sizes, cost grows faster than the increase of problem size because PCR is not linear across all resource types.

Similar to the case when the time deadline is relaxed, we observe correlation between the growth of largest Pareto-optimal problem size and the scaling function of the application. Moreover, we observe deviation of the curve from the expected trend at multiple points along the curve, similar to the observation in explained in Section 4.4.3. For example, as shown in Figures 4.11 and 4.12, there is a sharp deviation of 2 days curve at $750. After $750, the curve becomes flat. Similar deviation can be observed in Figure 4.13. The resource configuration at this point is \{5,5,5,5,5,5,5,5\}, meaning that all resources are used up, thus no further scaling is possible.

Furthermore, we observe that even for an application parameter with a linear relationship with resource demand, the improvement of largest problem size while increasing cost budget is not linear. For example, for \textit{n-body} 3 days deadline, a
two-fold increase of cost from $500 to $1000, improves $s$ by only two-thirds. This can be explained by considering the PCR for resource categories. When more resources are added to the configuration considering the cost-time efficiency of resources, the configuration would first be filled with resources with high PCR and move towards resource categories with low PCR once the high-PCR resources are not available. Therefore, the cost-efficiency of the resource configuration reduces gradually. While cloud poses itself as a platform for fixed-time scaling of applications, with these observations, we reiterate the importance of considering the cost-efficiency of cloud resources as well as the proposed PCR metric for a consumer to efficiently scale their application on cloud.

### 4.4.5 Resource Allocation with PCR

![Graph showing resource allocation with PCR](image)

Figure 4.15: Resource Allocation with PCR - $n$-body ($64k$, 8000)

To further investigate the effectiveness of using the PCR metric for allocating resources for a given application problem size, we compare the Pareto-optimal
resource configurations obtained with the brute force method with the resource configurations obtained by using PCR. When allocating resources with PCR, we first compute the PCR of each resource type for the given application and then allocate resources starting from the resource type with the highest PCR to the lowest. Figure 4.15 shows the actual Pareto frontier and the resources configurations obtained using PCR for $n$-body (64k, 8000) application with a cost budget of 200$ and time deadline of 24 hrs.

**Observation 6:**

*PCR obtained resource configurations are close to the real Pareto-frontier.*

As shown in Figure 4.15, there are 56 PCR obtained resource configurations whereas there are only 23 resource configurations in the real Pareto frontier. The cost ranges are 136$ - 173$ and 126$ - 167$ for PCR obtained configurations and Pareto-optimal configurations respectively, whereas the corresponding time ranges are 11.4 hrs and 23.7 hrs and 11.5 hrs - 23.7 hrs. Thus, although the configurations obtained with PCR do not exactly overlap with the real Pareto frontier, PCR based resource allocation can be used to obtain an approximate Pareto frontier. PCR based allocation is a greedy approach with logarithmic computational complexity (due to sorting) whereas the naive Pareto-optimization requires generating all different resource configurations which includes a Cartesian product operation resulting in exponential time complexity.

### 4.5 Summary

Clouds provide a theoretically infinite pool of resources with resource scalability and pay-per-use charging, thus, makes a suitable platform for executing scalable applications. However, making scaling decisions on cloud is challenging due to
the large configuration space and application dependent resource demand growth functions. This chapter presents a measurement-driven analytical modeling approach to determine the largest Pareto-optimal problem sizes executable for a given application with a time deadline and a cost budget, and, cloud resource configurations to execute them. We utilize baseline measurements from a non-virtualized server and on-cloud resources to determine the resource demand growth functions of applications with respect to problem size for different scaling parameters, and to characterize the performance of cloud resources. We evaluate our approach on Amazon EC2 cloud with more than ten million resource configurations and a subset of representative applications with different resource demand growth. Validation against time and cost measurements from Amazon EC2 shows that our model predictions are more than 81% accurate.

Using model results, we show the existence of cost-time-problem size Pareto frontier with multiple Pareto-optimal problem sizes for a given application with a time deadline and cost budget. Contrary to intuition, our model results show that relaxing the time deadline and putting in more cost budget does not guarantee execution of larger problem sizes. We introduce Performance Cost Ratio (PCR) metric to characterize the cost-performance of cloud resources and show that PCR can be utilized to determine cloud resource configurations for executing near-optimal largest Pareto-optimal problem sizes. We investigate the impact of scaling application problem size with a fixed time deadline on the cloud and show that the relationship between the Pareto-optimal problem size and cost is non-linear. Thus, when resources with different PCRs are used, the cost of execution grows faster than the increase in workload. Furthermore, we show that when the cost budget is fixed, the time deadline could be tightened for a relatively smaller reduction in problem size. For example, two-fold tightening of time deadline results in only a one-fifth reduction in largest Pareto-optimal problem size executable for an $n$-body...
simulation.
Chapter 5

Scaling Application Accuracy

Using a measurement-driven approach, this chapter investigates the impact of scaling application accuracy on fixed-cost scaling on cloud. Especially, we focus on applications where the accuracy of output could be traded-off for reduced execution time. To demonstrate the impact of fixed-cost scaling and the cost-accuracy and time-accuracy performance, we select Convolution Neural Networks (CNNs) as an example. Firstly, we define fixed-cost scaling with changing application accuracy in the cloud context. Secondly, we present our measurement-driven approach and time and cost models for determining inference time and cost followed by defining of metrics to represent cost and time performance with respect to accuracy. Thirdly, we present our evaluation containing an analysis of experiment results from our measurement-driven approach and discuss useful insights on fixed-cost scaling of applications with changing accuracy.

5.1 Fixed-Cost Scaling with Changing Accuracy

Cloud computing can be seen as the epitome of computing as a utility. Like any other utility, eg. electricity and water, cloud consumers are continuously
charged based on their resource usage. However, due to business constraints, cloud applications are often tied to cost budget. To ensure a given cloud application is completed within a fixed cost budget, we have a choice between two options (i) changing to a cheaper resource configuration or (ii) changing the application output’s accuracy.

Changing the resource configuration may result in longer execution time due to cheaper resources being less efficient compared to expensive ones. We explored this case in chapter 3 of this thesis. When it comes to changing the accuracy of application output, the trade-off is between the execution time and accuracy. Since execution time on cloud directly translates into the cost on cloud, changing accuracy directly impacts cost. Intuitively, the cloud resource configuration does not need to be changed to reduce cost when the accuracy of the application is traded-off in return for cost savings. However, there may be multiple cloud resource configurations to achieve the same accuracy. In this chapter, we investigate the cost-performance of scaling application accuracy using Convolution Neural Networks (CNNs), a popular machine learning application, as an example.

5.2 Convolution Neural Networks

Big data explosion and advancements in large applications such as real-time image classification, natural language processing, among others, pose new challenges for enterprises. There is a growing demand for compute power and parallel processing with the need to deliver results in real time or within expected time of Internet based applications. For example, in image detection and filtering processing in social media platforms in January 2019, Facebook social media network saw as many as 350 million photo uploads per day [21]. Increasingly, before these photos are published they go through a filtering process to determine whether they com-
Chapter 5. Scaling Application Accuracy

ply with the rules and regulations [29]. This filtering process has to be completed in near real-time speed for photos to appear on their profiles almost immediately. Today, Convolution Neural Networks (CNNs) are widely used in such applications. As CNNs perform a lot of computations, mainly convolutions, specialized hardware such as Graphic Processing Units (GPUs), Tensor Processing Units and Field Programmable Gate Arrays (FPGAs) are used to improve real-time performance. On the other hand, unlike traditional applications that produced exact results, CNNs output results with an associated accuracy percentage. Thus the expectation is to produce a close-enough result [40]. In image filtering on social media, it would be good enough to say that a given image is violating the rules with a 75% probability, so that the image could be forwarded for manual review.

A CNN application is generally divided into two main phases; training and inference. For example, in an image classification CNN, during the training phase the CNN’s parameters are tuned with labeled images and produces a trained CNN model. In inference phase, new non-labeled images are input into the trained model to get it classified into one of the labeled that the CNN was trained for. Although training is important and could take a long time to process, it is not performed frequently. In contrast, new inferences are initiated for new images by a multitude of users in parallel. For social media image classification example, once a model is trained, an inference process has to executed to classify each image input. In contrast to model training, we focus on inference queries that are performed frequently.

5.3 Cost-Accuracy Performance on Cloud

To investigate the trade-off between the cost and accuracy of applications on cloud, we propose a measurement driven approach and use image classification
CNN applications, Caffenet and Googlenet as example cloud applications. To vary the accuracy of CNN inference, we change the parameters in CNN layers thereby resulting in sparse layers. This technique is called pruning. The combination of layers pruned by different degrees results in versions of the CNNs that produce classification results with different accuracy. While pruning has been well-studied with respect to CNN algorithms, the impact of tuning such applications on different cloud configurations is non-trivial. Using a measurement-based analytical model we determine cost-accuracy and time-accuracy Pareto-optimal cloud resource configurations.

In contrast to majority of existing work that focus of the training performance of CNNs, we focus on inference performance. CNN training requires more time and resources and training performance is a major concern for application developers as well as cloud service providers. However, training is largely a one time job which creates the trained CNN. There are incremental training approaches that require successive training with new data to improve the accuracy of the CNN. Even in such incremental approaches, the frequency of conducting training is insignificant compared to frequency of CNN inference. It is true that a single CNN inference consumes much less resources and needs much less time compared to a single instance of training. However, due to its continuous and frequent invocation, inference performance is a major concern for a cloud consumer. Moreover, although there have been recent work in distributed CNN inference [106], as such approaches are still in the early research stage, we do not focus on distributed CNN inference in our performance study. In our approach, measurement results are obtained by running 50,000 unique inferences on Caffenet and Googlenet CNN image classification models that were trained with 1.2 million images on Amazon EC2 cloud instances with GPU for parallel processing.

Investigating the fixed-cost scaling with changing application accuracy, this
Chapter 5. Scaling Application Accuracy

The chapter makes the following contributions:

1. We present a measurement-driven model for investigating the trade-offs between time and accuracy, and, cost and accuracy, and show the existence of “sweet-spot” regions where the time and cost could be reduced with no reduction in application accuracy.

2. We show that multi-layer pruning is effective in reducing the inference time with minimal accuracy drop. For example, for Caffenet CNN on Amazon EC2, the inference time could be halved with just one-tenth drop in accuracy.

3. We show the existence of cost-accuracy and time-accuracy Pareto-optimal configurations spanning considerably large time and cost range, and opportunity to reduce cost and time. Selecting the right degree of pruning and resource configuration reduces execution time by up to 50% and cost by up to 55% for obtaining highest possible inference accuracy for Caffenet.

