Towards Elastic High Performance Distributed Storage Systems in the Cloud

YING LIU

Licentiate Thesis
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

TRITA-ICT/ECS AVH 15:02
Abstract

With the growing popularity of Internet-based services and the trend of hosting them in the Cloud, more powerful back-end storage systems are needed to support these services. On one hand, the storage system itself should be capable of handling workloads with higher concurrency and intensity. Distributed storage solutions are proposed. They are expected to achieve some of the following properties: scalability, availability, durability, consistency, and partition tolerance. It is difficult or even impossible to have a distributed storage solution that fully satisfies all of these properties. Thus, there are researches that focus on pushing the limit on achieving those properties in the designs of distributed storage solutions satisfying different usage scenarios. On the other hand, the increasing trend of Cloud computing brings new challenges to the design of distributed storage systems. In order to better enjoy the benefits from the pay-as-you-go pricing model in the Cloud, a dynamic resource provisioning middleware, i.e., an elasticity controller, is needed. An elasticity controller is able to help saving the provisioning cost of an application without compromising its performance. The design of an elasticity controller to auto-scale a distributed storage system is not trivial. The challenging issue is the data migration overhead (data consume time and resources to be transferred to added instances or transferred out from removed instances) when scaling the system.

The main goal of this thesis is to improve access latency in distributed storage services hosted in a Cloud environment. The goal is approached from two aspects: First, the dynamicity, in terms of intensity, concurrency and locality, in workloads introduces challenges to have a low latency storage solution. We propose smart middlewares, such as elasticity controllers, to effectively and efficiently achieve low latency with minimum resource provisioning under dynamic workloads. Another aspect that we consider is the efficiency of a storage solution itself. We improve access latency of a storage solution under the requirements of scalability, consistency, availability. We start by summarizing the current state of the research field by surveying representative design solutions and identify trends and open issues.

We proceed by presenting a performance evaluation of a distributed storage system (Swift) in a simulated Cloud platform. The project identifies the feasibility of deploying a distributed storage system in a community Cloud environment, which is different from a traditional data center environment. We start by characterizing environment parameters that influence the performance of Swift. We quantify the performance influences by varying the environment parameters including virtual machines (VMs) capability, network connectivity, and potential VM failures. We conclude that Swift is sensitive to network connectivity among storage instances and using a simple self-healing solution, it is able to survive with VM failures to some extent.

We continue by introducing GlobLease, a distributed storage system that uses leases to achieve consistent and low latency read/write accesses in a global scale. Leases guarantee that a data object is updated in a period of time. Thus, the data with valid lease can be read from any replica consistently. Leases are used to improve the access latency of read/write requests by reducing the communication cost among replicas.

Next, we describe BwMan, a bandwidth manager that arbitrates bandwidth allocation among multiple services sharing the same network infrastructure. BwMan guarantees network bandwidth allocation to specific service based on predefined rules or machine learning models. In the scenario of guarantee latency SLAs of a distributed storage system under dynamic workload, BwMan is able to significantly reduce SLA violations when there are other services sharing the same network.

Finally, we describe ProRenaTa, an elasticity controller for distributed storage systems that combines proactive and reactive control techniques to achieve better SLA commitments and resource utilization. ProRenaTa explicitly considers and models the data migration overhead when scaling a distributed storage system. Furthermore, ProRenaTa takes advantages of the complimentary nature of the proactive and reactive scaling approach and makes them operate coherently. As a result, ProRenaTa yields a better performance modeling and prediction of the system and thus achieving higher resource utilization and less SLA violation.
Acknowledgments

I am deeply thankful to the following people, without the help and support of whom, I would not have managed to complete this thesis:

My primary supervisor Vladimir Vlassov, for his guidance, constant feedback and encouragement throughout this work.

My secondary advisors, Peter Van Roy and Jim Dowling, for their valuable advice and insights.

Ahmad Al-Shishtawy, Jim Dowling and Seif Haridi, for reviewing my thesis proposal and for suggesting several improvements.

Sarunas Girdzijauskas for agreeing to serve as the internal reviewer of this thesis. His experience and valuable feedback significantly helped in improving the final version of this work.

Marios D. Dikaiakos for agreeing to serve as the thesis opponent.

