A Novel Cloud Broker-based Resource Elasticity Management and Pricing for Big Data Streaming Applications

by

Olubisi A. Runsewe

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electronic Business

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

The pervasive availability of streaming data from various sources is driving todays' enterprises to acquire low-latency big data streaming applications (BDSAs) for extracting useful information. In parallel, recent advances in technology have made it easier to collect, process and store these data streams in the cloud. For most enterprises, gaining insights from big data is immensely important for maintaining competitive advantage. However, majority of enterprises have difficulty managing the multitude of BDSAs and the complex issues cloud technologies present, giving rise to the incorporation of cloud service brokers (CSBs). Generally, the main objective of the CSB is to maintain the heterogeneous quality of service (QoS) of BDSAs while minimizing costs. To achieve this goal, the cloud, although with many desirable features, exhibits major challenges — resource prediction and resource allocation — for CSBs. First, most stream processing systems allocate a fixed amount of resources at runtime, which can lead to under- or over-provisioning as BDSA demands vary over time. Thus, obtaining optimal trade-off between QoS violation and cost requires accurate demand prediction methodology to prevent waste, degradation or shutdown of processing. Second, coordinating resource allocation and pricing decisions for self-interested BDSAs to achieve fairness and efficiency can be complex. This complexity is exacerbated with the recent introduction of containers.

This dissertation addresses the cloud resource elasticity management issues for CSBs as follows: **First**, we provide two contributions to the resource prediction challenge; we propose a novel layered multi-dimensional hidden Markov model (LMD-HMM) framework for managing time-bounded BDSAs and a layered multi-dimensional hidden semi-Markov model (LMD-HSMM) to address unbounded BDSAs. **Second**, we present a container resource allocation mechanism (CRAM) for optimal workload distribution to meet the real-time demands of competing containerized BDSAs. We formulate the problem as an n-player non-cooperative game among a set of heterogeneous containerized BDSAs. Finally, we incorporate a dynamic incentive-compatible pricing scheme that coordinates the decisions of self-interested BDSAs to maximize the CSB's surplus. Experimental results demonstrate the effectiveness of our approaches.

Acknowledgements

This dissertation would not have been possible without the invaluable support of many great personalities instrumental in shaping my PhD program.

First and foremost, I will like to express my hearty appreciation and gratitude to my Supervisor, Dr. Nancy Samaan, for her unwavering support and encouragement throughout the course of my study. Her extensive knowledge, unsurpassed expertise, forbearance and understanding made it possible for me to work on a topic that was of great interest to me. I also want to express my deepest thanks to her for believing in my ability and constantly motivating me to work harder. Mere writing is not enough to express the wealth of support and guidance I received from her on an academic and a personal level. My gratitude also goes to my thesis advisory committee members, Prof. Raheemi Bijan and Prof. Herna Viktor, for their invaluable and constructive comments, which helped to improve my dissertation.

Special thanks to my dear husband and lovely daughters, who served as my inspiration to pursue my goal that required so much understanding and stability throughout the years. Their wholehearted support for my decision is part of what kept me going. I thank my mum for her unequivocal support and prayers for which my mere expression of thanks does not suffice. To all my siblings for their constant encouragement, I am grateful. I will also like to express my sincere thanks to my friends and extended family members for being by my side throughout this journey, urging me on.

Finally, I will like to acknowledge the financial support of University of Ottawa in the award of an admission scholarship and Ontario Government Scholarship (OGS) that provided me the necessary financial support for my program. Above all, I thank God for making all my endeavours come to fruition.

To my beloved husband and girls.

Table of Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface

ASA Azure Stream Analytics

ASP Application Service Provider

AWS Amazon Web Services

BD Big Data

BDSA Big Data Streaming Application

CaaS Container as a Service

CLI Command Line Interface

CMS Container Management System

CPU Central Processing Unit

CRAM Container Resource Allocation Mechanism

CSB Cloud Service Broker

CSC Cloud Service Consumer

CSP Cloud Service Provider

DAG Directed Acyclic Graph

DBMS Database Management System

DC Datacenter

DSR Design Science Research

EC2 Elastic Compute Cloud

ESX Enterprise Server

ETL Extract, Transform, Load

EVCD Elastic Provisioning of Virtual Machines for Container Deployment

HDFS Hadoop Distributed File System

HMM Hidden Markov Model

HSMM Hidden Semi-Markov Model

I/O Input/Output

IaaS Infrastructure as a Service

IoT Internet of Things

ISP Internet Service Provider

JSON JavaScript Object Notation

KKT Karunsh-Kuhn-Tucker

LMD-HMM Layered Multi-dimensional Hidden Markov Model

LMD-HSMM Layered Multi-dimensional Hidden Semi-Markov Model

NIST National Institute of Standards and Technology

NoSQL Non-Structured Query Language

OLAP Online Analytical Processing

OS Operating System

PaaS Platform as a Service

QoS Quality of Service

RDD Resilient Distributed Datasets

RDMA Remote Directory Memory Access

RR Round-Robin

SaaS Software as a Service

SDK Software Development Kit

SDN Software Defined Network

SLA Service Level Agreements

SnF Store and Forward

SPEs Stream Processing Engines

SQL Structured Query Language

TCP Transmission Control Protocol

TE Traffic Engineering

ToR Top-of-the-Rack

TTM Time to Market

UDFs User Defined Functions

UDP User Datagram Protocol

UDT UDP-based Data Transfer

UI User Interface

VM Virtual Machine

VMM Virtual Machine Monitor

VN Virtual Network

WAN Wide Area Network

XML eXtended Markup Language

Chapter 1

Introduction

1.1 Overview

As the world becomes more interconnected, real-time streaming data from applications (e.g., social media) and ubiquitous devices (e.g., wearable devices, tablets and security cameras) have introduced a massive flow of data in either structured or unstructured formats referred to as big data [10]. Analyzing these big data streams in real-time under very short delays has become a crucial requirement for most enterprise to unearth values useful for informing business decisions, developing new products, minimizing risks and combating fraud [11]. Based on the real-time analytic requirements of big data streaming applications (BDSAs), traditional data management systems, which are prohibitively expensive when dealing with very large amounts of data, are less favoured. As a consequence, cloud computing has become a popular paradigm for quickly deploying BDSAs in an elastic setting through on-demand acquisition and release of resources [12], liberating cloud users (e.g., enterprises) from the expensive on-site infrastructure investment. Moreover, cloud platforms host many of the large-scale stream processing engines (SPEs) required to meet enterprises' demands, e.g., Storm [13], Spark [14], and S4 [15].

Besides the tremendous benefits the cloud offers for processing big data, choosing the

best cloud service provider (CSP) to manage enterprise BDSAs is a complex issue due to the variety of potential options and the number of variables to consider. Many BDSAs serve multiple users that are geographically distributed and as a result observe temporal and spatial workload dynamics. Also, as the size and complexity of these applications grow, deployment becomes more difficult. Although, processing through different CSPs can be beneficial for most enterprises, issues such as interoperability, legal contract maintenance, user account management, and periodic service monitoring are introduced [16], and hence, additional overhead. To address these challenges, a cloud service broker (CSB) is employed to act as an intermediary between the CSP and the enterprise [17]. CSBs provide an organizational pool for consolidating BDSAs' demands and negotiating the best service usage rates. In turn, enterprises are able to exploit diverse services from multiple CSPs [18] as well as avoid vendor lock-in [19].

A significant challenge facing CSBs, however, is how to dynamically manage resources to fulfill the quality of service (QoS) requirements of BDSAs while minimizing their investment and management costs. To achieve this goal, CSBs need an infrastructure such as the cloud, capable of scaling rapidly in case of peak load, but flexible enough to avoid resource over-provisioning during low demands. Generally, the mechanism through which the cloud is capable of achieving economies of scale is elasticity, which can be applied in form of size, location or number of resources. *"Elasticity aims at adapting a system to changes in workload through automatic provisioning and deprovisioning of resources, such that the available resources match the current demand"* [20]. In this case, utilizing cloud elasticity management solutions to reconfigure CSPs' resources becomes important to address the challenges of the CSBs.

This dissertation focuses on two major issues encountered by CSBs — resource prediction and resource allocation. Resource prediction can be useful for identifying when to provision/deprovision resources. However, it can be challenging for CSBs due to the need to estimate multiple BDSA workload demands utilizing a heterogeneous cluster of hosts. Moreover, CSBs are required to know in advance the resource needs of the applications or to continuously monitor their states to decide when a change in the amount of resources is required. Oftentimes, most stream processing systems allocate a fixed amount of resources at runtime. This "one size fits all" approach (i.e., static provisioning) can lead to under-provisioning or over-provisioning as BDSA demands vary over time. As a result, major CSPs provide tools for monitoring the incoming workload and the resource utilization patterns of applications. Unfortunately, these tools are reactive and can be inefficient when dealing with multiple BDSAs sharing a heterogeneous cluster of hosts. A desirable solution then would require a predictive approach, i.e., allocating resources a priori to match

workload demands.

Another factor that has contributed to the cloud resource elasticity issue is resource allocation. This can be challenging for CSBs due to the fluctuating demands of the BDSAs for resources. Additionally, a shift is also taking place whereby BDSAs are deployed on container-clusters to improve the performance of streaming applications. While containerization provides a great potential to elastically scale resources to match the demands of each BDSA, sharing of container-clusters by multiple BDSAs can lead to resource contention. Furthermore, given the limited CSBs' resources, when BDSAs act independently to maximize their level of satisfaction with no consideration for others, congestion can occur. Therefore, newer approaches are required that can fairly allocate resources to BDSAs to meet their QoS objectives as well as mechanisms that provide incentives for BDSAs to behave in ways that improve the overall resource utilization and performance.

The work presented in this dissertation aims at developing a dynamic cloud resource elasticity management framework for CSBs that incorporates mechanisms for assigning big data workloads in the cloud. The objective of the framework is to meet the QoS of multiple BDSAs running on a heterogeneous cluster of hosts while minimizing cost. In this chapter, we present the requirements for BDSAs, the cloud resource elasticity challenges, followed by a discussion on the motivation, research scope and methodology. Finally, the contributions of this work and thesis organization are presented.

1.2 Research Challenges and Motivation

1.2.1 Research Challenges

This section briefly discusses the research challenges encountered by CSBs managing multiple BDSAs on a heterogeneous cluster of hosts. It focuses on the resource prediction and resource allocation challenges of cloud resource elasticity management.

Resource Prediction Challenge: Lack of a flexible, proactive resource prediction mechanism may hinder resource scaling operations of BDSAs in the cloud. Resource scaling is a major challenge due to the stringent delay constraints, increasing parallel computation requirements, and dynamically changing data volume and velocity characteristic of big data streams. This challenge is further complicated by the lack of knowledge of the workload and resource usage patterns of BDSAs. To accommodate the fluctuating workload

demands, resource scaling operations may entail service interruption and shutdown and as a result, the application state may not be preserved [21]. Prior work for achieving resource scaling use different strategies, which can be a simple reactive approach that involves monitoring a metric (e.g., CPU utilization) or a proactive approach that involves forecasting the future resource need based on the demand history. In practice, most CSPs (e.g., Azure[1] and Amazon[2]) make use of the reactive approach. This approach usually requires that the application providers be aware of the resource usage patterns to set triggers for scaling activities. Since streaming applications have to adapt to varying resource requirements to meet user needs, reactively scaling computing resources at the time of usage may be insufficient as the demand occurs before additional resources are provisioned, i.e., exhibits inherent lag during provisioning [22]. Therefore, most CSBs are caught between expensive over-provisioning and reactively scaling resources for BDSAs. To meet QoS requirements of BDSAs, CSBs need an infrastructure capable of adding resources in the case of peak load, but flexible enough to avoid resource over-provisioning during low demands. We present proactive resource prediction mechanisms for CSBs to address the aforementioned problem in Chapter 4.

Resource Allocation Challenge: BDSAs generally require sufficient resources for processing to: i) reduce delay, ii) cope with sudden variations in load changes, and iii) avoid shutdown of processing. Applicable solutions involve optimal resource allocation and efcient utilization for BDSAs to meet their QoS requirements. Resource allocation is a key challenge for CSBs due to conflicting BDSA requests for resources. This is as a result of BDSA usage patterns coupled with the need to manage heterogeneous cluster of host resources. Therefore, fulfilling QoS requirements stated in the SLAs while minimizing costs is a challenging research problem [23]. Moreover, a key issue that arises with resource allocation is that each BDSA prefers a higher QoS level over lower level. Without a mechanism that induces self-interested BDSAs to behave in a way that aligns their individual objectives with that of the overall system can lead to congestion. Furthermore, how to optimally allocate resources to competing BDSAs running on container-clusters is still unclear. In Chapter 5, we address the aforesaid problem by introducing a novel resource allocation and incentive-compatible pricing mechanism for BDSAs.

In the next section, we present our research questions.

[1]https://azure.microsoft.com/en-us/
[2]https://aws.amazon.com/

1.2.2 Research Questions

Based on the challenges discussed in Section 1.2.1, the following research questions (RQs) are raised:

RQ (1) *How can cloud resource utilization be predicted for BDSAs running on a heterogeneous cluster of hosts?*

- How can we model time-bounded big data streaming applications, considering the multiple resource bottlenecks, e.g., CPU and memory?
- How can unbounded streaming applications be catered for to avoid incorrect prediction for long running streaming jobs?

RQ (2) *How can we optimally allocate resources to containerized BDSAs in a dynamic manner to meet their QoS as well as maximize the CSB's surplus?*

- How can resources for containerized BDSAs be efficiently allocated to achieve optimal performance and fairness?
- How can resources be allocated to avoid contention when dealing with multiple BDSAs running on shared heterogeneous cluster of hosts?
- How to incorporate a dynamic pricing mechanism that generates incentives for self-interested to make optimal decisions such that the net value of the CSBs' resources is maximized?

1.2.3 Motivation

Elasticity continues to be one of the most fundamental and important characteristic in the field of cloud computing [24] among other relevant features such as on-demand self-service, multi-tenancy, pay-per-use and ubiquitous access [25]. As CSBs become aware of the benefits that the cloud provides, they are increasingly adopting the cloud computing paradigm to provision resources for their end users (enterprises). For instance, Foursquare saves 53% in costs using Amazon EC2 to perform analytics across more than 5 million daily checkins [26]. Also, Netflix[3] can run critical encoding tasks with Amazon EC2 to serve its clients' video demands using a number of VM instances, and shuts them down when completed. Cloud computing is a promising solution for CSBs due to the ability to rapidly scale

[3]https://aws.amazon.com/solutions/case-studies/netflix/

up during peak load and flexible enough to avoid resource over-provisioning during scarce demand. With cloud solutions, CSBs are relieved from the burden of setting up their own datacenter and are able to instantiate elastic set of resources that can dynamically adapt to their needs [23]. For most CSPs, elasticity is achieved by implementing optimization and resource management mechanisms to allow cloud resource access at will. Whereas, for the CSBs, elasticity management involves finding the optimal trade-off between meeting their business goals and providing adequate QoS (i.e., meeting SLAs, scalability and deadline constraints) to their consumers.

Majority of the proposed elasticity management solutions in the literature are CSP-centric and mostly concerned with infrastructure optimization (VM migration, VM scheduling, and server consolidation) [27–30]. Limited studies have tackled the CSB-centric problem with few proposals presented in the literature to automate elasticity management as well as minimize CSB's intervention. Motivated by the need for CSBs to meet their business goals and provide QoS to their end users, we focus on some of the major issues that arise for CSBs when dealing with BDSAs — resource prediction and resource allocation. The requirements for designing an elastic resource management framework for BDSAs running on heterogeneous cluster of hosts (VMs or containers) are hereby identified as follows:

- *Multiple Data Sources:* Big data is generated by a number of external data sources, and even for a single domain, this can grow to be in the tens of thousands. Many of the data sources are dynamic, thus, SPEs should be able to adjust the amount of resources based on the current workload, i.e., scale resources. Moreover, the data sources can be extremely heterogeneous in their structure and differ in accuracy and timeliness. Furthermore, additional challenge can be experienced due to limited knowledge of the characteristics of the workload and a lack of understanding on the performance of the system.

- *Shared Cluster:* Early deployments of big data applications were on dedicated clusters. In an effort to improve cluster resource utilization and cluster management, a shift is taking place where these applications are now deployed on shared clusters. Shared clusters provide a great potential to elastically scale the resources to match demand from different applications, e.g., stream processing applications can obtain additional resources when needed from the cluster and give them back when demand subsides.

- *Workload Fluctuation:* Many BDSAs serve multiple users that are geographically distributed and as a result observe temporal and spatial workload dynamics. BDSAs

are long-running jobs, which can undergo sudden changes or spikes in workload arrival due to unexpected events. The workloads can be CPU-, memory- or I/O-intensive and as a result, overburden computing nodes. Since BDSAs typically run for days or weeks under heterogeneous conditions, making accurate estimation of the resources required to handle the workload in advance or adjusting resource configuration on-the-fly can be challenging [31]. In response, most existing BDSAs are allocated a fixed amount of resources at deployment time to handle changes in workload variations. Over-provisioning can result in resource underutilization, hence, increased budget while under-provisioning can lead to low performance, buffer overflow or data loss. For dynamic workloads, SPEs need to be able to automatically handle the fluctuating demands and scale accordingly without disrupting existing requests.

- *Real-time Analytic Requirements:* For real-time analysis, the majority of aggregated data sources are very dynamic and extremely heterogeneous, i.e., continuously changing data streams with highly unpredictable data arrival patterns, which can differ in accuracy and timeliness. Generally, response time is of utmost importance when processing data streams from these sources. This may involve time-unbounded applications that require data to be processed within very short time intervals with low latency requirements, e.g., financial analysis, fraud detection. Therefore, low response time is critical and any missed deadline can lead to results that are unusable. In contrast, these applications may involve the execution of complex simulations, often executed as time-bounded processes with medium to high latency requirements, e.g., surveillance and monitoring, smart-traffic management. In most cases, the deadlines of these streaming applications can range from milliseconds to hours — depending on the particular application constraints.

- *Large-state Maintenance:* BDSAs commonly require stateful computations such as window operations and joins. The volume of the internal states manipulated by stateful operators in many real-world scenarios can be very large (order of hundreds of gigabytes), beyond the memory capacity of a small computing cluster. Thus, effectively distributing the states of streaming applications to multiple computing nodes is very important [32].

- *Host Heterogeneity:* Datacenters are comprised of machines from several generations with varying features such as memory, processors and consequently different run-time speed.

These characteristics coupled with the need to adapt to high throughput efficiency,

parallel data processing and dynamic data-driven topology graphs of tasks introduce additional challenges that make resource prediction in the cloud a tedious task for CSBs. Therefore, correctly profiling each streaming application and its associated characteristics to forecast the resource demands is crucial for BDSA in the cloud. Furthermore, to dynamically allocate resources to big data, elasticity should be combined with dynamic load balancing to avoid provisioning new resources as a result of uneven load distribution. By this, new resources will only be provisioned when the cluster as a whole does not have enough capacity to cope with the incoming workload. To effectively address these resource elasticity challenges affecting CSBs, there is a need for a framework tailored to serving BDSA requests that; i) possesses the capability to handle multiple streaming applications utilizing shared clusters (VM or containers), ii) can utilize predictive techniques to proactively perform resource scaling tasks under specified SLAs, and iii) can adapt to the temporally changing resource requirements posed by competing streaming applications.

In the next sections, we summarize our research scope and methodology.

1.3 Research Scope

The scope of this dissertation's concentration, data collection, experiments and evaluations are discussed as follows:

- *Dissertation Concentration*: The primary focus of this dissertation is on designing efficient cloud resource elasticity management and dynamic pricing for CSBs managing multiple BDSAs running on a heterogeneous cluster of hosts.
- *Data Collection*: Our experiments are performed over synthetic and real-world data streams. To collect data for measuring the performance of our systems, experimental environments were set up and metrics from various big data streaming application benchmarks monitored, e.g., WordcountTopology, ExclaimationTopology, Sentiment Analysis and TophashTags. The benchmarks chosen were developed to access Twitter API due to lack of data repository that fits into our problem. In addition, synthetic data was utilized.
- *Experiments*: Apache Spark and Heron were used as the stream processing engines due to their deployment flexibility over available machines. The environment required for the prediction system was set up using Apache Spark on a cluster of virtual machines. Each node was installed with all the streaming services required

for collecting and storing data, e.g., flume and HDFS. At each interval, an amount of data was uploaded from Twitter and persisted in HDFS for immediate processing by the Spark streaming engine. The developed applications were launched at different time intervals on the VM cluster. Fig. 1.1 depicts the data processing flow.

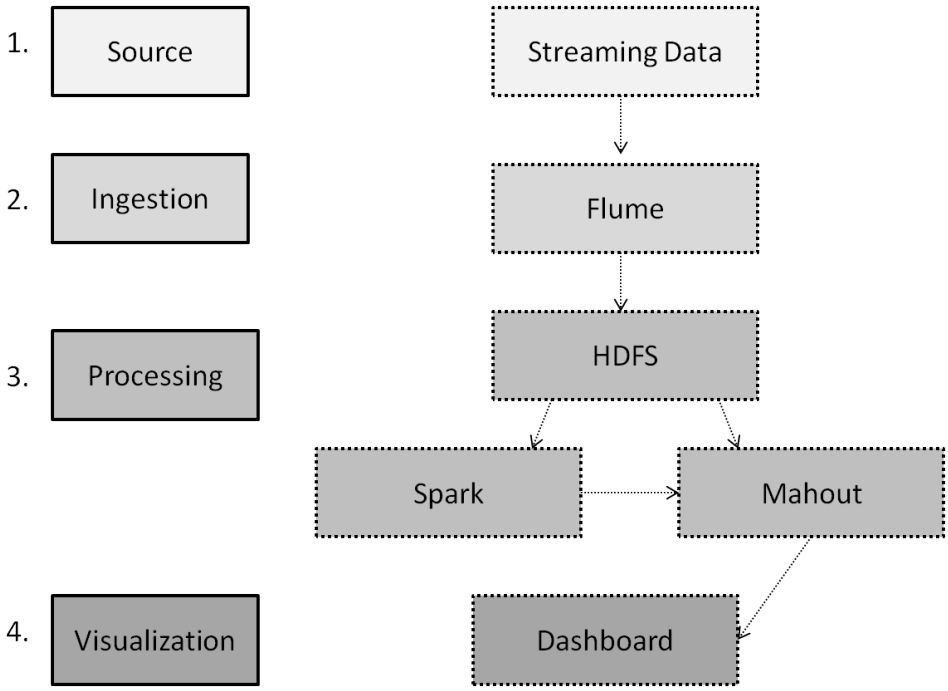


Figure 1.1: Processing Flow.

As for the container resource allocation mechanism, Apache Heron was selected as the stream processing engine of choice. Heron services – Zookeeper, Bookkeeper, Heron Tools (UI and tracker) and Heron API necessary for topology deployment were deployed before submitting the streaming applications (topologies). Information about the topology are handled by the Zookeeper upon submission of the topology and the Bookkeeper handles the topology artifact storage.

- *Evaluations*: We collected information on the workload arrival rates, number of streams and a mix of system performance metrics (CPU, memory) for each streaming application for the resource prediction system, which were then stored for further analysis. As for the resource allocation mechanism, profiles of each topology were obtained and statistics of the workload arrival and service rates at the container-cluster over a period were collected for evaluation.

1.4 Research Methodology

The main concept of Design Science Research (DSR), a method that is conducted under the paradigm of design science to operationalize research, is hereby presented. Also, its application to this dissertation as the research methodology is discussed. The research design strategy used is qualitative as well as quantitative. The qualitative strategy was used for setting performance targets to compare result with existing solution while the quantitative strategy was applied to the metrics for tracking changes.

1.4.1 Design Science Research

The Design Science Research (DSR) methodology involves the design of novel or innovative artifacts and the analysis of the use and/or performance of such artifacts to improve and understand the behavior of different aspects of Information Systems. According to Hevner et al. [33], the main goal of the DSR is to develop knowledge that can be used to develop solutions to problems. This mission can be compared to that of "explanatory sciences" like the natural sciences and sociology, which is to develop knowledge to describe, explain and predict. In DSR, as opposed to explanatory science research, academic research objectives are more of a logical nature. Hevner et al. [33] established seven guidelines to assist researchers, reviewers, editors, and readers understand the requirements for effective DSR as follows:

- *Design as an Artifact*: DSR must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
- *Problem Relevance*: The objective of DSR is to develop technology-based solutions to important and relevant business problems.
- *Design Evaluation*: The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
- *Research Contributions*: Effective DSR must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
- *Research Rigor:* DSR relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.

- *Design as a Search Process*: The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.

- *Research Communication*: DSR must be presented effectively both to technology-oriented as well as management-oriented audiences.

However, the DSR guidance by Hevner et al. [33], although highly regarded and widely cited, is seldom applied because the guidelines are presented at a high-level of abstraction [34]. This is a stark contrast to the research methodology by Peffers et al. [34], which provides guidance on methods. According to Peffers et al. [34], the DSR research model has six steps in its lifecycle as follows:

- *Step 1* is for the awareness of the problem, which is usually achieved by a new development or a reference in the discipline. The output of this phase is an initial research proposal.

- *Step 2* is suggestion of a solution with the output of a tentative design based on the information acquired in *Step 1*.

- *Step 3* focuses on designing and developing the solution, and it has an IT artifact as its output.

- *Step 4* demonstrates the use of the artifact to solve one or more instances of the problem.

- *Step 5* is to evaluate the resulting artifact against the initial problem, and the implicit and explicit criteria extracted in *Steps 1* and *2*. This step is iterative and the output of this step is the performance measures.

- *Step 6* is the final phase, which communicates the output of the overall research results.

Given the artificial nature of organizations and the information systems that support them, the design-science paradigm can play a significant role in resolving the fundamental dilemmas that have plagued information system (IS) research – rigor, relevance, discipline boundaries, behavior, and technology [33]. Within this research setting, the DSR paradigm is suitable with respect to technology as it focuses on creating and evaluating innovative IT artifacts that enable organizations to address important information-related tasks.

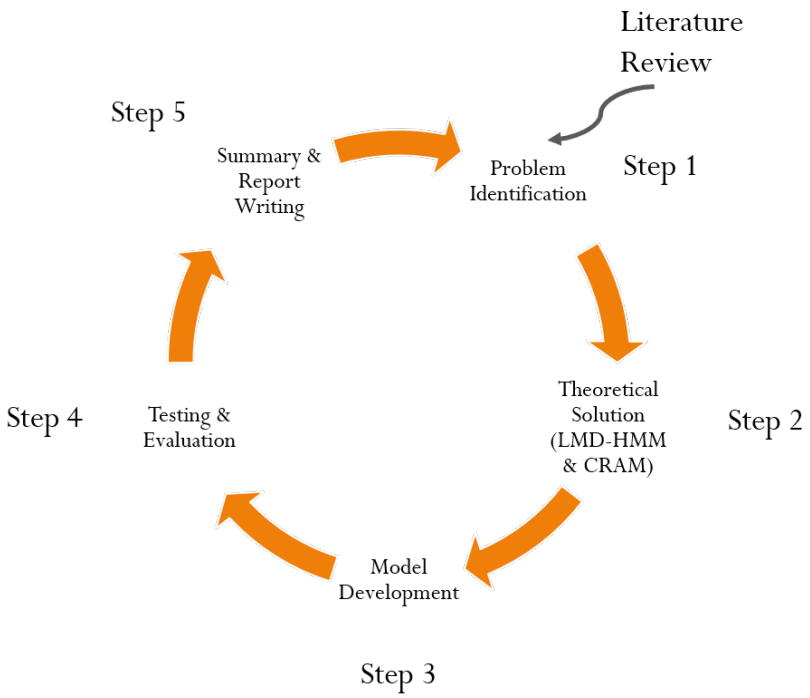


Figure 1.2: Research Model.

1.4.2 Research Model

We followed the Design Science Research in Information Technology methodology to establish our research model. It comprised of several closely related activities as proposed by Peffers et al. [34]. Activities in this model continuously overlap instead of following a sequence and the first step determines how the last step will be taken. Fig. 1.2 depicts our research model, which is discussed as follows:

- *Literature review:* An extensive literature review was conducted to gain in-depth knowledge of the subject matter and to make inferences from the work of experts in the field of Cloud Computing, Big Data, Resource Scaling, Resource Allocation and Containers. The sources of information stemmed from journals, conference proceedings, government reports, academic papers, books, case studies, international standards and best practices.

- *Problem identification and theoretical solution (DSR Steps 1 and 2):* After the problems were identified, theoretical solutions were arrived at.

- *Design and development (DSR Step 3):* Mechanisms for resource scaling and resource allocation were developed. This step required extensive engineering and re-

engineering.

- *Testing and evaluation (DSR Steps 4 and 5):* The mechanisms were tested in real-world and simulated environments, and compliance was checked against performance targets.

- *Summarizing and writing report (DSR Steps 6):* The results were summarized and documented.

1.5 Summary of Contributions

The major contributions of this dissertation are summarized in the following subsections:

1.5.1 LMD-HMM

We present a novel layered multi-dimensional HMM (LMD-HMM) framework with the goal of facilitating automatic scaling up and down of resources, i.e., add or decommission virtual machines. With the layered approach, each application can be modeled individually at the lower-layer while the upper-layer models interactions between groups of streaming applications running on a cluster. The LMD-HMM provides a feasible approach to model time-bounded streaming applications while considering multiple resource bottlenecks. In this case, each dimension of the HMM corresponds to the resource bottleneck being considered for each streaming job and different HMMs can be used to better reflect the nature of the sub-problem. This work has been published in [35].

1.5.2 LMD-HSMM

We consider the use of a layered multi-dimensional hidden semi-Markov model (LMD-HSMM) for modeling unbounded streaming applications that run for an infinite amount of time to accommodate the changes in behaviour from unbounded streaming applications. Since duration is often a critical consideration for streaming applications, the use of the LMD-HSMM allows for explicit modeling of duration in our framework. This helps to avert problems that can arise from taking incorrect scaling actions, e.g., shut down of processing due to lack of resources, fallout from using flawed or less precise models. This work has been published in [36].

1.5.3 CRAM

We investigate the container resource allocation problem and propose a game theoretical approach to address the aforementioned challenge. We focus on a competitive setting involving containerized BDSAs and consider maximizing their payoffs. Game theory, a powerful tool for analyzing and predicting outcomes of interactive decision-making processes, is used to derive the unique, efficient and Pareto optimal states at the Nash equilibrium. The mechanism interacts with BDSAs and periodically maximizes their utility in the presence of load variations to meet their QoS. The CRAM will help to establish fairness among competing containerized applications since the game ends at the equilibrium state. This work has been published in [37].

1.5.4 CRAM-P

In this contribution, we address the problem of dynamic resource pricing to determine the optimal workload allocation due to the conflicting goals of self-interested BDSAs that can lead to congestion. A pricing mechanism based on incentive is proposed to adjust the optimal workload allocation of BDSAs, which ultimately determines which BDSA will enjoy shorter delay. Incorporation of incentives will induce the BDSAs to behave in a way that aligns their objectives with that of the overall CSB system. This work has been submitted for publication in [38].

1.5.5 Publications

These contributions have been published or submitted for publication as follows:

- Cloud resource scaling for big data streaming applications using a layered multi-dimensional hidden Markov model. LMD-HMM has been published in [35].
- Cloud resource scaling for time-bounded and un-bounded big data streaming applications. LMD-HSMM has been published in [36].
- CRAM: a Container resource allocation mechanism for big data streaming applications. CRAM has been published in [37].
- CRAM-P: a Container resource allocation mechanism with dynamic pricing for big data streaming applications. This contribution has been submitted for publication in [38].

1.6 Thesis Organization

The remainder of this dissertation is organized as follows:

- *Chapter 2* briefly discusses the fundamental concepts of the big data paradigm, cloud computing, stream processing and key technologies used for big data stream processing. It presents the challenges of resource prediction and allocation for BDSAs and existing approaches used. Also, big data migration for efficient resource utilization, trends and future research opportunities are discussed.
- *Chapter 3* presents our proposed resource elasticity management and pricing framework.
- *Chapter 4* presents our resource prediction system based on a layered multi-dimensional HMM/HSMM (LMD-HMM and LMD-HSMM) prediction techniques.
- *Chapter 5* presents a game theoretical resource allocation mechanism along with the dynamic pricing scheme for BDSAs.
- *Chapter 6* concludes this dissertation and discusses future work.

Chapter 2

Background

This chapter presents an overview of the big data paradigm, sources and lifecycle. It showcases cloud as a suitable host for big data stream processing, and discusses the key cloud technologies used and big data migration for efficient resource utilization. In addition, major cloud resource elasticity issues — resource scaling and resource allocation — are presented. Some reference models used for the development of the mechanisms proposed in this dissertation are also reviewed.

2.1 Big Data Paradigm

In the past few decades, the continuous increase of computational power has produced an overwhelming flow of data referred to as big data (BD). Big data is not only becoming more available but also more understandable to computers, e.g., Facebook serves 570 billion page views per month, stores 3 billion new photos every month, and manages 25 billion pieces of content [10]. What then is big data? several definitions exist for big data.

According to Gartner, *"Big Data are high-volume, high-velocity, and/or high-variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization"* [10].

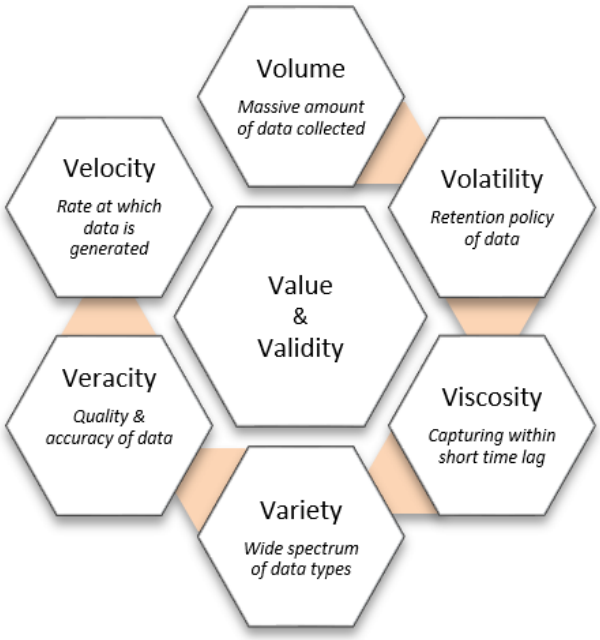


Figure 2.1: Multi-V Characteristics of Big Data.

Originally, IBM and Gartner described big data using three basic characteristics [39]: volume, variety and velocity. The first characteristic is the most dominant. It refers to the massive amount of data collected, processed and analyzed to improve any decision-making process. The gigantic volume of data has been the driving force in developing new forms of data processing applications. More precisely, parallel dataflow graph-based streaming applications, following the MapReduce framework [40], have become the dominant tools to make use of the cloud resources to achieve acceptable response times. Furthermore, newer storage technologies have been developed to facilitate big data hosting in the cloud [41]. However, transferring such massive volumes of data to cloud datacenters puts stringent demands on the needed bandwidth, especially if latency is of concern. Newer networking technologies to increase the available bandwidth is becoming a pressing requirement to meet the exponential growth in traffic. Variety refers to the wide spectrum of data types, formats, semantics and representations of big data. The velocity represents the rate at which data is generated, retrieved, processed and analyzed. For example, big data is retrieved for processing either in real-time, as a continuous stream of inputs near-real time, as a set of records or events in small time intervals, in batches at larger time intervals or offline as blocks of large files [42]. When real-time processing of data is performed, the impact of velocity on the underlying application becomes critical. Velocity can be related directly to elasticity, bandwidth, service availability and delay.

Other big data characteristics have also emerged gradually in the literature [42] as

shown in Fig. 2.1. For example, veracity, viscosity and volatility have also been used to describe big data [39]. Veracity refers to quality and accuracy of the data obtained, which is mostly affected by the nature of its sources but also can relate directly to the error and loss rates during generation, collection and migration. While Viscosity refers to the data being captured and used at an acceptable rate, or with a short time lag from its creation that deems it useful. This characteristic puts stringent requirements for tighter, frequent, elastic and more reliable communication between data sources, cloud resources and the consumers. Volatility has to do with the retention policies where important data is kept and data that is no longer needed is discarded. This can be a challenge big data management as requirements can exceed storage and security capacities of a single cloud.