4. We show that quantifying cost-accuracy and time-accuracy performance measured in Time Accuracy Ratio (TAR) and Cost Accuracy Ratio (CAR) is important in selecting efficient application and resource configurations, and demonstrate their usage as heuristics in a polynomial-time cloud configuration determination algorithm.

5.4 Approach

In this section, we present our approach which consists of (i) application characterization, (ii) a measurement-driven analysis and (iii) a measurement-driven analytical modeling approach for determining cloud resource configurations for a given accuracy within time and cost deadline constraints, as illustrated in Figure 5.1.
5.4.1 Overview

Given an application which produces results of different accuracy for different application configurations, and a set of cloud resources, the objective of our approach is to understand the effect of application accuracy on execution time and cost on cloud. The approach consists of three main stages. Firstly, to characterize the application we conduct a baseline execution and determine various experiment parameters. Secondly, we take time measurements for different application accuracy. Thirdly, we input these measurements into our analytical models for determining execution time and cost for different resource configurations. Cloud resource configurations are filtered using a Pareto optimization filter to determine time and cost Pareto optimal cloud configurations. Finally, with the measurements and analytical model predictions, we derive insights on the time-accuracy and cost-accuracy trade-offs in executing applications on cloud. Since we selected CNN as the representative application, we describe our approach in detail in the context of CNNs.

5.4.2 Application Characterization

To understand the CNN and to fix experiment parameters such as the number of parallel inferences, we characterize the CNN on cloud. Firstly, to understand the contribution of each CNN layer towards inference time, we analyze the execution
time distribution for each layer. Secondly, to determine whether there is still an opportunity to improve inference time, we evaluate the time taken for a single inference and how it changes with pruning. Thirdly, to ensure full utilization of the compute resources, we determine the maximum number parallel inferences on a given cloud resource.

5.4.2.1 Pruning of CNNs

CNN layers are stored in the memory as matrices. The CNN training process adjusts the values of the elements of these matrices in order to tune the CNN for the highest accuracy resulting in dense matrices. Pruning removes (changes to zero) selected elements in these matrices. There are different algorithms to determine the parameters (elements) that need to be pruned. Li et al. [55] remove entire regions of convolution layers instead of individual parameters based on L1 norm. Anvar et al. [15] present a similar approach to [55], but with a more complex scoring algorithm to rank parameters. Huang et al. [45] from Nvidia propose modeling the pruning problem as a combinatorial optimization problem with a cost function. Their objective is to determine the subset of parameters that minimizes the cost function when pruned. When CNN layers are pruned, the pruning algorithm selectively changes the matrices’ elements to zero, resulting in sparse matrices. For simplicity and implementation convenience, we use the pruning tools developed by Li et al. [55] in this work. Resulting sparse matrices can be efficiently processed with special sparse matrix computational libraries. In this work, we use an extended version of Caffe framework [101] for efficient sparse matrix computation.
5.4.2.2 Accuracy of CNN Inference

The accuracy of CNN inference refers to the percentage of correct predictions that the CNN does. For example, for an image classification CNN, the accuracy is the ratio between the number of images the CNN determined the correct label and the total number of images given as input for classifying. Keeping to the standards set by the machine learning research community [59,82], in this work, we use two widely used accuracy metrics as follows.

- Top 1 accuracy: The percentage of the number of times that the class with the highest probability is the expected class for the given input.

- Top 5 accuracy: The percentage of the number of times that the expected class for the given input is one of the 5 classes with the highest probability.

5.4.3 Measurements

For our analysis, we use measurements to determine both execution time and cost for CNN inference on a cloud resource instance for different accuracy. Firstly, for measuring the execution time, we execute CNN inference and record the time. To change the inference time, convolution layers of the CNN are pruned in different degrees. Secondly, to compute the cost, we retrieve the cost per unit time from the cloud provider and compute the inference cost by multiplying the unit cost of the cloud resources used by inference time. Thirdly, to compute the Top1 and Top5 accuracy, we count the number of accurate inferences and compute the metrics as defined in Section 5.4.2.2. Finally, we compute the TAR and CAR for each application configuration using the formula shown in Section 5.4.5. At the end of the measurement phase, we output a list of degrees of pruning with their inference time, cost, TAR, and CAR. To minimize the measurement error, we run each
Table 5.1: Symbols Used

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>a CNN application</td>
</tr>
<tr>
<td>$P$</td>
<td>set of $A$ pruned with different degrees of pruning</td>
</tr>
<tr>
<td>$p$</td>
<td>a degree of pruning in $P$</td>
</tr>
<tr>
<td>$a_p$</td>
<td>accuracy of $p$</td>
</tr>
<tr>
<td>$W$</td>
<td>number of images for inference</td>
</tr>
<tr>
<td>$n$</td>
<td>number of batches</td>
</tr>
</tbody>
</table>

Cloud Resources

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G$</td>
<td>set of all cloud resources</td>
</tr>
<tr>
<td>$R$</td>
<td>a cloud resource configuration of $G$</td>
</tr>
<tr>
<td>$i$</td>
<td>a cloud resource type in $R$</td>
</tr>
<tr>
<td>$v_i$</td>
<td>number of GPUs in $i$</td>
</tr>
<tr>
<td>$c_i$</td>
<td>cost per unit time for $i$</td>
</tr>
<tr>
<td>$b_i$</td>
<td>max parallel inference (batch size) of $i$</td>
</tr>
</tbody>
</table>

Model

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C'$</td>
<td>cost budget for inference of $W$</td>
</tr>
<tr>
<td>$T'$</td>
<td>time deadline for inference of $W$</td>
</tr>
<tr>
<td>$C$</td>
<td>total cost for inference of $W$</td>
</tr>
<tr>
<td>$T$</td>
<td>total time for inference of $W$</td>
</tr>
<tr>
<td>$t_{b,a_p}$</td>
<td>time for inferring $b$ with $a_p$</td>
</tr>
</tbody>
</table>

experiment three times and record the minimum time measurement.

5.4.4 Time and Cost Models

This section presents the derivation of our analytical models for determining execution cost and time on cloud resources for a given CNN with different degrees of pruning.

The CNN inference cost on cloud depends on the total inference time and the cost per unit time of the cloud resource configuration.

\[
C = T \sum_{i=0}^{\left| R \right|-1} c_i \tag{5.1}
\]

where $T$ is the inference time, and $c_i$ is the cost per unit time for cloud resource $i$. 

111
$T$ is ratio between the number of batches to be inferred and the time for inferring one batch for a given accuracy.

$$T = \max \left( \frac{n}{t_{b,a}} \right)$$ (5.2)

where $n$ is the number of batches and $t_{b,a}$ denotes the time for a single batch inference for a batch size $b$ and accuracy $a$. The inference accuracy of a CNN depends on its degree of pruning as described above in section 5.4.2.1. $n$ relies on the maximum number of parallel inferences possible on the GPU device (the batch size).

$$n = \frac{W}{b}$$ (5.3)

where $W$ is the total number of images to be inferred.

Among cloud resources in a given cloud resource configuration $R$, inference images are distributed as follows. If $W_i$ denotes the number of inference images for resource $i$,

$$W_i = \frac{W}{\sum_{j=1}^{[R]} v_j} v_i$$ (5.4)

Configurations predicted by the models are sent through Pareto-optimization to filter cost-accuracy and time-accuracy Pareto optimal configurations that satisfy time deadline $T'$ and cost budget $C'$.

5.4.5 Time Accuracy Ratio and Cost Accuracy Ratio

To quantify the trade-offs between time and accuracy, and, cost and accuracy, we introduce two metrics (i) Time Accuracy Ratio (TAR) and (ii) Cost Accuracy Ratio (CAR).

TAR is defined as

$$TAR = \frac{t}{a}$$
Chapter 5. Scaling Application Accuracy

where \( t \) is the inference time and \( a \) denotes inference accuracy.

Similarly, CAR is defined as

\[
CAR = \frac{c}{a}
\]

where \( c \) is the cost incurred on cloud to achieve inference accuracy \( a \).

TAR and CAR represent the time and cost respectively, for achieving a single unit of accuracy where \( t, c \in (0, \infty) \) and \( a \in [0, 1] \). Higher TAR value means that the time required to achieve a unit of accuracy is higher and higher CAR means that it incurs a higher amount of cost to achieve a unit of accuracy. Hence, for both these metrics a lower value indicates better performance when comparing system configurations.

5.5 Evaluation

In this section, we present the evaluation of our approach. Firstly, we present the experiment setup including application and cloud resources. Secondly, we show the impact of changing accuracy on the execution time on cloud. Thirdly, we present the impact of changing accuracy on execution cost. Finally, we present two metrics for quantifying the accuracy performance with respect to time and cost and discuss their usage.

5.5.1 Experiment Setup

5.5.1.1 Application

We selected two widely used image classification CNNs, namely Caffenet and Googlenet [96] implemented on caffe machine learning framework as representative applications for our analysis. Caffenet is a Caffe implementation of Alexnet [53] CNN model. As shown in Figure 5.2, Caffenet consists of five convolution layers
and three fully connected layers while Googlenet is much deeper with 56 convolution layers (two main convolution layers and nine “inception” layers each containing six convolution layers). The architecture of Googlenet is shown in Figure 5.3. As we are interested only in the inference phase, we obtained trained models of Caffenet and Googlenet trained with a subset of ImageNet [26] dataset containing about 1.2 million labeled images. This training dataset contains images belonging to 1000 categories with approximately 1000 images from each category. In total, Caffenet contains 650,000 neurons, 60 million parameters, and 630 million connections. A breakdown of the properties of each layer is shown in Table 5.2. \texttt{conv} is used to denote a convolution layer while \texttt{fc} denotes a fully-connected layer. The convolution layers are non-uniform in size, number of filters and the filter size. Convolution layer 1 is the largest in terms of image size and, convolution layer 2 comes next. The last three convolution layers are approximately equal in size. When it comes the fully-connected layers, unlike convolution layers which are three dimensional, these are vectors. Understandably the last fully-connected layer, \texttt{fc3}, has 1000 neurons that relate to the number of possible outputs of the model. Recall that the model is trained with a dataset having 1000 distinct image categories. In contrast to the Caffenet, despite bringing a deeper CNN, Googlenet has only 4 million parameters in its layers. We use the same RGB input image size of of size 224x224 pixels for both CNNs. We omit the details of Googlenet and its architecture diagram due to space constraints.