My co-authors, Vamis Xhagjika, Xiaxi Li, Navaneeth Rameshan, Enric Monte and Leandro Navarro. Working with them has been a great pleasure and indispensable learning experience.

The anonymous reviewers of the accepted papers included in this thesis, for providing observant and constructive feedback.

My emjd-dc colleagues and colleagues at KTH and UCL, especially Manuel Bravo, Zhongmiao Li, Navaneeth Rameshan, Hooman Peiro Sajjad, Paris Carbone, Kamal Hakimzadeh, Anis Nasir, Vamis Xhagjika and Ruma Paul for always being available to discuss ideas, for being supportive and for the good times we spent hanging out together.

Vasiliki Kalavri, for sharing her experiences with me throughout her PhD studies, for cheering me up and for always inviting me to social activities.

My family, for their continuous support, their faith in me, their patience and immeasurable understanding.

Stockholm, 31 March 2015

Ying Liu
## Contents

### List of Figures

<table>
<thead>
<tr>
<th>I Thesis Overview</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>3</td>
</tr>
<tr>
<td>1.1 Research Objectives</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Research Methodology</td>
<td>5</td>
</tr>
<tr>
<td>1.2.1 Design of Efficient Storage Solutions</td>
<td>5</td>
</tr>
<tr>
<td>1.2.2 Design of Elasticity Controller</td>
<td>5</td>
</tr>
<tr>
<td>1.2.3 Challenges</td>
<td>6</td>
</tr>
<tr>
<td>1.2.4 Experimental Evaluation</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Thesis Contributions</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Thesis Organization</td>
<td>7</td>
</tr>
<tr>
<td>2 Background</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Cloud Computing</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Distributed Storage System</td>
<td>11</td>
</tr>
<tr>
<td>2.2.1 Structures of Distributed Storage Systems</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2 Data Replication</td>
<td>13</td>
</tr>
<tr>
<td>2.2.3 Data Consistency</td>
<td>15</td>
</tr>
<tr>
<td>2.2.4 OpenStack Swift</td>
<td>16</td>
</tr>
<tr>
<td>2.3 Auto-scaling Techniques</td>
<td>16</td>
</tr>
<tr>
<td>2.3.1 Threshold-based Rules</td>
<td>16</td>
</tr>
<tr>
<td>2.3.2 Reinforcement Learning or Q-learning (RL)</td>
<td>16</td>
</tr>
<tr>
<td>2.3.3 Queuing Theory</td>
<td>17</td>
</tr>
<tr>
<td>2.3.4 Control Theory</td>
<td>17</td>
</tr>
<tr>
<td>2.3.5 Time Series Analysis</td>
<td>18</td>
</tr>
<tr>
<td>3 Performance Optimization for Distributed Storage Systems Hosted in the Cloud</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Optimizations on the Design of Distributed Storage Systems</td>
<td>19</td>
</tr>
<tr>
<td>3.1.1 Summary of Proposed Optimizations</td>
<td>20</td>
</tr>
<tr>
<td>3.2 Optimizations on the Load of Distributed Storage Systems</td>
<td>20</td>
</tr>
</tbody>
</table>
7.2 Replication in Distributed Storage ........................................... 59
  7.2.1 Replication for Availability ............................................. 59
  7.2.2 Active and Passive Replication Techniques ......................... 60
  7.2.3 Replication for Scalability ............................................ 61
  7.2.4 Replica Placement .......................................................... 62
7.3 Consistency Challenges ....................................................... 63
7.4 Existing Systems ............................................................... 65
  7.4.1 Key-value Store: Dynamo ................................................ 65
  7.4.2 Column-based Storage: Cassandra ..................................... 65
  7.4.3 Geographical Database: Spanner ....................................... 66
  7.4.4 Geographical Database: Yahoo! PNUTS ............................... 66
7.5 Discussion ............................................................................. 67
  7.5.1 Some Possible Future Directions and Open Issues .................. 69
7.6 Conclusion ............................................................................. 70

Bibliography .................................................................................. 71