Business-oriented characteristics, namely, validity and value have also been identified. Validity refers to the suitability of the big data for its intended usage while value refers to the business value that can be gained from the use of the data. The more valuable and valid the data is, the better opportunities various enterprises have to create new products, improve product's time to market (TTM) and quality, minimize their risks, provide better services, and gain insight into their processes [43]. Next, we identify the various sources of big data.

2.1.1 Taxonomy

Big Data can be mapped under five aspects to better understand their characterisitics: (i) data sources, (ii) content format, (iii) data stores, (iv) data staging, and (v) data processing [1]. Fig. 2.2 shows the taxonomy of big data.

2.1.1.1 Data Sources

These refer to the wide variety of sources big data can be collected from, which include:

1. **Sensor and machine generated:** These are large datasets generated by devices used to measure some biological or physical phenomena. The new paradigm, Internet of things (IoT) [44], is also another source for data in this category. The devices used range from fixed sensors that are collecting data at a fixed rate (e.g., sensors in weather stations) to mobile sensors that are used for location or behavior tracking with irregular data collection patterns. The velocity of generation and usage of this data is usually very high. For example, wearable devices are used nowadays to collect vital signs at very small intervals. Web logs and telecommunication networks also

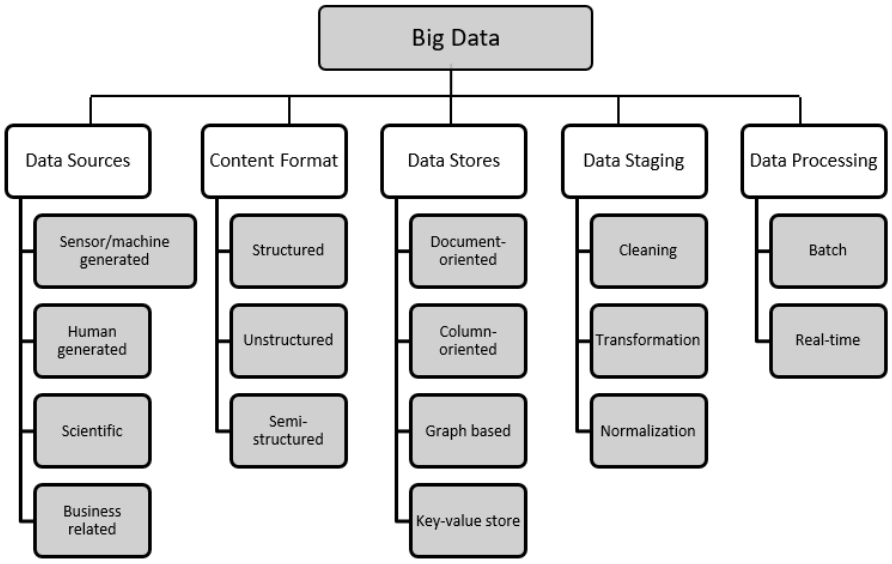


Figure 2.2: Taxonomy of Big Data. Adapted from [1].

generate big data related to web access behavior and network traffic, respectively. Various applications (e.g., home automation, traffic reports, and user tracking) rely on the processing of these datasets.

2. **Human-generated:** These are massive, often unstructured, data sets that are generated by users of social networks. Data is collected and stored in geographically distributed locations, usually closer to the end-users. Data in the cloud in this category is characterized with a very high write rate but analysis and value extraction from these social media datasets are carried out in batches of offline workflows using sequential data access [45]. Nevertheless, critical applications that provide the users with up-to-date statistics [45] require that these big datasets be available in a timely manner.

3. **Scientific:** These sets of data are collected to observe a scientific phenomenon in astronomy, high-energy physics, bioinformatics, etc. [46]. Some of the common features of these big datasets are their extremely large file sizes that are usually written once and accessed many times [47]. Due to their size, they are usually partitioned and stored using different DCs. Their processing applications are compute-intensive and are described using batch jobs. Each job is submitted as a workflow that is comprised of a graph of dependable tasks. The majority of the applications consuming scientific data are usually delay tolerant and are more concerned with monetary budget constraints [48].

4. **Business-related:** These are mostly streams of data records generated by computer-mediated transactions processing billions of financial transactions per day. In this category big datasets are characterized with a very high velocity. The most obvious example is the sales records from big merchants. These big datasets usually follow a common path of extraction, transformation and processing performed on big batches of data. They are then fed into delay tolerant machine learning and data mining applications. However, recently real-time analytics, in addition to some more traditional applications, such as fraud detection, require data to be processed as it is being streamed. They are usually associated with a flow completion time constraint.

2.1.1.2 Content Format

This refers to the structure of big data, which can be in structured, unstructured or semi-structured formats. Structured (e.g., tabular data stored in databases and spreadsheets) and unstructured binary formats (e.g., videos, movies, images, audio) mark the two ends of this spectrum. In between these two extremes, we have a wide range of data types that are neither raw data nor strictly typed (e.g., genome databases, data aggregated from different structured data sources referred to as semi-structured. They do not follow the conventional database system [41].

2.1.1.3 Data Stores

Data stores for big data can be categorized as document-oriented, column-oriented, graph database or key-value stores. The document-oriented data stores are designed to store collection of documents and support several data formats, e.g., JSON, XML [1]. The column-oriented database stores its content in columns, which differs from the approach for storing data in conventional database systems, i.e., row after row. The graph database is designed to store and represent data that utilize a graph model, e.g., Neo4j [49]. An alternative to the relational database system is the key-value store that store data design to scale to a very large size, e.g., Dynamo [50], Apache Cassandra [51], HBase [52], which uses HDFS.

2.1.1.4 Data Staging

The data staging involves the process of cleaning, transforming and normalization. Cleaning helps in identifying incomplete data, transforming allows the change of data to a form

suitable for analysis, while normalization allows the structuring of the database schema to minimize redundancy [53, 54].

2.1.1.5 Data Processing

Big data can be processed either in real-time, batch or as streams [42]. Whilst some big data applications process the arrival of data in batches, some applications require continuous and real-time analysis. Likewise, dataflows arriving via streams need to be analyzed immediately. In the next section, we discuss the phases required for processing big data.

2.1.2 Lifecycle Stages

Big data analytics involves using algorithms running on supporting platforms to unearth potentials concealed in big data, such as hidden patterns. Before any kind of actionable insights from data can be derived using advanced analysis techniques, several preprocessing steps are involved. The high-level phases through which big data is processed basically include data gathering, data management, modeling and visualization [2] as depicted in Fig. 2.3.

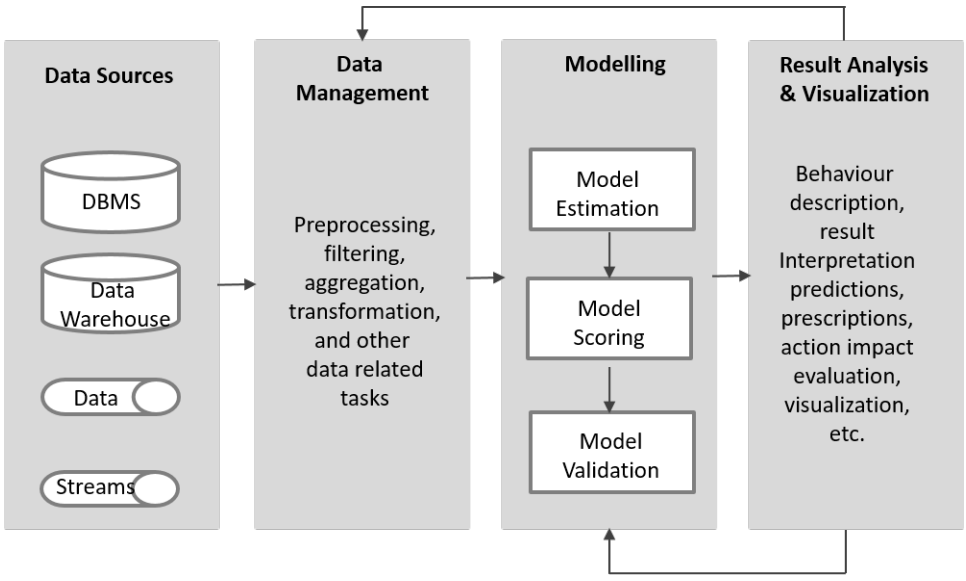


Figure 2.3: High-level Phases of Big Data Lifecycle [2].

The first step involves collecting big data from various sources. At the data management phase, data can be directly stored (e.g., in traditional databases or NoSQL storage) or

passed through a few stages of data preparation before storage. Examples of these stages include data cleaning, extraction, transformation and integration [55]. Big data storage poses a significant challenge due to its volume, hence, solutions that partition and distribute the data must often be employed. The unique characteristics of big data and continuous emergence of new data sources make traditional online analytical processing (OLAP), SQL-based tools and pre-packaged statistical tools unsuitable for processing [56]. Other operations such as data visualization can also take place at this phase. The main challenge in this step is the need for computing platforms that can scale well with respect to the size of the data and its growth rate [40].

The last phase focuses on the use of the developed tools and processes in order to gain insights on the collected data, extract business value and to support decision making processes. Based on the processing time obligation, big data analytics can be categorized into stream or batch processing. On the basis of analytic depth, it can be broadly classified into three categories: 1) *descriptive analytics*, which exploits the historical trends to extract useful information from the data, 2) *predictive analytics* that focuses on predicting future probability of occurrence of pattern or trends, and 3) *prescriptive analytics*, which focuses on decision making by gaining insights into the system behavior [57].

2.1.3 Services

Various services can be offered by the clouds to big data during its life cycle. The three main service models as identified by NIST [58] are platform as a service (PaaS), which provides tool and resources for development, testing and management of applications, software as a service (SaaS) is for provisioning of various applications on demand to cloud consumers, and infrastructure as a service (IaaS) is the basic service model where a virtual network of VMs and storage units communicate over virtual infrastructures.

Big data users can use the IaaS service to create their own virtual infrastructures in order to gain full control over various network, storage and software management functionalities (e.g., the user's own networking protocols, scheduling of VM jobs and data placement). Additional cloud services are also offered at this layer such as network-as-a-service [59], which provides traffic engineering services to the big data traffic. Rather than renting network resources, Transfer-as-a-service for big data [60] also advocates the idea of delegating the burden of data transfers from the users to the cloud providers. To provide other basic services to big data processing, database-as-a-service [61], big data-as-a-service and storage-as-a-service [62] are all necessary tools to enhance the efficiency and reduce

the cost of storing and managing big data in the cloud. Storage-as-a-service provides customers the illusion of unlimited storage by seamlessly storing big data across multiple DCs. Replication-as-a-service helps users to reduce the latency of data access and increase access reliability [63].

For big data users with a focus on developing big data applications, PaaS offerings are provided [64]. Cloud-based analytics-as-a-service was proposed by the authors in [65] as a platform to provide on-demand OLAP and mining services to cloud consumers. Data programming platforms (e.g., Hadoop, Dryad, Hive and Pig) [41] and similar platforms are commonly used now by programmers to describe their workflow-based jobs (e.g., a data mining operation) through user-interfaces [64]. Finally, SaaS offerings create value and knowledge from big data. Examples of such services include IoT applications, knowledge extraction, data analytics and business intelligence software tools [66]. In addition, identity-, policy management- and big data security-as-a-service can be used to facilitate user authentication and role assignments as well as ensure big data security.

2.1.4 Challenges

Big data offers substantial value to organizations willing to adopt it, but, it poses a considerable number of challenges for the realization of such added value. Research on big data still remains in its early stages despite its acceptance by many organizations. Several issues continue to emerge as existing challenges are being addressed. Issues that affect big data can be grouped into the storage and transport, processing, and management issues [67].

- *Storage and Transport Challenges:* These are challenges that relate to the data characteristics discussed in Section 2.1. The volume of data generated today has exploded — largely due to social media. Moreover, data is being created by everyone and everything. Current disk technology limits poses a challenge and transferring Exabytes of data from one access point to the other takes longer.

- *Processing Challenges:* These are challenges encountered when processing the big data that relates to its lifecycle as discussed in Section 2.1.2, i.e., from capturing to presenting the results. Traditional data processing and storage approaches were designed in an era when available hardware, storage and processing requirements were very different than they are today. Thus, those approaches are facing many challenges in addressing big data demands. In the big data community, MapReduce [40] has been identified as one of the key enabling approaches for meeting continuously

increasing demands on computing resources imposed by massive data sets. MapReduce is a highly scalable, fault-tolerant, simple programming paradigm capable of processing massive volumes of data by means of parallel execution on a large number of commodity computing nodes [68]. Recently, due to the real-time needs of organizations, various SPEs are being introduced.

- *Management Challenges:* These are challenges encountered while accessing, managing and governing big data. These include challenges such as security, privacy, data governance, ownership and cost [67]. Scalability and availability have also been identified to be a critical part of managing big data.

In the next section, we briefly provide an overview of cloud DCs as host for big data.

2.2 Cloud Datacenters

A cloud is comprised of multiple datacenters (DCs) located in distributed geographical locations. A DC can be viewed as a collection of compute resources (e.g., storage and servers) that are connected through networking devices (e.g., routers and switches). DC networks are connected to other Internet service providers (ISPs) to provide access to end-users. Fig. 2.4 shows a conventional datacenter network topology.

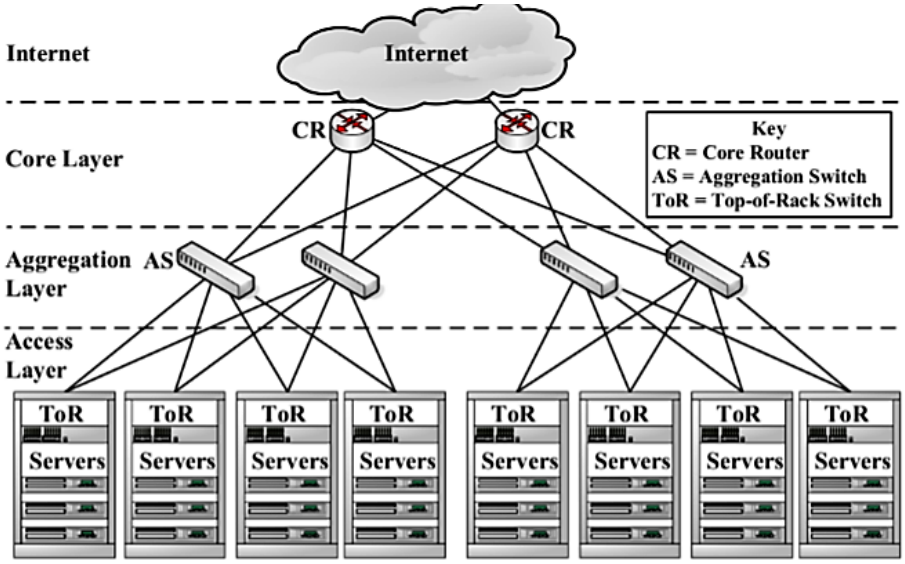


Figure 2.4: Datacenter Network Topology [3].

The most commonly used topology in DCs is the multi-rooted tree topology; the roots of the tree are Top-of-Rack (ToR) switches that are directly connected to the DC servers,

storage devices, database servers, etc. In larger DCs, the servers might be connected to a hierarchy of aggregation switches (AS). Within the DC, a ToR switch may then be connected directly or through one or more layer-two switches to border routers in order to communicate with access networks or other DCs. Such a topology has already proven to be inefficient for big data processing due to the high communication rate between pairs of VMs within the same racks or on different ones. To overcome this limitation, several other topologies have also been adopted to mitigate the congestion of the links connecting to the top tiers and to increase DC fault tolerance. For example, communication path redundancy, congestion reduction and reliability can be achieved by Clos networks, FatTrees and hyperCube-based topologies [69]. A full survey on DC architectures is presented by the authors in [69].

Ethernet switches are commonly used in DC networks with link capacities in the range of 1-10 Gbps (i.e., smaller capacities for links connecting servers and higher capacities for those connecting ToR switches) with the promise of future higher bandwidth of 40-100 Gbps [69, 70]. Nonetheless, optical switching has also been adopted within the DCs to produce either a hybrid or a purely optical DC [70]. Helios and C-through [71] are two examples of the former where in Helios optical core switches are used mainly to connect ToR switches.

2.2.0.1 Intra-clouds DC

The main DC network management functionalities include efficient traffic engineering (TE) (e.g., admission control, rate limiting, load balancing and traffic prioritization) for flows between pairs of VMs, handling the migration of data within servers and routing the resulting traffic among these VMs to and from the DC. Orchestration of these functionalities is carried out through interactions of various protocol layers along with the execution of various TE mechanisms as will be discussed in details in Section 2.3.4.

2.2.0.2 Inter-clouds DC

The inter-cloud DCs are interconnected through wide area networks (WANs) [72] to form one cloud if they are owned by the same provider or to construct a federation of clouds of cooperating providers [73]. The WANs may be privately constructed and owned (e.g., Google's B4 [72]). They can also be dynamically created on demand using the Internet backbone through virtual networks [59]. Recent trends are also using software defined network (SDN) technologies to create these WANs [74]. In all cases, the traffic within the

DC is aggregated by multiple edge routers before it leaves the DC. With current optical network technologies, these WANs can reach transfer rates in the range of 10 Gbps and up to Tbps [75]. Long-lived flow transfers between DCs can range from a few Terabytes or Petabytes with deadlines in the orders of a few hours to a couple of days.

2.2.1 Hypervisor and Container-based Virtualization

Virtualization techniques in the cloud has shown to provide opportunities to fully abstract logical resources from the underlying physical host resources. It improves agility, flexibility and reduces cost [76]. Virtualization enables multi-tenancy, i.e., allows multiple instances of virtualized applications (tenants) to share a physical server. It is the key enabler of DC functionalities, used in DCs and cloud environments to decouple applications from the hardware they run on. The basic component of virtualization is the virtual machine monitor (VMM), i.e., hypervisor. The VMM is a software-abstraction layer that enables the partitioning of the underlying hardware platform into one or more virtual machines (VMs) [77]. VMMs provide strong isolation between VMs, which enables multiplexing of multiple VMs over the same hardware. Different types of resources can be virtualized in the cloud such as server, storage, and network. There are two types of virtualization — hardware-level or hypervisor virtualization and operating system (OS) level virtualization.

The hypervisor virtualization consists of using a virtual machine monitor (VMM) on top of a host operating system (OS) that provides a full abstraction to the VM. In this case, each VM has its own OS that executes completely isolated from the others, allowing for multiple different OSs to be executed on a single host. Hypervisors enable system administrators to optimize the use of available resources and confine individual parts of the application infrastructure [78]. VMware ESX[1], Xen[2], and Hyper-V[3] are examples of hypervisors. With recent developments around Docker [79, 80], Kubernetes [81] and LXC [82, 83], a lightweight alternative to the hypervisors has emerged, i.e., container-based virtualization. While hardware-level virtualization is one of the most popular virtualization technology for deploying, packaging, and managing applications today, OS-level virtualization has surged in interest and deployment due to its promise of low-cost and improved performance [31].

OS virtualization virtualizes resources at the OS level by encapsulating OS processes and their dependencies to create containers, which are managed by the underlying OS kernel [84]. *"A container is a collection of operating system kernel utilities configured to*

[1]https://www.vmware.com/
[2]https://xenproject.org/
[3]https://docs.docker.com/machine/drivers/hyper-v/

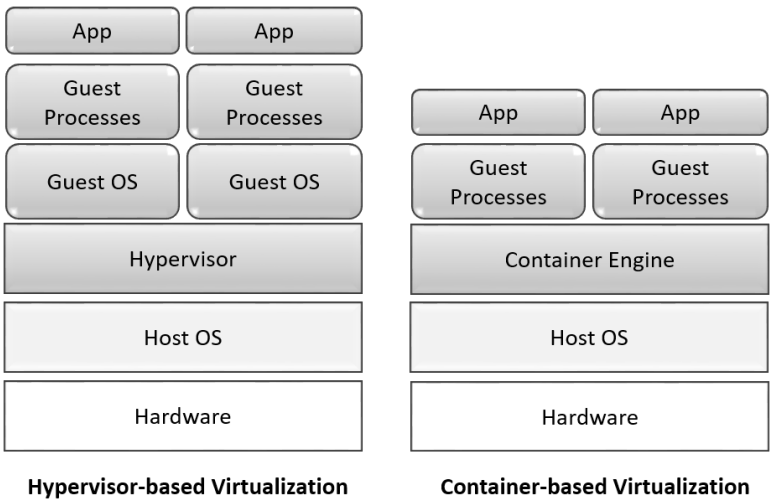


Figure 2.5: Comparison of Container-based and Hypervisor-based Virtualization

manage the physical hardware resources used by a particular application component" [85]. OS virtualization partitions the hosts' resources, creating multiple isolated instances on the same OS [84]. Fig. 2.5 provides key architectural differences between hypervisor-based and OS-level virtualization.

Compared to hypervisor-based virtualization, containers impose less overhead for resource virtualization since they share the OS kernel of the physical machine. One of the core technologies in container is the *Cgroup*, which is a mechanism for grouping and limiting resource of tasks. Until now, most research discuss the elasticity of stream processing system in VM-based cloud [31]. Since containers provide near-native performance and millisecond-level startup time, it is suitable for elastic stream processing system, which requires fast and efficient resource provisioning [31]. Also, through nesting, one or more containers, i.e., container-cluster, can be encapsulated inside a VM [86] and applications running in containers are able to benefit from the security and performance isolation provided by the VMs. While this approach provides an opportunity for reusing some existing hardware virtualization-based architecture for resource provisioning and management, some solutions may not be suitable when dealing with multiple containerized big data streaming applications. Also, in comparison to deploying containers on a physical host, a recent study by the authors in [83] found that the deployment of containers within VMs impose additional performance overhead and adds no extra benefit.

Container-clusters can be managed using solutions like Mesos[4] and Kubernetes[5] that provide applications with APIs for resource management and scheduling across cloud envi-

[4]https://mesos.apache.org/
[5]https://kubernetes.io/

ronments [86]. Kubernetes is an open-source technology for automating deployment, operations, and scaling of containerized applications [85]. Today, most Cloud service providers, e.g., Amazon, Azure, Google and IBM have extended their offerings to provide containers as a service (CaaS), enabling robust container-based platforms to build, deploy or host applications on a lightweight environment. While most research on resource management in the cloud have focused on hypervisor-based environment, the emergence of OS virtualization has reopened a number of issues related to resource management on container-based cloud. To deploy BDSAs on containers, the microservices approach is used. A microservices-based BDSA involves the interoperation of multiple microservices. Each microservice acts as a standalone application or component that can be developed, deployed and updated separately in contrast to the traditional monolithic development of applications. The ability to independently update and redeploy the code base of one or more microservices increases application's scalability, portability, updatability, and availability, but at the cost of expensive remote calls and increased overhead for cross-component synchronization [85].

Container technologies such as Docker can support multiple and heterogeneous microservices in a containerized environment. However, this can lead to unexpected interference and contention among microservices. Thus, how to deploy microservices in order to balance resource consumption and performance is a critical issue [85]. Also, tracking the performance of microservices in a containerized environment can be very challenging. Most available tools typically monitor CPU, memory, and filesystems. Monitoring frameworks such as Nagios[6] and Ganglia[7] monitor hardware metrics (cluster, CPU, and memory utilization) and cannot provide microservice-level performance metrics. Therefore, new monitoring frameworks are required. Some newer tools to monitor microservices are available from providers such as Sysdig[8] and Datadog[9]. Estimating microservice workload behavior in terms of request arrival rate can be a tedious task when dealing with multiple users with different types and mix of microservices. In addition, devising microservice-specific workload models to accurately learn and fit statistical functions of the monitored distributions is also challenging. The aforementioned challenges can be attributed to making decisions about the types and amount of resources to provision at a given time combined with the unpredictability of the availability, load, and throughput of resources. Many Cloud service providers utilize rule-based reactive approaches to scale resources for containers, e.g., Ama-

[6]https://www.nagios.org/
[7]http://ganglia.sourceforge.net/
[8]https://sysdig.com/
[9]https://www.datadoghq.com/

zon AutoScale[10]. Some container schedulers (e.g., Kubernetes[11]) also scale by observing the resource usage.

2.2.2 Cloud DCs as Host for Big Data

Today, we live in a universe where government and various business organizations are overflowing with data generated from Internet search, social media, mobile devices, the Internet of Things and business transactions among others. Traditional data processing technologies are unable to process this data within a tolerable elapsed time. Moreover, the datasets generated do not fit into the traditional data processing frameworks and are usually generated from multiple distributed data sources. Due to these identified characteristics, big data represents a new paradigm of data management (collection, processing, querying, data types, and scale) that is not well served by traditional data management systems. Consequently, as various sectors continue to use larger and larger data warehouses for their ever-increasing data processing needs, the performance requirements continue to outpace the capabilities of the traditional approaches, hence, the need for cloud DCs. To process data as they arrive, elastic and virtualized cloud DCs use data processing frameworks that can continuously process high-volume and high-velocity data streams. This development has attracted individuals and organizations and they are increasingly moving their data to the cloud for higher reliability, security and cost benefits.

Cloud services are a natural fit for processing big data streams because they allow data processing frameworks and algorithms to run at the scale required for handling uncertain data volume, variety, and velocity. Cloud computing promises on-demand access to affordable large-scale resources with no substantial upfront investment. Also, it offers a means for meeting the performance and scalability requirements of organizations by providing agility to the data management infrastructure. By this, cloud users can benefit from the pay-as-you-go model and adaptable resources. Furthermore, it frees up organizations from the need to purchase unneeded resources. Several other motivations for using the cloud computing model include the ease of use, elasticity, reliability, flexibility, performance, security among others [87]. Cloud computing allows workloads to be deployed quickly through rapid provisioning of virtualized resources [88].

The main deployment models of the cloud are the public, private, community or hybrid clouds. On the one hand, a private cloud (e.g., Microsoft's DCs), which provides the highest

[10]https://aws.amazon.com/autoscaling/
[11]https://kubernetes.io/

level of security, is deployed over private networks and can only be accessed by subscribed users. On the other hand, a public cloud, such as AWS, is deployed over the Internet or rented network resources. In between these two deployment models is a hybrid cloud that can be comprised of both private and public resources. Additionally, the community clouds serve organizations that have common management goals [58]. Acquiring computational resources as commercial services in a pay-per-use model according to the cloud stack (SaaS, PaaS, IaaS) is one of the key promises of cloud computing. Each of these models provides different views to the users for the type of resource available and how they can be accessed. For instance, in the IaaS model, a user's view is restricted to the operating system, whereas the PaaS model provides users with an environment for deploying their applications.

The provisioned resources are abstracted using virtualization technology and deployed to customers in form of VMs or containers. Sharing the underlying physical cloud infrastructure allows for workload consolidation, which leads to better system utilization and energy efficiency [4]. While most cloud service consumers enjoy the flexible and cost-effective services cloud computing provides, meeting SLA, scalability, and deadline constraints while achieving cost-effectiveness can be challenging. Hence, there is a need for frameworks that can effectively address the challenges plaguing resource usage in the cloud. Such frameworks tailored to serve application requests should i) be capable of handling different types of applications, ii) perform resource management tasks intelligently while meeting the QoS, and iii) adapt to fluctuating resource requirements posed by various applications.

Big data streams can be processed using cloud infrastructures, which depends on communication between virtual machines (VMs) or containers. Imparting cloud resource elasticity is the fact that most CSBs host BDSAs on massive geo-distributed datacenters (DCs), which involves huge data migrations across the underlying network infrastructure, leading to long migration delay and cost. Data transfer using the network becomes the bottleneck in the process [89]. While most efforts have been devoted to designing better resource management solutions for big data processing, big data migration for efficient resource utilization is an issue that has been largely left out. Therefore, there is a need for timely and cost-minimizing frameworks to address the big data migration challenge faced by CSBs.

2.3 Big Data Migration over Clouds

In this section, we discuss research contributions made on cloud big data migration in order to put into perspective a more holistic view of its challenges, existing solutions and effects on efficient CSB resource elasticity management. Big data migration to clouds was

initially driven by the scalability and computational performance limitations of traditional infrastructures and the elastic, low cost resources provided by the cloud. However, the use of cloud infrastructures comes with other challenges, which includes reliability, accessibility, cost reduction, result timeliness and outcome accuracy. The multitude of challenges encountered has ushered in numerous yet diverse research contributions from both industry and academia. These research contributions cover a wide spectrum of solutions such as developing newer protocols at various communication layers, adoption of software defined networks (SDN), in-network big data processing as well as opportunistic store-and-forward routing.

2.3.1 Requirements

The three levels of Big data movement over clouds can be separated into (i) big data movement to the cloud from external sources, (ii) intra-DC transfers, (iii) inter-DC transfers either through the communication of individual VMs at different DCs or interaction between peer DCs [90]. The types of traffic that can occur at these levels can be classified as follows:

- *Streaming data:* These transfers involve sending data as continuous real-time streams without the start and stop modes [91]. Streaming of data can use single or parallel streams in order to increase the transfer rate.
- *Bulk data traffic:* These transfers are delay-tolerant and have no explicit deadlines or have deadlines that span a few days. They can be transfers between the users and the clouds or between pairs of DCs [92]. They can also be as a result of maintenance operations that include migration and data replication due to load balancing.
- *Online batch data transfers:* These transfers are much smaller in volume than bulk transfers and are mostly related to MapReduce-like applications' operation [40].
- *Live VM migration traffic:* This operation is triggered by the DC management system and it is crucial to the administrative operations of the DCs. It allows the DC to free additional resources or perform maintenance operations on the hosting server [93].
- *Big data replication and migration traffic:* This type of traffic can be based on the triggers of replication and migration, e.g., Google and other DC operators hosting cloud-computing applications need to replicate and synchronize data across DCs [94]. Big data migration helps the cloud service providers to improve redundancy and increase the reliability of their DCs to prevent site failures.

The need for migrating big data over clouds can be addressed from both the CSBs' and CSPs' perspectives.

2.3.1.1 CSBs' Perspective

To a CSB, the following requirements are critical when migrating big data over clouds.

- *Sufficient flow transmission rates:* This is the most common requirements for all traffic patterns. However, it is usually traded-off for decrease in the service cost.
- *Latency:* This is the time delay experienced in migrating and processing data. This can be as a result of many factors such as the distance involved, routing path and the different technologies used in the transfer.
- *Cost:* Cost is a key factor in big data transfer, considering the geo-dispersion of users, the geo-distribution of DCs, and the characteristics of data involved. Attempts to find a balance between the various elements that affect big data transfer may result in added cost, e.g., the use of more cloud resources can reduce transfer delays but this leads to higher a cost for the consumers.
- *Reliability:* Storing big data sets across multiple DCs increases their access reliability but eliminates data locality and adds additional bandwidth needs between these DCs. Hence, a good balance between these two factors must be achieved.
- *Elasticity:* This parameter refers to the application's ability to temporarily consume additional or fewer resources dynamically without incurring any delay.
- *Optimal placement of big data:* This must be carried out by the CSB in order to meet users' specified business rules and policies.

2.3.1.2 CSP's Perspective

From a CSP's perspective, requirements to be met when migrating big data include the following:

- *Optimal resource utilization:* This is the main objective of any service provider, which in turn leads to increased profit. It can be achieved through good cross-layer orchestration of various network and application functionalities.

- *Scalability:* This reflects the ability of the provider to employ the resources in serving a large number of big data flows without experiencing any deterioration in the offered service performance.

- *Ease of solution deployment and maintenance:* The introduced newer protocols, algorithms and tools must be easily deployed on the large scale needed within DC networks, and compatible with existing ones.

- *Ease of management:* This requirement refers to the need for automation of resource configurations and dynamic adaptation.

- *Reduced operating cost:* Energy optimized solutions are needed in order to reduce the cost of operating the cloud infrastructures.

Next, we discuss the various research progress made to address the challenges of migrating big data over clouds, which can be classified into networking and non-networking approaches.

2.3.2 Non-networking Approaches

In this section, we examine various non-networking solutions, i.e., application strategies and big data management approaches that target the efficient migration and movement big data over clouds while satisfying the aforementioned requirements of the cloud users and providers. These approaches include the use of physical means to transfer data, volume reduction and data placement.

- *Shipping vs Internet:* Some research efforts [48, 95] have focused on investigating the trade-offs between physical big data transfers using hard-drives and the use of online services provided by clouds. Gorcitz et al. [96] focused on delay tolerant big data transfers between end-users and the cloud as well. They leveraged the storage capacity and communication capabilities of future smart vehicles in order to piggyback massive data that needs to be transferred between two locations. While performance results of data transfers between two cities showed a significant speed up as compared to Internet-based techniques, the proposed work is very difficult to implement in real life scenarios. Moreover, the scheme is dependent on several unpredictable parameters such as the weather and the traffic conditions.

- *Application and Big Data Adaptation:* This stream of research effort is motivated by the asymmetry between the fast growth rates of big data sizes and the slow growth in the available bandwidth in current network technologies. This involves the use of a compression algorithms to compact the data before transferring [97] and later decompress when needed. This approach was also used by the authors in [98] to optimize the input/output issues involved with big data transfer. Data deduplication is another means to reduce the big data volume before transfer [99]. This method is used to reformat the data into a more condensed form while allowing only necessary information to be transferred.

The main limitation of these approaches is that they require extensive compute capabilities on the source side. In addition, the majority of these schemes require profound knowledge of the transferred data types. Furthermore, compression decisions are still complex, i.e., the system must decide the granularity of compression (e.g., database records, blocks, single file or a set of files) and the appropriate compression algorithms.

2.3.3 Networking Approaches

In this section, we classify and analyze the main networking approach contributions for optimizing big data migration across the protocol layers – application, transport and network layers.

2.3.3.1 Application Layer Protocols and Tools

Conventional protocols at this layer such as the file transfer protocol (FTP) and the client-server HTTP fall short of meeting the needs of large scale data transfers in terms of the needed reliability and synchronization with existing data. Research efforts have either adapted these tools to suit big data transfers or have opted for developing newer protocols. Examples of tools proposed include GridFTP [100], Globus Online [101–103], BitTorrent[12], JetStream [104], Fast and Secure Protocol (FASP) [13], CloudStork [105], BBCP [106], Flexible Automated Synchronization Transfer (FAST) [107] and Genomic Text Transfer Protocol (GTTP) [108].

[12]wwww.bittorrent.com
[13]https://asperasoft.com/technology/transport/fasp/

2.3.3.2 Transport Layer Protocols

Today, most data transfers within the DC rely on the transmission control protocol (TCP). Unfortunately, TCP throughput collapse is frequently experienced in high bandwidth, low latency DC networks. In order to enhance the performance of the TCP protocol to suit big data application needs, the DC TCP protocol (DCTCP) [109] has been proposed but it fails to consider deadline sensitive flows as well as flow prioritization. To address this limitation, flow deadline and congestion avoidance have been simultaneously considered in the Adaptive-Acceleration DC TCP protocol (A^2DCTCP) [110]. Several research efforts have also focused on increasing the transfer throughput of TCP sessions either for individual files or for overall utilization of the network through the adjustment of the parameters of several transfer operations, i.e., pipelining, parallelism and concurrency. In general, the majority of the aforementioned techniques either have some performance issues or require significant modifications of the protocol either at the sender, receiver, and both sides or at the network switches hardware.

Alternatives to TCP such as UDP-based data transfer (UDT) [111] have also been introduced in the literature. In contrast to traditional TCP, UDT supports extremely high-speed networks and it can be used to transfer bulk datasets in excess of Terabytes. However, UDT suffers from extreme overhead requirements as well as the induction of significant delays in concurrent TCP transmissions. Finally, it is worth noting that recent trends in DCs advocate for the use of Remote Direct Memory Access (RDMA) and message passing either over RDMA ready networking technologies (e.g., Infiniband and Converged Ethernet (RoCE)). These technologies are clearly able to scale well to the big data needs but require expensive hardware support as well as a congestion-free layer-two circuit for the case of RoCE in order to work well. They also require significant modifications in the applications in order to make full use of this technology's potential [112].