5.5.1.2 Cloud Resources

As CNN processing involves a large number of matrix operations, we evaluate our approach on GPU instances from Amazon EC2 cloud. As shown in Table 5.3, we select six cloud resource types spanning two resource categories, \texttt{p2} and \texttt{g3}, with GPUs from Amazon EC2 Oregon region. Both \texttt{p2} and \texttt{g3} resource instances
Chapter 5. Scaling Application Accuracy

Figure 5.2: Caffenet CNN Architecture [53]

Table 5.2: Caffenet Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Dimensions</th>
<th>Number of Filters</th>
<th>Filter Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>224 x 224 x 3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>conv 1</td>
<td>55 x 55 x 96</td>
<td>96</td>
<td>11 x 11 x 3</td>
</tr>
<tr>
<td>conv 2</td>
<td>27 x 27 x 256</td>
<td>256</td>
<td>5 x 5 x 48</td>
</tr>
<tr>
<td>conv 3</td>
<td>13 x 13 x 384</td>
<td>384</td>
<td>3 x 3 x 256</td>
</tr>
<tr>
<td>conv 4</td>
<td>13 x 13 x 384</td>
<td>384</td>
<td>3 x 3 x 192</td>
</tr>
<tr>
<td>conv 5</td>
<td>13 x 13 x 256</td>
<td>256</td>
<td>3 x 3 x 192</td>
</tr>
<tr>
<td>fc1</td>
<td>4096</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>fc2</td>
<td>4096</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>fc3</td>
<td>1000</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.3: Amazon EC2 Cloud Resource Types

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>vCPUs</th>
<th>GPUs</th>
<th>Mem (GB)</th>
<th>GPU Mem (GB)</th>
<th>Price ($/hr)</th>
<th>GPU Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2.xlarge</td>
<td>4</td>
<td>1</td>
<td>61</td>
<td>12</td>
<td>0.9</td>
<td>NVIDIA K80</td>
</tr>
<tr>
<td>p2.8xlarge</td>
<td>32</td>
<td>8</td>
<td>488</td>
<td>96</td>
<td>7.2</td>
<td></td>
</tr>
<tr>
<td>p2.16xlarge</td>
<td>64</td>
<td>16</td>
<td>732</td>
<td>19</td>
<td>14.4</td>
<td></td>
</tr>
<tr>
<td>g3.4xlarge</td>
<td>16</td>
<td>1</td>
<td>122</td>
<td>8</td>
<td>1.14</td>
<td>NVIDIA M60</td>
</tr>
<tr>
<td>g3.8xlarge</td>
<td>32</td>
<td>2</td>
<td>244</td>
<td>16</td>
<td>2.28</td>
<td></td>
</tr>
<tr>
<td>g3.16xlarge</td>
<td>64</td>
<td>4</td>
<td>488</td>
<td>32</td>
<td>4.56</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5. Scaling Application Accuracy

Figure 5.3: Googlenet CNN Architecture [96]
are powered by Intel Xeon E5-2686 v4 CPUs with 2.3GHz base frequency. p2 instances have NVIDIA k80 GPUs with 2,496 parallel processing cores while g3 instances are equipped with NVIDIA M60 GPUs with 2048 parallel processing cores. It is important to note that although the system specification states that a GPU is attached, it is a virtual GPU. For example, although a single NVIDIA k80 unit consists of two GPUs, p2.xlarge user gets access to only one of them. Moreover, when charging for resource usage, the hourly price mentioned in the specification is pro-rated to the nearest second.

5.5.2 Application Characterization

In application characterization, we focus on three parts. Firstly, we determine the execution time distribution across CNN layers to find the most impactful layers. Secondly, we investigate to which extent CNN inference time can be improved with respect to a single inference. Lastly, we determine the maximum number of parallel inferences (batch size) on the GPU device for maximum utilization.

5.5.2.1 Execution Time Distribution of CNN Layers

To apply pruning, we first determine the layers that have a large execution time by individually measuring the time taken by each layer on caffe framework during inference. Figure 5.4 shows the distribution of execution time for different layers for Caffenet. We observe that the convolution layers contribute more to the execution time compared to other layers and the contribution of each convolution

Figure 5.4: Caffenet Execution Time Distribution of CNN Layers
layer is directly proportional to the values of the image parameters in the convolution layers. As shown in Table 5.2, conv1 has the largest image size (Figure 5.2) and conv2 comes second. conv3, conv5, and conv5 have an approximately similar number of parameters. Following a similar pattern, the execution time contribution is 51% for conv1, and 16% for conv2 followed by 9%, 10% and 7% for the last three convolution layers respectively. The contribution of fully-connected layers is very small compared to convolution layers. Although fully connected layers are dense, as they are not involved in the expensive convolution operation, their contribution to the execution time is smaller. Googlenet exhibits a similar execution time distribution among its layers where 92% of the time spent on convolution layers as shown in Figure 5.5.

5.5.2.2 Has Inference Performance Hit the Wall?

To determine whether there is still scope for improvement in the inference time, we apply pruning for a single inference and study the impact on execution time. While varying the prune percentage from 0% to 90% uniformly across all convolution layers, we recorded the execution time for a single inference. The resulting plot is shown in Figure 5.6. We observe that the inference time decreases with pruning percentage for both CNNs. From 0% to 90% pruning, the execution time of a single inference drops by about half from 0.09 seconds to 0.05 seconds for Caffenet and about one thirds from 0.16 seconds to 0.1 seconds for Googlenet. Thus, we can safely assume that there is still scope for performance improvement in CNN
Chapter 5. Scaling Application Accuracy

inference.

5.5.2.3 Parallel Inference on GPU

A GPU consists of hundreds of compute cores that operate in parallel. To get the maximum power of GPUs, processing needs to be parallelized such that the GPU resources are fully utilized. A simple way of increasing GPU utilization in inference is by increasing the number of inferences that run in parallel (also referred to as batch size). Since each GPU has its own characteristics such as the number of cores and memory, it is important to determine the amount of parallelism required to fully utilize the GPU. Although in an on-premise system, a GPU specification would be good enough to estimate this, cloud GPU instance capabilities may vary from the specification due to virtualization, multi-tenancy and other restrictions on the cloud. To estimate this accurately, we experimentally determined the number of parallel inferences required for full utilization of GPU. The results on Amazon EC2 p2.xlarge instance with Nvidia k80 is shown in Figure 5.7. We

![Figure 5.6: Time for a Single Inference](image)
observe that the GPU saturates around 300 parallel inferences, thus it is safe to assume that GPU cloud instances will be sufficiently utilized when the number of parallel inferences is 300 or above.

5.5.3 Effect of Accuracy on Inference Time

To understand the effect of accuracy on CNN inference time, we investigate the change in execution time while changing accuracy. As we use pruning to control accuracy, we prune CNN layers with different degrees and measure the inference time and accuracy. As shown in Figure 5.4, since convolution layers account for more than 90% of the total inference time, we focus only on the convolution layers. Firstly, we discuss the effect of accuracy on execution time for a CNN model hosted on a single standalone resource with a single GPU. We focus on the impact of pruning individual CNN layers and having dependency among CNN layers. Secondly, we extend to CNN inference hosted on a cloud resource configuration with multiple cloud resources with many GPUs and discuss the impact of accuracy
Figure 5.8: Caffenet Effect of Changing Accuracy with Individual Layer Pruning
Figure 5.9: Googlenet Effect of Changing Accuracy with Individual Layer Pruning
5.5.3.1 Pruning Single Layer Only

Figures 5.8 and 5.9 illustrate the execution time, Top 1 and Top 5 inference accuracy for different pruning ratios for Caffenet and Googlenet, respectively. The sub-figures correspond to pruning of each convolution layer. Figure 5.9 shows only six selected convolution layers of Googlenet from different levels of the CNN. We observe a near-linear decrease in inference time with pruning for all convolution layers of Caffenet whereas Googlenet has decrease in time albeit some fluctuations. The largest reduction in time for Caffenet is when pruning conv2 layer with one-fourth drop from 19 mins to 14 mins whereas the smallest reduction is for conv1 layer where the drop is about one-eighth from 19 mins to 16.6 mins. Therefore, by pruning only a single layer, a significant reduction of inference time up to 25% can be achieved for Caffenet. Similarly, conv2–3x3 of Googlenet has the strongest impact among the selected six layers where the time is reduced by about 30% from 13 mins to 9 mins.

Observation 1:

There exist “sweet-spot” regions where inference time reduces with an insignificant drop in accuracy.

Despite the reduction of inference time, we observe that Top 1 and Top 5 accuracy curves in Figure 5.8 remain unchanged for a range of pruning ratios starting from 0%. For example, as illustrated in Figure 5.8, in Caffenet both Top 1 and Top 5 accuracy remain almost unchanged until the pruning ratio reaches 50%. After 50%, both curves demonstrate a gradual drop. However, it is evident that for the same range where accuracy remains unchanged, the inference time demonstrates a gradual decrease. A similar pattern could be observed in all convolution layers.
For the first six layers of Googlenet as shown in Figure 5.9, the accuracy starts dropping only after 60% of pruning while the time is seen drastically reduced. We call this region where accuracy remains almost unchanged while inference time reduces, sweet-spot region. Existence of sweet-spot regions is important for a cloud consumer to achieve the maximum possible inference accuracy with shorter time by selecting a degree of pruning within the sweet-spot region. For example, for Caffenet CNN inference on Amazon EC2 p2.xlarge instance, we can achieve up to one-eights reduction of inference time with no accuracy change, just by pruning \textit{conv2} layer by 50%.

\textbf{Observation 2:}

\textit{Impact of pruning on accuracy and execution time is different across convolution layers and does not directly correlate with convolution layer parameter values.}

We observe that the variation of Top 1 and Top 5 inference accuracy differs across convolution layers. As shown in Figure 5.8, we observe the largest change in accuracy is in \textit{conv1} layer where the Top 5 accuracy drops from 80% to 0% when the prune ratio varies from 0% to 90%. This is expected since \textit{conv1} is the layer that receives the input image and also the largest in terms of the size of the image surface parameter. Rest of the layers demonstrate Top 5 accuracy drop from 80% to around 25% for the same prune ratio range. Similar variation pattern is observed for Top 1 accuracy as well. When it comes to execution time, \textit{conv2} demonstrates the largest time range from 19 mins to 14 mins and \textit{conv1} demonstrates smallest execution time range from 19 mins to 16.5 mins. When comparing these observations with the values of the parameters in each layer, it is apparent that the accuracy and execution time variation does not directly relate to the parameter values in the layer. As shown in Table 5.2 \textit{conv4} in Caffenet contains the highest number of compute operations suggesting that pruning \textit{conv4} would
have the highest impact on accuracy and execution time. But our measurements show that this is not the case. Therefore, it is not trivial to determine how to select the best layer and pruning ratio for achieving the highest accuracy with the lowest execution time.