8 GlobLease: A Globally Consistent and Elastic Storage System using Leases 75
8.1 Introduction ............................................................................. 76
8.2 System Architecture .............................................................. 77
  8.2.1 Nodes .............................................................................. 78
  8.2.2 Links .............................................................................. 78
8.3 Lease-based Consistency Protocol ............................................. 79
  8.3.1 Lease .............................................................................. 79
  8.3.2 Lease-based Consistency .................................................. 80
  8.3.3 Lease Maintenance .......................................................... 80
  8.3.4 Leasing Period ............................................................... 81
  8.3.5 Master Migration and Failure Recovery ............................... 81
  8.3.6 Handling Read and Write Requests .................................... 82
8.4 Scalability and Elasticity ........................................................ 82
  8.4.1 Affiliated Nodes .............................................................. 83
8.5 Evaluation ............................................................................... 85
  8.5.1 Experiment Setup ............................................................ 85
  8.5.2 Varying Load ................................................................. 86
  8.5.3 Lease Maintenance Overhead ............................................ 89
  8.5.4 Varying Read/Write Ratio ............................................... 91
  8.5.5 Varying Lease Length ..................................................... 92
  8.5.6 Skewed Read Workload ................................................... 92
  8.5.7 Elasticity with Spiky and Skewed Workload ......................... 92
8.6 Related Work .......................................................................... 93
  8.6.1 Distributed Hash Tables .................................................. 93
  8.6.2 Data Consistency ............................................................ 93
  8.6.3 Lease-based Consistency ................................................ 93
  8.6.4 Elasticity Issues ............................................................. 94
Bibliography

9 BwMan: Bandwidth Manager for Elastic Services in the Cloud
9.1 Introduction .................................................. 100
9.2 OpenStack Swift .............................................. 102
9.3 Predictive Models of the Target System ...................... 103
  9.3.1 User-oriented Performance versus Available Bandwidth ........ 104
  9.3.2 Data Recovery Speed versus Available Bandwidth .......... 104
9.4 BwMan: Bandwidth Manager .................................. 104
  9.4.1 BwMan Control Work-flow ................................ 106
  9.4.2 Tradeoff Scenario ....................................... 106
9.5 Evaluation ................................................... 107
  9.5.1 OpenStack Swift Storage .................................. 107
  9.5.2 Experiment Scenarios .................................... 107
  9.5.3 Experiment Setup ........................................ 108
  9.5.4 User-centric Workload Experiment ........................ 109
  9.5.5 System-centric Workload Experiment ...................... 109
  9.5.6 Policy-based Tradeoff Scenario .......................... 110
9.6 Related Work ................................................ 111
9.7 Conclusion and Future Work ................................. 112

Bibliography

10 ProRenaTa: Proactive and Reactive Tuning to Scale a Distributed Storage System
10.1 Introduction .................................................. 116
10.2 Observations and background ............................... 118
  10.2.1 Distributed storage systems ............................. 118
  10.2.2 Observations ........................................... 118
10.3 System design ............................................... 120
  10.3.1 Monitor ................................................ 120
  10.3.2 Analysis ............................................... 121
  10.3.3 Plan .................................................. 121
  10.3.4 Execute ............................................... 122
  10.3.5 Knowledge ............................................. 123
10.4 Workload Prediction ........................................ 123
  10.4.1 Representative workload types .......................... 125
  10.4.2 Prediction methodology ................................. 125
10.5 Controller models .......................................... 126
  10.5.1 Throughput performance model .......................... 126
  10.5.2 Data migration model ................................. 128
10.6 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 129
  10.6.1 Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . 129
  10.6.2 Evaluation results . . . . . . . . . . . . . . . . . . . . . . . . . . 131
10.7 Related work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 136
10.8 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 138

Bibliography 139

Appendices 142

A Predictor Implementation 143

B Analytical model for Data Migration 147
List of Figures

2.1 The Roles of Cloud Provider, Cloud Consumer, and End User in Cloud Computing .......................... 10
2.2 Storage Structure of Yahoo! PNUTS ......................................................................................... 12
2.3 Distributed Hash Table with Virtual Nodes .............................................................................. 13
2.4 Different Data Consistency Levels ............................................................................................. 15
2.5 Block Diagram of a Feedback Control System ........................................................................... 17