2.3.3.3 Network Level Protocols

In contrast to the application and transport layer approaches, research efforts at the network layer are mostly implemented by the CSP rather than the individual users (e.g., CSB) and focus mostly on the aggregated traffic between DCs. For example, Zhang et al. [113] focus on optimizing the monetary cost of transferring data to the cloud in order to be processed using MapReduce-like frameworks. Similar to the aforementioned approach, the authors in [60] consider big data transfers between clouds in a federation of cloud providers. However, the proposed work did not discuss the details of how fairness among the users'

transferred datasets can be achieved.

2.3.4 State-of-Art Networking Protocols

Links connecting the DCs are reportedly underutilized with an average of 40-60% utilization levels [114]. This problem is a direct result of the bursty nature of the traffic between the DCs. To overcome these problems network virtualization, software defined networks (SDN), in-network processing as well as store and forward (SnF) nodes have been used. These technologies are discussed below.

2.3.4.1 Network Virtualization

Current research efforts focus on intra-DC networks virtualization to provide bandwidth sharing between tenants in a single data center. For example, the authors in [115] created a VN topology for each MapReduce workflow and then tried to concurrently map the VM and links to the DC network with the objective of minimizing the data transmission latency and data processing rate. Another bandwidth allocation technique, termed Seawall [116], performs end-to-end proportional sharing of the network resources according to an associated weight with each traffic source using congestion-controlled tunnels. Similarly, Lam et al. [117] proposed Netshare, a statistical multiplexing mechanism that proportionally allocates bandwidth for the tenants. Unlike Seawall, Netshare shares link bandwidth among tenants rather than sender virtual machines to provide a fair allocation among different services.

2.3.4.2 Software Defined Networking (SDN)

The main idea behind SDN is the ability to provide a user-friendly programming interface to dynamically control the run-time configuration of the underlying network. This is achieved through the decoupling of the control (i.e., the messages needed to configure the various network devices) and the data plan (e.g., the paths that actual data flows take in the network). OpenFlow [72, 118] is the most commonly used implementation of SDN and it relies on configurable flow tables in each network device that contain a set of instructions on how to forward a received data packet. These tables are configured by a centralized SDN controller at run time and mostly upon receiving a first packet of the flow. Recently, research efforts have focused on exploring the suitability of SDNs to carry big data flows within the DC. For example, Wang et al. [119] focus on MapReduce applications and discuss

the potential of using application level information, including the expected communication pattern and the traffic demand matrix to configure the network at runtime.

SDNs are also emerging as a new technology for inter-DC communication. For example, Google's B4 WAN [72], using OpenFlow, is an SDN employed to connect its DCs with a promise of 70-100% utilization of its links. Similarly, Microsoft's SWAN [114] creates an SDN to enable its inter-DC links to carry more traffic, taking into consideration higher-priority services and fairness among similar services. The main challenge of employing SDNs for big data is the volume and the speed of the serviced data flows as well as the number of concurrent flows.

2.3.4.3 In-network Data Processing

In addition to the above approaches, newer research directions employ some processing functionalities to the data along their network path in order to reduce the size of the flows as they are transferred through the network. Research in this direction focuses on in-network processing for big data in order to reduce the overall needed bandwidth. This is carried out by developing various operations to modify the packets content during its trip in the network. The main tradeoff in these approaches is the difficulty and the cost associated with implementing, deploying and maintaining application specific functionalities that are customized for different flows along the network.

2.3.4.4 Store and Forward Techniques

Temporarily storing big data in network nodes along its way to its destination and only opportunistically sending data between these nodes is a technique that is usually referred to as *water filling* or *store and forward (SnF)*. Approaches in this category can help in overcoming network congestion during peak hours. However, implementing these approaches is not straightforward due to the varying arrival deadlines for different data transfers, the incurred cost of temporarily storing the data and the difficulty in predicting the needs of newly arriving transfers that might require a higher transfer priority over already stored data. One of the earliest approaches for SnF is NetStitcher introduced by Laoutaris et al. [93, 94]. Postcard [120] extends Netsticher's time expanded graph to consider multiple bulk data transfers, but results in a higher computational complexity to schedule these transfers. The success of these techniques hinges upon having the right proportion between the number of transfers that are more delay tolerant as compared to the more delay

sensitive traffic. Therefore, they are mostly realized for traffic that span DCs with different time zones and hence, different traffic patterns in each zone.

2.3.4.5 Traffic Engineering (TE)

Rate limiting and admission control are the most used TE mechanisms for controlling the behavior of big data migration. In general, rate limiting can take place at different levels — at the flow-, VM- or the job-level. In [75], the authors relied on prediction of the high priority traffic of big data between each pair of data centers. They used bandwidth reservations in future time blocks to serve long-lived flows and employed online temporal planning to appropriately pack these transfers. While the scheme aimed at minimizing the fraction of unsatisfied transfer, it did not guarantee any deadlines. In Amoeba [121], both admission control as well as per-flow rate limiting were employed to serve new incoming transfer requests. The system employed opportunistic rescheduling of existing transfers with the objective of meeting the transfer deadlines of all the flows. In [122], the authors developed MicroTE, an intra-DC TE mechanism that focused on isolating predictable traffic flows from those that cannot be predicted.

2.3.5 Optimization Objectives

In the last two sections, we examined existing approaches with respect to the protocol layers at which they function with the aim of coordinating their behaviour. In this section, we analyze a number of research contributions from their targeted objectives (e.g., service time reduction, cost minimization and increased performance). These objectives are heavily interdependent and sometimes contradicting. We aim at highlighting these interdependencies.

2.3.5.1 Time Reduction

Big data transfer time increases as the volume of data to be transferred increases. Hence, service time reduction becomes a higher priority for various mechanisms. Most often, to reduce the time involved for moving big data, policies are applied through approaches such as scheduling and resource allocation. In schemes that target flows and completion time need to consider several other factors such as fairness, deadline variations, and the life span of different flows. Also, additional resources employed to optimize the transfer time

increases the cost of such transfers and may not result in the optimal use of the underlying resources.

2.3.5.2 Performance Improvement

A large number of research efforts have focused on improving the performance of big data transfer using various protocol layers. For instance, multi-hop path splitting and multi-paths are solutions that employ multiple independent routes to transfer data [123,124]. The parallelism mechanism employed in the multi-hop and multi-path systems is an approach successfully used to scale data transfers but this may introduce bottlenecks from local constraints such as over-tasked CPUs, low disk I/O speeds [123]. Cross-layer optimization, introduced by the authors in [90] involves using information from different layers for efficient decision-making. In addition, issues encountered when transferring big data to the cloud can be addressed by in-network data optimization techniques [125] such as compression, de-duplication, caching and protocol optimization. Compression [98] and data de-duplication are used to reduce the size of the data before transferring.

2.3.5.3 Cost Minimization

Research work in this area focuses on cost-minimization strategies. Big data transfer between user location and the cloud incurs considerable cost, which can translate to millions for cloud service consumers. Hence, cost minimization strategies such as better models are required. The cost model employed involves passing the transfer charges by the Internet service providers (ISPs) to CSPs, and these are then passed on to the users. This charge is derived using a percentile charge model, which is different from the flat-rate charge model used for utility billings [126]. Studies carried out to minimize resource cost can sometimes be based on the heuristics that some delays are tolerable when transferring big data [94].

NetEx [127] was presented to address the challenge of high bandwidth costs charged by cloud service providers. This system utilizes unused bandwidth on ISPs' backbone links, offered at a lower price, to cheaply deliver content by using bandwidth aware-routing to dynamically adapt available bandwidth across multiple paths. Work carried out in [126] focused on how to minimize cost when moving dynamically generated, geo-dispersed data BD into the cloud for processing. Two online algorithms — the online lazy migration (OLM) algorithm and the randomized fixed horizon control (RFHC) algorithms — were proposed to optimize the choice of DC to aggregate and process data. Netstitcher [93] is also another example of a solution proposed to reduce the high cost of big data transfer.

The idea behind Netstitcher is to rescue unutilized bandwidth across DC and use it to transfer data for non-real-time applications using a store-and-forward algorithm, which can adapt to resource fluctuations.

2.3.6 Discussions and Future Opportunities

Migrating big data over clouds can be a complicated and an expensive process. Traditional transfer tools are found to be inefficient and exhibit poor performance when handling big data. Moreover, they are poorly documented and require considerable efforts for setting up. Minimizing costs and latencies are of interest when transferring big data. Having identified the major research contributions towards developing efficient frameworks for big data migration over clouds, Fig. 2.6 provides a holistic view of the aforementioned research contributions. Clearly, the main hurdle in the area of big data migration over clouds is the isolation of various research contributions at each of the layers. As shown in Fig. 2.6, big data requirements are not easily mapped from the top layer to the lower layer configurations due to lack of efficient models to represent these needs. While various strategies for the movement of big data have been proposed (e.g., shipping, Internet and private networks), these strategies do not consider the current or predicted availability of the underlying resources nor do they consider the dependency among various big data sets.

At the application level, various efficient strategies have been proposed to reduce the size of the transported big data and to configure optimal placement strategies with respect to the users' access delays. However, a decision model that can decide when, where and which big data manipulation functionalities (e.g., compression, aggregation and deduplication) to use as data is being moved over the clouds is required. This model can increase the efficiency of big data transfers and release the needed resources for more urgent traffic. Network layer solutions have also been receiving a considerable attention from various researchers. SDNs and VNs represent two promising approaches for the dynamic manageability of the intra- and inter-DC networks. However, scalability and speed of configurations are two main challenges that must still be overcome before fully adopting these technologies. Similarly, as the cost of hardware continues to decrease, in-network processing and SnF solutions will become viable for future migration and movement of big data over Inter-DC networks. In light of this analysis, we identify the following future research directions:

- To date, there still exists a big gap between big data application requirements and functionalities on one side, and the provisioning of the cloud resources on the other. Hence, the first step to overcome this limitation is to develop efficient models that

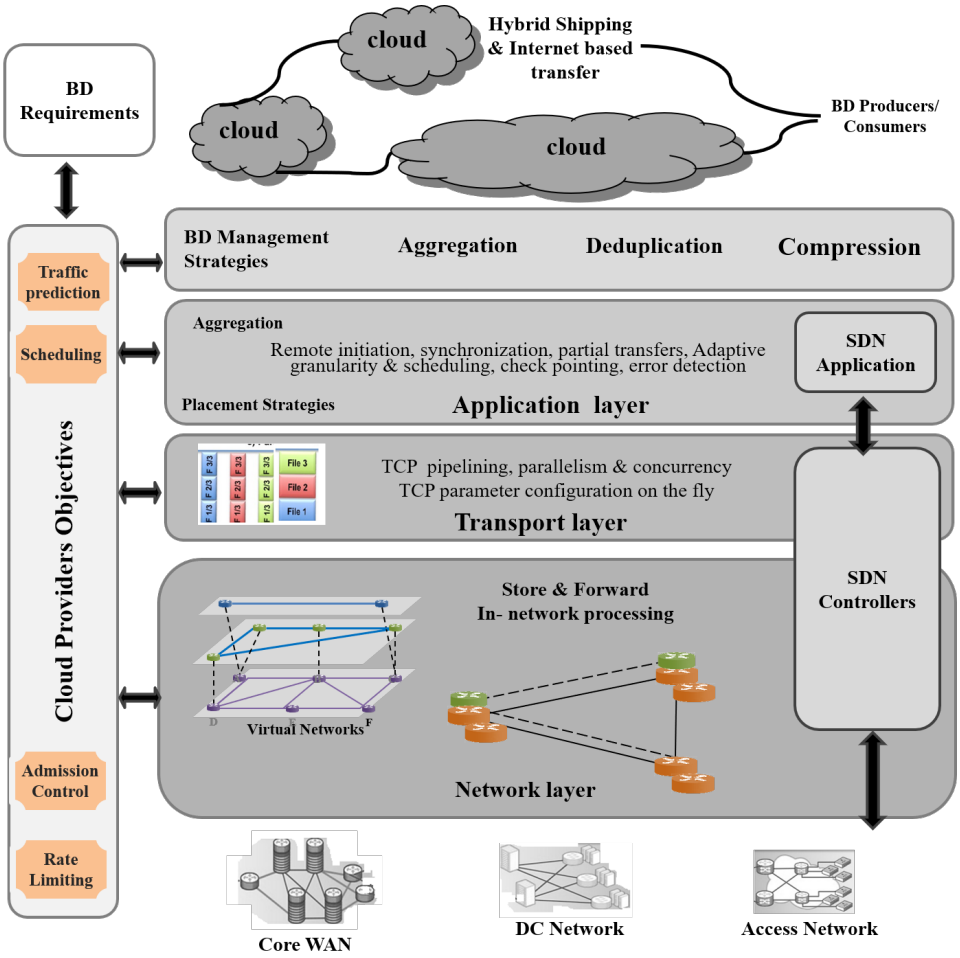


Figure 2.6: Cross-layer Solution for big data migration over clouds

can describe the varying needs of big data streams from the underlying networks as they are being transferred. An expressive model can easily be used to map these higher level requirements to the needed network configurations (e.g., in the form of a virtual network request). This will facilitate mapping the network performance parameters to application level parameters. For instance, during peak traffic periods an application can prioritize certain operations.

- The next step is to build network aware big applications that can dynamically adapt their performance based on a continuous feedback about the underlying network performance. If more elasticity is added to big data application, they can dynamically adjust their task schedules according to the current demands on the cloud resources. For example, similar to the store and forward routing methodologies, store and process based applications can delay part of their executions to benefit from periods where more resources are available. Also, resources can be used in a more optimal

manner if a more centralized view of the network resources and big data application needs is employed.

- In spite of the recent advances in network virtualization, the literature still lacks the needed research efforts to harness this tool to better serve big data migration. For instance, virtual networks can be created to precisely match the needed resources for big data applications with frequent access to their data sets but infrequent updates can use virtual links with asymmetric upload and download capacities.

- At the physical layer, current developments in the optical DC networks provide a promising solution for the high bandwidth demands of big data applications. However, their adoption is currently hindered by the higher replacement cost of traditional switches in current DCs. Clearly, bandwidth is the main bottleneck especially when using shared resources, e.g., streaming sites such as Netflix consume huge amount of available bandwidth daily. Therefore, in peak periods, users need to compete for the available bandwidth to successfully complete their big data transfers. Transfers to the cloud and between DCs using the WAN take a long time due to the transfer protocol and bandwidth size.

- Big data migration supports cloud resource elasticity management processes and it is an efficient way to improve resource utilization, however, it remains an important open issue [113]. Nowadays, CSPs own geo-distributed DCs connected by high bandwidth wide area networks (WAN) all over the globe to provide faster and better services. For most CSBs, running BDSAs on multiple DC systems can provide low-latency, high-quality and non-disruptive services. While geographically distributed BDSAs collect large amount of data for processing from various sources for better results, this may require aggregation at a single location and how to efficiently move data cost-effectively over time for processing has not been well studied.

- The execution of BDSAs may require migration onto different resource types or dynamic alocation and rescheduling to new cloud resources to ensure users requirements (e.g., end-to-end-latency) are met. While some efforts have been made in this direction, they are yet to be meaningfully integrated with CSB services [128].

2.4 Big Data Stream Processing in the Cloud

Big data stream processing is particularly suited for scenarios where massive amount of data must be processed with low latency. Under several emerging application scenarios

such as smart cities, social media, wearable devices, and IoT, continuous data streams must be processed within short delay intervals. Several solutions, including multiple software engines, have been developed for processing unbounded data streams in a scalable and efficient manner [4]. Unlike the storage before processing in traditional database systems, SPEs process tuples "on-the-fly" [129]. Fig. 2.7 shows a simple stream processing flow. Data stream flow from left to right, starting from the producers to the data store. The producers send events, which are appended to message queues, e.g., Flume[14], Kafka[15]. The consumer, i.e., a stream processing system, pulls the data out of the message queue and processes it. The result is then finally stored in a data store.

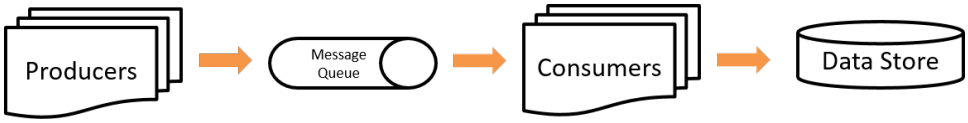


Figure 2.7: Simple Stream Processing Flow.

A stream is a potentially infinite sequence of tuples sharing a given schema. Queries in SPEs are defined as continuous since results are immediately pushed to the user each time the streaming data satisfies the query predicate. A query is a direct acyclic graph (DAG) where each node is an operator and edges define data flows. Typical query operators of SPEs can be classified as either stateless (e.g., Map) or stateful (e.g., Aggregate, Join) [130]. On the one hand, stateless operators do not keep state across tuples and perform their computation on a sliding window of input tuple over a period of time. On the other hand, stateful operators perform operations on sequence of tuples.

2.4.1 Stream Processing Architecture

Fig. 2.8 provides an overview of the basic components — collection, processing and storage — often found in a stream processing architecture.

- *Data collection:* collects data from multiple sources, e.g., social media, IoT and system logs. Data ingestion is performed by various tools that run close to the data sources and communicate via network connections, such as TCP and UDP [131].
- *Data processing:* consists of many systems such as stream/batch processing systems and data processing frameworks that ease the implementation of data analytics.

[14]https://flume.apache.org/
[15]https://kafka.apache.org/

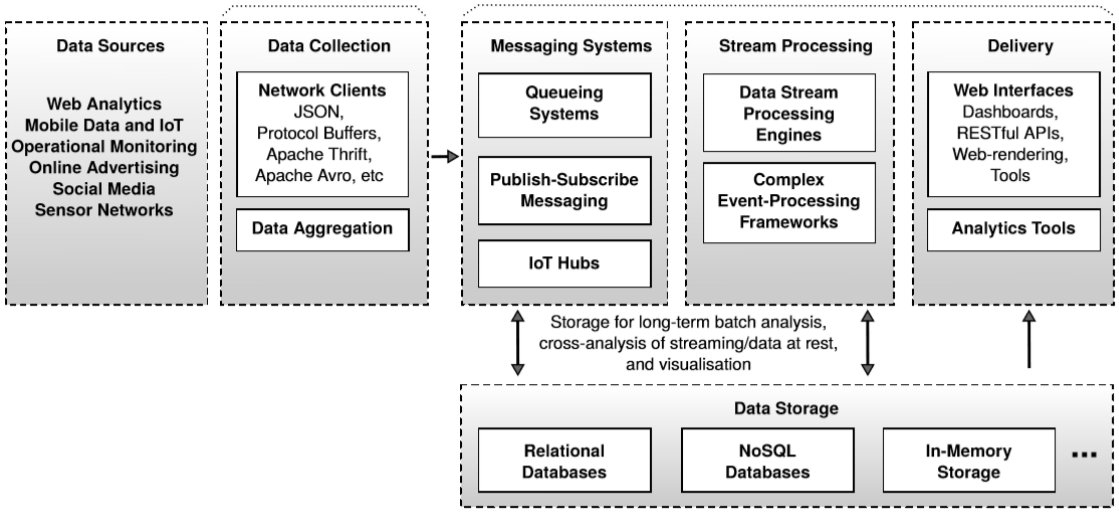


Figure 2.8: High-level Stream Processing Architecture [4].

- *Data storage:* consists of systems for storing and indexing in real-time architectures, e.g., NoSQL database, key-value stores and in-memory databases [132].

In addition, the stream processing architecture can comprise of multiple tiers of ingestion, processing and storage to incorporate modularity. The connection between the tiers are made by message brokers and queueing systems such as Apache ActiveMQ[16], RabbitMQ[17], Amazon Simple Queue[18] and Apache Kafka [133].

2.4.2 Stream Processing Applications

Today, a new class of stream processing applications have emerged in domains such as financial analysis, traffic monitoring, anomaly detection, healthcare, sensor data acquisition and multimedia in the past few years [11]. These applications allow data produced from heterogeneous, autonomous and globally-distributed data sources to be aggregated and analyzed dynamically to generate results in real-time. They are usually designed as Data-, Task- or Pipeline-Parallel applications containing structured directed acyclic graphs (DAG), where the vertices represent the operators (i.e., sources or sinks) and the edges are represented as data, task or pipeline streams [134], [5], respectively as shown in Fig 2.9.

On the one hand, the Data-Parallel application distributes computations involving the data streams to multiple nodes to reduce the computation latency of the application. An

[16]http://activemq.apache.org/
[17]https://www.rabbitmq.com/
[18]https://aws.amazon.com/sqs/

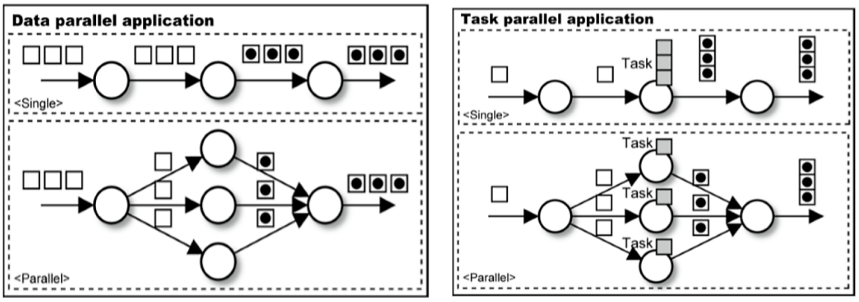


Figure 2.9: Task- and Data-Parallel Application Types [5]

example of this type of application is Twitter stream processing application for sentiment analysis. On the other hand, a Task-Parallel application, which requires long running time to complete, is one that distributes computation tasks (processes) to multiple nodes. Task pairs in a Task-Parallel application are executed on different parallel branches of the original stream graph with the output of each task never reaching the input of the other. In comparison to the aforementioned types, the Pipeline-Parallel applications offer low latency and the ability to run stateful operators in parallel by mapping chains of producers and consumers in a stream graph to different nodes.

Logically, a streaming application can be interpreted as a directed acyclic graph (DAG) whose nodes (operators/components) represent the computing elements and, the edges represent the stream of data flowing through the elements. An operator is a logical processing unit that applies user-defined functions on a stream of tuples. They perform functions such as filtering, aggregation and counting. The stream of tuples originate from the source operator of the DAG and exit out at the sink operator. The source operator pulls input streams from external sources while the sink operator processes the captured data streams. Separating the operators as a graph allows for exploring parallelism. Stream processing frameworks usually employ logical abstractions for specifying operators and how data flows between them, referred to as logical plan [8]. When the amount of parallelization is provided, i.e., number of instances required, it is referred to as the physical plan. This is provided to the scheduler for placing the operator instances on available cluster resources [4]. Stream grouping decides programmatically how tuples arriving at a given operator are split. Different kinds of grouping approaches are available for each streaming framework, e.g., shuffle, fields and global groupings. A number of stream processing systems are available today to process a wide variety of data, e.g., Twitter Heron [8], Storm [13], Spark [14] and Flink [135]. These systems rely on manual configuration for resource allocation, which is infeasible for dynamic streams.

2.4.3 Key Stream Processing Technologies

Stream processing has been an active area of research for many years. Traditional SPEs consisted of centralized solutions and were essentially extensions of DBMSs, developed to perform long running queries over dynamic data. Borealis [136] and STREAM [137] are examples of systems that are designed to provide stream processing middleware and languages for writing streaming applications. Also, System S [138] is used for large-scale, distributed stream processing middleware and application development. These systems are evidence of the past success of the stream processing paradigm. Due to the increasing popularity of stream processing engines (SPEs) and container technologies, there is a move to run BDSAs on containers for elastic workloads [139]. This is powered by virtualization techniques in the cloud that abstracts host resources. While most research efforts on resource management in the cloud have focused on the hypervisor-based environment, the emergence of OS virtualization has reopened a number of issues related to resource management in container-based environment.

In recent years, new innovative open source, off-the-shelf streaming engines such as Spark [14], Storm [13], and S4 [15] have been introduced to support high-volume, low latency stream processing applications and to address needs that are different from the analytical or extract, transform & load (ETL) domain typical of MapReduce [40] jobs. This generation of solutions resulted in more general application frameworks that enable the specification and execution of user defined functions (UDFs). Frameworks proposed typically follow two models: operator-graph model and micro-batch model. The operator-graph model processes ingested data tuple-by-tuple using DAG operators, while the micro-batch model groups incoming data at intervals, which are then processed at the end of each time window. Some examples of common SPEs that fall into these categories are described as follows:

2.4.3.1 Apache Storm

Storm [13] is a free and open-source distributed real-time computational system for processing large volumes of high-velocity data. It is commonly used by companies such as Twitter and Spotify to process high volume of data with low latency. Stream processing using Storm is achieved by distributing data streams across multiple machines as an unbounded sequence of tuples (key-value pairs). A Storm application is defined as spouts and bolts that run on clusters as DAG topologies. A spout is the source of a stream that pulls data from a messaging system such as Kafka [133], while a bolt processes the incoming

data stream, which in turn can be passed towards other set of bolts. Apache Storm [13] is known to perform Task-Parallel computations. Storm applications are configurable by the parallelism of the nodes, stream grouping and scheduling. The parallelism specifies the number of concurrent threads executing the same task (spout or bolt). Stream grouping determines the way a message is propagated to and handled by the receiving nodes. Scheduling algorithm deploys statically the spouts and bolts to the computational resources of the cluster.

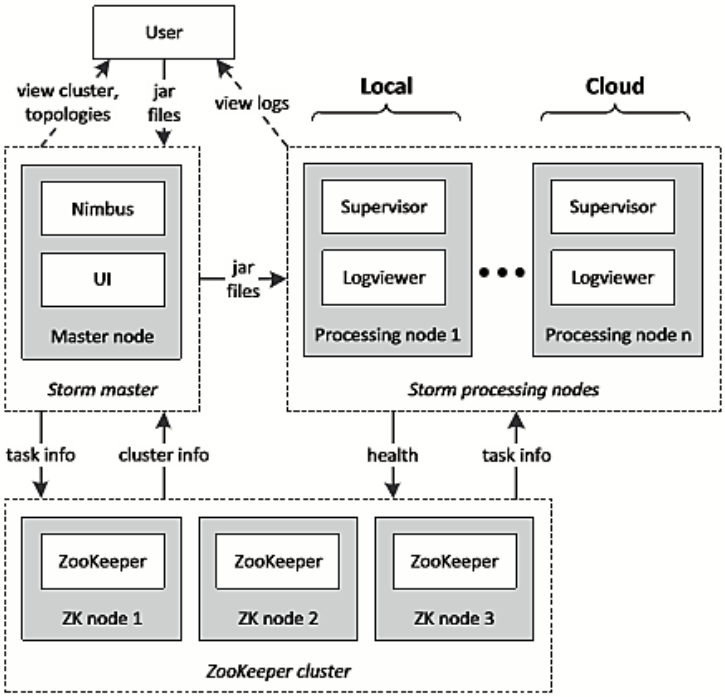


Figure 2.10: Components of a Storm Cluster [6].

Fig. 2.10 presents the components of a storm cluster. There are two main types of nodes in a Storm cluster—the master node and the processing nodes. The master node runs a daemon called Nimbus, which is responsible for distributing code around the cluster, assigning tasks, and monitoring their progression and failures. Processing nodes run a daemon called Supervisor, which listens for work assigned to its machine. It starts and stops processes as necessary, based on the work that was assigned to it by the master node. Each worker process within a processing node executes a subset of a topology, i.e., one or more spouts, bolts and metrics consumers. Coordination between the master node and the processing nodes is done through a ZooKeeper cluster. Also, information about assigned tasks, health of the worker processes and the cluster state are stored in the ZooKeeper [6].

2.4.3.2 Apache Spark

Spark [14] is an open-source distributed framework for data analysis. From the onset, Spark supported both batch and streaming workloads, which makes it different from other SPEs that are designed for only stream processing. Spark performs in-memory analytics on streams in a distributed and real-time manner by discretizing the streaming data into micro-batches before computation. Spark [14] is known to perform Data-Parallel computations. To cover a variety of workloads, Spark introduces an abstraction called Resilient Distributed Datasets (RDDs) that enables running computations in memory in a fault-tolerant manner. RDDs, which are immutable and partitioned collections of records, provide a programming interface for performing operations, such as map, filter and join, over multiple data items. To handle faults and stragglers more efficiently, the authors in [7] proposed D-Streams, a discretized stream processing based on Spark Streaming. D-Streams follows a micro-batch approach that organizes stream processing as batch computations carried out periodically over small time windows.

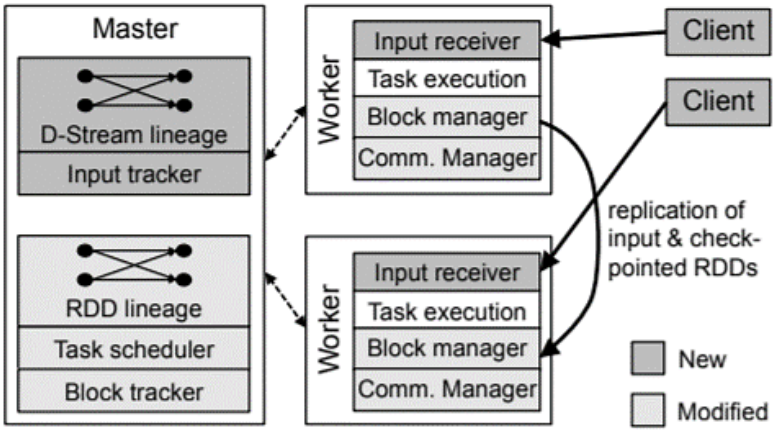


Figure 2.11: Spark Architecture [7].

Spark streaming architecture consists of three components as shown in Fig. 2.11. The master tracks the D-Stream lineage graph and schedules tasks to compute new RDD partitions, the worker nodes receive data, store the partitions of input and computed RDDs, and execute tasks, while the client library is used to send data into the system. Spark streaming differs from traditional streaming in that it divides computations into stateless and deterministic tasks that can run on any node in a cluster or even multiple nodes [7].

2.4.3.3 Apache Heron

Similar to Storm is Apache Heron [8] in which streaming applications are DAG. Heron [8] was built as an architectural improvement to Storm [13] with mechanisms to achieve better efficiency that addresses several of Storms' inadequacies. Heron runs topologies that are made up of bolts and spouts. The spouts generate the input tuples that are fed into the topology, and bolts carry out the actual computation. Also, Heron processing semantics are similar to that of Storm, which include at most once (i.e., no tuple is processed more than once) and at least once semantics (i.e., each tuple is guaranteed to be processed more than once). However, Heron topology is a departure from Storm where Nimbus was used for scheduling. In Heron, each topology is run as an Aurora[19] job consisting of several containers that run the Topology Master (TM), Stream Manager (SM), Metrics Manager (MM) and a number of processes called Heron Instances (HI) as shown in Fig. 2.12. The TM is responsible for managing the topology throughout its existence, the SM is used to manage the routing of tuples efficiently while each HI connects to its local SM to send and receive tuples.

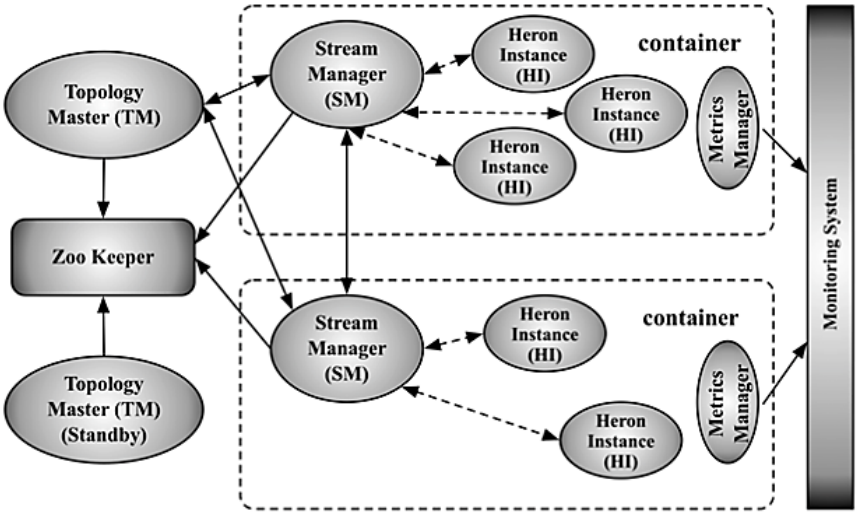


Figure 2.12: Heron Architecture [8].

Other popular SPEs include Yahoo S4 [15], Apache Flink [135], and Samza [140] among others. Yahoo S4 [15] has been proposed as an alternative framework for processing big data streams requiring real-time processing. S4 is a decentralized, scalable and fault-tolerant framework that provides a programming interface for processing data streams on top of a highly available cluster in a distributed computing environment. In addition to these

[19]http://aurora.apache.org/

engines, cloud service providers also provide many managed solutions for processing of streaming data such as Amazon Web Services (AWS) Kinesis[20], Google Cloud Dataflow[21], and Azure Stream Analytics (ASA)[22]. AWS Kinesis is a service that enables continuous data intake and processing for several types of applications. Google Cloud Dataflow is a programming model and managed service for developing and executing a variety of data processing patterns such as ETL tasks, batch processing, and continuous computing. ASA enables real-time analysis of streaming data from several sources such as devices, sensors, websites, and social media [4]. Although, some of these technologies allow for dynamic scaling of cluster resources allocated to applications, the required resources must be provisioned at run-time. Also, they do not provide the option of adding or removing virtual machines from clusters when they are no longer needed.

2.5 Cloud Resource Elasticity for BDSAs

Resource elasticity is an important distinguishing feature in the cloud computing context, which allows for an application or service to scale in/out according to fluctuating demands. This capability is essential in several settings as it helps service providers to minimise the number of allocated resources and to deliver adequate QoS levels, usually synonymous with low response times [4]. Although, elasticity has been extensively investigated for enterprise applications, stream processing poses challenges for achieving elastic systems when making efficient resource management decisions based on current load. Elasticity becomes even more challenging in highly distributed environments comprising of cloud computing resources [4]. Elasticity has been explored by researchers from academia and industry fields as many virtualization technologies, on which the cloud relies, continue to evolve. Thus, tremendous efforts are being invested in enabling the cloud systems to behave in an elastic manner.

Many definitions have been put forth in the literature to describe elasticity [141], [9]. The authors in [88] define elasticity as the ability of a system to add and remove resources (such as CPU cores, memory, VM and container instances) *on the fly* to adapt to the load variation in real time. According to NIST, *"elasticity is the ability for customers to quickly request, receive, and later release as many resources as needed"* [9]. The purpose of elasticity has been identified in [88] to include but not limited to improving performance, increasing resource capacity, saving energy, reducing cost and ensuring availability. While

[20]https://aws.amazon.com/kinesis/
[21]https://cloud.google.com/dataflow/
[22]https://azure.microsoft.com/en-us/services/stream-analytics/

several efforts are being made towards making stream processing more elastic, many issues remain unaddressed, which include task placement, bottleneck identification and application adaptation [4]. Since big data streaming applications are often long-running jobs, the initial task placement whereby processing elements are deployed on available computing resources becomes critical than in other systems.

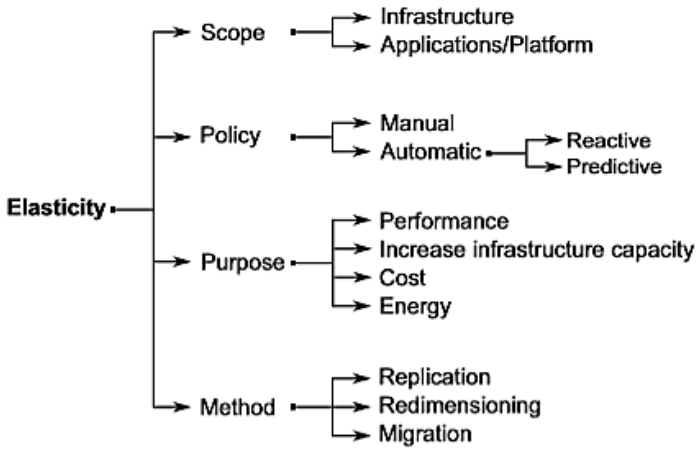


Figure 2.13: Classification of Elasticity Solutions [9].

Elasticity solutions can be classified based on four basic characteristics, which includes scope, policy, purpose and method as shown in Fig. 2.13. A more comprehensive classification is provided by the authors in [88]. The scope defines where elasticity actions are controlled, the policy specifies the needed interactions (manual or automatic), the purpose indicates the objective for the elastic actions (e.g., reduce cost or improve performance) and the method is related to the implementation types, i.e., vertical scaling, horizontal scaling or load balancing defined as follows: [9, 88].