### 5.5.3.2 Pruning Multiple Layers

To understand the implications on time and accuracy due to dependency between convolution layers in the CNN, we combine sweet-spots from multiple layers and generate a single CNN model. We take guidance from Figure 5.4 which shows that in Caffenet, $\text{conv1}$ and $\text{conv2}$ account for more than 60% of the inference time. As shown in Figure 5.10, we compare three degrees of pruning; (i) no-pruning (nonpruned), (ii) pruning only $\text{conv1}$ and $\text{conv2}$ until the last sweet-spot ($\text{conv1-2}$) and (iii) pruning all convolution layers until the last sweet-spot (all-conv).

![Figure 5.10: Caffenet Effect of Changing Accuracy with Multi-Layer Pruning](image-url)
Observation 3:

*Combining CNN layer sweet-spots from multiple layers results in lower execution time, but may result in a drop in accuracy as well.*

Considering the Top5 accuracy, last sweet-spots in Caffenet with lowest exe-
execution time for \texttt{conv1} and \texttt{conv2} are at 30% and 50% prune ratios. As shown in Figure 5.10, \texttt{conv1-2} which has the first two layers pruned recorded an execution time of 13 mins. In comparison, when the layers are pruned individually, the corresponding execution times for \texttt{conv1} and \texttt{conv2} are 18.4 mins and 16.7 mins respectively. Thus, by pruning the two layers together, we achieve one-fifth reduction in execution time. When it comes to accuracy, \texttt{conv1-2} recorded 70% Top5 accuracy, a 10% drop from the original accuracy of 80%. When all five layers are pruned until their last sweet-spots, as shown in \texttt{all-conv}, the recorded execution time is 11 mins and the Top5 accuracy is 62%. Thus, pruning all layers resulted in one-third reduction of execution time and 18% drop in accuracy. Similar observations could be made on Top1 accuracy as well. Therefore, in addition to the impact of individual convolution layers, it is important to understand the effect of dependency between convolution layers.

Since there are multiple ways to prune a given CNN, there exist many degrees of pruning. Every degree of pruning is associated with an inference accuracy. Moreover, each of them can be hosted on many cloud resource configurations. Thus we investigate the impact of inference on accuracy on execution time. For simplicity, we focus on the simpler Caffenet CNN and select a set of 60 versions of Caffenet CNN pruned in different degrees spanning a wide accuracy range. We select a resource configuration space consisting of three Amazon EC2 resource types from \texttt{p2} category with three resource instances from each type and assume a time deadline of ten hours. Figure 5.11 is the resulting plot showing all feasible configurations for inferring one million images with Caffenet.
Figure 5.12: Impact of Accuracy on Cloud Cost
5.5.3.3 Time-Accuracy Pareto Frontier

**Observation 4:**

*Given a time deadline, there exist many feasible configurations for CNN inference. Among them there are multiple Pareto-optimal configurations.*

We observe that there 7654 feasible configurations for executing the Caffenet inference within the 10 hour time deadline. Among them, there are five Pareto-optimal configurations each for Top1 accuracy and Top5 accuracy. The Pareto-frontier is shown as a line in Figures 5.11 (a) and (b). Even among the Pareto-optimal configurations, Top1 accuracy varies from 27% to 53% whereas Top5 accuracy varies from 45% to 78% with execution time ranging from 3 to 5 hours in both cases. Selecting a Pareto-optimal configuration over other configurations significantly reduces the execution time. For example, as shown in Figure 5.11 (a), selecting the Pareto-optimal configuration with the highest accuracy reduces execution time by 50% compared to other configurations with the same accuracy.

5.5.4 Effect of Accuracy on Inference Cost

The effect of accuracy on cloud cost can be divided into two parts; (i) within a single resource type and (ii) across different resource types. Within a single resource type, the cost of a cloud resource instance solely depends on the time used. Thus, the cost incurred for CNN inference is a function of inference time. i.e. *inference cost = inference time* *cost per unit time*. However, when it comes to using a combination of resources the relationship becomes more complex since we have to take into the cost-accuracy performance of different cloud resources. Figure 5.12 shows the total number of configurations that are capable of executing Caffenet within a $300 cost budget.
Observation 5:

*Given a cost budget, there exist many feasible configurations for CNN inference including multiple cost-accuracy Pareto-optimal configurations.*

We observe that there are 1042 feasible configurations for executing the CaffeNet inference within the $300 cost budget. Among them, there are five Pareto-optimal configurations for both Top1 and Top5 accuracy. Moreover, among the Pareto-optimal configurations, the Top1 accuracy varies from 27% to 53% and Top5 accuracy varies from with cost ranging from $69 to $119. The cost-accuracy Pareto-frontier overlaps with time-accuracy Pareto-frontier due to cost being the restricting factor when allocating resources in both cases. Cloud consumer can save cost significantly by selecting Pareto-optimal configurations over the rest. For example, as observed from Figure 5.12 (a), selecting the Pareto-optimal configuration for the highest accuracy saves up to 55% cost.

Investigating cost-performance scaling of applications on cloud has been proposed by Han [41]. They model the performance of cloud applications based on arrival and service rates and propose algorithms to scale the performance appropriately based on a cost estimation. However, this approach can be applied only on a specific set of adaptive applications they call elastic algorithms. To circumvent this limitation, they propose techniques to transform existing applications (algorithms) into elastic algorithms before deploying them on the cloud. In contrast, our measurement-driven analytical modeling approach can be applied on applications without special modifications as long as the application inherently supports variable accuracy. The elastic algorithms proposed by Han are compatible and complementary to our scaling approach. Moreover, since our cost estimation is conducted based on measurements, our approach captures the application profile better as well as makes it convenient for a cloud consumer to apply for real world
5.5.5 Quantifying Accuracy Performance

As shown in Figures 5.11 and 5.12, there exist a series of configurations spanning vertically on time axis and cost axis respectively for a given accuracy. This behaviour is caused by having multiple degrees of pruning that yield same accuracy but with difference performance.

5.5.5.1 Time-Accuracy Performance

To understand the accuracy performance of the application, we plot the execution time of Caffenet degrees of pruning in Figure 5.13. We measure the inference time and accuracy by changing the prune ratio for conv1 and conv2 of Caffenet. Leveraging on the sweet-spot regions observed in Figure 5.8, we vary the prune ratio from 0% to 40% for conv1 and 0% to 50% for conv2 in 10% increments. The TAR value for each pruning configuration is labeled next to the configuration on the figure. Since TAR represents the relative change between time and accuracy, for a given accuracy, the configuration with a lower TAR gives the least inference time. Thus, TAR can be used to determine how the trade-offs offered by each degree of pruning compare among each other.

5.5.5.2 Cost-Accuracy Performance

The effect of accuracy on cloud cost can be divided into two parts; (i) within a single resource type and (ii) across different resource types.

Within a single resource type, the cost of a cloud resource instance solely depends on the time used. Thus, the cost incurred for CNN inference is a function of inference time. i.e. \( \text{inference cost} = \text{inference time} \times \text{cost per unit time} \). Therefore, TAR is sufficient to understand both the trade-offs between time and
Chapter 5. Scaling Application Accuracy

Figure 5.13: Time-Accuracy of Degrees of Pruning with TAR

Figure 5.14: Caffenet CAR Across Resource Types
accuracy and, cost and accuracy for a single cloud resource type. When it comes to a comparison between different resources types, as shown in Table 5.3, different cloud resource types have different cost and performance. Thus, to quantify the cost of execution of cloud resource with respect to accuracy, we use CAR metric.

Figure 5.14 shows computed CAR values for Caffenet with first two convolution layers pruned by 20%, across six types of cloud resource types from two resource categories, p2 and g3. Figure 5.14 (a) shows the CAR when all GPUs are allocated for inference and Figure 5.14 (b) shows when only one GPU is allocated. We observe that the CAR is approximately same for resource types within a resource category and varies across resource categories. For example, p2 has a CAR of approximately $0.57 whereas g3 has a CAR of approximately $0.35 with all GPUs allocated. When allocating cloud resources for CNN inference, it is ideal utilize all GPUs in the allocated resource. However there may be circumstances where the application restricts the number of GPUs it can utilize due to requirements such as memory and storage.

**Algorithm 5.1** Resource Allocation with TAR and CAR

1: sort $P$ in (i) accuracy descending order and (ii) TAR ascending order for elements with same accuracy
2: for each $p \in P$ do
3:   sort $G$ in ascending order of CAR
4:   $R \leftarrow \emptyset$
5:   for each $g \in G$ do
6:     $R \leftarrow g$ //add resource with lowest CAR
7:     distribute workload in $R$
8:     compute $T$ and $C$
9:     if $T < T'$ and $C < C'$ then
10:        return $p, R, T, C$
11:     end if
12: end for
13: end for
14: return $\emptyset$ //no feasible resource allocation
5.5.5.3 Efficient Cloud Resource Allocation

To determine the cloud resource configuration for executing CNN inference efficiently, we propose using TAR and CAR as a heuristics to decide on the order of resource allocation. Given a CNN pruned in different degrees resulting in a set of CNNs with different accuracy ($P$), a set cloud resource instances ($G$), and a time deadline ($T'$) and a cost budget ($C'$), Algorithm 5.1 illustrates our proposed resource allocation approach. The outputs are the resource configuration $R$, estimated time $T$ and estimated cost $C$.

First we order elements in $P$ by descending order of accuracy. If there are multiple elements with same accuracy, then those are ordered in ascending order of TAR. Then, the algorithm iterates through $P$ and allocates cloud resources in the ascending order starting from the lowest CAR until a resource configuration is found that completes the inference within the given time deadline and cost budget. Whenever the cloud resource configuration is changed, the workload distribution across the new resource configuration is performed as described in section 5.4.

Efficiency

A major challenge in exploring the cloud configuration space is due to the large computational time required [79]. Configuration space exploration is a non polynomial time problem and is upper bounded by $O(2^{|G|})$. By using CAR as a heuristic to pick resources greedily, our algorithm performs under $O(|G|\log|G|)$.

5.6 Summary

This chapter presents a measurement-driven approach for investigating the cost-accuracy and time-accuracy trade-offs in executing applications on cloud. To represent applications with variable accuracy, we select CNN as the example and
Chapter 5. Scaling Application Accuracy

apply our approach on two widely used and well established image detection CNNs, Caffenet and Googlenet. In contrast to existing work that focus on CNN training performance, we investigate the CNN inference performance. We train our CNNs with more than one million images and use parameter pruning technique to change the inference accuracy. We perform our analysis on six types of cloud resources with GPUs from Amazon EC2 cloud with an inference dataset containing 50,000 images.