6.1 Read latency of Swift using proxies with different VM flavors .................................................... 44
6.2 Write latency of Swift using proxies with different VM flavors ..................................................... 44
6.3 Read Performance of Swift under Network Latencies ................................................................. 45
6.4 Write Performance of Swift under Network Latencies ............................................................... 46
6.5 Performance of Swift with Bandwidth Limits on Storage Servers ............................................. 47
6.6 Workload Patterns ..................................................................................................................... 48
6.7 Proxy Server Outbound Bandwidth Usage for Reads and Writes .............................................. 48
6.8 Control Flow of the Self-healing Algorithm ................................................................................ 49
6.9 Self-healing Validation under the Control System .................................................................... 50

7.1 Active and Passive Replication ..................................................................................................... 60

8.1 GlobLease system structure having three replicated DHTs ......................................................... 78
8.2 Impact of varying intensity of read dominant workload on the request latency ....................... 86
8.3 Impact of varying intensity of write dominant workload on the request latency ...................... 87
8.4 Latency distribution of GlobLease and Cassandra under two read dominant workloads ................... 87
8.5 High latency requests ................................................................................................................... 89
8.6 Impact of varying read:write ratio on the leasing overhead ....................................................... 89
8.7 Impact of varying read:write ratio on the average latency ......................................................... 90
8.8 Impact of varying lengths of leases on the average request latency ......................................... 90
8.9 Impact of varying intensity of skewed workload on the request latency ................................... 91
8.10 Elasticity experiment of GlobLease ............................................................................................ 91

9.1 Regression Model for System Throughput vs. Available Bandwidth .......................................... 103
9.2 Regression Model for Recovery Speed vs. Available Bandwidth ................................................ 105
9.3 MAPE Control Loop of Bandwidth Manager .............................................................................. 105
9.4 Control Workflow ................................................. 106
9.5 Throughput under Dynamic Bandwidth Allocation using BwMan .......... 109
9.6 Data Recovery under Dynamic Bandwidth Allocation using BwMan ........ 109
9.7 Throughput of Swift without BwMan ................................ 110
9.8 Throughput of Swift with BwMan .................................. 110

10.1 Observation of SLA violations during scaling up. (a) denotes a simple increasing workload pattern; (b) scales up the system using a proactive approach; (c) scales up the system using a reactive approach .............................. 119
10.2 ProRenaTa control framework ....................................... 120
10.3 Scheduling of reactive and proactive scaling plans .......................... 122
10.4 Standard deviation of load on 20 servers running a 24 hours Wikipedia workload trace. With larger number of virtual tokens assigned to each server, the standard deviation of load among servers decreases ......................... 127
10.5 Throughput performance model ...................................... 128
10.6 Data migration model under throughput and SLA constraints ............. 129
10.7 Aggregated CDF of latency for different approaches ...................... 132
10.8 Aggregated CDF of CPU utilization for different approaches ............ 133
10.9 Actual workload and predicted workload and aggregated VM hours used corresponding to the workload .................................................. 134
10.10 SLA commitment comparing ideal, feedback and predict approaches with ProRenaTa ......................................................... 135
10.11 Utility for different approaches ...................................... 136
Part I

Thesis Overview
Chapter 1

Introduction

With the growing popularity of Internet-based services, more powerful back-end storage systems are needed to match the increasing workloads in terms of concurrency, intensity, and locality. When designing a high performance storage system, a number of major properties, including scalability, elasticity, availability, consistency guaranties and partition tolerance, need to be satisfied.

**Scalability** is one of the core aspects in the design of high performance storage solutions. Centralized storage solutions are no longer able to support large scale web applications because of the high access concurrency and intensity. Under this scenario, distributed storage solutions, which are designed with greater scalability, are proposed. A distributed storage solution provides an unified storage service by aggregating and managing a large number of storage instances. A scalable distributed storage system can, in theory, aggregate unlimited number of storage instances, therefore providing unlimited storage capacity if there are no bottlenecks in the system.