- Horizontal elasticity involves adding or removing instances of computing resources associated with an application in contrast to vertical elasticity that involves increasing or decreasing computing resources, such as CPU time, cores, memory, and network bandwidth. Basically, horizontal elasticity is coarse-grained and it is incapable of handling the dynamic resource requirements of streaming applications due to the static nature and fixed VM sizes [142].

- Vertical elasticity is fine-grained and applicable to any application. It eliminates the overhead caused by booting instances in horizontal elasticity and does not require running of additional machines such as load balancers. Also, vertical elasticity increases performance, provides higher throughput and reduces SLA violations [142].

- Load balancing involves the distribution of requests evenly among the available resources.

Manual policy actions require cloud users to interact with the virtual environment through an API and perform elasticity actions. Most elastic policy actions are usually carried out in an automatic mode, which can either be reactive or proactive. In the reactive mode, elastic actions are triggered based on certain thresholds or rules. Most cloud platforms such as Amazon EC2 use this technique. Meanwhile, the proactive mode uses forecasting techniques to anticipate future needs. These techniques include the following [88]:

1. *Time Series Analysis:* This approach is responsible for making an estimation of the future resource and workload utilization. It focuses on two objectives 1) predicting the future, and 2) identifying repeated patterns. Several techniques such as Moving-Average, Auto-Regression, ARIMA and machine learning are used for predicting the future while approaches such as Fast Fourier Transform (FFT) and histogram can be used to achieve the second objective.

2. *Reinforcement Learning:* This approach depends on learning through interactions between an agent and the system or environment. The agent (decision-maker) is responsible for taking decisions for each state of the environment and tries to maximize the return reward. The agent learns from the feedback of the previous states and rewards of the system, and then it tries to scale up or down the system by choosing the right action.

3. *Control Theory:* Makes use of three types of controllers: Openloop controllers that computes the input to the system Feedback controllers. The Feedback controllers that monitor the output of the system and correct the deviation against the desired goal and the Feedback-forward controllers that predict errors in the output.

4. *Queuing Theory:* This approach studies queues in the system, taking into consideration the waiting time, arrival rate and service time.

In the next sections, we briefly discuss the resource prediction and allocation challenges in the cloud.

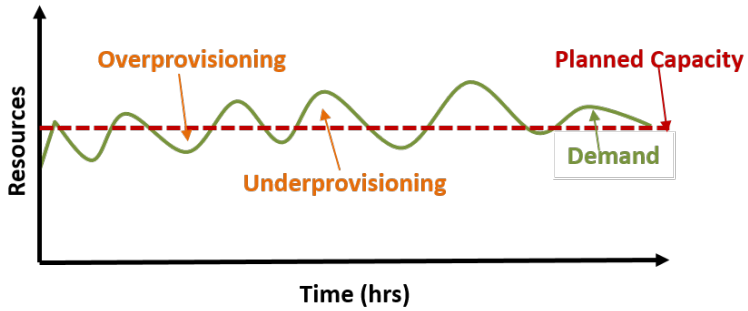


Figure 2.14: Cloud Resource Over-provisioning/Under-provisioning.

2.5.1 Resource Prediction

Resource prediction, at the core of auto-scaling, aims at identifying the amount of resources required to process a workload. To meet the expected QoS requirements of BDSAs, it is increasingly important to gain an understanding or characterize the workloads running in the cloud environment. Predictive approaches for resource estimation falls into one of the following categories; rule-based, fuzzy inference, profiling, analytical modeling, machine learning and hybrid approaches. A discussion on these approaches was presented by the authors in [143]. Since many streaming applications exhibit sudden, dramatic changes in the workload between peak and off-peak hours, the use of an erroneous prediction system will lead to over-provisioning of resources, hence, waste resources outside peak hours, or under-provisioning of resources, which can lead to bad performance and SLA violations during peak hours. Fig. 2.14 shows maintaining sufficient resources to meet fluctuating workload demands can be costly. Currently, there is a lack of research on the relationship between multiple BDSAs, CSBs and associated factors driven by demands such as resource consumption and workload, as an integrated concept.

2.5.2 Resource Allocation

As a hot issue of using cloud infrastructures, resource allocation has been well investigated and dealt with in several manners using either the static or dynamic approaches. The static allocation approach assigns fixed resources, which can lead to resource starvation in streaming applications as workload changes. Studies on the dynamic approaches generally involve applying the SLA-based, utility-based, market-based, or priority-based custom policies [144–146]. These strategies attempt to conduct resource fairness but do not take into consideration the multiple objectives of competing containerized streaming applications, and mostly focus on the hypervisor-based VM deployment. Alternatively, resources

can be dynamically provisioned using various auto-scaling approaches, e.g., work proposed by the authors in [35] can be adopted. However, auto-scalers are useful for scaling up/down in resource-sufficient environments since containers are limited to the resources of the host (e.g., VM) they reside.

An important feature of the SPE is its capability to scale out applications to meet the requirements of workloads. This employs the instantiation of parallel operators called tasks. The number of instances for a certain operator is fixed by users during streaming application configuration. Since the task workload can fluctuate unpredictably in SPEs, resources should be allocated to each streaming application dynamically to satisfy the real-time constraints with minimal resource consumption. In addition, balancing the load among these instances is important for better resource utilization and reduced response times.

2.5.3 Container Elasticity

Containers have emerged in recent years as a lighter-weight version of VMs and they have become a popular deployment approach in cloud infrastructures. Likewise, the use of containers for running BDSAs is gaining widespread adoption due to the flexibility it provides for packaging, delivering and orchestrating applications. Despite that most available works [147], [148] proposed for cloud elasticity are focused on VM mechanisms, some researches [149], [150], [151], [152] are being dedicated to containers. Kukade et al. [150] proposed a design for developing and deploying microservices applications into Docker containers. During load spikes, instances of the application are added through horizontal scaling. The memory load and number of requests are constantly monitored until a threshold is reached whereby containers are scaled in/out. Paraiso et al. [151] proposed a tool to ensure the deployability and the management of Docker containers. The framework allows coordinating infrastructure and platform adaptation. In addition, it allows to vertically scale the size of container or VM resources by inserting agents for monitoring and adaptation. However, web applications are targeted in these proposals.

Despite the amount of studies on cloud elasticity, many open issues still remain on the challenges of the emerging container technology. These challenges include monitoring, elasticity of containers running on VMs and how to incorporate elasticity into container orchestration [88]. Monitoring containers is not an easy task as containers hold all the applications and their dependencies. Also, many containers can be hosted on a machine. For containers running on VMs as illustrated in Fig. 2.15, vertical elasticity becomes an

issues since the containers are limited to the resources of the host on which they reside. Static limits are used to allocate resources to containers or containers share resources with other containers hosted. Although, container orchestration tools can be used to manage containers, integrating vertical and horizontal scaling is still an open research challenge. While the elasticity management mechanisms proposed for managing VM resource scaling and allocation in the cloud can be applied to containers, some modifications are required in order to implement such mechanisms [88].

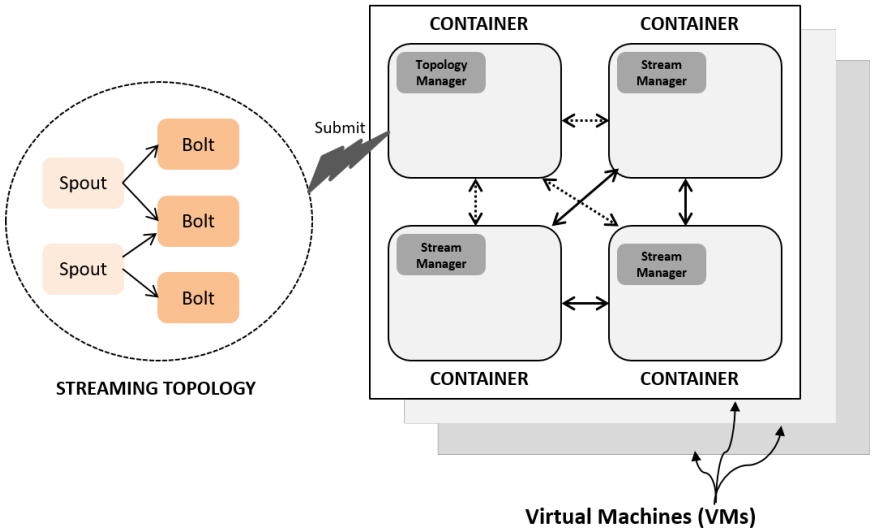


Figure 2.15: A Streaming Topology on a Container-Cluster.

2.6 Reference Models

In this section, we present some reference models used in this dissertation.

2.6.1 Hidden Markov Models

Hidden Markov Model (HMM) [153] is a widely employed and powerful statistical tool for modeling and analyzing sequential data. HMMs have been extensively used in many applications, such as speech recognition, video activity recognition, and gesture tracking.

In the single-layer, one-dimension HMM model, an observation O_t at time t is produced by a stochastic process, but the state $q_t = S_i$ of this process cannot be directly observed, i.e., it is hidden. This hidden process is assumed to satisfy the Markov property, where state q_t at time t depends only on the previous state, S_{t-1} at time $t-1$.

The HMM is described as a process evolving over discrete time intervals. Let $S = \{S_t, \quad t = 1, ..., T\}$ denote a sequence of hidden states, drawn from a finite state space of size N and let $O = \{O_t, \quad t = 1, ...T\}$ denote a corresponding set of observable states with M discrete possible observation symbols $O = O_1, O_2, ...O_M$. The standard HMM has the form:

$$P(S, O) = P(S_o) \prod_{t=1}^{T} P(S_t|S_{t-1})P(O_t|S_t) \tag{2.1}$$

where $P(S_o)$ is the initial probability distribution denoted as π, $P(S_t|S_{t-1})$ is the transition probability distribution of moving from state $S_{t-1} = i$ to state $S_t = j$, where $\{i, j \in 1, .., N\}$, denoted as A, i.e., $N \times N$ matrix and $P(O_t|S_t)$ is the observation probability distribution denoted by B, i.e., $N \times M$ matrix. Hence the model $\lambda = (\pi, A, B)$.

Duration in HMM is usually characterized by a probability distribution function $P_i(d)$, e.g., in state i, the value of $P_i(d)$ is the probability of remaining in state i for d time units. Given the Markov assumption and from probability theory, $P_i(d)$ is the product of all the probabilities:

$$P_i(d) = a_{ii}^{d-1}(1 - a_{ii}) \tag{2.2}$$

where a_{ii} is the self-loop probability of state i. $P_i(d)$ is a geometrically decaying function of d and when applied directly to resource usage prediction of big data streaming applications, the probability that the underlying process remains in a resource usage state i for d time units is a geometrically decaying function of d.

HSMM [154] is an extension of HMM designed to remove the constant or geometric distribution of state durations assumed in HMM by introducing the concept of explicit state durations. The HSMM is similar to HMM in which an embedded Markov chain represents the transitions between the hidden states. However, it incorporates a discrete state occupancy distribution $D = P_i(d)$, representing the duration in non-absorbing states, i.e., transient states. The HSMM model is specified as $\lambda = (\pi, A, B, D)$, where D models the time spent in a particular state.

In the HSMM, each state variable has variable time intervals and a sequence of observations $O_{t_1}, ..., O_{t_2}$ are generated while in the state as opposed to a single observation O_t in the HMM. The length of the observation sequence for each state is determined by the duration spent in each state. In some real-time streaming applications that run for a

longer period of time, accounting for changes in behaviour at different time intervals is of importance for a more accurate prediction.

2.6.2 Game Theory

Game theory [154] has been explored in the past for allocating resources in the cloud. It has impacted various fields such as biology, political science and more recently, computer science. Game theory provides a straightforward tool that can be used to study the resource allocation problems in competitive big data streaming application environments. The strategic behaviour of a rational profit maximizing agents in a non-cooperative game is predicted to be a Nash equilibrium [155], which is a balanced state with a strategy profile, from which no game player has the incentive to deviate. In practice, being able to compute the Nash equilibrium to represent the predictions of the outcome is very important. Computing Nash equilibrium in a small game that involves two players is trivial but with many players, this becomes very difficult.

A strategic non-cooperative game models the interactions among multiple decision-players, in this case, the streaming applications. The non-cooperative game can be defined to consist of a set of players, and each player has a finite set of actions and a pay-off function. Each player chooses their strategy simultaneously and the players' payoffs characterize their preference over the set of strategy profiles. The strategy of the players can either be pure or mixed. A mixed strategy is the probability distribution over the set of the players pure strategies. The Nash equilibria can be computed by enumerating all possible totally mixed or pure strategy profiles and checking if a Nash equilibrium exists. Thus, the algorithm should be able to generate all possible combinations of to represent the total strategies.

Some existing works applying game theory in the cloud are as follows; To maximize cloud provider profit while satisfying client requests, Srinivasa et al. [156] proposed a Min-Max game approach. They utilized a new factor in the game called utility factor, which considers the time and budget constraint of every user and resources are provided for tasks having the highest utility for corresponding resource. Mao et al. [157] proposed a congestion game whereby services are considered as players with the strategy of selecting a subset of resources to maximize their payoffs, which is the sum of the payoffs received from each selected resource. However, none of these approaches were proposed to address containerized big data streaming applications.

2.7 Summary

In this chapter, we reviewed the big data paradigm, and presented emerging characteristics, deployment services, lifecycle and issues related to utilizing cloud technologies. We introduced the cloud DC from a networking perspective, and virtualization techniques (i.e., hypervisor- and container-based virtualization) powering DC resource provisioning are presented. This is followed by the support cloud provides and why it is a perfect fit for hosting big data. We further discussed the challenges and requirements for migrating big data over clouds. An overview of big data stream processing and the key technologies used are then presented. Finally, we introduced the container technology as a newer approach for handling BDSAs.

Chapter 3

Proposed Framework

In this chapter, we present our framework for managing the resource elasticity and pricing challenges for CSBs in the cloud. In this framework, a proactive approach is introduced to improve resource scaling, which takes into consideration multiple streaming applications running on a heterogeneous cluster of hosts. While research related to resource management for containers has not been considered actively, we include a resource allocation and pricing module to ensure fair resource sharing and utilization among containerized big data streaming applications. This framework provides useful resource management functions for CSBs, and more importantly, they can optimize the resource usage of containerized big data streaming applications in order to meet their QoS.

3.1 Cloud Ecosystem and Application Scenario

Cloud computing is a new cloud operations model that combines existing technologies such as virtualization, utility-based pricing, autonomic computing and grid computing [158]. Around this new operations model, a business ecosystem has evolved whereby new market players have emerged breaking up the traditional value chain of IT provision. The

ecosystem includes parties such as the cloud service provider (CSP), cloud tenants/users (e.g., ASPs or cloud brokers) and end users (e.g., cloud service consumers (CSC)).

- *Cloud Service Provider (CSP):* The CSPs manage a set of computing resources through their datacenters on a pay-per-use basis and are responsible for allocating resources in order to meet the SLAs of the CSBs.
- *Cloud Service Consumer (CSC):* The CSCs have applications with fluctuating loads and lease resources from CSBs to run their applications.
- *Cloud tenant/User:* The cloud tenant can be an Application Service Provider (ASP) or a CSB. For instance, Netflix is an example of an ASP for video on-demand. The tenant acts as an intermediary between the CSC and CSP, and manages the use, performance and delivery of cloud services. Attracted by the cost-reduction that the cloud provides, CSBs offer economically efficient services to CSCs using hardware resources provisioned by the CSPs, which can be directly accessed by the CSCs. The CSB is responsible for meeting the SLAs agreed with the CSCs. CSBs leverage their expertise by aggregating, integrating, deploying and managing multiple BDSAs to deliver a fully managed solution to the CSCs. Benefits provided to the CSCs include efficiency, technical expertise, customization, and simplified billing.

Beyond the scope of this dissertation, other players such as cloud auditors and cloud carriers can also participate in the cloud ecosystem [25]. Taking the ecosystem as a whole, all players jointly create services in a loosely coupled manner to meet end users' needs [159]. CSBs and CSPs use SLAs, which state the QoS to be delivered to the consumers, to guarantee QoS. As more and more enterprises leverage cloud computing services or contract their computing footprints to be more competitive in a global world, BDSAs are being deployed in private, public or hybrid cloud settings. However, managing computing resources requires sophisticated programmers to understand the various low-level cloud service application programming interfaces (APIs) and command line interfaces (CLI) tools as well as maintain complex resource configurations. This development lacks flexibility and introduces complexity for many enterprises, hence, higher costs. In addition, as more cloud services and resources are utilized, the intensity of deployment difficulty increases. The complexity of cloud integration, customization and management prevents enterprises from exploring opportunities to integrate cloud services and features. Hence, the need for CSBs.

The target application scenario of our resource management framework is a big data CSB that relies on the cloud infrastructure to provide services to CSCs. More precisely,

the CSB rents heterogeneous resources (containers or VMs) to a number of CSCs to run their streaming applications. The CSB aggregates different types of resources from a number of CSPs and sublets these resources to the requesting CSCs' streaming applications. Among the available resources, the CSB finds the best resources and proceeds through allocation policies for the BDSAs while avoiding QoS violations. From the perspective of the CSB willing to deploy the CSCs' applications, the framework enables them to focus on the business side of scalability and application development of their service while ignoring details about the underlying infrastructure actually supporting the applications. The resources are charged by the CSP using pay-per-use billing scheme while the CSB charges the CSC on a monthly basis. Requests are submitted by the CSC who specifies their QoS preferences (shorter response time and lower cost).

3.2 The Cloud Business Model

CSPs basically offer three main service models (SaaS, PaaS and IaaS) to the CSBs or directly to the CSCs. To pay for virtual resources, most CSPs provide two main ways — on-demand or reserved approach [160]. The on-demand approach, usually recommended for short-term workloads, enables CSCs to pay for compute capacity hourly while the reserved approach, recommended for long-term workloads, enables CSCs to rent virtual resources for longer periods with significant discount [160]. Since short-term CSCs tend to pay more, CSBs have entered the realm to save cost for the CSCs. Relieved of the burden of setting up their own DCs, CSBs usually rent resources with the reserved price from the CSPs and outsource the resources on-demand to the CSCs in order to host the CSCs' applications at lower prices. The revenue of the CSB is largely determined by two factors – the CSC's workload arrival and the resource price. Also, it is worth noting that the CSB's price has a great impact on the demand of the CSC. Whenever the CSB increases its price, the CSCs' demands simultaneously decreases. Likewise, when the CSB decreases its price, the CSCs' demands increases.

Pricing approaches have been applied in the cloud environment to address resource management using common mechanisms such as market-based pricing and game theoretic pricing. In the game theoretic context, models and mechanisms used to determine resource prices include non-cooperative game, auction theory, Stackelberg game and bargaining game. In this dissertation, we focus on the non-cooperative game. Generally, participants in the game are the cloud providers CSPs), service providers (CSBs), cloud tenants/consumers (CSCs), and end-users. To provision resources to the CSBs and CSCs,

CSPs utilize the fixed-price and auction-based mechanisms to provide cost-effective options for obtaining cloud resources. Whereas, the CSBs transform the cloud market into a commodity-like service to offer packages with distinct features, e.g., resource types and pricing schemes, to the CSCs. As a two-way market, the primary market involves the CSPs and the CSBs, while the relationship between the CSBs and the CSCs form the secondary market.

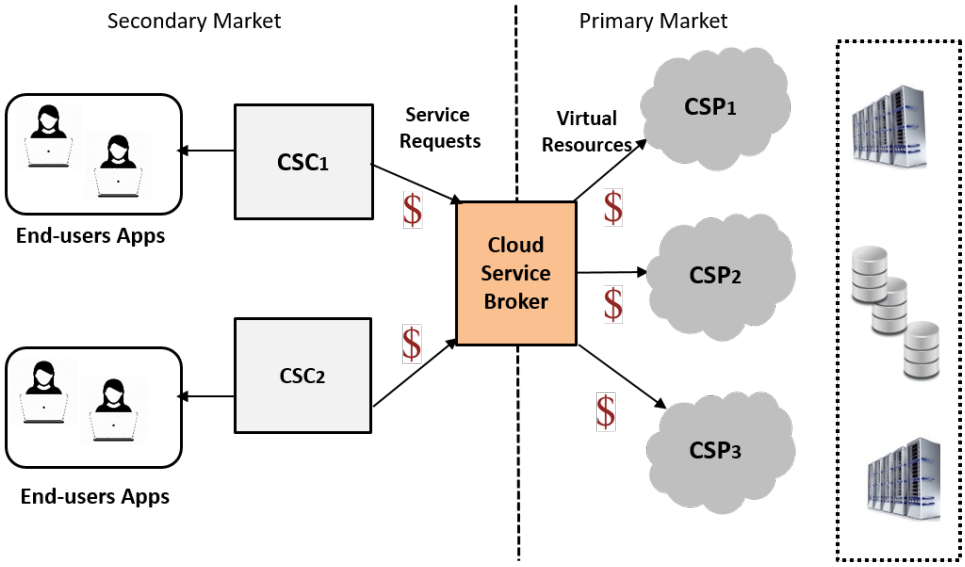


Figure 3.1: Cloud Market Structure.

We consider a secondary market in which the CSBs are the sellers, and the CSCs, as buyers, are typically selfish. Fig. 3.1 shows the relationship between the CSP, CSB and CSC. In this market, the CSB dominates the control of resource price for the CSC. The CSC's obligation is to ensure maximum resource utilization by competing in a non-cooperative manner for the CSB's resources. The surplus of the CSC is determined by their utility minus the price charged by the CSB. Hence, the CSCs make optimal demand response decisions to maximize their surplus (i.e., utility minus cost), given the resource utilization prices by the CSBs. To achieve an equilibrium point, the CSC changes their resource demand strategy to match that of other CSCs.

Due to the need to consider the heterogeneity of streaming applications as well as that of the cluster in which they run makes resource allocation and pricing a non-trivial issue for CSBs. Moreover, despite the technological advances containers have made, there is yet to be a standard, performance-optimized resource allocation and pricing scheme for managing streaming applications in containerized environments. Therefore, newer approaches are required to automate the selection of resource configurations and their mapping to het-

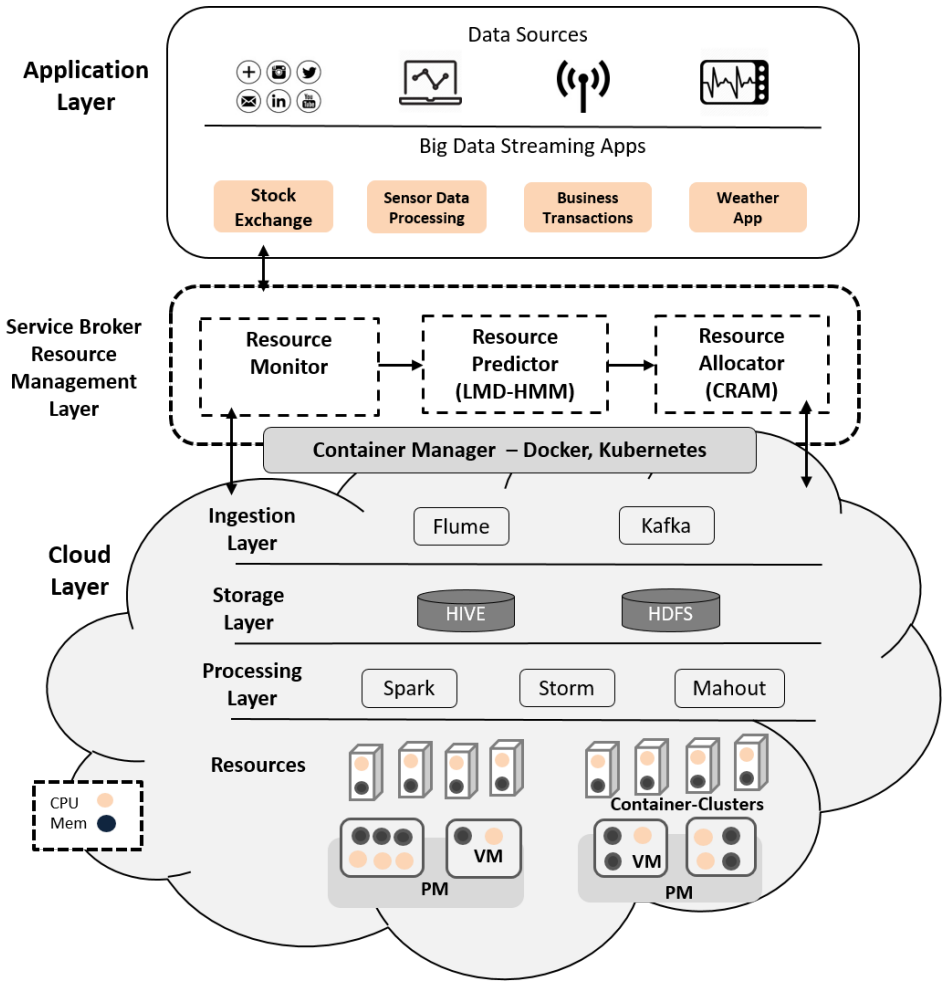


Figure 3.2: Proposed Resource Elasticty Management and Pricing Framework

erogeneous host resources. To address these issues, we propose a cloud-based framework for dynamically handling resource elasticity of big data streaming applications for CSBs in the cloud.

3.3 Framework Layers

Fig. 3.2 illustrates the proposed framework detailing the required components for achieving resource prediction and allocation. It depicts the relationship between the BDSAs, the CSB and the virtual cloud host resources. The architecture consists of three layers; the application layer, resource management layer and cloud resource layer.

3.3.1 Application Layer

The application layer is made up of multiple heterogeneous BDSAs (e.g., stock exchange, sensor data processing applications) that processes data streams collected from various sources. It provides interfaces to enable the easy extraction of data from various sources such as sensors, application logs and social media, to be delivered to the applications. Each application has varying resource demands thereby complicating how resources are allocated to process the data streams. The application layer also provides functions for the design, specification and visualization.

3.3.2 Resource Management Layer

The resource management layer acts as the middleware between the application layer and the cloud resource layer. It manages the process of provisioning and allocating resources to satisfy BDSAs' resource requirements and other placement constraints. It is made up of modules such as the monitor, profiler, resource predictor (LMD-HMM/HSMM), container resource allocator (CRAM) and container engine for managing the resources. The resource management layer is responsible for resource management and provisioning, and responding to requests from upper-layer and supporting various scheduling frameworks. All the operations that need to access the underlying resources are encapsulated in this layer. At this layer, the CSB can interact with services of the other layers. Tools such as APIs, CLIs and SDKs are used to expose services to manipulate cloud resources and policies. It also provides tools and user interfaces for submission. Fig. 3.3 represents the operational overview of the resource management layer.

The mechanism will observe the system states by collecting metrics from the underlying streaming system and attempts to adjust the load. When a streaming application is submitted to the framework, the profiling system checks whether the profiled log exists. If not, the profiler will be activated first to collect the execution pattern of this newly arrived streaming application. Otherwise, the submitted application will start execution immediately. At the resource layer, the resource predictor predicts the resource demand of the submitted streaming applications according to its profiled log. Considering the current resource usage captured by the resource monitor, the resource provisioning mechanism will be triggered as needed in order to improve the resource utilization.

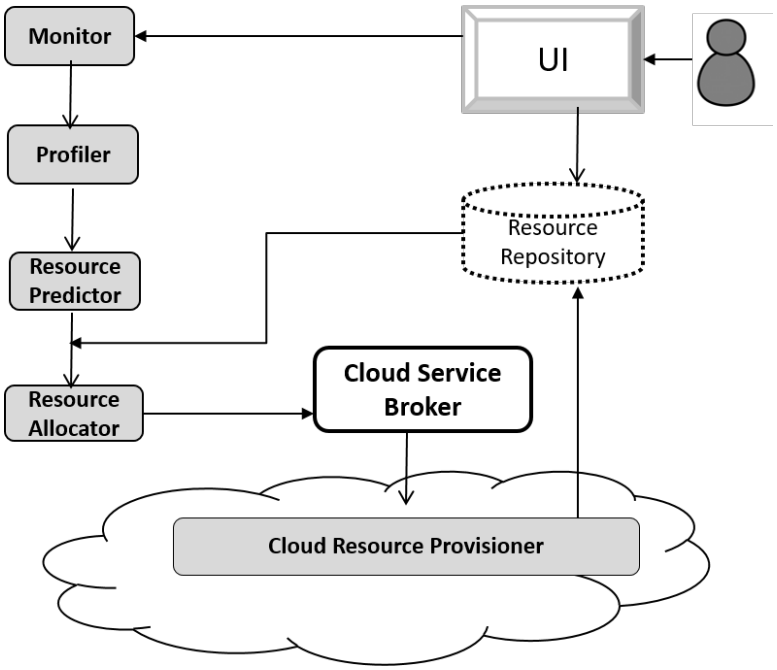


Figure 3.3: Operational Overview of the Resource Management Layer

3.3.3 Cloud Layer

The cloud resource layer is the lowest layer which provides the host resources (VMs or container-clusters) required by the applications. It contains the stream processing engine (e.g., Spark, Heron) and cloud orchestration services (e.g., Kubernetes and Docker) to run multiple containerized streaming applications. The containers for the streaming applications are created by assembling them from individual base images. Each application is packaged into containers and deployed on a cluster of hosts. The cloud resource layer also handles stream processing management through the ingestion layer, processing layer and storage layer. The ingestion layer provides services such as Flume[1] and Kafka [133] that facilitates access to heterogeneous data sources for each streaming event while concealing the technical facets of the data streams. The ingestion layer is capable of integrating data streams from multiple sources as required by each streaming event. The processing layer contains SPEs such as Spark [14] and Storm [13] that supports the analysis of the streamed data in order to identify meaningful patterns in the data streams and trigger actions. The storage layer supports record ordering and strong consistency to enable fast reads and writes of large streams of data whenever required. Storage services, e.g., HDFS[2], in this layer may serve as an intermediate or final storage point for stream analysis results.

[1]https://flume.apache.org/
[2]http://hadoop.apache.org/

3.4 Function of Actors

3.4.1 Cloud Service Consumers

The CSC make use of cloud interfaces, which are location-independent to access cloud services. Requests are sent by the CSC to the CSB with QoS constraints such as deadline, budget and penalty rate. This information is utilized by the CSB to process the CSC's request and a formal agreement (SLA) is signed between both parties to guarantee the QoS requirements.

3.4.2 Cloud Service Brokers

The CSB acts as an intermediary between CSP and CSC. CSBs rent a certain number of virtual machines (VMs) or Containers from the CSP to run their streaming applications. They make it easier to deliver value to the CSC while also assisting with coordinating the services they use. Examples of CSBs are APPIRIO, Appregata, BOOMI that provide solutions for CSPs and mid-level size companies. The CSB performs three key roles among others in helping CSP deliver services to their customers [161, 162]:

- *Aggregation:* The CSB bundles many individual services together and present them as a unified service.
- *Integration:* Similar to aggregation, the CSB can integrate multiple services collectively to provide new functionality, i.e., the CSB has the flexibility to choose services from multiple CSPs.
- *Customization:* A number of services can be customized by the CSB to meet the needs of individual CSC.

While the CSB provides cloud services to CSCs, they also support the CSPs. They can implement services and solutions that may not be part of the provider's core business and can negotiate better prices for CSCs while delivering more customers to the CSPs. Many CSPs work with CSBs to facilitate relationships with the CSCs. For instance, Verizon serves as a CSB for Google and Amazon to connect various enterprises with appropriate cloud services. Services that support CSBs include resource discovery, resource monitoring, SLA management, and scheduling. In order to benefit from the elasticity the cloud

provides, CSBs need to continuously monitor the end-to-end QoS of their cloud-hosted applications. Thus, they can deploy strategies to scale up and down the resources allocated to each application.

3.4.3 Cloud Service Providers

The CSP provides computing, storage, and network resources required for hosting CSB services and maintains SLA with the CSBs on availability of infrastructure services. The CSP ensures their resources are integrated such that usage is optimized. They provides the ability to add or remove resources on demand and implement a clear means to charge CSBs for resource usage. They are responsible for dispatching VM or container images to create instances that run on their physical resources. Optimal selection of services by a CSB depends on the availability of complete and updated information on the services offered by the CSP.

3.5 Framework Components

To manage cloud resources, various services are used to select, configure, deploy, and monitor cloud resources. From the CSB's perspective, these services connect the resource operations thereby providing an abstraction layer for the CSB to focus on their main objectives. The main services required to attain resource elasticity for the CSB are discussed as follows:

3.5.1 Resource Profiler

Workloads characterization is very crucial in a cloud computing system in order to achieve proper resources provisioning, cost estimation, and SLA considerations. To effectively manage streaming applications and resources, it is important to use models and tools that create an application profile, which are used to create forecasting models for determining resource usage. Application profiling involves four phases, which are data granularity, definition, monitoring, processing and storing. Each phase has its own challenges as discussed by the authors in [163]. Although some research progress has been made to develop profilers for the cloud, only a few take into consideration the dynamic nature of the cloud. The profiler is responsible for collecting the performance metrics such as service rate, which are then sent to the container resource allocator. The data collected is aggregated for making

allocation decisions in an efficient manner. While the log of resource usage is captured by the profiler, the monitor captures the current resource usage. The benefits of profiling includes cost, application and resource management [163]. We discuss in details the role of the monitor next.

3.5.2 Resource Monitor

Resource monitoring plays a crucial role in providing streaming application guarantees like performance, availability, and security. A resource monitor is in charge of collecting real-time measures about key performance and other useful workload information by communicating with VM and container platform frequently. The information collected (e.g., CPU usage, memory, service rate, arrival rates) by the resource monitor can help the resource predictor and resource allocator to manage the VM and containers. It provides system-wide visibility into streaming application characteristics and workload changes. Monitoring is beneficial to CSBs since it helps them to analyze their resource requirements, and ensure that they get the requested amount of resources they are paying for. It also enables them to relinquish any underutilized resources and determine the amount of resources required to run their applications.

Continuous monitoring of the cloud supplies CSBs and users with information on the workload generated as well as the performance and QoS offered through the cloud [164]. While monitoring is of importance to provide elastic solutions in the cloud, it is not an easy task especially when dealing with containers. Since applications and all their dependencies are bundled into containers, which may be hosted on the same machine, accurately monitoring containers becomes tedious. There are several monitoring platforms that can be used to monitor resources in the cloud provided by CSPs (e.g., CloudWatch[3]), open-source (e.g., Nagios[4], Ganglia[5]) and developed for containers (e.g., Sysdig[6]). Information gathered from these tools are passed on to the resource predictor to create a model, and evaluation results from the predictor can be used by the resource allocator to perform scaling operations for running applications. For BDSAs running on a cluster of hosts, the resource prediction model needs to be robust enough to represent the typical variations in the underlying streaming jobs.

[3]https://aws.amazon.com/cloudwatch/
[4]https://www.nagios.org/
[5]https://developer.nvidia.com/ganglia-monitoring-system
[6]https://sysdig.com/

3.5.3 Resource Predictor

Techniques for prediction can be used in the context of cloud computing to help CSBs to optimize the utilization of resources. By correctly estimating the future demand for resources required, the right amount of resources that can deliver the expected SLA with minimum wastage can be achieved. By this, CSBs need to be able to dynamically adapt the cloud operational environment on the y to cope with variations in streaming workload dynamics. Hence, an approach to predict changes in the streaming workload patterns and adjusts promptly when variations in the patterns occur is needed. Adding a resource predictor to the framework will enable the CSB to predict the future workload according to the historical workload of the streaming applications.

3.5.4 Resource Allocator

The Resource Allocator controls computing resources practically. Through this module, the CSB can control resources, i.e., a dynamic resource allocation function can be implemented. The container resource allocator is the mechanism that optimizes the resource allocation for container instances and determines on which host the container workload should run. The CRAM module is the load balancer that takes the container-cluster system states and workload arrival from the CSB and attempts to adjust the load for the streaming applications to minimize the delay.

3.6 Resources

"A resource is any physical or virtual component of limited availability within the cloud environment [165]." Cloud resources typically include processors, memory, and peripheral devices offered by the CSP to the CSB as short-term on-demand and long-term reservation plans. In this section, we introduce the resource constituents of the proposed framework.