We show the existence of “sweet-spot” regions where the inference time can be reduced with no or insignificant reduction in inference accuracy. With different application configurations, we investigate the dependency among CNN layers and their impact on inference time. We show that inference time can be halved for just a one-tenth accuracy reduction with multi-layer pruning for Caffenet. We expose the existence of time-accuracy and cost-accuracy Pareto-optimal configurations in the large resource configuration space where the consumer is able to reduce the time by 50% and cost by 55% for achieving the highest possible inference accuracy for Caffenet CNN by selecting Pareto-optimal configurations. For quantifying the trade-off between inference time and accuracy, we introduce Time Accuracy Ratio (TAR) metric and show its usefulness by comparing across different degrees of pruning to save time for a given accuracy bound. To quantify the trade-off between cost and accuracy, we introduce Cost Accuracy Ratio (CAR) metric. We show that TAR and CAR can be utilized to efficiently determine cloud resource configurations for achieving the best inference accuracy within the given time deadline and cost budget constraints in polynomial-time.
Chapter 6

Conclusion

6.1 Summary

This thesis focuses on application scalability and proposes fixed-cost application scaling on cloud, and investigates the impact of resource configuration, scaling application problem size, and scaling application accuracy. The inherent resource scalability and pay-per-use charging have attracted a wide range of large scale applications into cloud. However, the large cloud resource configuration space makes it challenging for a cloud consumer to select the best cloud resource configuration for their application. In enterprise computing, cloud applications often come with time deadlines and cost budgets, and thus, this problem becomes multidimensional. On the other hand, the rise of scalable applications where the resource demand of the application is a function of problem size or accuracy opens up a new opportunity to further leverage the inherent resource scalability on cloud.

We propose a fixed-cost scaling law on cloud and address these challenges with respect to fixed-cost scaling using a measurement-driven analytical modeling approach. We investigate impact of fixed-cost scaling on cloud in three directions; (i) cloud resource configurations, (ii) scaling application problem size, and (iii)
Chapter 6. Conclusion

scaling application accuracy. We evaluate our approach on cloud resources from Amazon EC2 cloud.

6.1.1 Proposed Fixed-Cost Scaling

Building upon Amdahl’s and Gustafson’s scaling laws, the proposed fixed-cost scaling law for cloud is (expression 1.6):

\[ S_C = \alpha + \frac{(1 - \alpha) C}{\theta} \]

where \( S_C \) is the fixed-cost speedup, \( \alpha \) is the sequential fraction of the application, \( C \) is the cost budget and \( \theta \) is the cost of cloud resource. The number of resource instances allocated for the workload is determined by the cost budget \( C \). With the presence of multiple cloud resource types with different cost and performance, fixed-cost scaling law can be expressed as (expression 1.9):

\[ S_C = \alpha + \left\{ \frac{\beta_1 n_1 C_1}{\theta_1} + \frac{\beta_2 n_2 C_2}{\theta_2} + ... + \frac{\beta_i n_i C_i}{\theta_i} + ... + \frac{\beta_k n_k C_k}{\theta_k} \right\} \]

where \( \beta_i \) denotes the portion of workload assigned for resource type \( i \) and

\[ \beta_1 + \beta_2 + ... + \beta_i + ... + \beta_k = 1 - \alpha \]

\( n_i \) denotes the number of instances taken from resource type \( i \), and \( \theta_i \) is the cost of using one instance of resource type \( i \).

Firstly, the cost of scaling application on cloud is affected by the cost-performance of the cloud resource configuration due to non-uniformity and non-linearity of cost-performance among cloud resources. Secondly, the application’s problem size of the results produced by the application affects the resource demand of the application. Thirdly, the application performance can be changed by trading-off accuracy.
of the application. Thus, changing the problem size and/or accuracy of the application results in a different execution cost and time. This thesis investigates fixed-time scaling on cloud by investigating these three main directions.

### 6.1.2 Measurement-driven Analytical Modeling Approach

We propose a measurement-driven analytical modeling approach for determining cost-time efficient cloud configurations, cost-time efficient application problem sizes, and cost-time efficient application accuracy. Our approach uses baseline measurements on on-premise and cloud resources to estimate the application resource demand and the capacity of cloud resources.

**Cost-time Efficient Cloud Configurations**

Public cloud platforms expose a large resource configuration space due to the availability of a wide variety of cloud resource types with different cost and performance. While it is an advantage to have a large configuration space to find a better match to a given application based on its resource demand and cloud consumers’ business constraints such as time deadline and cost budget, finding this match is a key challenge. This thesis presents an approach that models both the cost and time performance of cloud resources and matches the resource demand of the application with cloud resources. Moreover, our approach predicts the time and cost to execute the application on the predicted resource configuration.

**Cost-time Efficient Application Problem Size**

Due to recent advancements in computer science such as big data analytics, machine learning, and advanced scientific simulations, among others, the size of cloud applications is becoming increasingly larger. In addition to the large resource configuration space on cloud, these scalable applications exhibiting different re-
source demand growth adds to the challenge faced by cloud consumers to make application scaling decisions within business constraints. This leads to a multi-objective optimization between execution cost, time, and the application problem size. Thus, it is important for a cloud consumer to understand the trade-offs involved in executing a larger workload and what is the implication of scaling application problem size. This thesis presents a measurement-driven analytical modeling approach for determining the cost-time efficient application problem sizes within a given time deadline and a cost budget.

Scaling Application Accuracy

The emergence of applications that produce results of variable accuracy opens up a new opportunity for cloud consumers to trade-off application accuracy for execution cost and time on cloud. The relationship between accuracy and resource demand of the application is non-linear and non-trivial. Therefore, cloud consumers are faced with new challenges when making application scaling decisions and selecting the cost-accuracy and time-accuracy efficient cloud resources. This thesis presents a measurement-driven approach for determining cost and time efficient application accuracy and cloud resource configurations for executing the application with the determined accuracy.

6.1.3 Insights from Fixed-Cost Scaling Analysis

Fixed-Cost Scaling with Fixed Workload

For a given fixed workload, we derive the following key insights based on the results obtained using our approach:

1. Cost-time Pareto optimal cloud resource configurations: many Pareto-optimal configurations exist that represent either the lowest cost for the particular
execution time or the lowest time for the particular cost.

2. Due to heterogeneity and different cost performance of resources, the cost of scaling resources grows faster than the resource demand: when the configuration consists of different types of cloud resources, their cost-performance affects how the cost scales.

3. For a fixed-workload, the time deadline could be tightened with relatively smaller increase in cost by selecting the right cloud configuration based on its cost-performance.

The existence of multiple Pareto-optimal resource configurations and a Pareto-frontier consisting of configurations that yield the lowest cost for a given execution time shows the opportunity for the cloud consumer to scale the performance of the application in terms of cost efficiency for a fixed-workload. Our representative applications demonstrate up to 30% cost savings by selecting Pareto-optimal cloud resource configurations. Moreover, having multiple Pareto-optimal configurations within the cost budget implies the opportunity for fixed-cost scaling.

Fixed-Cost Scaling with Varying Workload

For a given application, we derive the following key insights for regarding the impact of varying application workload (problem size) on the execution cost and time:

1. Cost-time Pareto-optimal largest problem sizes: three-dimensional Pareto-optimization between cost, time, and problem size. A Pareto-optimal largest problem size represents the largest problem size achievable without increasing either the cost or time associated with the problem size.
2. Increasing the cost budget and/or relaxing the time deadline does not always result in executing a larger problem size: the cost-performance of cloud resources with respect to the application needs to be considered while scaling.

3. The trade-off between cost, time, and problem size is non-linear and non-trivial as it depends on both the application resource demand and resource capacity of cloud resources: there is an opportunity for tightening time deadline for relatively smaller reduction of problem size.

4. Performance Cost Ratio (PCR) metric can be used to determine cloud resource configurations that execute cost-time efficient near-optimal problem sizes.

Similar to fixed-cost scaling with a fixed workload, the existence of multiple Pareto-optimal largest problem sizes in the three-dimensional cost-time-problem size space implies the opportunity for the cloud consumer to scale the application problem size to best satisfy cost and time constraints. Depending on the more critical constraint, either the time deadline or the cost budget, the largest application problem size and the resource configuration to execute it can be chosen. The resource demand growth function of the application can be leveraged to allocate cloud resources while scaling the problem size of the application to reduce execution cost. Moreover, for time constrained applications, there is an opportunity for reducing time with relatively smaller reduction of problem size. For example, our representative application requires only one-fifth reduction in problem size to tighten time deadline by half. Furthermore, PCR metric which represents the cost-performance of cloud resources with respect to the application can be used to derive resource configurations for a given problem size without exhaustively exploring the large resource configuration space.
Scaling Application Accuracy

For a given application with varying accuracy, we derive the following key insights on the impact of accuracy towards execution cost and time:

1. Cost-accuracy “Sweet Spots”: application accuracy can be varied by altering different application parameters and there exist application settings where time and cost could be reduced with no or small reduction in accuracy.

2. Cost-accuracy and time-accuracy Pareto-optimal configurations: these configurations either yield the lowest cost or lowest time for the given accuracy.

3. Cost Accuracy Ratio (CAR) and Time Accuracy Ratio (TAR) for selecting efficient application and resource configurations: to select cost and time near-optimal application settings and cloud resources configurations, CAR and TAR metrics can be used.

Applications that produce results of varying accuracy open up new avenues to better leverage the inherent resource scalability on cloud. Depending on the consumer’s requirement, the accuracy can be traded-off for cost savings or for meeting a strict time deadline for a time-critical application. The existence of cost-accuracy and time-accuracy Pareto-optimal configurations provides the consumer with an additional handle to further optimize the application execution on cloud while meeting accuracy requirements as well as satisfying cost and time deadline constraints. For example, our representative CNN applications show that up to 55% savings in execution cost and time can be achieved by selecting accuracy-efficient Pareto-optimal configurations. Usage of CAR and TAR metrics that quantify cost-accuracy and time-accuracy performance to guide the heuristics for choosing configurations simplifies the complexity resource selection in the large configuration space.
6.2 Limitations

While our measurement-driven analytical modeling approach is designed for static applications, a baseline phase characterizes the application and resources to determine application resource demand and resource capacity. This is a one-time overhead for each cloud application.

For a given application, we assume that the application workload can be proportionately mapped to a given number of virtual cloud processors and the sequential portion of the application is negligible compared to the parallel portion of the application. While these are reasonable assumptions for larger application workloads, some applications may have a higher sequential processing time which is not currently captured in the baseline measurements.

Since we focus only on the compute-intensive applications, memory, communication, or storage intensive applications will require extending the baseline measurements and model. For example, as described in Chapter 3, for sand application which performs a comparatively higher inter-node communication compared to other compute-intensive applications, the model error is about 5% higher.

6.3 Future Work

We discuss two possible future directions to extend our work.

6.3.1 Fixed-Cost Scaling for Serverless Computing

Traditionally, the cloud consumer manages IaaS resources, operating system, scaling, and other application configurations. In contrast, in serverless computing, the cloud consumer is relieved of cloud resource provisioning, and the cloud provider manages compute resources including resource scaling. Thus, the consumer fo-
cuses solely on defining the application. Serverless computing services include Amazon AWS Lambda \cite{11} and IBM Cloud Functions \cite{47}.

In this thesis, we discussed three types of fixed-cost scaling: (i) scaling with fixed-workload (ii) scaling application problem size, and (iii) scaling application accuracy. With a fixed-workload, we achieve scaling by changing the resource configuration whereas in the other two cases we varied application size. In serverless computing, since we do not have control over the resource configuration, the number of function calls that execute in parallel can be varied. This will trigger a scaling decision on the cloud serverless service backend. By studying the trade-off between the cost, time, and number of parallel functions, there is a potential to find cost-performance sweet spots. Scaling application problem size and accuracy is applicable on serverless platforms as well because the consumer has control over the application parameters and the program code. However, to make scaling decisions that benefit the consumer, a thorough study on all the trade-offs involved is required.

6.3.2 Hybrid Edge-cloud Computing

Edge computing enables data collected at the edge to be pre-processed before transmitting to the cloud for wider data analytics. This allows for cost reductions compared to the completely centralized data analytics where all raw data is processed at the cloud \cite{86,91,92}. However, the high-level data analysis on cloud could still be expensive.

The measurement driven analytical modeling approach proposed in this thesis for determining cost-time efficient cloud resource configurations can be applied in this context by considering cloud and edge as one hybrid resource pool. Based on the cost-performance of executing the application on cloud resources as well as the edge nodes, the optimal resource configurations could be derived.
List of Publications


References


REFERENCES


REFERENCES


REFERENCES


REFERENCES


IEEE/ACM/IFIP International Conference on Hardware/Software Code-

[74] D. Rajan, A. Canino, J. A. Izaguirre, D. Thain, Converting a High Per-
formance Application to an Elastic Cloud Application, Proc. of 3rd In-
ternational Conference on Cloud Computing Technology and Science, pages

[75] L. Ramapantulu, D. Loghin, Y. M. Teo, An Approach for Energy Effi-
cient Execution of Hybrid Parallel Programs, International Parallel and

[76] L. Ramapantulu, D. Loghin, Y. M. Teo, On energy proportionality and
time-energy performance of heterogeneous clusters, 2016 IEEE International

[77] L. Ramapantulu, B. M. Tudor, D. Loghin, T. Vu, Y. M. Teo, Modeling the
energy efficiency of heterogeneous clusters, 2014 43rd International Confer-


[79] S. Rathnayake, D. Loghin, Y. M. Teo, CELIA: Cost-time Performance of
Elastic Applications on Cloud, Proc. of 46th International Conference on

[80] S. Rathnayake, L. Ramapantulu, Y. M. Teo, Cost-time performance of
scaling applications on the cloud, 2018 IEEE International Conference on
Cloud Computing Technology and Science (CloudCom), pages 30–33. IEEE,
2018.
REFERENCES


REFERENCES


Appendix A

Experiment Setup and Baseline Execution

A.1 Cloud Applications

To investigate the implications of application scaling on cloud, we selected applications that demonstrate different types of relationships between application parameters and application resource demand. These are four fixed-workload applications with fixed problem size (bt, ft, lu, and sp), three variable-workload applications with variable problem size (x264, n-body and sand) and two CNN applications (Caffenet and Googlenet) that represent applications with variable accuracy. These applications belong to domains such as multimedia, scientific simulation, bioinformatics and machine learning.

bt

bt is one of the MPI benchmark applications in NAS NPB benchmark suite. bt solves three sets of uncoupled systems of equations in three directions x,y and z. These systems are block tridiagonal with 5x5 blocks. This runs on power-of-two
number of processors. For our experiments, we fix the grid size at $162 \times 162 \times 162$ with $200 \times 1024$ iterations and $0.0001$ time step.

\textbf{ft}

$ft$ is one of the MPI kernel benchmarks in NAS NPB benchmark suite that performs the three dimensional fast Fourier transform. For our experiments, we fix the grid size at $512 \times 512 \times 512$ with $20 \times 1024$ iterations.

\textbf{is}

$is$ is one of the MPI kernel benchmarks in NAS NPB benchmark suite that performs the integer sort computation. For our experiments, we fix the number of keys at $2^{27}$, maximum key value at $2^{28}$ and number of iterations at $2^{14}$.

\textbf{x264}

$x264$ [2] is a commercial video encoding application that takes a number of video clips, $n$, and distributes them among a number of independent processes running on one or multiple cluster nodes. The video encoding is done with an accuracy represented by the compression factor, $f$, that varies from 1 to 51. For simplicity, we fix the size of a single video clip to 75MB in our experiments. The resource demand (workload) of $x264$ grows linearly with the number of video clips and quadratically with the compression factor.

\textbf{n-body Simulation}

$n$-body [54] implements the n-body simulation of masses in a n-body where input parameters are the number of masses, $n$, and the number of simulation steps $s$. Masses are distributed among MPI processes to compute the new positions for
each mass at each simulation step. Simulation accuracy improves with increasing number of steps, thus, we use $s$ as a proxy for accuracy. There are no theoretical upper bounds for $n$ and $s$. The resource demand of $n$-body grows linearly with the number of simulation steps, and quadratically with the number of bodies in the system.

sand Genome Sequencing

sand [71] is an application for genome sequence assembly. It aligns compatible genome sequences from a list of candidate sequences of size $n$. The quality threshold, $t$, specifies the degree of similarity for two candidate sequences to be aligned together. This application is implemented using a master-slave approach built on Work Queue platform [74]. The master process takes the list of candidate sequences, creates a set of alignment tasks and distributes them among slave processes. There is no upper bound for $n$ and the meaningful range for $t$ is from 0 to 1. The resource demand of sand grows linearly with the number of genome sequences, and logarithmically with the quality threshold.

In addition to these, we use Caffenet and Googlenet image Classification CNNs which are characterized and studied in chapter 5.

A.2 Cloud Resources

To substantiate the application of our approaches on the cloud, we selected nine Amazon EC2 resource types and up to five instances from each type, representing a large configuration space of 10,077,695 configurations. These resources are possessing different cost-performance, as shown in Table A.1. Amazon cloud resources are classified based on their performance characteristics into different categories such as compute-intensive, general-purpose and memory-optimized, among oth-
Table A.1: Amazon EC2 Cloud Resource Types

<table>
<thead>
<tr>
<th>Type</th>
<th>vCPUs</th>
<th>Frequency (GHz)</th>
<th>Memory (GB)</th>
<th>Storage</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>2</td>
<td>2.9</td>
<td>3.75</td>
<td>EBS</td>
<td>0.105</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>2.9</td>
<td>7.5</td>
<td>EBS</td>
<td>0.209</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>2.9</td>
<td>15</td>
<td>EBS</td>
<td>0.419</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
<td>2.3</td>
<td>8</td>
<td>EBS</td>
<td>0.133</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>2.3</td>
<td>16</td>
<td>EBS</td>
<td>0.266</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>2.3</td>
<td>32</td>
<td>EBS</td>
<td>0.532</td>
</tr>
<tr>
<td>r3.large</td>
<td>2</td>
<td>2.5</td>
<td>15</td>
<td>32</td>
<td>0.166</td>
</tr>
<tr>
<td>r3.xlarge</td>
<td>4</td>
<td>2.5</td>
<td>30.5</td>
<td>80</td>
<td>0.333</td>
</tr>
<tr>
<td>r3.2xlarge</td>
<td>8</td>
<td>2.5</td>
<td>61</td>
<td>160</td>
<td>0.664</td>
</tr>
</tbody>
</table>

1 EBS = Amazon Elastic Block Storage

ers. Each of these categories consist of multiple resource types such as *large*, *xlarge* and *2xlarge*, labeled based on the number of vCPUs each resource type possesses.

We select three resource categories representing compute-intensive *c4*, general-purpose *m4* and memory-optimized *r3*, and three resource types, *large*, *xlarge* and *2xlarge* which possess two, four and eight vCPUs, respectively. Compute-intensive *c4* instances are powered by Intel Xeon E5-2666 v3 processors, *m4* instances are powered by Intel Xeon E5-2676 v3 processors and *r3* instances are powered by Intel Xeon E5-2670 processors. All cloud resources are fully virtualized instances running Ubuntu Linux 16.04 LTS. Maximum of five instances per resource type are allowed in the configuration. All cloud instances are selected from Amazon EC2 Oregon region and the hourly prices range from $0.105 to $0.664.

### A.3 Baseline Execution

Due to the virtualization of cloud resources, traditional performance monitoring techniques such as using CPU performance counters cannot be applied. Thus, we are constrained to use a local server with the same micro-architecture as target cloud resources to measure instruction count. Moreover, as each elastic applica-
tion has multiple application parameters and different execution profile, we are required to establish the relationship between application parameters and application resource demand.

The instruction count is measured using non-intrusive hardware counters available on most modern processors. However, due to virtualization and security issues, cloud providers do not allow the usage of hardware counters and, thus, we are constrained to use a local server for these measurements. To estimate cloud resources capacities, we run the same scale-down versions of the application on each cloud instance type and measure the execution time. We compute the resource capacity of a cloud resource in terms of instructions executed per unit of time as the ratio between measured instruction count and measured execution time. This rate includes virtualization overhead imposed by the cloud platform, thus, we do not need to take it into account separately.