- *Compute Resource:* The compute resource is the core processing capability used to execute software instructions, which comprises of the CPU and memory. The CPU performs most of the processing inside the VM or container. CPU utilization is an issue in cloud computing, which varies depending on the amount and type of managed computing tasks. Some tasks may be CPU-intensive whereas others are less.

- *Storage Resource:* Datacenters usually provide access to internal storage as well as a storage area network that abstracts the complexity of accessing storage throughout the datacenter. Cloud storage systems generally rely on hundreds of data servers. This allows data to be maintained, managed and backed up remotely in the cloud using the network. It is used as an extension to the primary memory due to lower cost.

- *Network Resource:* A network resource is a set of virtual networking resources: virtual nodes (end-hosts, load balancers, switches, firewalls, routers) and virtual links. The network resource's weight in the cloud market has alerted network service providers to build their own distributed DCs with a vision to enter the cloud market [166]. A CSP provides network services to CSBs to: 1) connect CSBs' clients to VMs reserved in the DCs, 2) connect VMs inter-clouds to facilitate data exchange between them, and 3) connect VMs intra-clouds. There is an emerging trend towards virtualizing DC networks in addition to server virtualization to create multiple virtual networks (VNs) on top of a shared physical network substrate [3]. This allows each VN to be implemented and managed independently. This way, customized network protocols and management policies can be introduced. In addition, manageability and isolation to minimize security threats and better performance can be provided.

3.7 Summary

As an alternative to static provisioning, dynamically allocating resources to BDSAs by forecasting the expected workload demands is important to reduce unnecessary resource provisioning while guaranteeing the QoS. Based on the proposed framework, we address the resource prediction challenge by introducing proactive mechanisms to forecast the resource needs of multiple heterogeneous BDSAs utilizing a cluster of hosts in Chapter 4. In the cloud, mechanisms that ensure fair resource allocation determine whether CSBs should engage BDSAs in dynamic, shared cloud clusters. Such mechanisms should ensure that agents (i.e., BDSAs) that participate in shared environment cannot gain by misrepresenting their actions. In Chapter 5, we propose a novel container resource allocation mechanism along with a pricing scheme to address the resource allocation challenge.

Chapter 4

Cloud Resource Scaling for Big Data Streaming Applications

Majority of research efforts on resource scaling in the cloud are investigated from the cloud provider's perspective with little consideration for multiple resource bottlenecks. This chapter addresses the resource prediction problem from a CSB's point of view such that efficient scaling decisions can be made. It provides two contributions to the study of resource scaling for BDSAs in the cloud. First, a Layered Multi-dimensional Hidden Markov Model (LMD-HMM) is presented for managing time-bounded streaming applications. Second, to cater to unbounded streaming applications, a Layered Multi-dimensional Hidden Semi-Markov Model (LMD-HSMM) is proposed. The parameters in the models are evaluated using modified Forward and Backward algorithms. Detailed experimental evaluation results show that LMD-HMM is very effective with respect to cloud resource prediction for bounded streaming applications running for shorter periods while the LMD-HSMM accurately predicts the resource usage for streaming applications running for longer periods. A summary of the contributions in this chapter has been published in [35, 36].

4.1 Introduction

Recent years have witnessed a new class of real-time big data streams generated from social media, web clicks, IP logs, and sensing devices [10]. Processing these big data streams in real-time to extract value has become a crucial requirement for many time-bounded applications, running for a short period of time or unbounded big data stream processing applications that execute for a long period of time. With the real-time stream processing requirements for big data, traditional data management systems are not suitable and are prohibitively expensive when dealing with very large amounts of data [11]. To handle fast and massive streaming data, cloud services are increasingly leveraged. Cloud computing is a powerful paradigm that allows users to quickly deploy their applications in an elastic setting through on-demand acquisition/release of resources at a low cost [12]. Oftentimes, the cloud hosts many of the stream processing engines (SPEs) (e.g., Storm [13], Spark [14], and S4 [15]), which users can make use of depending on their requirements. These SPEs allow for the continuous processing of data as it is generated [11]. They also alleviate the performance problems faced by the traditional store-and-process model of data management since they process data directly in-memory.

Nevertheless, it is naturally challenging for big data application service providers to determine the right amount of resources needed to meet the QoS requirements of streaming applications, especially when multiple applications with varying resource bottlenecks (e.g., CPU, memory) running on a shared cluster of hosts are involved. A statically provisioned SPE (i.e., an SPE deployed on a fixed number of processing nodes) can lead to overprovisioning of resources for peak loads or to underprovisioning, which can have a huge impact on the performance of the streaming applications. Thus, to efficiently process dynamic streams, SPEs need to be able to scale up/down the used cloud resources accordingly. Different scaling techniques are used to reduce the cost of utilizing cloud resources. These involve i) adding virtual machine instances, i.e., horizontal scaling, ii) changing the amount of resources allocated to a virtual machine, i.e., vertical scaling, or iii) distributing workload among available virtual machines.

Prior work on resource scaling for streaming applications can be classified as either reactive or proactive. Although, reactive techniques are computationally attractive, i) resource sensitive applications may suffer from a degraded performance or even terminate before the need for resource up scaling is detected [22], ii) the experienced delay resulting from the scaling operations (e.g., adding a new virtual machine (VM), searching for a spot at a datacenter, allocating IP address, configuring and booting the OS [167]) may not be

tolerated by most streaming applications [144] and, iii) they perform poorly when multiple resources (e.g., CPU and memory) must be adjusted simultaneously [168]. Proactive techniques [169,170] overcome the aforementioned limitations by predicting future applications resource needs before resource bottlenecks occur. However, existing predictive approaches proposed for streaming applications can be cumbersome and may lead to inaccurate results. This problem is further complicated by the stringent delay constraints, increasing parallel computation requirements, dynamically changing data volumes and arrival rates of the data streams. Overall, these techniques often address the auto-scaling problem from a cloud provider's point of view and mostly focus on web applications.

We proposed the use of a layered multi-dimensional HMM (LMD-HMM) framework to facilitate the automatic scaling up/down of resources (i.e., add or decommission virtual machine) for time-bounded streaming applications. The proposed design provides an approach for modelling each application individually at the lower-layer while the upper-layer models interactions between groups of streaming applications running on a cluster. However, our experimental results showed that the LMD-HMM may not be sufficiently accurate for predicting the resource needs of unbounded streaming applications that run over very long periods. This can be attributed to its implicit assumption of constant or geometric distribution of resource consumption state durations that do not change over time. We overcome the aforementioned problem by proposing a novel Layered Multi-dimensional Hidden Semi-Markov Model (LMD-HSMM) that can accurately capture the resource demands of unbounded big data streaming applications. More precisely, the main contributions of this model are twofold. First, we introduce a novel mathematical model for predicting the resource usage patterns of multiple unbounded streaming applications running simultaneously on a cluster of cloud nodes. Second, we present a modified forward-backward algorithm [153], commonly used in calculating the predicted parameters of the Markov model that takes into consideration the prediction of multiple resource bottlenecks at each layer of the model. The proposed layered approach simplifies resource prediction for a large cluster of nodes by modelling the behavior of each streaming application individually at the lower layer of the model.

Furthermore, the multi-dimensional Markov process is used for each of these streams to separately capture the individual resource (e.g., memory, CPU and I/O) needs. The model also captures at a finer grain the frequency of resource consumption state changes per resource for each stream, which are then fed to the upper-layer in the model. The upper-layer then models the dependency and collective resource usage for the unbounded streaming applications in the cluster. This layered approach not only allows for the usage of different Markov models simultaneously but also accurately captures varying applica-

tion needs while avoiding the need to use a larger dataset to train since each streaming application model is trained individually. The frequency of retraining these models can be adjusted for each application.

The rest of this chapter is organized as follows: Section 4.2 briefly discusses the existing resource scaling approaches. Section 4.3 provides an overview of the prediction system. It also presents the learning and inference model for the resource prediction mechanisms proposed based on the layered multi-dimensional HMM/HSMM prediction technique. Section 4.4 reports our experimental results, and Section 4.5 concludes this chapter.

4.2 Related Work

Reactive scaling schemes in the cloud are known to allocate resources based on the workload changes detected using threshold-based rules. This approach requires that the application providers be aware of the resource usage patterns in order to set triggers for scaling activities. In practice, most cloud service providers, e.g., Amazon autoscale[1] and Azure[2], describe a set of service provider-defined rules that govern how the services scale up or down at run-time in reaction to changes in traffic. This approach can lead to a large delay before resource is provisioned. To address this limitation, proactive schemes are used whereby resource utilization patterns predicted through forecasting are provided in advance of workload changes and allocations are adjusted accordingly prior to any change.

Various predictive techniques [169–174] have been proposed for the cloud based on approaches such as time series analysis (e.g., linear regression, autoregression, Neural Networks, and ARIMA), control theory, reinforcement learning and queuing theory. The time series approach [175] incorporates temporal and spatial correlations in order to provide prediction results. An approach based on time series forecasting methodology was proposed by Baldan et al. [176] to forecast cloud workloads by combining statistical methods, cost function and visual analysis with focus on a short-time horizon. However, no seasonal pattern was found since the analyzed series were non-linear. Control theory [175] has been applied to automate management of resources in various engineering fields, such as storage systems, data centers and cloud computing platforms. The main objective of a controller is to maintain the output of the target system (e.g., performance of a cloud environment) to a desired level by adjusting the control input (e.g., number of VMs). The suitability

[1]https://aws.amazon.com/autoscaling/
[2]https://azure.microsoft.com/en-us/features/autoscale/

of control theory approaches depends on the type of controller and the dynamics of the target system.

Queuing theory [175] is used to find the relationship between jobs arriving and leaving a system. It is mostly used for Internet applications, however, it is not an efficient approach when used with elastic applications that have to deal with changing workloads and a varying pool of resources. While reinforcement learning [177] automates scaling tasks without prior knowledge of the application workload, it can lead to scalability problems when the number of states grow. Also, it yields bad performance since the policy has to be readapted as the environment conditions (e.g., workload pattern) change. Vijayakumar et al. [178] developed an algorithm that can handle dynamic patterns in data arrival to identify overflow or underflow conditions. Different from our work, this is a reactive approach that focuses mainly on CPU adaptation whereby streaming applications are implemented as a set of pipelined stages. Here, each stage acts as a server for the next.

Nikravesh et al. [179] proposed an auto-scaling system based on Hidden Markov Model (HMM). In their research, the states are associated with system conditions and the observation are the sequence of generated scaling actions. Their experiments targeted e-commerce websites, i.e., multi-tier applications, which typically includes a business-logic tier, containing the application logic as well as a database tier. While this approach makes use of HMM, its applicability directly to big data stream processing applications is limited because it does not take into consideration multiple streaming applications, which can lead to ample retraining of the model in response to changes in active streaming jobs, and overfitting of data. In contrast, our approach makes use of a two-layered multi-dimensional Hidden Markov/semi-Markov model; the lower-layer is used for training individual streaming jobs without altering the upper-layer, and whenever a job is not running, the model does not need to be recalculated while the upper-layer is used for training group applications running on clusters. By so doing, the outcome of the upper-layer will be less sensitive because it is based on the collective behaviour of the streaming jobs from the lower-layer that are expected to be well trained. Finally, our framework can accommodate different streaming applications and it is application independent.

Several predictive models have been developed in the past for short-term (bounded) and long-term (unbounded) predictions of performance metrics for VM clusters using machine learning techniques. However, few works exist for big data streaming applications. PLAStiCC [180] uses short term workload and infrastructure performance predictions to actively manage resource mapping for continuous dataflows to meet QoS goals while limiting resource costs. For time-unbounded streaming application, the work in [181] proposed

a resource scheduler which dynamically allocates processors to operators to ensure a real-time response. This work focuses on streaming operator scaling while our research is aimed at scaling resources. The authors in [182] proposed Elysium, a fine-grained model for estimating the resource utilization of a stream processing application, which enables the independent scaling of operators and resources using the reactive and proactive approaches. However, they did not consider multiple applications with varying resource bottlenecks.

The use of containers has been exploited recently for scaling stream processing operators. Pahl & Lee [183] reviewed the suitability of container and cluster technology as a means to address elasticity in highly distributed environments. While the use of containers can provide for faster provisioning during scale in/out as compared to VMs, streaming applications will need to be designed as lightweight SOA-style independently deployable microservices to gain this benefit. Moreover, some containers run on VMs instead of bare metal machines and are limited to the amount of VMs provisioned. Hence, VM scaling is required. To the best of the author's knowledge, the application of HSMM to unbounded big data streaming applications is novel.

4.3 Proposed Prediction Model

We consider a CSB that provides big data stream processing services for cloud users in a scalable distributed environment as shown in Fig. 3.2 from Chapter 3. The applications collect data from a wide variety of sources, and based on user requirements, integrate them to detect interesting events and patterns. We hereby present our solution for handling the resource prediction of big data streaming applications, whereby resource needs are forecasted in advance by monitoring each streaming job and its associated characteristics. Our solution combines knowledge of stream processing with statistical Markov models. The key idea of our approach is to decouple the problem space into layers, whereby each streaming job is independently modeled as a multi-dimensional HMM/HSMM at the lower-layer and the generated observations are concatenated afterwards at the upper-layer.

We extend the standard HMM and HSMM to a layered multi-dimensional HMM/HSMM for the resource predictor. The layered approach allows for the decomposition of the parameter space into distinct layers, whereby each layer is connected to the next by inferential results. In our case, we make use of a two-layered model (upper-layer and lower-layer) as shown in Fig. 4.1 where each streaming application can be modeled as either a multi-dimensional HMM or multi-dimensional HSMM at the lower layer. The lower-layer allows for the abstraction of individual streaming jobs' observations at particular times and also

ensures independence from others while the upper-layer is used to predict the resource usage pattern for the group streaming applications running on a cluster.

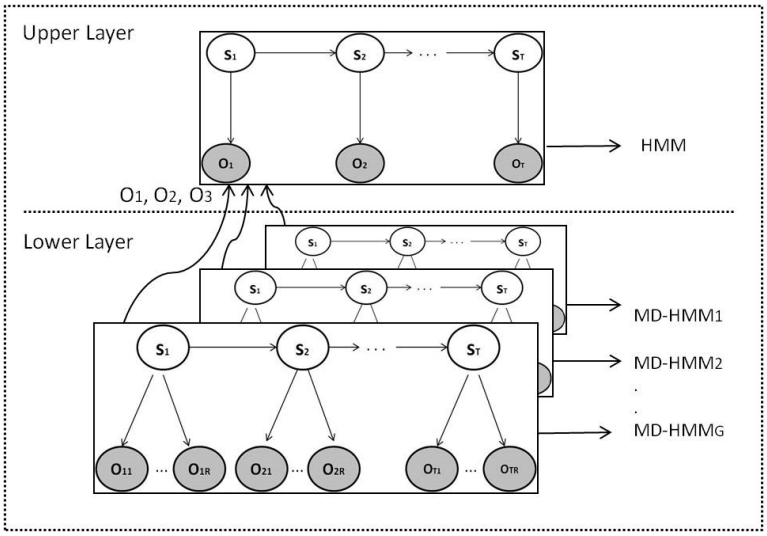


Figure 4.1: A Layered Multi-dimensional HMM.

4.3.1 The Lower-Layer

The lower-layer denotes the resource utilization for each individual streaming job (e.g., traffic analysis, weather analysis, and anomaly detection), where each streaming job's HMM/HSMM has more than one observable symbols at each time t. We make use of the multi-dimensional HMM (MD-HMM) or multi-dimensional semi-HMM (MD-HSMM) at the lower-layer to represent individual streaming job's resource usage pattern, whereby the system bottlenecks (e.g., CPU, memory) being considered form the dimensions to the process. The observations from the submodel MD-HMM/MD-HSMM corresponding to each streaming job from the lower layer are then concatenated afterwards into a new single layer HMM at the upper-layer. Next, we discuss estimation and learning algorithms for the models. We list in Tbl. 1 important notations used in this chapter.

4.3.2 MD-HMM Inference & Learning Mechanisms

The parameters in our multi-dimensional HMM can be defined as follows:

- N: Number of states associated with the resource usage patterns for each streaming job. This is denoted as $S = \{S_1, S_2, ..., S_N\}$, and the state at time t as q_t.

Table 4.1: Notations used in the LMD-HMM & LMD-HSMM models

Variables	Definitions
N	Number of hidden states in the model representing the resource usage patterns
R	Number of observable dimensions for each streaming job
M	Number of distinct observation symbols, which describes the set of states for individual streaming job's dimension
A	Hidden state transition probability distribution for each streaming job
B	Set of observable probability distribution matrices
O	$O = \{O_1, O_2, ..., O_T\}$ Observation sequence
S	$S = \{q_1, q_2, ..., q_T\}$ State sequence
D	Time spent in a particular state
$b_j(O_k^r)$	Probability of emitting observation symbol O_k^r in state j for each dimension r
π	Initial probability of distribution
$\alpha_t(i)$	Forward variable that denotes the probability of generating $O_{1:T}$ and ending in state S_i at time t
$\beta_t(i)$	Backward variable that denotes the probability of generating $O_{t+1:T}$ and begins in state S_i at time t
$\gamma_t(i)$	Probability of being in S_i at time t, given the observation sequence $O_{1:T}$
$\xi_t(i,j)$	Conditional the probability of being in state S_i at time t and in state S_j at time $t+1$, given the observation sequence $O_{1:T}$
$\bar{\lambda_g}$	Re-estimated parameters of the HMM/HSMM model for each streaming job

- R: Number of observable dimensions for each streaming job. In our model, system bottlenecks such as CPU and memory utilization are dimensions in the process.

- M: Number of distinct observation symbols in each state for individual streaming job's dimension. Hence, for a given streaming job, in state $q_t = S_i$, there will be $M \times R$ distinct observations, where $O^r = \{O_1^r, O_2^r, ..., O_M^r\}$ are individual observation sequences for each dimension $1 \leq r \leq R$. For simplicity, we assume that the number of states for each dimension is the same.

- A: Hidden state transition probability distribution for each streaming job $A = \{a_{ij}\}$,

where $a_{ij} = P\{S_{t+1} = S_j \mid q_t = S_i\}, \quad S_i, S_j \in S, \quad 1 \leq i, j \leq N$, described by the characteristics of individual streaming job at time t.

- **B**: A set of observable probability distribution matrices, where $b_j(O_k^r)$ is the probability of emitting observation symbol O_k^r in state j for each dimension r, i.e., $b_j(O_k^r) = P[O_t = O_k^r | q_t = S_j]$ for $\quad 1 \leq k \leq M$ and $\quad 1 \leq r \leq R$. In the multi-dimensional HMM, there will be R B-matrices, i.e., $\mathbf{B} = \{B_1, B_2, ..., B_R\}$, where each B_r characterizes the stochastic distribution for each dimension of the individual streaming job.

- π: Initial probability of distribution $\pi_i = P(q_1 = S_i), \quad 1 \leq i \leq N$.

Given these parameters, a multi-dimensional HMM (MD-HMM) can be used to derive the sequence of observed symbols $O = \{O_1, O_t, ..., O_T\}$, where $O_t = \{O_t^1, O_t^2, ..., O_t^R\}$.

The complete set of the model is represented as $\lambda = (N, R, M, A, B, \pi)$. The MD-HMM model for each streaming job $\lambda_g = (N_g, R_g, M_g, A_g, B_g, \pi_g)$, where $g = 1, ..., G$, can be used to solve three basic problems (i.e., evaluation, decoding and training) that arise:

- How to compute the probability $P(O|\lambda_g)$ of the observation sequence $O = \{O_1, O_2, ..., O_T\}$ given the model λ_g. This problem corresponds to evaluation, which can be solved using the Forward algorithm.

- How to compute the corresponding state sequence that best explains the observation. This is a decoding problem in which the Viterbi algorithm [153], a dynamic algorithm that computes both the maximum probability and the state sequence given the observation sequence, is used.

- Given the observation O, how to find the model λ_g that maximizes $P(O|\lambda_g)$. This problem corresponds to learning to determine the model that best fits the training data. The Baum-Welch algorithm [184], which can be extended to reflect the multi-dimensional properties based on the independence assumption, can be used for solving this problem. The learning problem adjusts the model parameter λ_g, such that $P(O|\lambda_g)$ is maximized.

For simplicity, we will drop the subscript g thereafter since we are focusing on individual streaming job at the lower layer.

Given a model λ, an observation sequence $O = \{O_1, O_2, ..., O_T\}$, and a state sequence $Q = \{q_1, q_2, ..., q_T\}$ where $q_t = S_i$ at time t, all the parameters for solving the problems

depend on the forward-backward variables. The forward variable $\alpha_t(i)$ is the joint probability of the partial observation sequence $O_1, O_2, ..., O_t$, given the hidden state at time t is $q_t = S_i$. The backward variable $\beta_t(i)$ is the joint probability of the partial observation sequence $O_{t+1}, O_{t+2}...O_T$, given the hidden state at time t is $q_t = S_i$.

The forward procedure is a recursive algorithm for calculating $\alpha_t(i)$ for the observation sequence of increasing length t. In our framework, since each observed dimension is assumed to be independent, the forward variable $\alpha_t(i)$ can be modified and the output for each state i recomputed as the Cartesian product of the observation probabilities of each dimension as follows:

$$\alpha_t(i) = P(O_1, O_2...O_t, q_t = S_i|\lambda) \tag{4.1}$$

$$\alpha_1(i) = \pi_i \prod_{r=1}^{R} b_i(O_1^r), \quad 1 \leq i \leq N \tag{4.2}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] \prod_{r=1}^{R} b_j(O_{t+1}^r), \quad 1 \leq t \leq T-1, \quad 1 \leq j \leq N \tag{4.3}$$

where $\alpha_t(i)$ is the probability of the partial observation sequence $O_1, O_2, ..., O_t$ with state $q_t = S_i$, given the model λ and $\alpha_{t+1}(j)$ is the probability of the partial observation sequence $O_1, O_2, ..., O_t, O_{t+1}$ at state S_j at time $t+1$.

Likewise, recursion described in the forward algorithm can as well be used backwards in time in the backward algorithm. The backward variable can be recomputed as:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}...O_T|q_t = S_i, \lambda) \tag{4.4}$$

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{4.5}$$

$$\beta_t(i) = \left[\sum_{j=1}^{N} a_{ij} \beta_{t+1}(j)\right] \prod_{r=1}^{R} b_j(O_{t+1}^r), \quad 1 \leq t \leq T-1, 1 \leq i \leq N \tag{4.6}$$

where $\beta_t(i)$ is the probability of the partial observation sequence $O_1, O_2, ..., O_T$ from $t+1$ to the end, given the state $q_t = S_i$ at time t and the model λ. The state transition probability distribution from S_i to S_j is a_{ij}, and $b_j(O_{t+1}^r)$ is the observation symbol probability distribution of the occurrence of observable value (O_{t+1}^r), given state S_i at time t. b_j

is normalized based on the outputs of each dimension. The observation evaluation is then:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i), \quad \forall t. \tag{4.7}$$

especially the forward variable as defined in Equation 4.1,

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{4.8}$$

To address the training problem of finding the optimum model parameter vector $\lambda \in \Lambda$ that maximizes $P(O|\lambda)$, given an observation sequence O, the Baum-Welch algorithm [184] can be extended for the MD-HMM in terms of the joint events and state variables, respectively:

$$\begin{aligned} \xi_t(i,j) &= P(q_t = S_i, q_{t+1} = S_j|O,\lambda) \\ &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} \\ &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N} \alpha_T(i)} \end{aligned} \tag{4.9}$$

$$\begin{aligned} \gamma_t(i) &= P(q_t = S_i|O,\lambda) \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_T(i)} \end{aligned} \tag{4.10}$$

$\xi_t(i,j)$ is the probability of being in state S_i at time t and in state S_j at time $t+1$, given the observation sequence $O_1, O_2, ..., O_T$ and $\gamma_t(i)$ is the probability of being in S_i at time t.

The variables $\alpha_t(i)$, $\beta_t(i)$, $\xi_t(i,j)$, $\gamma_t(i)$ are found for every training sample and then, re-estimation is performed on the accumulated values. To generate the new better model $\bar{\lambda}$, the state transition, observation symbol and initial state probabilities can be updated as follows:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, 1 \leq i \leq N, 1 \leq j \leq N \tag{4.11}$$

$$\bar{b}_j^r(k) = \frac{\sum_{t=1, O_t^r = O_k^r}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}, \tag{4.12}$$

$$1 \leq r \leq R, 1 \leq j \leq N, 1 \leq k \leq M$$

$$\bar{\pi}_i = \gamma_1(i), 1 \leq i \leq N \tag{4.13}$$

A set of new model parameters $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$, which are the average of the result from each individual training sequence, is obtained after the update and the process is repeated until changes in the parameters are smaller than a predefined threshold, i.e., optimal is reached.

In the testing phase, a test sequence is feed to each of the MD-HMM and the maximum likelihood given the observation is evaluated using the Viterbi algorithm [153]. The Viterbi algorithm is used to obtain the probable sequences using the forward item at time t in state i based on the output of the learned parameters from training.

4.3.3 MD-HSMM Inference & Learning Mechanisms

In this section, we extend the standard HSMM in order to address the need to model multiple resource bottlenecks. HSMM is very useful in applications that run unbounded for longer periods to accommodate shifts in behaviour from the HMM-based model. This model relaxes the duration assumption on resource usage states for big data streaming applications. To define the parameters for the MD-HSMM, duration is included in addition to the parameters defined for the MD-HMM in Section 4.3.2 as:

- D: The probability distribution matrix of remaining in state i for d time intervals, i.e., $P_i(d)$.

We defined the lower-layer model MD-HSMM as $\lambda = (\pi, N, R, M, A, B, D)$ to reflect the duration of resource usage states for each streaming application, i.e., each streaming application is modeled as a multi-dimensional hidden semi-markov (MD-HSMM) at the

lower-layer. Therefore, given the observable sequence in the past T duration, we need to find its most likely state sequence S^* and the most likely state model MD-HSMM λ^* for each streaming application. This involves solving the basic problems of evaluation, recognition and training as described next.

The process of training and recognition involves redefining the forward and backward variables. The addition of duration in the model makes the algorithm more complex than the HMM-based model. We define a forward variable $\alpha_t(i)$ as the probability of generating $O_1, O_2, ..., O_t$ and ending in state i as defined in Eqn 4.1. Assuming that the first state begins at time 1 and the last state ends at time T, the initial conditions for the forward recursion formula become:

$$\alpha_1(i) = 1, \tag{4.14}$$

$$\alpha_t(j) = \sum_{i=1}^{N} \sum_{d=1}^{D} \alpha_{t-d}(i) a_{ij} P_j(d) \prod_{r=1}^{R} \prod_{s=t-d+1}^{t} b_j(O_s^r) \tag{4.15}$$

Assuming $\alpha_{t-d}(i)$ has been determined where the duration in state j is d, a_{ij} is the state transition probability from state i to state j, $b_j(O_s^r)$ is the output probability of the observation O_s^r for each dimension from state i and $P_j(d)$ is the state duration probability of state j. Also, N is the number of states in the MD-HSMM and D is the maximum duration in state i.

Likewise, the backward variable can be written as Eqn 4.4 and the value for the initial conditions of the backward recursion formula $t = T$ is as written in Eqn 4.5. Summing over all the states N and all possible states duration d, we have the recurrence relations:

$$\beta_t(i) = \sum_{i=1}^{N} \sum_{d=1}^{D} a_{ij} P_j(d) \beta_{t+d}(j) \prod_{r=1}^{R} \prod_{s=t+1}^{t} b_j(O_s^r) \tag{4.16}$$

We then define three variables to obtain the re-estimation formulae for the MD-HSMM as follows:

$$\alpha_{t,t'}(i,j) = P(O_1^{t'}, q_t = S_i, q_{t'} = S_j|\lambda) \tag{4.17}$$

$$\Omega_{t,t'}(i,j) = \sum_{d=1}^{D}[P(d = t' - t|i) \cdot$$

$$P(O_{t+1}^{t'}|q_t = S_i, q_{t'} = S_j, \lambda)] \tag{4.18}$$

$$\xi_{t,t'}(i,j) = P(q_t = i, q_t' = S_j|O_1^T, \lambda) \tag{4.19}$$

where $\alpha_{t,t'}(i,j)$ is the probability of the partial observation sequence and state i at time t and state j at time $t', (t' = t + d)$, $\Omega_{t,t'}(i,j)$ is the mean value of the probability of being in state i for d time intervals and moving to state j and $\xi_{t,t'}(i,j)$ is the probability of being in state i for d time intervals and moving to state j given $O = O_1, O_2, .., O_T$.

$\alpha_{t,t'}$ can be described in terms of $\Omega_{t,t'}(i,j)$ as:

$$\alpha_{t,t'}(i,j) = P(O_1^{t'}, q_t = S_i|\lambda) \cdot$$

$$P(O_{t+1}^{t'}, q_t = i, q_{t'} = j|O_1^t, q_t = i, \lambda) \tag{4.20}$$

$$= \alpha_t(i)a_{ij}\Omega_{t,t'}(i,j) \tag{4.21}$$

and the relationship between $\alpha_t(i)$ and $\alpha_{t,t'}(i,j)$ is given as:

$$\alpha_{t'}(j) = P(O_1^{t'}, q_{t'} = S_j|\lambda) \tag{4.22}$$

$$= \sum_{i=1}^{N}\sum_{d=1}^{D}[P(d = t' - t|S_j)\alpha_{t,t'}(i,j) \tag{4.23}$$

The $\xi_{t,t'}(i,j)$ can be derived as follows:

$$\xi_{t,t'}(i,j) = \frac{\sum_{d=1}^{D}\alpha_t(i)a_{ij}\Omega_{t,t'}(i,j)\beta_{t'}(j)}{\beta_0(i = s_0)} \tag{4.24}$$

where s_0 is the initial resource usage state. Given the initial value of the model λ^0, the Expected-Maximization (EM) algorithm is used to find the best model λ^* using the recursive computing procedure. The re-estimation formulas for the MD-HSMM can be written as follows:

- The re-estimation for the initial distribution is the probability that state i was the first state given O.

$$\bar{\pi}_i = \frac{\pi_i[\sum_{d=1}^{D} \beta_d(i) P(d|i) b_j(O_1^d)]}{P(O_1^T|\lambda)} \tag{4.25}$$

- The re-estimation of state transition probabilities is the ratio of the expected number of transitions from state i to state j to the expected number of transitions from state i.

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{j=1}^{N} \sum_{t=1}^{T} \xi_{t,t'}(i,j)} \tag{4.26}$$

- The re-estimation formula for the observation distribution is the expected number of times that observation $O_t = v_k$ occurred in state i, normalized by the expected number of times that any observation occurred in state i.

$$\bar{b}_i(k) = \frac{\sum_{t=1}^{T} \alpha_t(i) \left[\frac{\Omega_{t,t'}(i,j)}{\sum_{d=1}^{D} P(d=t'-t|i)} \right] \beta_t(i)}{\sum_{t=1}^{T} \alpha_t(i) \left[\frac{\Omega_{t,t'}(i,j)}{\sum_{d=1}^{D} P(d=t'-t|i)} \right] \beta_t(i)} \tag{4.27}$$

The optimal states S^* and model λ^* are then continuously updated by replacing λ with λ^* if it gives a higher probability, until a limit is reached. The limit represents the number of iterations and signifies when to stop the re-estimation or find the probability of observing the sequence O cannot be improved beyond a certain threshold.

A problem with duration of the HSMM-based model is the large number of parameters associated with each state. This makes the re-estimation problem more difficult than the HMM-based model. To alleviate this, we make use of a parametric state duration density instead of the non-parametric duration density that requires less training. We adopt the Gaussian distribution method. The theorem states that the mean of any set of variates with any distribution having a finite mean and variance tends to the Gaussian distribution. The mean $\mu(j)$ and the variance $\sigma^2(j)$ of the duration of state j can be determined as follows:

$$\mu(j) = \frac{\sum_{d=1}^{D} \{ \sum_{t=1}^{T-d} (\sum_{t=1}^{N} \alpha_t(i) a_{ij}) P_j(d|j) b_j(O_{t+1}^{t+d}) \beta_{t+d}(j) \} d}{\sum_{d=1}^{D} \sum_{t=1}^{T-d} (\sum_{t=1}^{N} \alpha_t(i) a_{ij}) P_j(d|j) b_j(O_{t+1}^{t+d}) \beta_{t+d}(j)} \tag{4.28}$$

$$\sigma^2(j) = \frac{\sum_{d=1}^{D}\{\sum_{t=1}^{T-d}(\sum_{t=1}^{N}\alpha_t(i)a_{ij})P_j(d|j)b_j(O_{t+1}^{t+d})\beta_{t+d}(j)\}d^2}{\sum_{d=1}^{D}\sum_{t=1}^{T-d}(\sum_{t=1}^{N}\alpha_t(i)a_{ij})P_j(d|j)b_j(O_{t+1}^{t+d})\beta_{t+d}(j)} - \mu^2(j) \tag{4.29}$$

4.3.4 The Upper-Layer

Since the lower-layer does not model the relationships between the streaming jobs, the outputs from the lower layer, which are expected to be well-trained are used as the new observation O' sequence for the upper-layer. Therefore, devising the right mechanism for determining how the outputs from the lower-layer will be used as input features into the upper-layer becomes one of the issues. We hereby discuss our approach to this problem.

Suppose G MD-HMMs or MD-HSMMs are trained at the lower-layer, with $g = \{1, .., G\}$. To transform the lower layer MD-HMM/MD-HSMM into a single HMM/HSMM at the upper layer, the likelihood of observing the g^{th} MD-HMM/MD-HSMM in state i is used in the recursion formula of the upper layer HMM/HSMM to pass the probability distribution up. More precisely, the predicted states from the trained model λ^* from the lower layer MD-HMM/MD-HSMM becomes the input observations for the upper layer HMM/HSMM.

We define the observation sequence from the lower-layer as $O' = O'_{1:T}$, where T is the length of the observable sequence. We also define the probability $\rho(i,t) = \alpha(i,t)$ of being in state i at time t for each model, where $\alpha(i,t)$ is the forward variable of the Baum-Welch algorithm [184] for the observed sequence $O'_{1:T}$, with $\alpha(i,t) \triangleq P(O'_{1:T}, q_t = i)$. We normalize the probability $\rho(i,t)$ for all states of all the models, i.e.,

$$\sum_{j=1}^{N_s} P(q_t = j) = 1 \tag{4.30}$$

where N_s is the number of all states for all models. Then, the probability $P(q_t = i|O'_{1:T})$ of state i given a sequence $O'_{1:T}$ is:

$$P(q_t = i|O'_{1:T}) = \frac{P(q_t = i, O'_{1:T})}{P(O'_{1:T})} \tag{4.31}$$

$$= \frac{P(q_t = i, O'_{1:T})}{\sum_{j=1}^{N_s} P(q_t = j, O'_{1:T})} \tag{4.32}$$

$$= \frac{\rho(i,t)}{\sum_{j=1}^{N_s} \rho(j,t)} \tag{4.33}$$

where i is the state in the model, which is a subset of the states of all models, and N_s is the total number of states.

The observations at the upper-layer is the probability distribution, $\{P(1:G)_1, .., P(1:G)_T\}$, for each of the lower-layer model λ_g for the time interval, i.e., a vector of G reals for each time interval, will be used as input to the upper-layer. The upper-layer HMM/HSMM is then used to complete the probability of a certain state as a function of time given the observation sequence produced by the lower layer MD-HMMs/MD-HSMMs.

At the upper- layer, a single HMM/HSMM can then be used to model the resource usage of group applications running on the cluster, where each state in the HMM/HSMM corresponds to a resource scaling action. For instance, the HMM for the upper-layer is formally defined as $\lambda' = (N', M', A', B', \pi')$, where N' describes the resource scaling actions (which can be scaling up, down or no scaling), M' is the set of observation symbols for the resource demand states learned from the lower-layer observations, π' is the initial state distribution, A' is the state transition probability distribution matrix and B' is observable probability distribution matrix. With this new model, training can be done using the same learning and inference mechanisms used for HMM. In the next section, we implement our proposed models, making use of the modified variables and parameters.