Figure A.1: Baseline Execution
A.3.1 Baseline Execution Measurements on Local Server

Table A.2: Instruction Count - *bt*, *ft* and *is*

<table>
<thead>
<tr>
<th>application</th>
<th>instructions (billions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bt</td>
<td>6950445</td>
</tr>
<tr>
<td>ft</td>
<td>1533759</td>
</tr>
<tr>
<td>is</td>
<td>1248216</td>
</tr>
</tbody>
</table>

Table A.3: Instruction Count - *n-body* masses

<table>
<thead>
<tr>
<th>masses</th>
<th>instructions (billions)</th>
<th>1000 steps</th>
<th>2000 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>8192</td>
<td>20750</td>
<td>41833</td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>82984</td>
<td>167293</td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td>331906</td>
<td>669113</td>
<td></td>
</tr>
<tr>
<td>65536</td>
<td>1327209</td>
<td>2676293</td>
<td></td>
</tr>
</tbody>
</table>

Table A.4: Instruction Count - *n-body* steps

<table>
<thead>
<tr>
<th>steps</th>
<th>instructions (billions)</th>
<th>8192 masses</th>
<th>16384 masses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>20750</td>
<td>82984</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>41833</td>
<td>167293</td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>82793</td>
<td>336259</td>
<td></td>
</tr>
<tr>
<td>8000</td>
<td>167374</td>
<td>669155</td>
<td></td>
</tr>
</tbody>
</table>

Table A.5: Instruction Count - *sand* sequences

<table>
<thead>
<tr>
<th>sequences (millions)</th>
<th>instructions (billions)</th>
<th>t=0.04</th>
<th>t=0.08</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>944</td>
<td>1699.2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1868</td>
<td>3362.4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3469</td>
<td>6244.2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>6917</td>
<td>12450.6</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>13800</td>
<td>24840</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>28315</td>
<td>50067</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>54585</td>
<td>98253</td>
<td></td>
</tr>
</tbody>
</table>
### Table A.6: Instruction Count - *sand* threshold

<table>
<thead>
<tr>
<th>$t$</th>
<th>$n=8000000$</th>
<th>$n=16000000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>2970</td>
<td>5940</td>
</tr>
<tr>
<td>0.02</td>
<td>4248</td>
<td>8496</td>
</tr>
<tr>
<td>0.04</td>
<td>6686</td>
<td>13372</td>
</tr>
<tr>
<td>0.08</td>
<td>12121</td>
<td>24242</td>
</tr>
<tr>
<td>0.16</td>
<td>22389</td>
<td>44778</td>
</tr>
<tr>
<td>0.32</td>
<td>42008</td>
<td>84016</td>
</tr>
<tr>
<td>0.64</td>
<td>76938</td>
<td>153876</td>
</tr>
<tr>
<td>1</td>
<td>108614</td>
<td>217228</td>
</tr>
</tbody>
</table>

### Table A.7: Instruction Count - *x264* Number of Clips

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f=10$</th>
<th>$f=20$</th>
<th>$n=4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1872</td>
<td>1288</td>
<td>868</td>
</tr>
<tr>
<td>4</td>
<td>3725</td>
<td>2563</td>
<td>1144</td>
</tr>
<tr>
<td>8</td>
<td>7488</td>
<td>5152</td>
<td>1656</td>
</tr>
<tr>
<td>16</td>
<td>14976</td>
<td>10304</td>
<td>2576</td>
</tr>
<tr>
<td>32</td>
<td>29801</td>
<td>20504</td>
<td>3744</td>
</tr>
</tbody>
</table>

### Table A.8: Instruction Count - *x264* compression factor

<table>
<thead>
<tr>
<th>compression factor</th>
<th>$n=1$</th>
<th>$n=2$</th>
<th>$n=4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>217</td>
<td>434</td>
<td>868</td>
</tr>
<tr>
<td>40</td>
<td>286</td>
<td>572</td>
<td>1144</td>
</tr>
<tr>
<td>30</td>
<td>414</td>
<td>828</td>
<td>1656</td>
</tr>
<tr>
<td>20</td>
<td>644</td>
<td>1288</td>
<td>2576</td>
</tr>
<tr>
<td>10</td>
<td>936</td>
<td>1872</td>
<td>3744</td>
</tr>
</tbody>
</table>
### A.3.2 Baseline Execution Measurements on Cloud Resources

Table A.9: Execution Time - *n-body* 16384 masses and 1000 steps

<table>
<thead>
<tr>
<th>resource type</th>
<th>vcpus</th>
<th>time(sec)</th>
<th>instr(bil) /sec</th>
<th>instr(bil) /sec/vcpu</th>
<th>cost/hr</th>
<th>instr(bil)/sec/cost/hr</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>2</td>
<td>9099</td>
<td>2.76</td>
<td>1.38</td>
<td>0.105</td>
<td>26.27</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>4560</td>
<td>5.50</td>
<td>1.38</td>
<td>0.21</td>
<td>26.21</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>2297</td>
<td>10.92</td>
<td>1.37</td>
<td>0.42</td>
<td>26.01</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
<td>10822</td>
<td>2.32</td>
<td>1.16</td>
<td>0.133</td>
<td>17.43</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>5044</td>
<td>4.98</td>
<td>1.24</td>
<td>0.266</td>
<td>18.70</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>2549</td>
<td>9.84</td>
<td>1.23</td>
<td>0.532</td>
<td>18.50</td>
</tr>
<tr>
<td>r3.large</td>
<td>2</td>
<td>10994</td>
<td>2.28</td>
<td>1.14</td>
<td>0.166</td>
<td>13.75</td>
</tr>
<tr>
<td>r3.xlarge</td>
<td>4</td>
<td>5623</td>
<td>4.46</td>
<td>1.12</td>
<td>0.332</td>
<td>13.44</td>
</tr>
<tr>
<td>r3.2xlarge</td>
<td>8</td>
<td>2823</td>
<td>8.89</td>
<td>1.11</td>
<td>0.664</td>
<td>13.39</td>
</tr>
</tbody>
</table>

Table A.10: Execution Time - *sand* 16 million sequences and 0.04 threshold

<table>
<thead>
<tr>
<th>resource type</th>
<th>vcpus</th>
<th>time(sec)</th>
<th>instr(bil) /sec</th>
<th>instr(bil) /sec/vcpu</th>
<th>cost/hr</th>
<th>instr(bil)/sec/cost/hr</th>
</tr>
</thead>
<tbody>
<tr>
<td>r3.large</td>
<td>2</td>
<td>2055</td>
<td>6.72</td>
<td>3.36</td>
<td>0.105</td>
<td>40.45</td>
</tr>
<tr>
<td>r3.xlarge</td>
<td>4</td>
<td>1040</td>
<td>13.27</td>
<td>3.32</td>
<td>0.21</td>
<td>39.85</td>
</tr>
<tr>
<td>r3.2xlarge</td>
<td>8</td>
<td>528</td>
<td>26.14</td>
<td>3.27</td>
<td>0.42</td>
<td>39.30</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
<td>1744</td>
<td>7.91</td>
<td>3.96</td>
<td>0.133</td>
<td>59.50</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>855</td>
<td>16.14</td>
<td>4.04</td>
<td>0.266</td>
<td>60.68</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>434</td>
<td>31.80</td>
<td>3.97</td>
<td>0.532</td>
<td>59.77</td>
</tr>
<tr>
<td>c4.large</td>
<td>2</td>
<td>1591</td>
<td>8.67</td>
<td>4.34</td>
<td>0.166</td>
<td>82.61</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>803</td>
<td>17.19</td>
<td>4.30</td>
<td>0.332</td>
<td>82.23</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>411</td>
<td>33.58</td>
<td>4.20</td>
<td>0.664</td>
<td>80.14</td>
</tr>
</tbody>
</table>
## Table A.11: Execution Time - \( bt \)

<table>
<thead>
<tr>
<th>resource type</th>
<th>vcpus</th>
<th>time(sec)</th>
<th>instr(bil) /sec</th>
<th>instr(bil) /sec/vcpu</th>
<th>cost/hr</th>
<th>instr(bil)/sec /cost/hr</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>2</td>
<td>848</td>
<td>8.00</td>
<td>4.00</td>
<td>0.105</td>
<td>76.23</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>440</td>
<td>15.43</td>
<td>3.86</td>
<td>0.21</td>
<td>73.46</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>232</td>
<td>29.26</td>
<td>3.66</td>
<td>0.42</td>
<td>69.66</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
<td>990</td>
<td>6.86</td>
<td>3.43</td>
<td>0.133</td>
<td>51.55</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>501</td>
<td>13.55</td>
<td>3.39</td>
<td>0.266</td>
<td>50.93</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>253</td>
<td>26.83</td>
<td>3.35</td>
<td>0.532</td>
<td>50.43</td>
</tr>
<tr>
<td>r3.large</td>
<td>2</td>
<td>1016</td>
<td>6.68</td>
<td>3.34</td>
<td>0.166</td>
<td>40.24</td>
</tr>
<tr>
<td>r3.xlarge</td>
<td>4</td>
<td>510</td>
<td>13.31</td>
<td>3.33</td>
<td>0.332</td>
<td>40.09</td>
</tr>
<tr>
<td>r3.2xlarge</td>
<td>8</td>
<td>260</td>
<td>26.11</td>
<td>3.26</td>
<td>0.664</td>
<td>39.32</td>
</tr>
</tbody>
</table>

## Table A.12: Execution Time - \( ft \)

<table>
<thead>
<tr>
<th>resource type</th>
<th>vcpus</th>
<th>time(sec)</th>
<th>instr(bil) /sec</th>
<th>instr(bil) /sec/vcpu</th>
<th>cost/hr</th>
<th>instr(bil)/sec /cost/hr</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>2</td>
<td>234</td>
<td>29.01</td>
<td>14.50</td>
<td>0.105</td>
<td>276.25</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>120</td>
<td>56.56</td>
<td>14.14</td>
<td>0.21</td>
<td>269.35</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>59</td>
<td>115.04</td>
<td>14.38</td>
<td>0.42</td>
<td>273.91</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
<td>266</td>
<td>25.52</td>
<td>12.76</td>
<td>0.133</td>
<td>191.86</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>136</td>
<td>49.91</td>
<td>12.48</td>
<td>0.266</td>
<td>187.63</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>71</td>
<td>95.60</td>
<td>11.95</td>
<td>0.532</td>
<td>179.70</td>
</tr>
<tr>
<td>r3.large</td>
<td>2</td>
<td>270</td>
<td>25.14</td>
<td>12.57</td>
<td>0.166</td>
<td>151.44</td>
</tr>
<tr>
<td>r3.xlarge</td>
<td>4</td>
<td>136</td>
<td>49.91</td>
<td>12.48</td>
<td>0.332</td>
<td>150.33</td>
</tr>
<tr>
<td>r3.2xlarge</td>
<td>8</td>
<td>69</td>
<td>98.37</td>
<td>12.30</td>
<td>0.664</td>
<td>148.15</td>
</tr>
</tbody>
</table>

## Table A.13: Execution Time - \( is \)

<table>
<thead>
<tr>
<th>resource type</th>
<th>vcpus</th>
<th>time(sec)</th>
<th>instr(bil) /sec</th>
<th>instr(bil) /sec/vcpu</th>
<th>cost/hr</th>
<th>instr(bil)/sec /cost/hr</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>2</td>
<td>24</td>
<td>282.81</td>
<td>141.41</td>
<td>0.105</td>
<td>2693.47</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>13</td>
<td>522.12</td>
<td>130.53</td>
<td>0.21</td>
<td>2486.28</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>7</td>
<td>969.65</td>
<td>121.21</td>
<td>0.42</td>
<td>2308.69</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
<td>24</td>
<td>282.81</td>
<td>141.41</td>
<td>0.133</td>
<td>2126.42</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>12</td>
<td>565.63</td>
<td>141.41</td>
<td>0.266</td>
<td>2126.42</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>6</td>
<td>1131.26</td>
<td>141.41</td>
<td>0.532</td>
<td>2126.42</td>
</tr>
<tr>
<td>r3.large</td>
<td>2</td>
<td>36</td>
<td>188.54</td>
<td>94.27</td>
<td>0.166</td>
<td>1135.80</td>
</tr>
<tr>
<td>r3.xlarge</td>
<td>4</td>
<td>19</td>
<td>357.24</td>
<td>89.31</td>
<td>0.332</td>
<td>1076.02</td>
</tr>
<tr>
<td>r3.2xlarge</td>
<td>8</td>
<td>10</td>
<td>678.75</td>
<td>84.84</td>
<td>0.664</td>
<td>1022.22</td>
</tr>
</tbody>
</table>
Appendix B

Knapsack Approach for Determining Largest Problem Size

Given an application with a time deadline and a cost budget, and a set of cloud resources, we can map the problem of determining the largest executable problem size to a knapsack problem. In this section we show a knapsack based algorithm for solving this problem. However, with this approach, we can only determine one optimal solution.

B.1 Determining Largest Problem Size

Resources are represented as $|R| = \{R_i \mid R_i \text{ is a resource type}\}$. Each resource type $R_i$ has $|R_i|$ available instances, each having a computational capacity $\Delta_{R_i}$ and a cost per unit time $c_{R_i}$. The user specifies cost budget $C$ and the time budget $T$. Our objective is to find $P_{S_{\text{max}}}$, and the subset $|R'|$, along with the number of instances of each resource type and the time $T'$ that we want to run
them for. It is important to note that although we can have a variable number of instances from each resource type, the execution time must be the same across resource type. This is a restriction of our model imposed by the use-case scenario, since most distributed applications rely on the availability of the nodes throughout execution.

We split this problem in 2 cases: **Case 1** Choose a subset of $R$ of one resource to maximize workload. **Case 2** Choose a subset of $R$ of multiple resources to maximize workload.

### B.1.1 Case 1: Choosing a one resource

If we are restricting the method to selecting one resource we can resort to a *divide et impera* approach and use the homogeneous case approach to solve each sub-problem. We compute the maximum achievable workload for each resource, and we chose, as an optimal solution, the one with the maximum amount of work being done.

### B.1.2 Case 2: Choosing a subset of multiple resources

The number of configurations to evaluate increases exponentially with the number of resources in the set. We propose a method that finds an optimal configuration, that would yield the maximum runnable size.

We use *divide et impera* method to split the problem: we multiplex the problem in time. We split the problem in $T$ sub-problems. Each sub-problem $t$ determines the maximum achievable workload by running a subset of $|R|$ for time $t$. The optimal configuration is determined by choosing the maximum workload size from the sub-problems.

To solve the sub-problems we reduce our problem to the knapsack problem.
Chapter B. Knapsack Approach for Determining Largest Problem Size

and use a dynamic programming approach to solve it in pseudo-polynomial time.

The original knapsack problem is an optimization problem. Given a knapsack of volume $V$, and a set of $n$ objects that each has attached a price $p_i$ and a volume $v_i$, the objective is to determine which object to pick as to maximize the sum of prices, while the objects still fit in the knapsack.

The objective is:

$$\max \left( \sum_{i=1}^{n} x_i \times p_i \right) \quad (B.1)$$

where $x_i$ is binary value, if $x_i$ is 1 then the object gets included, if $x_i$ is 0 then the object is not included in the subset.

The constraint is:

$$\sum_{i=1}^{n} x_i \times v_i \leq V \quad (B.2)$$

If we take a sub-problem $t$, of our initial problem, we can model it on the knapsack problem. We consider the cost budget as the capacity of our knapsack and each instance of each resource type will be an object, where the price of each object will be $t \times \Delta R_i$, and $t \times c_{R_i}$ is the volume of each object, $R_i$ is the resource type that the instance belongs in.

$$\max \left( \sum_{i=1}^{\left|R_i\right|} t \times \Delta R_{i,j} \times x_{i,j} \right) \quad (B.3)$$

$$dp[i][j] = \max \left\{ \begin{array}{c}
  dp[i-1][j] \\
  dp[i-1][j-t \times c_{R_i}] + t \times \Delta R_{i,j} \quad if \ j \geq c_i
  \end{array} \right\} \quad (B.4)$$

Knapsack problems can be solved in pseudo-polynomial time (for the original problem $O(n \times V)$, and in our case $O(\sum_{i=1}^{\left|R_i\right|} |R_i| \times C)$) using a dynamic programming approach. The solution is defined by the formula B.4. For memoization of partial
Chapter B. Knapsack Approach for Determining Largest Problem Size

Table B.1: Dynamic Programming Table

| \( \{x | x \in R_i, R_i \in R \} \) | C \( \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & \ldots & C_D - 1 & C_D \\ \hline \end{array} \) |
|---|---|---|---|---|
| 0 | | | | |
| \( R_{\{1,1\}} \) | | | | |
| \( R_{\{1,2\}} \) | | | | |
| \( \ldots \) | | | | |
| \( R_{\{2,1\}} \) | | | | |
| \( R_{\{2,2\}} \) | | | | |
| \( \ldots \) | | | | |
| \( R_{\{|R_i|,|R_i|\}} \) | | | | |

Algorithm B.1 Knapsack solution

1: Initialize matrix \( \text{mem} \) to zero
2: Initialize array \( R \) with all available instances
3: Initialize array \( S \) with relative speed-ups to baseline resource
4: for \( t = 1..T \) do
5: for each \( R_i \) in \( R \) do
6: \( \text{cost}_{\text{instance}} = R_i \times t \)
7: for \( \text{cost} = 1..C \) do
8: if \( \text{cost}_{\text{instance}} > \text{cost} \) then
9: \( \text{mem}[t,i,\text{cost}] = \text{mem}[t,i-1,\text{cost}] \)
10: else
11: \( \text{mem}[t,i,\text{cost}] = \max(\text{mem}[t,i+1,\text{cost} - \text{cost}_{\text{instance}}] + S_i \times t, \text{mem}[t,i,\text{cost}]) \)
12: end if
13: end for
14: end for
15: end for

solutions we use Table B.1.

\[
W(S) = a \times S + b \tag{B.5}
\]

The solution to solve the resulting knapsack problem is described in Algorithm B.1.
Table B.2: Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Domain</th>
<th>Input</th>
<th>Problem Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>object detection (detect)</td>
<td>deep learning</td>
<td>number of images, number of steps (n)</td>
<td>(n)</td>
</tr>
<tr>
<td>n-body simulation (n-body)</td>
<td>astrophysics</td>
<td>number of masses (n), simulation steps (s)</td>
<td>(n, s)</td>
</tr>
<tr>
<td>SAND genome alignment (sand)</td>
<td>bioinformatics</td>
<td>genome sequences (n), threshold (t)</td>
<td>(n, t)</td>
</tr>
</tbody>
</table>

Table B.3: Amazon EC2 Cloud Resource Types

<table>
<thead>
<tr>
<th>Type</th>
<th>vCPUs</th>
<th>Frequency (GHz)</th>
<th>Memory (GB)</th>
<th>Storage¹</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>2</td>
<td>2.9</td>
<td>3.75</td>
<td>EBS</td>
<td>0.105</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>2.9</td>
<td>7.5</td>
<td>EBS</td>
<td>0.209</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>2.9</td>
<td>15</td>
<td>EBS</td>
<td>0.419</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
<td>2.3</td>
<td>8</td>
<td>EBS</td>
<td>0.133</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>2.3</td>
<td>16</td>
<td>EBS</td>
<td>0.266</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>2.3</td>
<td>32</td>
<td>EBS</td>
<td>0.532</td>
</tr>
<tr>
<td>r3.large</td>
<td>2</td>
<td>2.5</td>
<td>15</td>
<td>32</td>
<td>0.166</td>
</tr>
<tr>
<td>r3.xlarge</td>
<td>4</td>
<td>2.5</td>
<td>30.5</td>
<td>80</td>
<td>0.333</td>
</tr>
<tr>
<td>r3.2xlarge</td>
<td>8</td>
<td>2.5</td>
<td>61</td>
<td>160</td>
<td>0.664</td>
</tr>
</tbody>
</table>

¹ EBS = Amazon Elastic Block Storage

B.2 Results

We present some results and insights gained through our approach with three applications in the Table B.2 on Amazon EC2 and Google Cloud. To determine largest possible problem size executable for a given application with a given time deadline and cost budget. We investigate this under two cases; with homogeneous cloud resources, and with heterogeneous cloud resources. To investigate the homogeneous case, we use detect, and to investigate heterogeneous case we use n-body and sand.

B.2.1 detect on Homogeneous Cloud Resources

In figure B.1, the effect of scaling out on Google Cloud for detect are analyzed. For configurations we use 1, up to 4 workers, and we plot the number of steps executed with respect to time.
Observation 1: As local runs showed, the application scales linearly with the number of steps. Observation 2: Even though it the workload is highly parallelizable, scaling-out the system does not lead to a similar increase in execution speed. This trend is due to network latency and memory accesses, and can be improved by adding multiple servers to store the parameters of the model, but this is not the object of this thesis.

B.2.2  \textit{n-body} and \textit{sand} on Heterogeneous Cloud Resources

As shown in Figures B.2 and B.3, to investigate the impact of the cost budget on the largest possible problem size executable, we fix the time deadline at 12hr, 24hr, 36hr and 48hr, and, observe the change to the largest executable problem while increasing the cost budget from $50 to $250.

Observation 3: For both \textit{n-body} and \textit{sand} we observe that the largest pos-
Chapter B. Knapsack Approach for Determining Largest Problem Size

Figure B.2: Largest Possible Problem Size - *n-body*

Figure B.3: Largest Possible Problem Size - *sand*
sible problem size is increases until flattening out suddenly. The sudden cease of
growth is due to utilizing all resources with no remaining resources to allocate
with more money.

Observation 4: \textit{n-body} demonstrates a sublinear growth of problem size
when the cost budget is increased with a fixed time deadline as shown in Figure
B.2 while \textit{sand} demonstrates a linear growth in problem size as shown in Figure
B.3. These different growth patterns in each case could be explained using the
application characterization of \textit{n-body} and \textit{sand}. For \textit{n-body}, the execution time
grows quadratically with the problem size $n$. Thus, the resource demand growth
gradient keeps increasing with problem size. This is reflected in Figure B.2 by the
decreasing gradient of problem size growth curve. Similarly, the linear resource
demand growth of \textit{sand} is reflected in Figure B.3 by the linear growth of largest
problem size curve.