4.4 Experiments

The focus of our experiment is to validate the LMD-HMM and LMD-HSMM models. We first describe the experimental setup including the environment and datasets. To verify the flexibility and general applicability of the proposed models on predicting the resource usage for big data streaming applications, the framework was tested on applications developed to access Twitter API due to lack of big data streaming benchmarks and data repository

that fits into our problem. Apache Spark was chosen as the stream processing engine due to its flexibility of deployment over available machines.

4.4.1 Experimental Setup

The Spark cluster was setup on a physical machine with 8 core i7-3770 CPU @ 3.40 GHz Intel processors and 16GB of RAM. Two VM instance types were considered with configurations of 4GB and 8GB RAM with 2VCPUs each. Each node was installed with Hadoop 2.6, Spark 1.6, Scala 2.11 along with JDK/JRE v1.8. One node was configured as the master while the other nodes were configured as slaves. The master node was equipped with a 195GB SSD drive and the slave nodes with 50GB SSD drives each.

Hadoop was configured to make use of HDFS for providing storage for the streaming data and Apache Spark was configured to run on top of Hadoop since it does not provide an independent storage system. The streaming data was collected using Apache Flume, a distributed system for collecting, aggregating data and transporting the events generated. At each interval, an amount of data is downloaded and persisted in HDFS for immediate processing by the Spark engine. Two streaming application programs (Tophashtags and Sentiment Analysis) were used in evaluating the prediction model. The developed applications were launched at different time intervals on the master node and monitored.

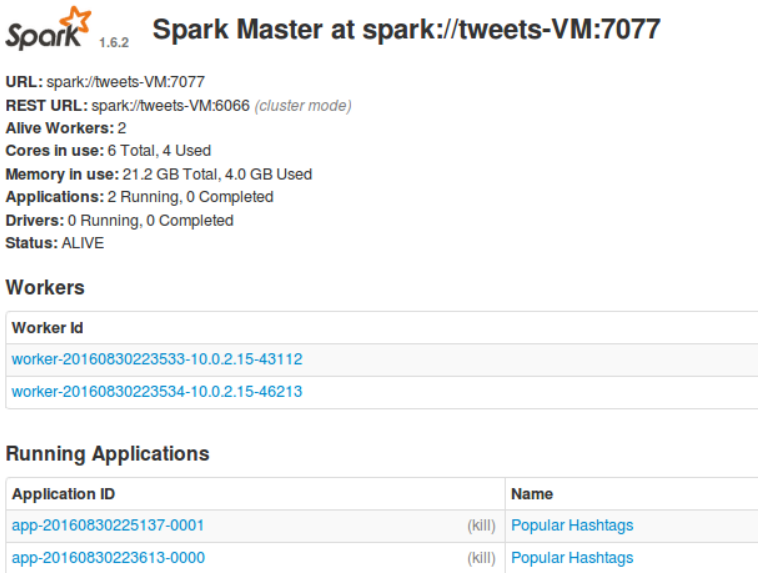


Figure 4.2: Big Data Streaming Applications

Fig. 4.2 shows a snapshot of application submission to the Spark Master VM. The workload arrival rates, number of streams and a mix of system performance metrics (CPU,

memory) for each streaming applications were collected and stored for further analysis. A snapshot of the multi-dimensional resource usage (CPU utilization & Memory) for one of the applications recorded at 1 min intervals is presented in Fig. 4.3.

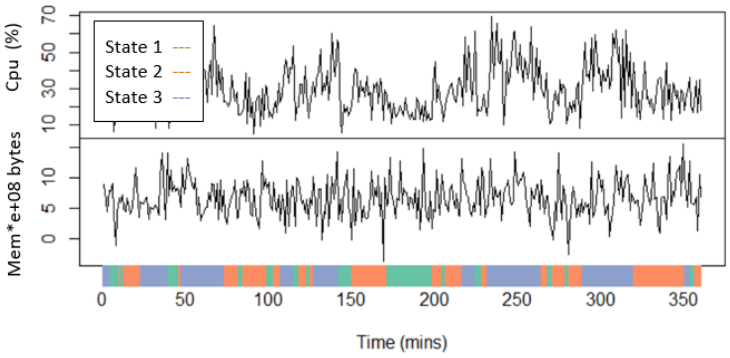


Figure 4.3: Multi-dimensional (CPU & Memory) Observation for a stream

4.4.2 LMD-HMM Implementation

To implement the LMD-HMM model, our goal is to first evaluate the lower-layer MD-HMMs with respect to each streaming job. The collected datasets were partitioned into two sets: a training set and a prediction set. After the models have been processed using the training set, we tested them against the prediction sets to determine their accuracy. The results from the MD-HMM were then used as input features to the upper-layer HMM to make the final resource scaling decision. For training and testing, we make use of RStudio, a statistical software for R.

4.4.2.1 Training the MD-HMMs

We employ our modified Baum-Welch expectation maximization algorithm to fit the model when the dataset is not trained. When the prediction model is trained, the Viterbi algorithm can be applied to determine the hidden states. The objective of the training algorithm is to optimize the MD-HMM parameters such that the overall training sequence likelihood is maximized or until convergence is reached. We observed the resource usage for the streaming applications for a period ranging from a few minutes up to $60hrs$.

The optimal number of states for the MD-HMMs is determined using the K-means clustering algorithm to find the initial parameters of the distributions belonging to each state. It is worth noting that our choice to employ the K-means algorithm was due to

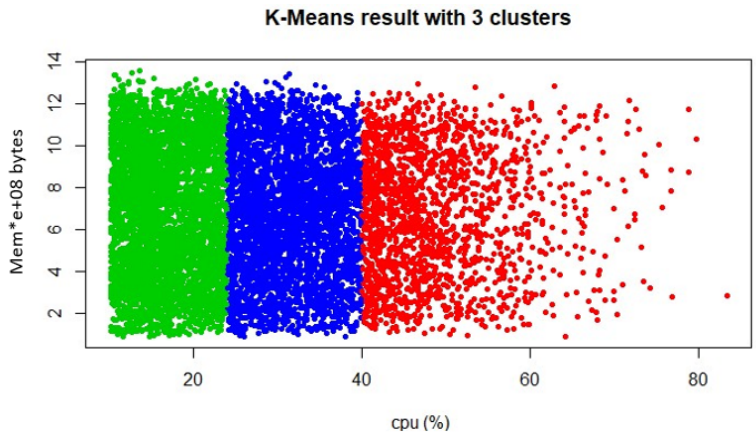


Figure 4.4: K-Means using CPU & Memory Values

its simplicity, however, any clustering algorithm can be used. The K-means algorithm grouped the multi-dimensional data into k clusters during the clustering process using the data points as shown in Fig. 4.4, where each colored cluster represents the resource usage states. From the elbow graph in Fig. 4.5, it can be seen that the location of a bend in the graph is between 2 to 4 points, which is an indicator of the appropriate number of clusters. Considering the within-cluster sum of squares (SS), we selected 3 states from the result of the clusters as determined by the K-means algorithm. Each cluster contains a pair of values representing the mean and data points that belong in the cluster.

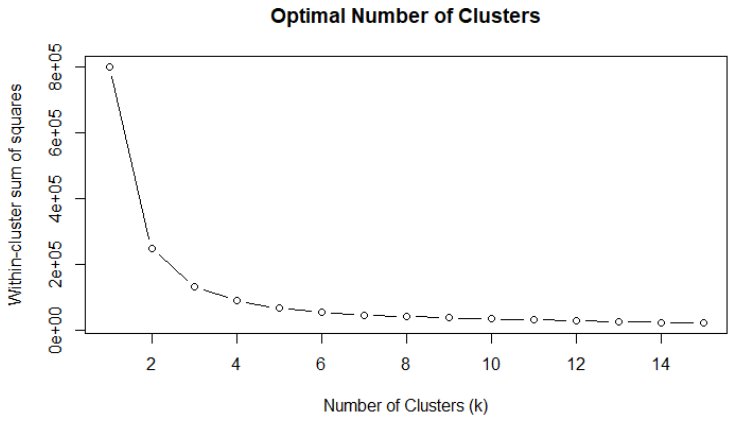


Figure 4.5: K-Means Optimal Number of Clusters

We use the k cluster value derived from the K-means clustering as input to the training algorithm and set the initial parameters as shown in Tbl. 4.2. Since it starts from a randomly initialized MD-HMM, the algorithm is run several times with different initializations and the best model is kept for prediction.

- We set states in the initial probability distribution matrix as $\pi = [0 \quad 0 \quad 1]$.
- In our 3-states model, there will be nine transition probabilities A.
- For the observation probabilities, we use Gaussian mixture distribution. We define the mean μ and variance σ^2 for each dimension of the model.

Table 4.2: Probabilities Matrix

Transition States (A)		
0.858	0.142	0.000
0.053	0.926	0.021
0.013	0.019	0.968
Observation (μ_1) - cpu (%)		
16.877	26.351	38.981
Observation (μ_2) - mem (b) $* e + 08$		
6.795	6.889	6.986
Variance (σ_1^2) - cpu (%)		
18.883	76.428	214.096
Variance (σ_2^2) - mem (b) $* e + 08$		
9.657	8.961	8.716

With the help of the defined variables and the re-estimation formula, we then find the best MD-HMM model λ^*. The modified Baum-Welch algorithm is used to train the model to produce the state probabilities, transition state probabilities, and observation mean and variances. The Baum-welch estimates the model such that the log-likelihood probability of generating the observation sequence O is maximized. The model is trained by varying the initial parameters (λ) with multiple runs to obtain the updated parameters ($\bar{\lambda}$). Fig. 4.6 shows that the log-likelihood achieves its maximum at about 3 iterations.

Determining the best model is a model selection problem tackled to prevent overfitting of data. Since there is no simple, theoretically correct way of making model choices, techniques like the Bayes Information Criterion (BIC) can be used for determining the best model. The BIC uses the likelihood as a measure of fit and penalizes complexity (increased number of states) as a means of mitigating the risk of overfitting. Also, BIC takes the number of parameters into account when determining the best model. The graph of BIC values obtained with different choices of k states will show increase with k

to a certain point before reaching a plateau. The smallest k for which the BIC no longer increases signals a better model. To select the best model, we trained a series of candidate MD-HMM with an increasing number of states (k).

4.4.2.2 Predicting the MD-HMMs

This step involves computing the current state of the system and using this to predict the future state. Here, we try to generate the sequence of states responsible for producing the observation sequence. After training the MD-HMMs for each streaming job, the Viterbi algorithm [153] is used to compute the most likely sequence of hidden variables which could have generated our observations. We use as inputs values (initial, transition and observation) from the defined MD-HMM model and expect the list of states as outputs. In order to determine the effectiveness of our approach in predicting workloads of varying sizes, we varied the training set period and the prediction period as shown in Tbl. 4.3 to determine the effect that changes in training and prediction periods will have on the accuracy of the result. The MAPE and RSME values indicate the level of correlation between predicted and referenced durations for the test data. As shown in the table, the accuracy of the result decreases as the prediction period increases, which indicates greater accuracy for shorter prediction intervals. Therefore, the MD-HMM model is ideal for predicting accurately the resource usage patterns for time-bounded streaming applications with shorter durations of up to 4hrs.

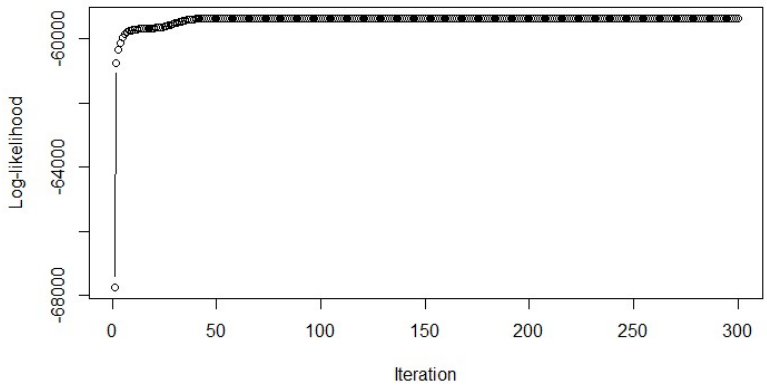


Figure 4.6: Log-likelihood of the MD-HMM Model

4.4.2.3 Validating the MD-HMMs

The accuracy of our model is determined by how well the model performs on new datasets that were not used when fitting the model. We evaluated the prediction accuracy of

the MD-HMM models based on two performance indexes, which are the Mean Absolute Percentage Error (MAPE), a commonly used scale-independent measure, and the Root Mean Square Error (RSME), a scale-dependent measure to determine the % error in our model. MAPE is defined as the sum of the absolute deviation by the total of the predictions. The calculation is written as:

$$MAPE = \frac{1}{n} \sum \left(\frac{|Actual - Forecast|}{|Actual|} * 100 \right) \tag{4.34}$$

This was calculated for the next forecast interval in time divided by the number of interval.

The RSME is calculated as:

$$RSME = \sqrt{\sum_{t=1}^{n} \frac{(Actual - Forecast)^2}{n}} * 100 \tag{4.35}$$

Table 4.3: Prediction Set Size Variation & Accuracy of the MD-HMM

Experiment Period	Prediction Period	RSME (%)	MAPE (%)
24hrs	4hrs	88.7	91.2
24hrs	6hrs	88.2	89.0
24hrs	8hrs	86.7	86.0
48hrs	8hrs	88.6	89.0
48hrs	12hrs	86.3	86.5

4.4.2.4 Upper-layer HMM

The upper-layer HMM is trained on the outputs from the lower-layer MD-HMMs using the normalized probabilities of the states of the MD-HMMs as input. We compute the A' and B' matrices, which are also trained using the Baum-Welch algorithm [184] and the most probable states determined using the Viterbi algorithm [153]. The graph in Fig. 4.7 shows a prediction scenario where the x-axis indicates an interval of 1 min for a 6hr prediction period depicting the accuracy of the prediction system for highly variable workload and the y-axis shows the resource scaling states, (i.e., 1-down, 2-no scaling, and 3-up).

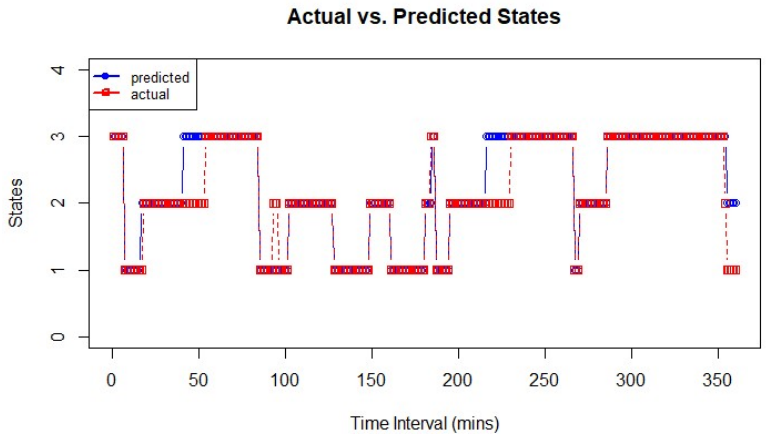


Figure 4.7: Actual vs. Predicted States

4.4.3 LMD-HSMM Implementation

In big data streaming applications, measurements of real-time streaming workloads often indicate a significant amount of workload variability is present over time. The variability is as reflected in the off-peak and peak periods that often characterizes the workload arrival. The exhibited temporal patterns can have significant impact on the resources required by streaming applications, hence, demand resources be scaled up or down accordingly. A major advantage of using the HSMM-based model is in its capability of making predictions for longer periods. The conditional independence in the HSMM-based model is ensured when the process moves from one distinct state to another. In this section, we use the LMD-HSMM to evaluate the resource usage states of big data streaming applications that run for a longer period of time.

4.4.3.1 Training the MD-HSMM

In this model, we also classify the resource prediction process into 3 states. We ensure the initial value of D is sufficiently large enough to cover the maximum duration of any state while being small enough to provide efficient computation, we use $D = 60hrs$. The order of transition A from each state signifies the change in time and the duration of a state D represents the time spent in each state. The observation sequence of each state throughout its duration represents the constraints being modeled, e.g., CPU, memory. The sample variables used as inputs to the MD-HSMM are as listed in Tbl. 4.4. We set the initial values for the initial state probabilities π (0.33 0.33, 0.34), the state transition probabilities a_{ij}, the state duration distributions $P_i(d)$ and the observation $b_i(O^r)$. A Gaussian mixture

model is used to describe the observation, where (μ_1, μ_2) indicate the mean values of the observation dimensions and (σ_1^2, σ_2^2) represent the variances.

Table 4.4: Probabilities Matrix

Transition States (A)		
0.00	0.50	0.500
0.50	0.00	0.500
0.47	0.53	0.000
Observation (μ_1) - cpu (%)		
16.607	25.672	38.355
Observation (μ_2) - mem(b) $* e + 08$		
6.728	6.806	6.892
Variance (σ_1^2) - cpu (%)		
19.228	64.884	158.598
Variance (σ_2^2) - mem (b) $* e + 08$		
9.709	9.269	8.793
Duration		
Shape	0.732	1.663	1.139
Scale	6.826	3.693	5.709

Given these initial states for the MD-HSMM, we then use the modified Forward-Backward algorithm to re-estimate the model parameters and obtain results. The MD-HSMM training model is as shown in Fig. 4.8 indicating the convergence of the parameters in the MD-HSMM. Working with the log-likelihood makes it more convenient to decrease the value returned by likelihood function. The x-axis shows the training steps and the y-axis represents the likelihood probability of different states. The progression of the states reaches the set error in less than 3 steps. This indicates that an appropriate MD-HSMM model is generated within 3 k iterations.

4.4.3.2 Predicting and Validating the MD-HSMM

Using the observations and the MD-HSMM model, we executed the Viterbi algorithm to determine the states that must have given the observations. After obtaining the parameters of the model, we use a simulation approach to produce our own sequence of observations.

Fig. 4.9 is a snapshot of the CPU usage and memory data. The curve shows the values

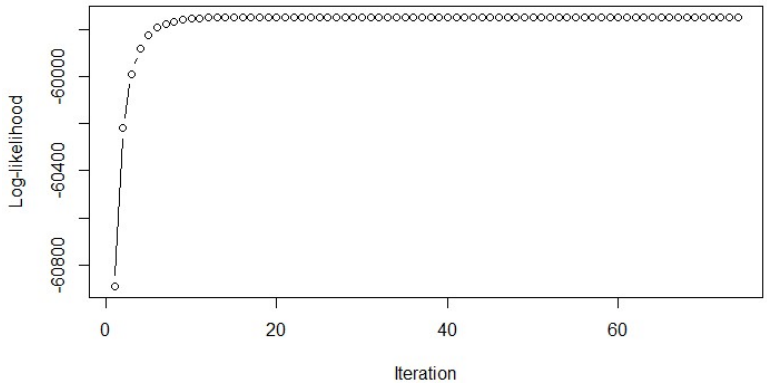


Figure 4.8: Log-likelihood of the MD-HSMM model

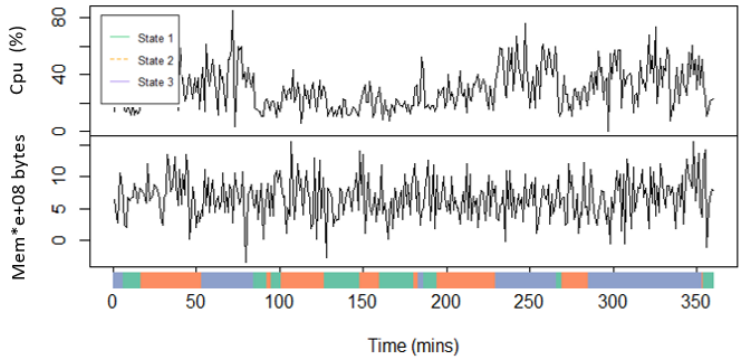


Figure 4.9: Simulated data and states for resource usage observations

from the observation distribution while the horizontal bar indicates the different states. We then compute the means of the model and the prediction data to validate the model. Tbl. 4.5 presents the results of the training and accuracy of prediction.

Table 4.5: Prediction Period Variation & Accuracy of the MD-HSMM

Experiment Period	Prediction Period	RSME (%)	MAPE (%)
24hrs	4hrs	91.2	91.5
24hrs	6hrs	89.6	90.5
24hrs	8hrs	90.8	92.1
48hrs	8hrs	91.6	91.7
48hrs	12hrs	91.1	91.4

Fig. 4.10 shows the predictive accuracy of the model over a 6hr period. Fig. 4.11 depicts the accuracy of the MD-HSMM in predicting the resource usage states of streaming applications, highlighting its consistency in retaining its predictive properties for a longer period of time.

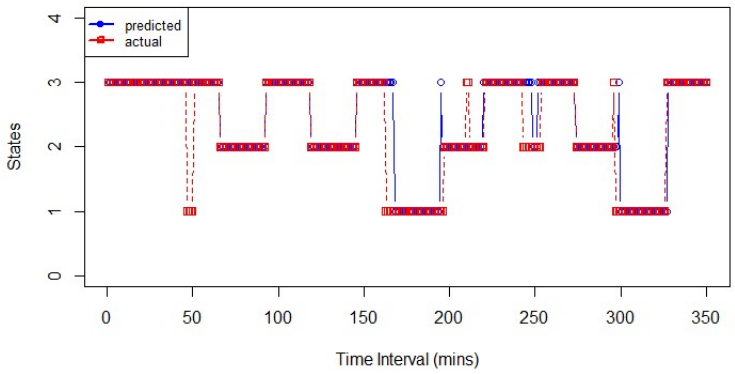


Figure 4.10: Actual vs. Predicted States

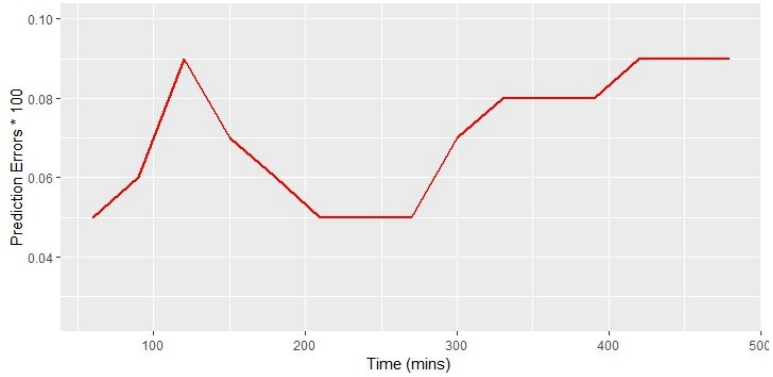


Figure 4.11: Predictive Accuracy of the MD-HSMM

4.5 Summary

In this chapter, we investigated the challenges of scaling resources for big data streaming applications. Based on our findings, we presented a layered multi-dimensional HMM (LMD-HMM) and multi-dimensional HSMM (LMD-HSMM) approaches for resource usage prediction of time-bounded and unbounded big data streaming applications, respectively. Our proposed methodology used multi-dimensional HMM/HSMM at the lower-layer, which are well-trained and the states concatenated at the upper-layer for resource scaling. Experimental evaluation results showed that LMD-HMM has a good potential for predicting the resource usage of time-bounded big data streaming applications. Thus, as long as the lower-layer produce consistent models during training and testing, the LMD-HMM has a better advantage than the single-layer HMM since it is able to perform well under varying streaming environments.

We also proposed the LMD-HSMM framework for predicting the resource usage of streaming applications by explicitly modeling the duration of states. A modified forward-

backward algorithm is used to estimate the parameters of the MD-HSMM in which new variables are defined and the corresponding re-estimation formulae is based on the new variables derived. By incorporating explicit temporal structure into the framework, we are able to predict the resource usage of unbounded big data streaming applications. Evaluation of our proposed frameworks is carried out using developed big data streaming applications running within the Spark streaming environment. Our experimental results show that the HSMM-based framework has a much better performance than the HMM-based model when applied to unbounded applications since it explicitly models the duration of states.

In the next chapter, we will address the resource allocation and pricing challenge for containerized BDSAs running on a heterogeneous cluster of hosts. We leverage game theory to ensure fairness among big data streaming applications when allocating resources by balancing the workload across the clusters.

Chapter 5

CRAM-P: a Container Resource Allocation Mechanism with Dynamic Pricing for BDSAs

Containerization provides a lightweight alternative to the use of virtual machines (VMs) for potentially reducing service cost and improving cloud resource utilization. This chapter addresses the resource allocation problem for CSBs managing multiple competing containerized big data streaming applications (BDSAs) with varying quality of service (QoS) demands running on a heterogeneous cluster of hosts. Also, due to BDSA's demand uncertainty, another fundamental problem for the CSB is how to incorporate a pricing mechanism such that the overall long-term profit is maximized. We hereby propose a container resource allocation mechanism with dynamic pricing (CRAM-P) to address the resource allocation challenge with the objectives of maximizing: i) the payoffs of the containerized BDSAs while satisfying their QoS, and ii) the CSB's surplus. A summary of the contributions in this chapter has been published in [37] and submitted for publication in [38].

5.1 Introduction

Big data streaming applications (BDSAs) that process high-volume of data streams in real-time are pushing the limits of traditional data processing infrastructures. These applications are widely used in several domains (e.g., IoT and healthcare) to analyze data from various sources (e.g., sensors, social media). Due to the challenging requirements of the BDSAs, cloud infrastructures are increasingly leveraged for processing big data streams to improve performance and reduce costs [185]. In addition, the increased adoption of the cloud is fueled by its unique promise of elasticity, flexibility, and pay-as-you-go pricing among others [58]. The cloud platform provides the opportunity to rent distributed resources, typically, virtual machines (VMs) or containers, located at geographically dispersed datacenters (DCs). Through hardware virtualization, BDSAs can be isolated to obtain operational flexibility [186, 187]. Unfortunately, this approach incurs huge resource overhead and poor performance [84] due to dedicating systems' resources to run the VM hosting environment. A simple alternative introduced recently for deploying and managing BDSAs is containerization (i.e., operating system (OS) virtualization).

Typically, containerization allows BDSAs to share the underlying resources of the host as well as build an exact image of the environment needed to run applications. In addition, the use of containers makes the BDSA deployment process easier, lowers time and cost, enhances continuous integration of software components, and increases application scalability [83, 84]. However, coexisting containerized BDSAs on shared computing resources can lead to resource contention, and hence, slower execution times and increased service costs [139]. Since containers running on a shared cluster of hosts do not have resource constraints by default [142], this can introduce non-cooperative behaviours among containerized BDSAs, i.e., selfish containers can monopolize the resources thereby negatively impacting the performance of others. Generally, *Cgroup*, a Linux Kernel feature that limits system resources, can be used to prevent application interference in such multi-tenanted environments [188]. Unfortunately, Cgroup fail in resource-pressured environments as BDSAs compete for free OS resources.

One critical problem in such a heterogeneous environment that needs to be addressed is how to allocate resources efficiently among competing containerized streaming applications to reduce delay and cost. Resource allocation for stream processing applications is complicated by a number of factors: i) they are typically long running jobs, spanning days/weeks, ii) they can be faced with spikes due to unpredictable and fluctuating load arrival patterns iii) they may be of a specific resource-intensive type, e.g., CPU, iv) many

run under heterogeneous host conditions, and, v) trade-offs can exist among competing objectives such as response time and resource cost. To address these challenges, users often rely on container parallelism tuning to alleviate delays. Since this approach is only suitable in a resource-sufficient environment, newer approaches are required to meet the challenges of BDSAs in resource-constrained environments.

In the cloud market, there are three major players involved—cloud service providers (CSPs), cloud service brokers (CSBs) and cloud service consumers (CSCs)—and pricing is a major and complex decision area in this market. Naturally, container resources are priced differently due to the different resource costs obtained by the CSB from the CSPs. Given the CSB's limited resources, it is crucial to determine how the container resources should be utilized while maintaining the QoS of the BDSAs. Congestion can occur when BDSAs act independently and selfishly to maximize their level of satisfaction with no consideration for others. Therefore, the CSB requires a mechanism that provides incentives for BDSAs to behave in ways that improve the overall resource utilization and performance. Under uncertain demand circumstances, the key questions with significant economic implications then are: What strategy can each BDSA use to select the container-cluster for processing, in order to maximize their utility? Given each BDSA's strategy, how can the CSB dynamically modify the prices of the container resources to maximize the overall profit? We address these questions by jointly investigating the optimal resource allocation and pricing policy for a CSB facing uncertain BDSA demands.

Although, a plethora body of research work [144–146] exists on solutions for dynamically managing resource allocation in the cloud, these approaches are addressed from the CSP's viewpoint and focus overwhelmingly on VM allocation challenges. In this chapter, we propose CRAM-P, a mechanism that can adapt to stream processing conditions for optimal resource allocation with pricing. In particular, we focus on a competitive setting involving containerized BDSAs utilizing a heterogeneous cluster of hosts and consider maximizing their payoffs and the overall CSB's surplus. The proposed model allows the BDSAs to efficiently utilize the procured cloud resources from the CSBs. To this end, our main contributions are as follows:

- We tackle the container resource allocation problem and present a game theoretical approach to address the aforementioned challenges, a resource allocation strategy and an algorithm that can adapt to stream processing conditions are proposed.
- We formulate the resource allocation problem using queueing theory [189], which takes each streaming applications arrival rates, and the waiting times and service

rates of each container-cluster host into account to guarantee optimality as well as maximize the payoffs of the streaming applications.

- We also include a pricing mechanism in our model to adjust the BDSAs' demands for efficient resource utilization. The proposed mechanism makes use of the workload volume price to control the requests of the BDSAs in order to maximize the CSB's surplus.

From theoretical analysis, we obtain the optimal Nash Equilibrium state where no player (i.e., BDSA) can further improve its performance without impairing others. Experimental results demonstrate the effectiveness of our approach – which attempts to equally satisfy each containerized streaming application's request as compared to existing techniques that may treat some applications unfairly. The rest of the chapter is organized as follows: Section 5.2 presents existing work on resource allocation for stream processing applications. Section 5.3 introduces the system model, a formalization of the problem and the CRAM-P mechanism as a non-cooperative game along with the pricing method. Finally, a validation of our proposed solution is considered in Section 5.4 followed by conclusions and future work in Section 5.5.

5.2 Related Work

Recently, researchers [144,190] have been dealing with the issue of cloud resource allocation in several manners, using either the static or dynamic approaches. The static allocation approach assigns fixed resources, therefore, users need to be aware of the resource requirements of each application. This can lead to resource starvation in streaming applications as streaming workloads change. Studies on the dynamic approaches involve applying the SLA-based, utility-based, market-based, or priority-based custom policies [145, 146, 191–193]. Most of these approaches are focused on the hypervisor-based VM deployment on physical machines whereby the VMM allocate resources according to observed workload or changes in the environment. Only a few studies have been carried out on how to allocate resources in container-based deployments. Moreover, they mostly make use of reactive approaches. Popular strategies for deploying containers on hosts, e.g., Docker Swarm use the bin-packing, spread or round-robin strategies [194]. These strategies attempt to conduct resource fairness but do not take into consideration the multiple objectives of competing containerized streaming applications.

There are a number of researches on stream processing, which mainly focus on operator-based systems. The authors in [181] propose a resource scheduler which dynamically allocates processors to operators to ensure real-time response. They model the relationship between resources and response time and use optimization models to get processor allocation strategy. The authors in [195] proposed an adaptive approach for provisioning VMs for stream processing system based on input rates but latency guarantees are ignored. In addition, the authors in [32] introduced ChronoStream, which provides vertical and horizontal elasticity. The authors in [31] proposed an elastic batched stream processing system based on Spark Streaming, which transparently adjusts available resource to handle workload changes and uneven distribution in container cloud. However, there are few researches on resource management for containerized BDSAs.

We review some efforts that have been made in this area. In the literature, only few works specifically deal with the resource allocation problem for big data streaming applications running on containers. Nardelli et al. [194] presented a formulation to address the elastic provisioning of virtual machines for container deployment (EVCD) as an Integer Linear Programming problem, which takes explicitly into account the heterogeneity of container requirements and virtual machine resources. EVCD determines the container deployment on virtual machines while optimizing Quality of Service (QoS) metrics. We address the resource allocation problem in a non-cooperative environment with the goal of enforcing cooperation among containers.

Piraghaj et al. [196] investigated the problem of inefficient utilization of data center infrastructure resulting from over-estimation of required resources at the VM level. They proposed the container as a service (CaaS) cloud service model and presented a technique for finding efficient virtual machine sizes for hosting containers considering their actual resource usage instead of users estimated amounts. With lack of intelligent framework resource scheduling in the cloud for managing resources, Awada and Barker [197] proposed a cloud-based Container Management System (CMS) framework for deployment, scalability and resource efficiency in the cloud. The CMS provides an approach for optimizing containerized applications in multiple cloud region container-instance clusters, taking into consideration the heterogeneous requirements of the containerized applications. This work does not address multiple QoS objectives of big data streaming applications in a competitive environment.

Game theory has specifically been explored in the past for resource management and load distribution in the cloud. In order to maximize cloud provider profit while satisfying client requests, Srinivasa et al. [156] proposed a Min-Max game approach. They utilized a

new factor in the game called utility factor which considers the time and budget constraint of every user and resources are provided for tasks having the highest utility for corresponding resource. Mao et al. [157] proposed a congestion game whereby services are considered as players with the strategy of selecting a subset of resources to maximize their payoffs, which is the sum of the payoffs received from each selected resource. However, none of these approaches were proposed to address containerized big data streaming applications. We consider the container resource allocation problem among competing big data streaming applications, which we model as a Nash Equilibrium problem.

5.3 System Model

In this section, we describe the resource allocation problem and present a utility function for the containerized BDSAs with consideration for their QoS. Then, we give a detailed description of the pricing model for profit maximization and efficient resource utilization.

5.3.1 Problem Statement

We study the container resource allocation and pricing problem in which a set of I, where $\mathcal{I} = \{1, 2, .., i, .., I\}$, heterogeneous containerized streaming application agents controlled by a CSB seek to optimize their workload allocation decision to k container-cluster hosts, $k = 1, ..., K$, as a non-cooperative game. In our scheme as shown in Fig. 5.1, we associate a waiting time goal and service cost with each streaming application. Due to the delay-sensitive characteristics of the streaming applications, the waiting time is commonly used as a critical quality of service (QoS) metric to measure the delay before processing each application's request.

We assume every streaming application can access multiple container-cluster hosts in order to enhance its performance. The amount of tuple sequence arriving from the event sources for each application's agent indicates the workload of the streaming applications. Suppose the workload arrival for each application's agent from the event sources follows a Poisson process. Each streaming application agent sends their information to the CSB that decides the fraction of workload to be sent to each container-cluster host. Let γ_i be the average arrival of workload to the CSB for application agent i. Each container-cluster host is modeled as an $M/M/c_k$ queueing system (i.e., Poisson arrivals and exponentially distributed service times), where c_k is the capacity of each container-cluster host. We define $1/\mu_k$ as the average service time at each container-cluster host k.

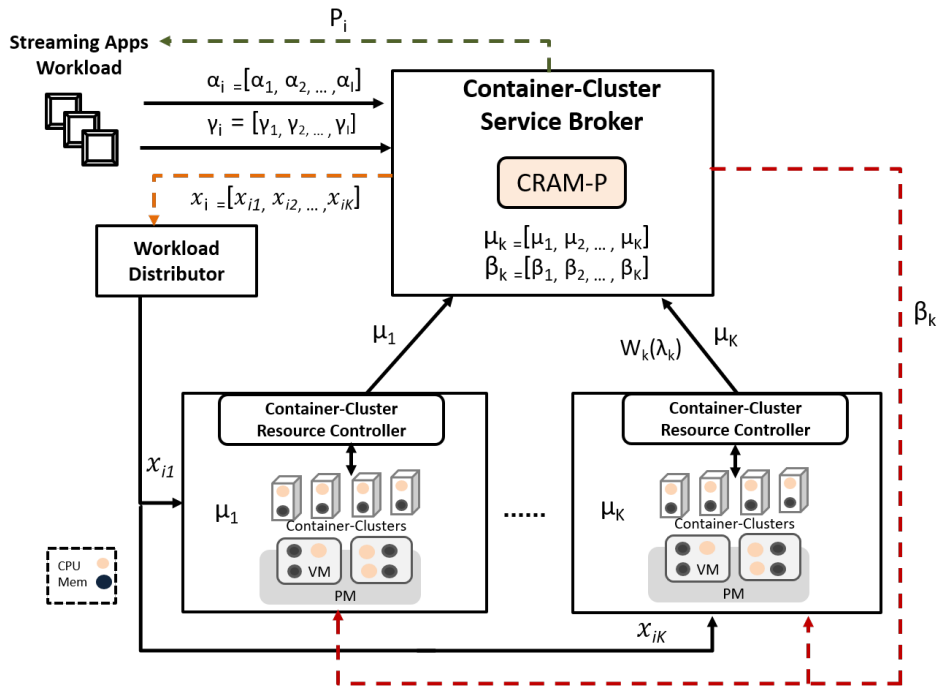


Figure 5.1: Control Structure for CRAM-P.

Given the Poisson input assumptions, this decision can be expressed as a vector $\tilde{x_i} = \{x_{i1}, x_{i2}, ..., x_{iK}\}$, where x_{ik} is the fraction of workload arriving from application agent i to the CSB and sent to the container-cluster host k and the decision from the CSB with resource allocation strategy for application agent i. Then, let the set of actions that each application agent i can take be specified as H_i:

$$H_i = \left\{ \tilde{x_i} \geq 0; \sum_{k=1}^{K} x_{ik} = \gamma_i \right\} \tag{5.1}$$

Hence, the average arrival of workload to the container-cluster host k, denoted by λ_k, is given by $\lambda_k = \sum_{i=1}^{I} x_{ik}$. Each application agent aims to minimize the average waiting time and the service cost when utilizing the container-cluster host. The expected workload from each application agent i sent to the CSB and service at each container-cluster host k is then given by $\sum_{k=1}^{K} \frac{x_{ik}}{\mu_k}$. Also, the average waiting time for each application agent i at container-cluster host k is thus $(\frac{1}{\gamma_i}) \sum_{k=1}^{K} x_{ik} w_k(\lambda_k)$, where $w_k(\lambda_k)$, a function of λ_k, is the expected waiting time for the application at container-cluster host k.

Considering each application agent i's and given the decision of other agents $\tilde{x_j}$, where $j \neq i$, each agent will seek to find $\tilde{x_i}$ such that $(\frac{\alpha_i}{\gamma_i}) \sum_{k=1}^{K} x_{ik} w_k(\lambda_k) + \sum_{k=1}^{K} \beta_k \frac{x_{ik}}{\mu_k}$ is minimized. α_i and β_k are used to represent the weights of the waiting time and service cost

respectively, e.g., for a delay sensitive application, the parameter α_i is always set lower to reflect delay affects the total objective. The weights allow the model to be adaptable to objective preferences in different streaming applications.

We define the strategic form of an i-player container resource allocation game as $G = (\mathcal{I}, (X_i)_{i\in I}, (U_i)_{i\in I})$ where $\mathcal{I} = 1, 2, ..., I$ represents the set of players (i.e., application agents). X_i is the set of pure strategies representing the workload distribution, where $\tilde{x_i}$ represents the strategy of a player. The strategy space X_i is defined by $X_i = \{\tilde{x_i}| \quad \forall i \in I, \tilde{x_i} \geq 0\}$. We denote U_i as the utility obtained for player i, where $U_i(\tilde{x_i}, \mathbf{X}_{-i})$ is the utility of all the player i and $\mathbf{X}_{-i}$ matrix repesents all players except i. The workload distribution $\mathbf{X}_i$ indicates a possible resource allocation status for each application i based on the demand of the host.

A feasible resource allocation strategy $\tilde{x_i}$ must satisfy the following constraints:

- Positivity: $\tilde{x_i} \geq 0$;
- Conservation: $\sum_{k=1}^{K} x_{ik} = \gamma_i$;
- Stability: $\lambda_k < c_k\mu_k$ for an optimal solution as a result of $\sum_{i=1}^{I} \gamma_i < \sum_{k=1}^{K} c_k\mu_k$;

When all the streaming applications satisfy these conditions, resources can be allocated based on the optimal strategy, which leads to maximum payoffs for the streaming applications. We hereby adopt the Nash equilibrium concept to achieve optimality.

Definition 1. *Nash equilibrium is a balanced state with a strategy profile from which no game player can improve its utility by changing its own workload distribution scheme unilaterally, i.e., for every $i \in I$, $U_i(x_i^*, \boldsymbol{X}_{-i}^*) \geq U_i(x_i, \boldsymbol{X}_{-i}^*)$ for all $x_i \in \boldsymbol{X}_i$, then $\boldsymbol{X}_i^* = [x_1^*, x_2^*, ..., x_I^*]$ is considered the result of the Nash equilibrium.*

In our mechanism, each application agent i is regarded as a player controlled by the CSB distributedly optimizing its performance by distributing their workloads. The decision made for application agent i depends on the decisions of other agents. Key notations used in this chapter are as shown in Table 5.1.

5.3.2 CRAM Design as a Non-Cooperative Game

In this section, we formulate the problem as a container resource allocation game and prove the existence of an equilibrium point to the game. At the Nash equilibrium, the stability

Table 5.1: Notations used in CRAM

Variables	Definitions
I	set of streaming applications
K	set of hosts
c_k	capacity of container-cluster host k
w_k	response time of the container-cluster host k
α_i	weight given to the waiting time
β_k	weight given to the service rate
x_{ik}	optimization variable signifying containerized application agent i assigned to container-cluster host k
$U(i)$	utility function for each streaming application agent i
γ_i	workload arrival from streaming application agent i
μ_k	service rate at container-cluster host k

condition should be satisfied because of the fact that the arrival rates does not exceed the service rate. Thus, we only consider the problem with two constraints, i.e., positivity and conservation, which are represented in H_i. Therefore, the optimization problem becomes:

$$\max_{\tilde{x_i}} \quad U_i(\tilde{x_i}, \mathbf{X}_{-i}) \tag{5.2}$$

$$s.t. \quad \tilde{x_i} \in H_i \tag{5.3}$$

$$\begin{aligned} U_i(\tilde{x_i}, \mathbf{X}_{-i}) &= -(\frac{\alpha_i}{\gamma_i}) \sum_{k=1}^{K} x_{ik} w_k(\lambda_k) - \sum_{k=1}^{K} \beta_k \frac{x_{ik}}{\mu_k} \\ &= -(\frac{\alpha_i}{\gamma_i}) \sum_{k=1}^{K} x_{ik} w_k(\sum_{j=1}^{I} x_{jk}) - \sum_{k=1}^{K} \beta_k \frac{x_{ik}}{\mu_k} \end{aligned} \tag{5.4}$$

Eqn. 5.4 shows that the goal is a monotonically decreasing function of the waiting time and a monotonically increasing function of the service rate.

This completes the model formulation and the application agents are now in a gaming situation whereby each agent competes for resources to minimize their payoff. Next, we determine whether an operational equilibrium steady state exists. To establish the exis-

tence of Nash equilibrium in the container resource allocation game, we make use of the existence theorem presented by the authors in [198]. We state the existence theorem as follows:

Theorem 1. *There exist an equilibrium to the agent's container resource allocation game* $G = (\mathcal{I}, (X_i)_{i\in I}, (U_i)_{i\in I})$.

Proof. Nash equilibrium exists when the following conditions are satisfied:

- The streaming application agent i's strategy space given by X_i is a compact convex set in a topological linear space.
- The streaming application agent i's utility function, $U_i(\tilde{x}_i, \mathbf{X}_{-i})$, is concave with respect to its strategy $\tilde{x}_i$.

From Eqn. 5.4,

$$\frac{\partial x_{ik} w_k(\lambda_k)}{\partial x_{ik}} = x_{ik}\frac{\partial w_k(\lambda_k)}{\partial \lambda_k} + w_k(\lambda_k) \tag{5.5}$$

Then,

$$\frac{\partial^2 x_{ik} w_k(\lambda_k)}{\partial x_{ik}\partial x_{il}} = \begin{cases} x_{ik}\frac{\partial^2 w_k(\lambda_k)}{\partial^2 \lambda_k} + \frac{2\partial w_k(\lambda_k)}{\partial \lambda_k} & \text{for} \quad k = l, \\ x_{ik}\frac{\partial^2 w_k(\lambda_k)}{\partial^2 \lambda_k} + \frac{2\partial w_k(\lambda_k)}{\partial \lambda_k} & \text{for} \quad k \neq l, \end{cases}$$

which shows that it is positive and the Hessian matrix of U_i is negative definite. Therefore, the U_i is concave in $\tilde{x}_i$. Since the $U_i(\tilde{x}_i)$ is convex, there is always a strategy file $\tilde{x}_i{}^*$ to make the total cost minimum. As a result, Nash equilibriums always exist.

We then find the Nash equilibrium through theoretical analysis. Applying the Karunsh-Kuhn-Tucker (KKT) conditions [198] and introducing the Lagrange Multiplier ξ_i, we have from Eqn. 5.4

$$\mathcal{L}_i = -(\frac{\alpha_i}{\gamma_i})\sum_{k=1}^{K} x_{ik} w_k(\lambda_k) - \sum_{k=1}^{K}\beta_k \frac{x_{ik}}{\mu_k} + \xi_i \tag{5.6}$$

$$\frac{\partial \mathcal{L}_i}{\partial x_{ik}} = x_{ik}\left(\frac{\alpha_i}{\gamma_i}\right)\frac{\partial w_k(\lambda_k)}{\partial \lambda_k} + \left(\frac{\alpha_i}{\gamma_i}\right) w_k(\lambda_k) + \frac{\beta_k}{\mu_k} + \xi_i = 0 \tag{5.7}$$

The implies that x_{ik} is the optimal solution to the problem if and only if there exists $\xi_i \geq 0$ such that $\frac{\partial \mathcal{L}_i}{\partial x_{ik}} = 0$.

Let

$$a_{ik} = \left(\frac{\alpha_i}{\gamma_i}\right)\frac{\partial w_k(\lambda_k)}{\partial \lambda_k} \quad \text{and} \quad b_{ik} = \left(\frac{\alpha_i}{\gamma_i}\right)w_k(\lambda_k) + \frac{\beta_k}{\mu_k} \tag{5.8}$$

Simplifying, we have

$$x_{ik} = \frac{-\xi_i - b_{ik}}{a_{ik}} \tag{5.9}$$

We recall that $\sum_{k=1}^{K} x_{ik} = \gamma_i$, therefore

$$\sum_{k=1}^{K} x_{ik} = \gamma_i = \sum_{k=1}^{K}\left[\frac{-\xi_i - b_{ik}}{a_{ik}}\right] \tag{5.10}$$

$$\sum_{k=1}^{K}\frac{\xi_i}{a_{ik}} = -\gamma_i - \sum_{k=1}^{K}\frac{b_{ik}}{a_{ik}} \tag{5.11}$$

Which implies

$$\xi_i = -\frac{\gamma_i + \sum_{k=1}^{K}\frac{b_{ik}}{a_{ik}}}{\sum_{k=1}^{K}\frac{1}{a_{ik}}} \tag{5.12}$$

Considering all agents and substituting ξ_i from Eqn. 5.12 in Eqn. 5.9, the optimal resource allocation strategy can be derived as follows:

$$x^*_{ik} = \frac{1}{a_{ik}}\left[\frac{\gamma_i + \sum_{k=1}^{K}\frac{b_{ik}}{a_{ik}}}{\sum_{k=1}^{K}\frac{1}{a_{ik}}} - b_{ik}\right] \tag{5.13}$$

5.3.3 Dynamic Pricing

In this section, we examine the pricing scheme to coordinate the decisions of the BDSA agents. Given the limited resources of the CSBs, it is important to determine how they

will be efficiently utilized while maintaining an acceptable level of QoS. Valuation of the container-clusters by the CSBs (i.e., charging a fee) can provide a decision control mechanism for the BDSA agents. For the CSB, the issue of social optimization arises because the BDSA agents' decisions are based on self-interest, which are not optimal. Incentive-pricing allows the alignment of the objectives of self-interested BDSAs to the goal of the overall system. The question then is how does the CSB price the container-cluster resources to achieve this goal? To derive an incentive-compatible pricing for the $M/M/c_k$ queue, we use the observed performance as input to the pricing formula to maximize the net value of the CSBs' resources.

We recall that the external arrival rate of the BDSA workload is γ_i, which follows a Poisson process. Assuming $W_i(\gamma_i)$ is the average waiting time for a single BDSA agent i in the system and α_i is the cost. The expected waiting cost faced by the BDSA agent is $\alpha_i\gamma_i.W_i(\gamma_i)$. Let P_i denote the service uasage price per BDSA agent i charged by the CSB. Each BDSA agent's expected total cost consists of the cost of waiting and service usage, i.e., $\alpha_i\gamma_i.W_i(\gamma_i) + P_i$. The value that each BDSA agent i derives from using the container-cluster, which may differ, should then be equal to the cost (waiting and price charged) at equilibrium. The value is captured as V_i, a cumulative function, twice-differentiable and concave. An agent chooses to use the container-cluster if the value exceeds the expected total cost, i.e., workload will be added to the container-cluster as long as the value $V'(\gamma_i)$ exceeds the cost. At equilibrium, the demand relationship is given by:

$$V_i'(\gamma_i) = \alpha_i\gamma_i W_i(\gamma_i) + P_i \tag{5.14}$$

We also recall that the waiting time w_k at the container-cluster k from Section 5.3.2 is a function of the effective arrival rate λ_k, i.e., $w_k(\lambda_k)$. This differs from W_i, which is the mean waiting time for a BDSA agent i in the system. They are related through:

$$W_i(\gamma_i, x_i) = \frac{1}{\gamma_i}\sum_{k=1}^{K} x_{ik} w_k(\lambda_k) \tag{5.15}$$

λ_k is the effective arrival-rate for each streaming application to container-cluster k both from external and internal.

The problem of finding the optimal arrival rates to maximize the net value, which is given as $\sum_{i=1}^{I}\left[V_i(\gamma_i) - \alpha_i\gamma_i W_i(\gamma_i)\right]$, can be formulated as:

$$\max_{\gamma_i} NV(\gamma_i) = \sum_{i=1}^{I} \left[V_i(\gamma_i) - \alpha_i \gamma_i W_i(\gamma_i) \right] \tag{5.16}$$

Differentiating Eqn. 5.16 with respect to the arrival rate γ_i, this gives the socially optimal arrival rate as:

$$V_i'(\gamma_i) - \alpha_i W_i(\gamma_i) - \sum_{j=1}^{I} \alpha_j \gamma_j \frac{\partial W_j}{\partial \gamma_i} = 0 \tag{5.17}$$

Eqn. 5.17 shows that the price a BDSA agent has to pay is dependent on the delay inflicted by other BDSA agents utilizing the system. The optimal price for each BDSA i can then be the cost of waiting caused by other agents. It then follows from Eqn. 5.14 and 5.17 that the net value maximizing price is given by:

$$P_i^* = \sum_{j=1}^{I} \alpha_j \gamma_j^* \frac{\partial W_j(x_i)}{\partial \gamma_i} \tag{5.18}$$

P_i^* is the cost, per unit of time, inflicted on the system by increasing the workload allocation. Since, charging the price per BDSA stream will achieve the same efficiency as per container-cluster pricing for each BDSA agent i. Therefore, the optimal price β_k that maximizes the net value of the CSB can be derived thus:

$$\sum_{k=1}^{K} \beta_k \frac{x_{ik}}{\mu_k} = P_i \tag{5.19}$$

To compute the optimal price P_i, the CSB first needs to solve Eqn. 5.17 to find the optimal allocation γ_i and then use Eqn. 5.18 to find the optimal price. From the optimal price, the price per container-cluster β_k can be determined using Eqn. 5.19.

5.3.4 Algorithm Design

Based on the proposed mechanism, we develop an algorithm that periodically adapts to changes in the workload arrival, waiting time and service rates at the container-clusters. We propose the use of a central control mechanism managed by the CSB for implementing the algorithm as shown in Fig. 5.1. The CSB collects information on the service rates and waiting times from the container-cluster hosts and the streaming application agents

submit the arrival rates to the CSB. The CSB uses the information collected to optimize the workload distribution for proper resource management. The arrival rates, waiting time and service rates are determined by monitoring the streaming applications and container-clusters respectively. This data is then sent to the CSB to determine the resource allocation strategy. When the CSB receives all the inputs, it will obtain the optimal strategy by applying Eqn. 5.4. We establish that there is a unique optimal Nash Equilibrium for the resource allocation in Section 5.3.2.

As outlined in Algorithm 1, Step 6 generates the optimal resource allocation strategy. The mechanism considers all the streaming workows running on the container-clusters as a whole and decides the amount of input data to allocate to each streaming application. The key idea is to collect statistics periodically and adjust the amount of utility for each streaming application agent i such that the waiting time and service cost are minimized. The algorithm is activated once per monitoring period. The container-cluster is in an equilibrium state when all the streaming applications reach their maximum utility. If not in an equilibrium state, the mechanism adjusts the workload for each streaming application periodically in each iteration.

Algorithm 1 CRAM Algorithm

Input:
I: Number of streaming applications
μ_k: Service rate of each container-cluster
γ_i: Arrival rate of each streaming application
$w_k(\lambda_k)$: Waiting time at each container-cluster for the arrival
α_i: Weight assigned to the waiting time of each streaming application
β_k: Weight assigned to the service rate at each container-cluster

Output:
X_i: Resource allocation strategy for the streaming applications

1: **procedure** CRAM Algorithm
2: Calculate $a_{ik} = \left(\frac{\alpha_i}{\gamma_i}\right)\frac{\partial w_k(\lambda_k)}{\partial \lambda_k}$
3: Calculate $b_{ik} = \left(\frac{\alpha_i}{\gamma_i}\right) w_k(\lambda_k) + \frac{\beta_k}{\mu_k}$
4: **for all** $i = 1; i < I; i++$ **do**
5: Generate the optimal resource strategy
6: $x^*_{ik} = \frac{1}{a_{ik}}\left[\frac{\gamma_i + \sum_{k=1}^{K} \frac{b_{ik}}{a_{ik}}}{\sum_{k=1}^{K} \frac{1}{a_{ik}}} - b_{ik}\right]$
7: **return** $X_i = (\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_I})$

5.4 Performance Evaluation

In this section, we evaluate the performance of the proposed CRAM approach and compare it with the traditional Round-Robin (RR) scheme, which is the default packing algorithm implemented in Apache Heron used to assign threads to containers. We provide simulation experiments to verify the rationality of our proposed model and examine the possibility of a Nash equilibrium in a two-player game. Our default experimental settings and parameters are given in Table 5.2.

Table 5.2: Container-Cluster Configurations

Parameters	App1	App2
α	1	10
Container-Cluster1	1GB mem/node, 3 CPUs $\beta_1 = 10$	
Container-Cluster2	500MB mem/node, 2 CPUs $\beta_2 = 5$	

5.4.1 Experimental Setup

The platform used for the experiment was setup on a physical machine with 8 cores i7-8550U CPU @ 1.80GHz GHz Intel processors and 16GB RAM. The container-clusters were set up to support Apache Heron streaming engine deployment. Heron deployment make use of Docker as the containerization format for Heron topologies. Following the common configurations of Apache Heron, we set up Heron services - Zookeeper, BookKeeper, Heron Tools (UI and tracker) and Heron API, necessary for the streaming application deployment. Information about the topology are handled by the Zookeeper upon submission of the topology and the BookKeeper handles the topology artifact storage.

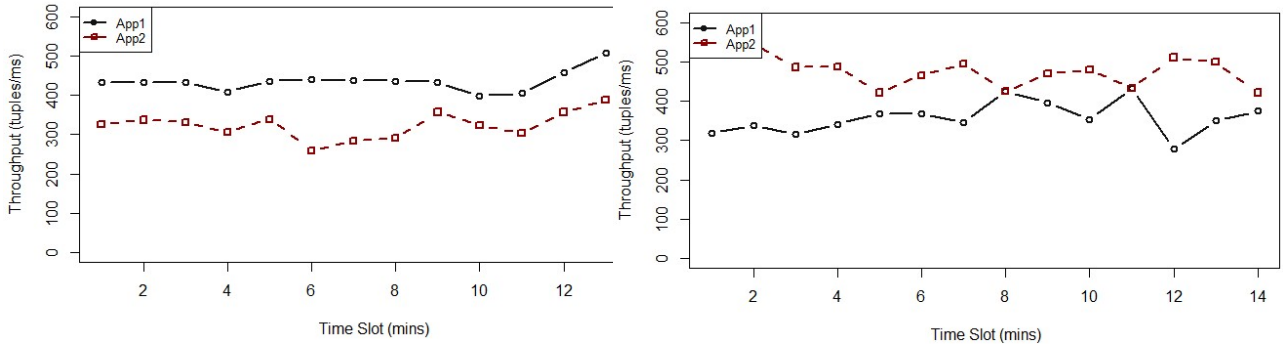
5.4.2 Testing Applications

Two benchmark topologies representative of streaming applications were considered to evaluate the mechanism, namely ExclamationTopology and WordCountTopology. The WordCountTopology consists of a WordSpout that generates a random word stream and

distributes the words to the ConsumerBolt, which then counts the number of times distinct word occurred, while the ExclamationTopology consumes tuples of words and appends an exclamation at the end. The ExclamationTopology and WordCountTopology were used to run the workloads. We implemented our algorithm in Java and varied α and β to find the values at which the Nash equilibrium of the game is achieved. When the resulting streaming applications have the same priority, then $\alpha_1 = \alpha_2$.

5.4.3 Experimental Results

In this experiment, we repeat the scenarios by randomly varying the initial profile of the configuration. We examine the effects of variation in three factors: α, β, and the workload distribution. Each streaming application is deployed to run on the container-clusters, which can be scaled up or down by adding or removing instances. Since each container-cluster has a queue for receiving tuples of data, the arrival rate for each streaming application is the number of tuples that are inserted into the queue and the waiting time is derived from the arrival rate as an estimate of how many milliseconds each tuple sits in the queue before being processed. We set the time slot to a 1 minute interval and run the system for a period of about 20 minutes. We collected the metrics for the waiting and service times for each container-cluster.



a b

Figure 5.2: (a) Streaming App1 with a higher priority (b) Streaming App1 with a lower priority.

Fig. 5.2a and Fig. 5.2b show the throughput of the streaming applications with varying α configurations, which highlight the negotiation between the streaming applications. Fig. 5.2a shows App1 with a higher priority (i.e., $\alpha_1 = 1, \alpha_2 = 10$) and Fig. 5.2b depicts App1 with a lower priority (i.e., $\alpha_1 = 10, \alpha_2 = 1$). In comparison, Fig. 5.3 shows the throughput of the streaming applications using the RR mechanism, whereby App2 takes

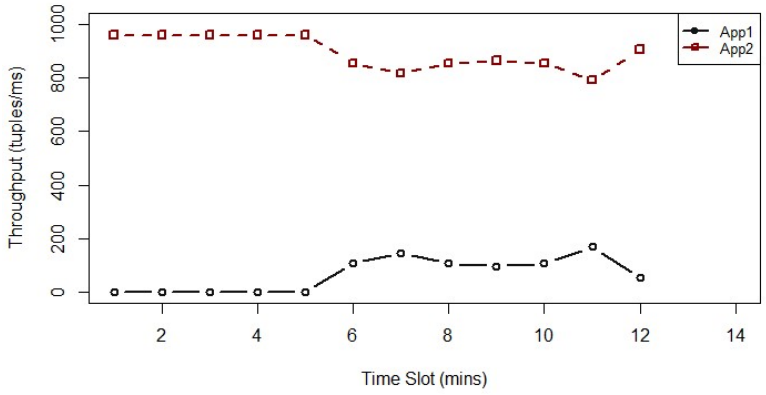


Figure 5.3: Throughput of Streaming Applications using the RR Mechanism

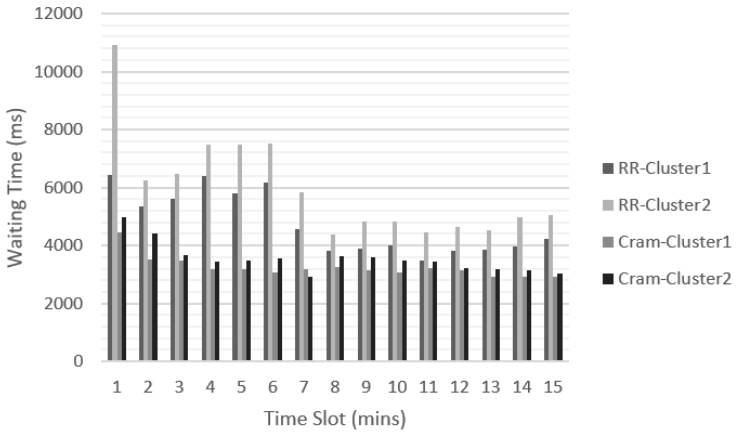


Figure 5.4: Delay at each Container-Cluster for CRAM vs. Round-Robin mechanism.

over the container-cluster resources. Fig. 5.4 represents the resulting delay improvement performance of the proposed CRAM in comparison to the RR mechanism popularly implemented. We found that the waiting time can be reduced by approximately 20% in most cases, which indicates that the proposed model reduces the completion time of the streaming applications and can save cost. This demonstrates the benefits of the proposed resource allocation scheme in the case of selfish workloads sharing the container-clusters.

From our experimental results, we can draw the following conclusions:

- Since the resources allocated to a streaming application to accomplish a certain task is proportional to the workload at the time, our result shows that the proposed strategy is more effective in allocating container resources than the RR mechanism.
- The optimized performance of our mechanism is sensitive to the delay conditions at each container-cluster. Fig. 5.2a and Fig. 5.2b present a more balanced state for the

streaming applications as compared to applications using the RR mechanism shown in Fig. 5.3, which does not give consideration to the priority of individual streaming application. In this case, greedy applications can take over the cluster, leaving little or no resources for the other applications running. These results show that our proposed strategy is more adaptive to delay conditions at the container-cluster than the RR mechanism.

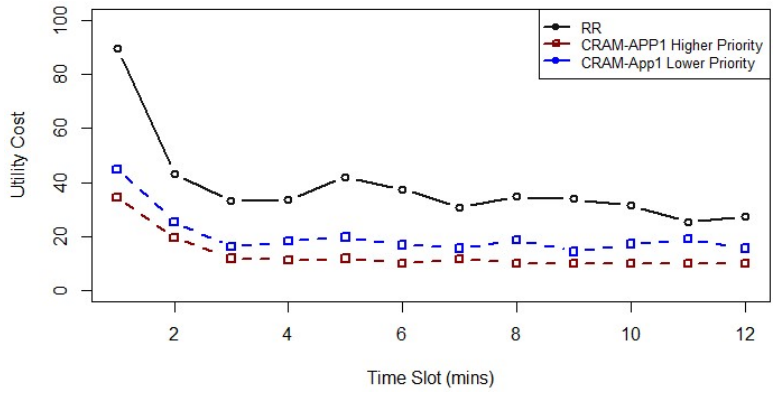


Figure 5.5: Utility Cost of CRAM vs. RR

The utilities of the streaming applications are derived from Eqn. 5.4 with the parameters (e.g., $\alpha_1 = 1$, $\alpha_2 = 10$, $\beta_1 = 10$, $\beta_2 = 5$ and $\gamma_i = 950$ tuples/ms, the default tuple arrival conguration for the streaming application benchmark). Fig. 5.5 represents the utility cost of the streaming applications with varying priorities as compared to the RR mechanism. As shown in the figure, the proposed mechanism maximizes the utility (i.e., minimizes the cost) of the streaming applications at the Nash equilibrium. This indicates that the algorithm fulfils the streaming workload requirements, which achieves fairness as well as optimality.

5.5 Summary

In this chapter, we presented a non-cooperative game to manage the container resource allocation problem for big data stream processing applications. We proposed a dynamic container resource allocation mechanism (CRAM) for assigning big data streaming application workloads to container-cluster hosts due to the resource contention that can occur among heterogeneous stream processing applications in resource-constrained environments. A game theoretical approach was applied to find the optimal allocation such that the payoff for each streaming application is maximized. The payoffs are derived from the objectives —

waiting and service times — specified. For efficient resource utilization, we included a CSB pricing mechanism to compel BDSA to adjust their workload allocation to the container-clusters. We proved the existence of a unique Nash equilibrium and evaluated the game to study the existence thereof using two micro-benchmarks. Experimental results show that our mechanism greatly improves the performance of big data streaming applications.

Chapter 6

Conclusion and Future Work

In this dissertation, we present our approaches for addressing the challenges encountered by CSBs managing multiple BDSAs on a heterogeneous cluster of hosts. We focus on the resource prediction and allocation schemes to handle fluctuating BDSA demands, and provide a framework to automate the processes efficiently. In this chapter, we summarize the value of the proposed mechanisms for the CSB's business in Section 6.1, the contributions of this research in Section 6.2, and suggest various future research directions in Section 6.3.

6.1 Business Value of Proposed Framework

As CSBs utilize widely distributed and dynamic multi-cloud infrastructures, there are needs for tools that provide global view into performance and control of resource capacities. Consequently, efficient cloud resource elasticity management and pricing for a CSB has become a continuous effort that provides more than a myopic view into the resource capacity and performance required to support BDSAs. While several efforts have been made to address the resource elasticity management challenges plaguing CSBs, our research contributions step up efforts in this area to provide efficient resource scaling and allocation mechanisms for CSBs, and hence, save cost, improve resource utilization, ensure fairness, and enhance

performance while achieving QoS for BDSAs. In addition, streaming application failures that can occur due to lack of resources can be avoided. We hereby summarize the business values of utilizing our proposed framework as follows:

- *Visibility and Control:* The proposed framework will provide a real-time unified view of resources being utilized across multiple cloud platforms, i.e., provide the visibility needed to deploy new resources to address BDSA needs, unexpected workload spikes or even to control costs. By this, CSBs are able to minimize redundancies, optimize resource utilization and gain control of the cloud resource consumption.

- *Agility:* Agility refers to the ability of the CSB's system to rapidly respond to workload demands. Agility is achieved when manual processes are eliminated from the CSBs' resource management processes. As focused tools for automating resource prediction and allocation, the proposed framework will help make decisions faster and simplify the deployment of cloud resources.

- *Cost Reduction and Greater ROI:* Return on Investment (ROI) is the most widely used measure of success in business. It is the proportionate increase in the value of an investment over a period of time. Factors driving ROI include improved productivity (i.e., get more done with less resources), speed (e.g., faster provisioning and deprovisioning), and QoS. Without an effective resource elasticity management framework, CSBs will continue to struggle to maximize their ROI when utilizing cloud resources. When managing cloud costs, the continued expansion of resources has often resulted in higher costs due to inefficiency and waste brought by poor management. Automated policies tied to identifying resource usage can increase the CSB's ability to pinpoint potential cost savings and take action. Therefore, optimizing resource usage through efficient allocation and prediction using intelligent and highly automated frameworks will help to reduce the overall CSB's resource costs.

- *Competitive Advantage:* Innovation, a key to the CSB's success, acts as mediator that enables CSBs to gain and sustain competitive advantage. Most CSBs have limited resources to manage their multitude of streaming applications, therefore, the advantage that they have over their competitors lie in innovating newer frameworks to manage resources for better efficiency. By adopting innovative resource elasticity management mechanisms that can customize the CSPs' resources to the needs of the BDSAs, CSBs are able to maintain competitive advantage. The benefits of gaining competitive advantage can be realized in areas that include making precise business

decisions, and generating repeat business and loyalty. In addition, implementing dynamic frameworks for resource management will take CSBs farther along the learning curve before their competitors catch up.

- *Enhanced CSC Experience:* Satisfied CSCs are unlikely to give up in frustration or switch to competitor due to subpar experiences. Ensuring the QoS of CSCs are met (e.g., reduced latency for BDSAs) by providing adequate resources through the use of dynamic frameworks can also be advantageous for CSBs.

6.2 Research Contributions

As a first step, we conducted a study of the literature on big data, cloud computing, stream processing and technologies used and identified the major challenges faced to elastically manage resources in the cloud for big data processing. Particularly, we aimed at addressing two major questions; how can cloud resource utilization be predicted for BDSAs running on a heterogeneous cluster of hosts?, and how can resources be optimally allocated to BDSAs in a dynamic manner to meet the QoS of each containerized streaming application while maximizing the CSB's surplus? Based on the progress made thus far in this field, a novel cloud elasticity resource management and pricing framework was proposed to address these questions. We hereby present a summary of our contributions to the field of cloud computing at large.

- **An efficient and proactive approach for managing the auto-scaling of time-bounded BDSAs [35]:** In Chapter 4, we tackled the resource prediction problem from a CSB's point of view such that efficient scaling decisions can be made. We proposed the use of a LMD-HMM for managing time-bounded streaming applications that run for a short period of time. The proposed design provided an approach for modeling each application individually at a lower-layer while the upper-layer models interactions between the groups of streaming applications running on a cluster. The prediction system will enable efficient resource provisioning without prior knowledge of running the applications in the cloud.

- **A novel approach for predicting the resource usage of time-unbounded BDSAs [36]:** We extended the LMD-HMM mechanism in Chapter 4 by proposing a novel LMD-HSMM that can accurately capture the resource demands of unbounded BDSAs. The LMD-HSMM framework will enable the resource usage prediction of

BDSAs by explicitly modeling the duration of states. The main advantage of making use of the LMD-HSMM is that it provides CSBs with predictive capabilities for handling resource scaling actions in the cloud in a timely manner. The derived model can also be beneficial when addressing load balancing, defining scheduling policies, and optimizing resource usage for BDSAs.

- **An automated framework for ensuring fair resource allocation to multiple containerized BDSAs [37]:** Chapter 5 addressed the container resource allocation problem by focusing on workload distribution for optimal resource allocation to meet the real-time demands of competing containerized BDSAs. We introduced a container resource allocation mechanism (CRAM) as a non-cooperative non zero-sum game among BDSA agents intermediated by a CSB. CRAM takes the tuple arrival, waiting times, service rates and resource capacity limits as input and optimally allocates workloads to maximize the payoff of each BDSA. As compared to existing techniques, experimental result shows our mechanism equally satisfies each BDSA's request without affecting the performance of other applications. This is a model-based approach for CSBs to optimize resource allocation decisions of BDSAs.

- **A dynamic pricing mechanism for equilibrium and stability of BDSAs [38]:** In Chapter 5.3.2, we presented a pricing mechanism that induces optimal workload allocation when CSBs face uncertain BDSA demands. In particular, pricing mechanisms are used to deal with individual self-interested BDSAs that make decisions to maximize their own objective with no consideration for other applications, which can result in congestion especially in environments with scarce CSB resources. We contribute to meeting this challenge by proposing a dynamic pricing scheme that uses the system performance as input to the pricing formula for optimal resource allocation, i.e., the arrival rates and waiting times of the BDSAs jointly maximize the expected net value of the CSB.

6.3 Future Directions

Resources and services offered in the cloud have rapidly changed in the last decade with emerging computing architectures (e.g., fog, edge, container, serverless, and software-defined computing), and changing infrastructures (e.g., multi-cloud, cloudlets, and heterogeneous clouds). In addition, many streaming applications make use of heterogeneous resources from multiple CSPs with multiple geographically distributed DCs. Hence, changes

are required to the resource management layers. Many strategies and algorithms have been proposed to address resource provisioning, allocation, scaling and scheduling, among others in the cloud. As a key-goal in future research, the following interesting areas will be explored:

- **Integrate predictive mechanisms for scaling BDSA resources in container-clusters:** Generally, it is challenging to make decisions on the types and scale of resources to provision without knowing the workload behaviour of applications. With containers, estimating workload pattern of BDSAs can be cumbersome using parameters such as the arrival rate, CPU usage, processing rate, I/O behaviours and number of connected users. Kubernetes offers a container reconfiguration feature for resource scaling based on basic CPU usage monitoring. Several other similar rule-based reactive mechanisms are used in the cloud, e.g., Amazon Web Services (AWS) Auto-scaler, but predictive mechanisms to enable the reconfiguration of containerized BDSAs in the cloud are yet to be unveiled. Therefore, state-of-art predictive models are required to determine the workload of multiple containerized BDSAs deployed in the cloud.

- **Deal with resource provisioning and scheduling for containerized BDSAs in the cloud:** The framework discussed in Chapter 3.2 incorporates a cloud resource provisioner. As future work, a resource provisioning and scheduling mechanism for CSBs can be develop as there is yet to be a standard large-scale, performance-optimized scheduling platform for managing an ecosystem of containerized BDSAs. The elastic scheduling of containerized BDSA is a complex research problem due to several runtime uncertainties — since BDSAs have dataflow dependencies, challenges exist when dealing with heterogeneous configurations of BDSAs and cloud datacenter resources, driven by heterogeneous QoS requirements and constraints. Despite the clear technological advances in container technologies, resource provisioning and scheduling can be challenging.

- **Implement predictive schedulers for BDSAs in geo-distributed DCs:** The volume and velocity of data generated today necessitates newer and innovative methods to help manage and process the data streams in a timely manner. Scheduling in the cloud for processing large amounts of data streams poses significant challenges, leading to longer processing times, waiting times and costs. This problem becomes even more complicated when scheduling to multiple geo-distributed datacenters. Most SPEs are equipped with a basic default scheduler that evenly distributes

the execution of topology components on the available nodes using a round-robin strategy. Unfortunately, this approach impacts heavily on the stream processing latency. Hence, there is a need to implement a predictive scheduler for improved response times while minimizing resource cost. In this case, such mechanisms should be able to predict the average tuple processing time of BDSAs to derive the scheduling solution. Existing researches proposed offline and online schedulers for minimizing inter-node traffic while preventing overload, however, the time lag before the scheduler kicks-in can lead to additional delay in processing BDSAs.

- **Application of complex ML approaches, e.g, deep learning:** In recent years, deep learning, a radically different approach to artificial intelligence, has produced major breakthroughs for complex tasks. Deep learning is a branch of machine learning algorithms based on learning multiple levels of representation. The vast majority of deep learning algorithms are built upon artificial neural networks (ANN) framework. Considering the problem of predicting the resource needs of multiple streaming applications in federated DCs, given the number of input variables presented, can be complex. Since resource management problems in federated clouds often manifest as difficult online decision-making tasks, application of simple traditional approaches, e.g., logistic regression, may not be feasible as the number of interactions grow. In this case, we will determine if more complex machine learning approaches, e.g., deep learning, can be utilized. As an important technology for most industries today, future studies will focus on the application of deep learning approaches to deal with the complex input-output mappings for BDSA cloud resource prediction.

Bibliography

[1] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of big data on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, 2015.

[2] Marcos D Assuncao, Rodrigo N Calheiros, Silvia Bianchi, Marco AS Netto, and Rajkumar Buyya. Big data computing and clouds: challenges, solutions, and future directions. *arXiv preprint arXiv:1312.4722*, pages 1–39, 2013.

[3] Md Faizul Bari, Raouf Boutaba, Rafael Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Md Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data center network virtualization: A survey. *IEEE Communications Surveys & Tutorials*, 15(2):909–928, 2013.

[4] Marcos Dias de Assuncao, Alexandre da Silva Veith, and Rajkumar Buyya. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications*, 103:1–17, 2018.

[5] Atsushi Ishii and Toyotaro Suzumura. Elastic stream computing with clouds. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD)*, pages 195–202. IEEE, 2011.

[6] Jan Sipke van der Veen, Bram van der Waaij, Elena Lazovik, Wilco Wijbrandi, and Robert J Meijer. Dynamically scaling apache storm for the analysis of streaming data. In *Proceedings of the 2015 IEEE First International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 154–161. IEEE, 2015.

[7] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 423–438. ACM, 2013.

[8] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 239–250. ACM, 2015.

[9] Guilherme Galante and Luis Carlos E de Bona. A survey on cloud computing elasticity. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, pages 263–270. IEEE Computer Society, 2012.

[10] Changqing Ji, Yu Li, Wenming Qiu, Uchechukwu Awada, and Keqiu Li. Big data processing in cloud computing environments. In *Pervasive Systems, Algorithms and Networks (ISPAN), 2012 12th International Symposium on*, pages 17–23. IEEE, 2012.

[11] Rajiv Ranjan. Streaming big data processing in datacenter clouds. *IEEE Cloud Computing*, 1(1):78–83, 2014.

[12] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[13] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.

[14] Apache Spark. Spark streaming programming guide. *Acessado em*, 4(02):2017, 2014.

[15] Fatos Xhafa, Victor Naranjo, and Santi Caballé. Processing and analytics of big data streams with yahoo! s4. In *Proceedings of the 2015 IEEE 29th International Conference on Advanced Information Networking and Applications (AINA)*, pages 263–270. IEEE, 2015.

[16] M Usha, J Akilandeswari, and AS Syed Fiaz. An efficient qos framework for cloud brokerage services. In *Proceedings of the 2012 International Symposium on Cloud and Services Computing (ISCOS)*, pages 76–79. IEEE, 2012.

[17] David S Linthicum. Are cloud service brokers (csbs) still relevant? *IEEE Cloud Computing*, 4(4):18–21, 2017.

[18] Mateusz Guzek, Alicja Gniewek, Pascal Bouvry, Jedrzej Musial, and Jacek Blazewicz. Cloud brokering: Current practices and upcoming challenges. *IEEE Cloud Computing*, 2(2):40–47, 2015.

[19] Gaetano F Anastasi, Emanuele Carlini, Massimo Coppola, and Patrizio Dazzi. Qbrokage: A genetic approach for qos cloud brokering. In *Proceedings of the 2014 IEEE 7th International Conference on Cloud Computing (CLOUD)*, pages 304–311. IEEE, 2014.

[20] Athanasios Naskos, Emmanouela Stachtiari, Anastasios Gounaris, Panagiotis Katsaros, Dimitrios Tsoumakos, Ioannis Konstantinou, and Spyros Sioutas. Cloud elasticity using probabilistic model checking. *arXiv preprint arXiv:1405.4699*, 2014.

[21] Jack Li, Calton Pu, Yuan Chen, Daniel Gmach, and Dejan Milojicic. Enabling elastic stream processing in shared clusters. In *Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 108–115. IEEE, 2016.

[22] RS Shariffdeen, DTSP Munasinghe, HS Bhathiya, UKJU Bandara, and HMN Dilum Bandara. Adaptive workload prediction for proactive auto scaling in paas systems. In *Proceedings of the 2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, pages 22–29. IEEE, 2016.

[23] Valeria Cardellini, Emiliano Casalicchio, Francesco Lo Presti, and Luca Silvestri. Sla-aware resource management for application service providers in the cloud. In *Proceedings of the 2011 First International Symposium on Network Cloud Computing and Applications (NCCA)*, pages 20–27. IEEE, 2011.

[24] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[25] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. Nist cloud computing reference architecture. *NIST special publication*, 500(2011):1–28, 2011.

[26] Han Zhao, Miao Pan, Xinxin Liu, Xiaolin Li, and Yuguang Fang. Exploring fine-grained resource rental planning in cloud computing. *IEEE Transactions on Cloud Computing*, 3(3):304–317, 2015.

[27] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Abdullah Yousafzai, and Feng Xia. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of network and computer applications*, 52:11–25, 2015.

[28] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Feng Xia, and Sajjad A Madani. Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues. *The Journal of Supercomputing*, 71(7):2473–2515, 2015.

[29] Poonam Singh, Maitreyee Dutta, and Naveen Aggarwal. A review of task scheduling based on meta-heuristics approach in cloud computing. *Knowledge and Information Systems*, 52(1):1–51, 2017.

[30] Luiz F Bittencourt, Alfredo Goldman, Edmundo RM Madeira, Nelson LS da Fonseca, and Rizos Sakellariou. Scheduling in distributed systems: A cloud computing perspective. *Computer Science Review*, 30:31–54, 2018.

[31] Song Wu, Xingjun Wang, Hai Jin, and Haibao Chen. Elastic resource provisioning for batched stream processing system in container cloud. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, pages 411–426. Springer, 2017.

[32] Yingjun Wu and Kian-Lee Tan. Chronostream: Elastic stateful stream computation in the cloud. In *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering (ICDE)*, pages 723–734. IEEE, 2015.

[33] R Hevner Von Alan, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[34] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[35] Olubisi Runsewe and Nancy Samaan. Cloud resource scaling for big data streaming applications using a layered multi-dimensional hidden markov model. In *Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 848–857. IEEE, 2017.

[36] Olubisi Runsewe and Nancy Samaan. Cloud resource scaling for time-bounded and unbounded big data streaming applications. *IEEE Transactions on Cloud Computing*, 2018.

[37] Olubisi Runsewe and Nancy Samaan. Cram: a container resource allocation mechanism for big data streaming applications. In *Proceedings of the 2019 19th IEEE/ACM*

International Symposium on Cluster, Cloud and Grid Computing (CCGRID). IEEE, 2019.

[38] Olubisi Runsewe and Nancy Samaan. Cram-p: a container resource allocation mechanism with dynamic pricing for big data streaming applications. In *IEEE Transactions on Cloud Computing*. IEEE.

[39] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile Networks and Applications, Springer*, 19(2):171–209, 2014.

[40] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[41] S. Sakr, A. Liu, D.M. Batista, and M. Alomari. A survey of large scale data management approaches in cloud environments. *Communications Surveys Tutorials, IEEE*, 13(3):311–336, 2011.

[42] Marcos D Assunção, Rodrigo N Calheiros, Silvia Bianchi, Marco AS Netto, and Rajkumar Buyya. Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79:3–15, 2015.

[43] Katina Michael and Keith W Miller. Big data: New opportunities and new challenges. *Computer, IEEE*, 46(6):22–24, 2013.

[44] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376, 2015.

[45] Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel Rash, et al. Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1071–1080, 2011.

[46] Vivien Marx. Biology: The big challenges of big data. *Nature*, 498(7453):255–260, 2013.

[47] Radu Tudoran, Alexandru Costan, and Gabriel Antoniu. Overflow: multi-site aware big data management for scientific workflows on clouds. *IEEE Transactions on Cloud Computing*, 4(1):76–89, 2015.

[48] Brian Cho and Indranil Gupta. Budget-constrained bulk data transfer via internet and shipping networks. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 71–80, 2011.

[49] Justin J Miller. Graph database applications and concepts with neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, volume 2324, page 36, 2013.

[50] Grisha Weintraub. Dynamo and bigtablereview and comparison. In *Proceedings of the 2014 IEEE 28th Convention of Electrical & Electronics Engineers in Israel (IEEEI)*, pages 1–5. IEEE, 2014.

[51] Artem Chebotko, Andrey Kashlev, and Shiyong Lu. A big data modeling methodology for apache cassandra. In *Proceedings of the 2015 IEEE International Congress on Big Data (BigData Congress)*, pages 238–245. IEEE, 2015.

[52] Mehul Nalin Vora. Hadoop-hbase for large-scale data. In *Proceedings of the 2011 international conference on Computer science and network technology (ICCSNT)*, volume 1, pages 601–605. IEEE, 2011.

[53] Zuhair Khayyat, Ihab F Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. Bigdansing: A system for big data cleansing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1215–1230. ACM, 2015.

[54] Ikbal Taleb, Rachida Dssouli, and Mohamed Adel Serhani. Big data pre-processing: a quality framework. In *Proceedings of the 2015 IEEE International Congress on Big Data (BigData Congress),*, pages 191–198. IEEE, 2015.

[55] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107, 2014.

[56] Alfredo Cuzzocrea, Il-Yeol Song, and Karen C. Davis. Analytics over large-scale multidimensional data: The big data revolution! In *Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP*, DOLAP '11, pages 101–104, 2011.

[57] Deepali Arora and Piyush Malik. Analytics: Key to go from generating big data to deriving business value. In *Proceedings of the 2015 IEEE First International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 446–452. IEEE, 2015.

[58] Michael Hogan, Fang Liu, Annie Sokol, and Jin Tong. Nist cloud computing standards roadmap. *NIST Special Publication*, 35, 2011.

[59] Bassem Wanis, Nancy Samaan, and Ahmed Karmouch. Efficient modeling and demand allocation for differentiated cloud virtual-network as-a service offerings. *IEEE Transactions on Cloud Computing*, 4(4):376–391, 2015.

[60] R. Tudoran, A. Costan, and G. Antoniu. Transfer as a service: Towards a cost-effective model for multi-site cloud data management. In *Reliable Distributed Systems (SRDS), 2014 IEEE 33rd International Symposium on*, pages 51–56, Oct 2014.

[61] Minqi Zhou, Rong Zhang, Dadan Zeng, and Weining Qian. Services in the cloud computing era: A survey. In *Proceedings of the 2010 4th International Universal Communication Symposium (IUCS)*, pages 40–46, 2010.

[62] Zibin Zheng, Jieming Zhu, and Michael R Lyu. Service-generated big data and big data-as-a-service: An overview. In *Proceedings of the 2013 IEEE International Congress on Big Data (BigData Congress)*, pages 403–410, 2013.

[63] Xinhua E, Jing Han, Yasong Wang, and Lianru Liu. Big data-as-a-service: Definition and architecture. In *Proceedings of the 2013 15th IEEE International Conference on Communication Technology (ICCT)*, pages 738–742, Nov 2013.

[64] D. Talia. Clouds for scalable big data analytics. *Computer*, 46(5):98–101, May 2013.

[65] F. Zulkernine, P. Martin, Ying Zou, M. Bauer, F. Gwadry-Sridhar, and A. Aboulnaga. Towards cloud-based analytics-as-a-service (claaas) for big data analytics in the cloud. In *Proceedings of the 2013 IEEE International Congress on Big Data (BigData Congress)*, pages 62–69, June 2013.

[66] K. Slavakis, G.B. Giannakis, and G. Mateos. Modeling and optimization for big data analytics: (statistical) learning tools for our era of data deluge. *Signal Processing Magazine, IEEE*, 31(5):18–31, Sept 2014.

[67] Uthayasankar Sivarajah, Muhammad Mustafa Kamal, Zahir Irani, and Vishanth Weerakkody. Critical analysis of big data challenges and analytical methods. *Journal of Business Research*, 70:263–286, 2017.

[68] Katarina Grolinger, Michael Hayes, Wilson A Higashino, Alexandra L'Heureux, David S Allison, and Miriam AM Capretz. Challenges for mapreduce in big data.

In *Proceedings of the 2014 IEEE World Congress on Services (SERVICES)*, pages 182–189. IEEE, 2014.

[69] Ali Hammadi and Lotfi Mhamdi. A survey on architectures and energy efficiency in data center networks. *Computer Communications*, 40:1–21, 2014.

[70] Christoforos Kachris and Ioannis Tomkos. A survey on optical interconnects for data centers. *Communications Surveys & Tutorials, IEEE*, 14(4):1021–1036, 2012.

[71] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T.S. Eugene Ng, Michael Kozuch, and Michael Ryan. c-Through: part-time optics in data centers. *SIGCOMM Comput. Commun. Rev.*, 41(4):327–338, August 2010.

[72] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a Globally-deployed Software Defined Wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, August 2013.

[73] Nancy Samaan. A novel economic sharing model in a federation of selfish cloud providers. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):12–21, 2014.

[74] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, D. Niyato, and Haiyong Xie. A survey on software-defined networking. *Communications Surveys Tutorials, IEEE*, 17(1):27–51, 2015.

[75] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. Calendaring for wide area networks. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 515–526, 2014.

[76] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM'09.*, pages 44–51. IEEE, 2009.

[77] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39–47, May 2005.

[78] Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. lightweight virtualization: a performance comparison. In *Proceedings of the 2015 IEEE International Conference on Cloud Engineering (IC2E)*, pages 386–393. IEEE, 2015.

[79] Charles Anderson. Docker [software engineering]. *IEEE Software*, 32(3):102–c3, 2015.

[80] Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[81] Víctor Medel, Omer Rana, José Ángel Bañares, and Unai Arronategui. Adaptive application scheduling under interference in kubernetes. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pages 426–427. ACM, 2016.

[82] Ann Mary Joy. Performance comparison between linux containers and virtual machines. In *Proceedings of the 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA)*, pages 342–346. IEEE, 2015.

[83] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. In *Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172. IEEE, 2015.

[84] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy, and YC Tay. Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference*, pages 1–13. ACM, 2016.

[85] Maria Fazio, Antonio Celesti, Rajiv Ranjan, Chang Liu, Lydia Chen, and Massimo Villari. Open issues in scheduling microservices in the cloud. *IEEE Cloud Computing*, 3(5):81–88, 2016.

[86] Vangelis Koukis, Constantinos Venetsanopoulos, and Nectarios Koziris. ˜ okeanos: Building a cloud, cluster by cluster. *IEEE internet computing*, 17(3):67–71, 2013.

[87] Amandeep Khurana. Bringing big data systems to the cloud. *IEEE Cloud Computing*, 1(3):72–75, 2014.

[88] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. Elasticity in cloud computing: state of the art and research challenges. *IEEE Transactions on Services Computing*, 11(2):430–447, 2018.

[89] Cairong Yan, Ming Zhu, Xin Yang, Ze Yu, Min Li, Youqun Shi, and Xiaolin Li. Affinity-aware virtual cluster optimization for mapreduce applications. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 63–71, Sept 2012.

[90] Eun-Sung Jung and Rajkumar Kettimuthu. An overview of parallelism exploitation and cross-layer optimization for big data transfers. In *Proc. IEEE IPDPS*, 2013.

[91] Romeo Kienzler, Remy Bruggmann, Anand Ranganathan, and Nesime Tatbul. Stream as you go: The case for incremental data access and processing in the cloud. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops (ICDEW)*, pages 159–166, 2012.

[92] Di Xie, Ning Ding, Y. Charlie Hu, and Ramana Kompella. The only constant is change: Incorporating time-varying network reservations in data centers. *SIGCOMM Comput. Commun. Rev.*, 42(4):199–210, August 2012.

[93] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. Inter-datacenter bulk transfers with netstitcher. *ACM SIGCOMM Computer Communication Review*, 41(4):74–85, 2011.

[94] Nikolaos Laoutaris, Georgios Smaragdakis, Pablo Rodriguez, and Ravi Sundaram. Delay tolerant bulk data transfers on the Internet. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 229–238, 2009.

[95] B. Cho and I. Gupta. New algorithms for planning bulk transfer via internet and shipping networks. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems (ICDCS)*, pages 305–314, June 2010.

[96] R.A. Gorcitz, Y. Jarma, P. Spathis, M. Dias de Amorim, R. Wakikawa, J. Whitbeck, V. Conan, and S. Fdida. Vehicular carriers for big data transfers. In *Proceedings of the 2012 IEEE Vehicular Networking Conference (VNC)*, pages 109–114, Nov 2012.

[97] Phlip Shilane, Mark Huang, Grant Wallace, and Windsor Hsu. Wan-optimized replication of backup datasets using stream-informed delta compression. *ACM Transactions on Storage (TOS)*, 8(4):13, 2012.

[98] Zhenghua Xue, Geng Shen, Jianhui Li, Qian Xu, Yang Zhang, and Jing Shao. Compression-aware i/o performance analysis for big data clustering. In *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, BigMine '12, pages 45–52, 2012.

[99] Yinjin Fu, Hong Jiang, Nong Xiao, Lei Tian, Fang Liu, and Lei Xu. Application-aware local-global source deduplication for cloud backup services of personal storage. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1155–1165, 2014.

[100] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The globus striped gridftp framework and server. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, Washington, DC, USA, 2005.

[101] Ian Foster. Globus online: Accelerating and democratizing science through cloud-based services. *IEEE Internet Computing*, 15(3):70–73, 2011.

[102] B. et al. Allen. Software as a service for data scientists. *Communications of the ACM*, 55(2):81–88, 2012.

[103] Kyle Chard, Steven Tuecke, and Ian Foster. Efficient and secure transfer, synchronization, and sharing of big data. *Cloud Computing, IEEE*, 1(3):46–55, 2014.

[104] Radu Tudoran, Olivier Nano, Ivo Santos, Alexandru Costan, Hakan Soncu, Luc Bougé, and Gabriel Antoniu. Jetstream: Enabling high performance event streaming across cloud data-centers. In *Proceedings of the 8th ACM International Conference on Distributed Event-based Systems*, pages 23–34, 2014.

[105] Tevfik Kosar, Engin Arslan, Brandon Ross, and Bing Zhang. Storkcloud: Data transfer scheduling and optimization as a service. In *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pages 29–36, 2013.

[106] Hai Ah Nam, Jason Hill, and Suzanne Parete-Koon. The practical obstacles of data transfer: why researchers still love scp. In *Proceedings of the Third International Workshop on Network-Aware Data Management*, 2013.

[107] Rosa Filgueira, Iraklis Klampanos, and Yusuke Tanimura. Fast: flexible automated synchronization transfer. In *Proceedings of the sixth international workshop on Data intensive distributed computing*, pages 47–52, 2014.

[108] M. Aledhari and F. Saeed. Design and implementation of network transfer protocol for big genomic data. In *Proceedings of the 2015 IEEE International Congress on Big Data (BigData Congress)*, pages 281–288, June 2015.

[109] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *SIGCOMM Comput. Commun. Rev.*, 40(4):63–74, August 2010.

[110] T. Zhang, J. Wang, J. Huang, Y. Huang, J. Chen, and Y. Pan. Adaptive-acceleration data center tcp. *IEEE Transactions on Computers*, 64(6):1522–1533, June 2015.

[111] Yunhong Gu and Robert L. Grossman. Udt: Udp-based data transfer for high-speed wide area networks. *Computer Networks, Elsevier*, 51(7):1777 – 1799, 2007.

[112] Brian Tierney, Ezra Kissel, Martin Swany, and Eric Pouyoul. Efficient data transfer protocols for big data. In *Proceedings of the 2012 IEEE 8th International Conference on E-Science (e-Science)*, 2012.

[113] Linquan Zhang, Chuan Wu, Zongpeng Li, Chuanxiong Guo, Minghua Chen, and Francis C. M. Lau. Moving big data to the cloud: An online cost-minimizing approach. *IEEE Journal on Selected areas of Communications*, 31(12):2710–2721, Dec 2013.

[114] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):15–26, aug 2013.

[115] Cong Xu, Jiahai Yang, Hui Yu, Haizhuo Lin, and Hui Zhang. Optimizing the topologies of virtual networks for cloud-based big data processing. In *Proceedings of the 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, pages 189–196, 2014.

[116] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 23–23, 2011.

[117] Vinh The Lam, Sivasankar Radhakrishnan, Rong Pan, Amin Vahdat, and George Varghese. Netshare and stochastic netshare: predictable bandwidth allocation for data centers. *SIGCOMM Comput. Commun. Rev.*, 42(3):5–11, June 2012.

[118] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. *Communications Surveys & Tutorials, IEEE*, 16(1):493–512, 2014.

[119] Guohui Wang, T.S. Eugene Ng, and Anees Shaikh. Programming your network at run-time for big data applications. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 103–108, 2012.

[120] Yuan Feng, Baochun Li, and Bo Li. Postcard: Minimizing costs on inter-datacenter traffic with store-and-forward. In *Proceedings of the 2012 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 43–50, 2012.

[121] Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang. Guaranteeing deadlines for inter-datacenter transfers. In *Proceedings of the Tenth European Conference on Computer Systems*, 2015.

[122] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. MicroTE: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh Conference on emerging Networking Experiments and Technologies*, page 8, 2011.

[123] R. Tudoran, A. Costan, Rui Wang, L. Bouge, and G. Antoniu. Bridging data in the clouds: An environment-aware system for geographically distributed data transfers. In *Proceedings of the 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 92–101, May 2014.

[124] Ahmet Soran, Furkan Mustafa Akdemir, and Murat Yuksel. Parallel routing on multi-core routers for big data transfers. In *Proceedings of the 2013 workshop on Student workhop*, pages 35–38, 2013.

[125] M Nirmala. WAN Optimization Tools, Techniques and Research Issues for Cloud-Based Big Data Analytics. In *Proceedings of the 2014 World Congress on Computing and Communication Technologies (WCCCT)*, pages 280–285, 2014.

[126] Linquan Zhang, Zongpeng Li, Chuan Wu, and Minghua Chen. Online algorithms for uploading deferrable big data to the cloud. In *INFOCOM, 2014 Proceedings IEEE*, pages 2022–2030, April 2014.

[127] M. Marcon, N. Santos, K.P. Gummadi, N. Laoutaris, P. Rodriguez, and A. Vahdat. Netex: Efficient and cost-effective internet bulk content delivery. In *Proceedings of the 2010 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Oct 2010.

[128] Adam Barker, Blesson Varghese, and Long Thai. Cloud services brokerage: A survey and research roadmap. In *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, pages 1029–1032. IEEE, 2015.

[129] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, and Patrick Valduriez. Streamcloud: An elastic and scalable data streaming system. *IEEE Transactions on Parallel and Distributed Systems*, 23(12):2351–2365, 2012.

[130] Daniel J Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: a new

model and architecture for data stream management. *the VLDB Journal*, 12(2):120–139, 2003.

[131] Marco Centenaro, Lorenzo Vangelista, Andrea Zanella, and Michele Zorzi. Long-range communications in unlicensed bands: The rising stars in the iot and smart city scenarios. *IEEE Wireless Communications*, 23(5):60–67, 2016.

[132] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on nosql database. In *Proceedings of the 2011 6th international conference on Pervasive computing and applications (ICPCA)*, pages 363–366. IEEE, 2011.

[133] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, pages 1–7, 2011.

[134] Michael I Gordon, William Thies, and Saman Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. *ACM SIGARCH Computer Architecture News*, 34(5):151–162, 2006.

[135] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[136] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, et al. The design of the borealis stream processing engine. In *Cidr*, volume 5, pages 277–289, 2005.

[137] STREAM Group et al. Stream: The stanford stream data manager. Technical report, Stanford InfoLab, 2003.

[138] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S Yu, and Myungcheol Doo. Spade: the system s declarative stream processing engine. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1123–1134. ACM, 2008.

[139] Kejiang Ye and Yunjie Ji. Performance tuning and modeling for big data applications in docker containers. In *Proceedings of the 2017 International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–6. IEEE, 2017.

[140] Shadi A Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H Campbell. Samza: stateful scalable stream processing at linkedin. *Proceedings of the VLDB Endowment*, 10(12):1634–1645, 2017.

[141] Emanuel Ferreira Coutinho, Flávio Rubens de Carvalho Sousa, Paulo Antonio Leal Rego, Danielo Gonçalves Gomes, and José Neuman de Souza. Elasticity in cloud computing: a survey. *annals of telecommunications-annales des télécommunications*, 70(7-8):289–309, 2015.

[142] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. Autonomic vertical elasticity of docker containers with elasticdocker. In *Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 472–479. IEEE, 2017.

[143] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys (CSUR)*, 51(4):73, 2018.

[144] Lei Wei, Chuan Heng Foh, Bingsheng He, and Jianfei Cai. Towards efficient resource allocation for heterogeneous workloads in iaas clouds. *IEEE Transactions on Cloud Computing*, 2015.

[145] Amin Nezarat and GH Dastghaibifard. Efficient nash equilibrium resource allocation based on game theory mechanism in cloud computing by using auction. *PloS one*, 10(10):e0138424, 2015.

[146] Xiaoming Nan, Yifeng He, and Ling Guan. Towards optimal resource allocation for differentiated multimedia services in cloud computing environment. In *Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 684–688. IEEE, 2014.

[147] Wesam Dawoud, Ibrahim Takouna, and Christoph Meinel. Elastic virtual machine for fine-grained cloud resource provisioning. In *Global Trends in Computing and Communication Systems*, pages 11–25. Springer, 2012.

[148] Soodeh Farokhi, Ewnetu Bayuh Lakew, Cristian Klein, Ivona Brandic, and Erik Elmroth. Coordinating cpu and memory elasticity controllers to meet service response time constraints. In *Proceedings of the 2015 International Conference on Cloud and Autonomic Computing (ICCAC)*, pages 69–80. IEEE, 2015.

[149] Jose Monsalve, Aaron Landwehr, and Michela Taufer. Dynamic cpu resource allocation in containerized cloud environments. In *Proceedings of the 2015 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 535–536. IEEE, 2015.

[150] Priyanka P Kukade and Geetanjali Kale. Auto-scaling of micro-services using containerization. *International Journal of Science and Research (IJSR)*, 4(9):1960–1963, 2015.

[151] Fawaz Paraiso, Stéphanie Challita, Yahya Al-Dhuraibi, and Philippe Merle. Model-driven management of docker containers. In *9th IEEE International Conference on Cloud Computing (CLOUD)*, pages 718–725, 2016.

[152] Chuanqi Kan. Docloud: An elastic cloud platform for web applications based on docker. In *Proceedings of the 2016 18th International Conference on Advanced Communication Technology (ICACT)*, pages 478–483. IEEE, 2016.

[153] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[154] Shun-Zheng Yu. Hidden semi-markov models. *Artificial intelligence*, 174(2):215–243, 2010.

[155] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[156] KG Srinivasa, K Sharath Kumar, U Shashank Kaushik, S Srinidhi, Vignesh Shenvi, and Kushagra Mishra. Game theoretic resource allocation in cloud computing. In *Proceedings of the 2014 Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*, pages 36–42. IEEE, 2014.

[157] Zexiang Mao, Jingqi Yang, Yanlei Shang, Chuanchang Liu, and Junliang Chen. A game theory of cloud service deployment. In *Proceedings of the 2013 IEEE Ninth World Congress on Services (SERVICES)*, pages 436–443. IEEE, 2013.

[158] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[159] Stefanie Leimeister, Markus Böhm, Christoph Riedl, and Helmut Krcmar. The business perspective of cloud computing: Actors, roles and value networks. In *ECIS*, page 56, 2010.

[160] Jing Mei, Kenli Li, Zhao Tong, Qiang Li, and Keqin Li. Profit maximization for cloud brokers in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 30(1):190–203, 2019.

[161] Alba Amato, Beniamino Di Martino, and Salvatore Venticinque. Cloud brokering as a service. In *Proceedings of the 2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 9–16. IEEE, 2013.

[162] Ehsan Mostajeran, Bukhary Ikhwan Ismail, Mohammad Fairus Khalid, and Hong Ong. A survey on sla-based brokering for inter-cloud computing. In *Proceedings of the 2015 Second International Conference on Computing Technology and Information Management (ICCTIM)*, pages 25–31. IEEE, 2015.

[163] Rafael Weingärtner, Gabriel Beims Bräscher, and Carlos Becker Westphall. Cloud resource management: A survey on forecasting and profiling models. *Journal of Network and Computer Applications*, 47:99–106, 2015.

[164] Giuseppe Aceto, Alessio Botta, Walter De Donato, and Antonio Pescapè. Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115, 2013.

[165] Sunilkumar S. Manvi and Gopal Krishna Shyam. Resource management for infrastructure as a service (iaas) in cloud computing: A survey. *Journal of Network and Computer Applications*, 41:424 – 440, 2014.

[166] Mohamed Abu Sharkh, Manar Jammal, Abdallah Shami, and Abdelkader Ouda. Resource allocation in a network-based cloud computing environment: design challenges. *IEEE Communications Magazine*, 51(11):46–52, 2013.

[167] Paul C Brebner. Is your cloud elastic enough?: performance modelling the elasticity of infrastructure as a service (iaas) cloud applications. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 263–266. ACM, 2012.

[168] Anshuman Biswas, Shikharesh Majumdar, Biswajit Nandy, and Ali El-Haraki. A hybrid auto-scaling technique for clouds processing applications with service level agreements. *Journal of Cloud Computing*, 6(1):29, 2017.

[169] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD)*, pages 500–507. IEEE, 2011.

[170] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Proceedings of the 2010 International Conference on Network and Service Management (CNSM)*, pages 9–16. Ieee, 2010.

[171] Jing Bi, Zhiliang Zhu, Ruixiong Tian, and Qingbo Wang. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In *Proceedings of the 2010 IEEE 3rd international conference on Cloud Computing (CLOUD)*, pages 370–377. IEEE, 2010.

[172] Waheed Iqbal, Matthew N Dailey, David Carrera, and Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, 27(6):871–879, 2011.

[173] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 5. ACM, 2011.

[174] Rui Han, Li Guo, Moustafa M Ghanem, and Yike Guo. Lightweight resource scaling for cloud applications. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 644–651. IEEE, 2012.

[175] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing*, 12(4):559–592, 2014.

[176] Francisco Javier Baldan, Sergio Ramirez-Gallego, Christoph Bergmeir, Jose M Benitez-Sanchez, and Francisco Herrera. A forecasting methodology for workload forecasting in cloud systems. *IEEE Transactions on Cloud Computing*, 2016.

[177] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th international conference on Autonomic computing*, pages 137–146. ACM, 2009.

[178] Smita Vijayakumar, Qian Zhu, and Gagan Agrawal. Dynamic resource provisioning for data streaming applications in a cloud environment. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 441–448. IEEE, 2010.

[179] Ali Yadavar Nikravesh, Samuel A Ajila, and Chung-Horng Lung. Cloud resource auto-scaling system based on hidden markov model (hmm). In *Proceedings of the 2014 IEEE International Conference on Semantic Computing (ICSC)*, pages 124–127. IEEE, 2014.

[180] Alok Gautam Kumbhare, Yogesh Simmhan, and Viktor K Prasanna. Plasticc: Predictive look-ahead scheduling for continuous dataflows on clouds. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 344–353. IEEE, 2014.

[181] Tom ZJ Fu, Jianbing Ding, Richard TB Ma, Marianne Winslett, Yin Yang, and Zhenjie Zhang. Drs: dynamic resource scheduling for real-time analytics over fast streams. In *Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems (ICDCS)*, pages 411–420. IEEE, 2015.

[182] Federico Lombardi, Leonardo Aniello, Silvia Bonomi, and Leonardo Querzoni. Elastic symbiotic scaling of operators and resources in stream processing systems. *IEEE Transactions on Parallel and Distributed Systems*, 29(3):572–585, 2018.

[183] Claus Pahl and Brian Lee. Containers and clusters for edge cloud architectures–a technology review. In *Proceedings of the 2015 3rd International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 379–386. IEEE, 2015.

[184] Xiaolin Li, Marc Parizeau, and Réjean Plamondon. Training hidden markov models with multiple observations-a combinatorial method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(4):371–377, 2000.

[185] Xunyun Liu and Rajkumar Buyya. Performance-oriented deployment of streaming applications on cloud. *IEEE Transactions on Big Data*, 2017.

[186] Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia. Virtualization vs containerization to support paas. In *Proceedings of the 2014 IEEE International Conference on Cloud Engineering (IC2E)*, pages 610–614. IEEE, 2014.

[187] Claus Pahl. Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31, 2015.

[188] Asraa Abdulrazak Ali Mardan and Kenji Kono. Containers or hypervisors: Which is better for database consolidation? In *Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 564–571. IEEE, 2016.

[189] John F Shortle, James M Thompson, Donald Gross, and Carl M Harris. *Fundamentals of queueing theory*, volume 399. John Wiley & Sons, 2018.

[190] Gunho Lee, Byung-gon Chun, and Y H Katz. Heterogeneity aware resource allocation and scheduling. In *University of California, Berkeley*. Citeseer, 2011.

[191] Hadi Goudarzi and Massoud Pedram. Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD)*, pages 324–331. IEEE, 2011.

[192] Dorian Minarolli and Bernd Freisleben. Utility-based resource allocation for virtual machines in cloud computing. In *Proceedings of the 2011 IEEE Symposium on Computers and Communications (ISCC)*, pages 410–417. IEEE, 2011.

[193] Mohamed Abu Sharkh, Abdallah Shami, and Abdelkader Ouda. Optimal and suboptimal resource allocation techniques in cloud computing data centers. *Journal of Cloud Computing*, 6(1):6, 2017.

[194] Matteo Nardelli, Christoph Hochreiner, and Stefan Schulte. Elastic provisioning of virtual machines for container deployment. In *Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering Companion*, pages 5–10. ACM, 2017.

[195] Javier Cervino, Evangelia Kalyvianaki, Joaquin Salvachua, and Peter Pietzuch. Adaptive provisioning of stream processing systems in the cloud. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops (ICDEW)*, pages 295–301. IEEE, 2012.

[196] Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N Calheiros, and Rajkumar Buyya. Efficient virtual machine sizing for hosting containers as a service (services 2015). In *Proceedings of the 2015 IEEE World Congress on Services (SERVICES)*, pages 31–38. IEEE, 2015.

[197] Uchechukwu Awada and Adam Barker. Improving resource efficiency of container-instance clusters on clouds. In *Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 929–934. IEEE, 2017.

[198] J Ben Rosen. Existence and uniqueness of equilibrium points for concave n-person games. *Econometrica: Journal of the Econometric Society*, pages 520–534, 1965.