**Availability** is another desired property for a storage system. Availability means that data stored in the system are safe and always or most of the time available to its clients. Replications are usually implemented in a distributed storage system in order to guarantee data availability in the presence of server or network failures. Specifically, several copies of the same data are preserved in the system at different servers, racks, or data centers. Thus, in the case of server or network failures, data can still be served to clients from functional and accessible servers that having copies of the data. Maintaining multiple copies of the same data brings the challenge of **consistency** issues. Based on application requirements and usage scenarios, a storage solution is expected to provide some level of consistency guarantee. For example, strong consistency provides that all the data copies act synchronously like one single copy and it is desired because of its predictability. Other consistency models, for example, eventual consistency allows data copies to diverge within a short period of time. In the general case, stricter consistency involves more overhead for a system to achieve and maintain. Hosting multiple data replicas in multiple servers also needs to survive with **network partitions**. Partition of networks blocks the communications among data copies. As a result, either inconsistent result or no result can be returned to clients in this scenario.
Despite researching on the above properties to design high performance storage solutions, we narrow down the design scope by targeting storage systems that are hosted in public/private Cloud platforms. Cloud computing not only shifts the paradigm that companies used to host their Internet-based businesses, but also provides end users a brand new way of accessing services and data. A Cloud is the integration of data center hardware and software that provides "X as a service (XaaS)"; value of X can be infrastructure, hardware, platform, and software. The scope of Cloud computing focused in this thesis is infrastructure as a service (IaaS), where Cloud resources are provided to consumers in the form of physical or more often virtual machines (VMs). Cloud computing provides the choice for companies to host their services without provisioning a data center. Moreover, the pay-as-you-go business model allows companies to use Cloud resources on a short-term basis as needed. On one hand, companies benefit from letting resources go when they are no longer needed. On the other hand, companies are able to request more resources anytime from the Cloud platform when their businesses grow without planning ahead for provisioning. The autonomic tuning of resources according to the needs of applications hosted in the Cloud is often called dynamic resource provisioning, auto-scaling or elasticity.

1.1 Research Objectives

The main goal of this work is to optimize service latency of distributed storage systems that are hosted in a Cloud environment. From a high-level view, there are two main factors that significantly impact the service latency of a distributed storage system, assuming a static execution environment and available resources: (a) the efficiency of the storage solution itself and (b) the intensity of the workload that need to be handled by the system. Naturally, a less efficient storage solution slows down the processing of requests, whereas an intensive workload might saturate the system. Thus, in order to provide low service latency guarantees, we define two main corresponding goals:

1. to improve the efficiency of distributed storage solutions, and

2. to make storage systems adapt to workload changes to achieve the low latency guarantees at a low cost

Our vision towards the first goal is make storage systems to be distributed in a larger scale, so that more requests can be served by servers that are close to clients, which significantly reduce the portion of high latency requests. We specify optimizations on data consistency algorithm in this scenario with the objective of reducing replica communication overhead under the requirements of scalability, consistency, and availability. The core challenge to achieve the second goal is introduced by the complexity of workload patterns, which can be highly dynamic in intensity, concurrency, and locality. We propose smart middlewares, i.e., elasticity controllers, to effectively and efficiently provision resources allocated to a distributed storage system, which allows it to achieve low latency at a reduced cost.

With respect to these goals, our paper [1] is a study on state of the art optimizations on designing efficient storage solutions, our solution [2] is a work done with respect to
1.2. RESEARCH METHODOLOGY

improving the efficiency of a distributed storage system, our work [3] presents a bandwidth manager for distributed storage systems under dynamic workload patterns, our system [4] contains work towards improving resource utilization and guaranteeing service latency under dynamic workload.

1.2 Research Methodology

In this section, we describe the methods that we used in this research work. We provide descriptions of the process that we followed and design decisions that we made in order to achieve our goals. We also discuss challenges that we faced during the process and how we overcame them.

1.2.1 Design of Efficient Storage Solutions

The work on this matter does not follow analytical, mathematical optimization methods, but is rather based on an empirical approach. We approach the problem by first studying techniques/algorithms used in the design of distributed storage solutions. This process provides us the knowledge base for inventing new algorithms to improve the efficiency of storage solutions. After understanding the state of the art solutions, we then start investigating the usage scenarios of distributed storage solutions. The efficiency of a storage solution varies on different usage scenarios. We focus on analyzing solutions’ efficiency with respect to the operating overhead when deployed in different usage scenarios. This overhead usually differs because of different system architectures, implementations and algorithms. We place our innovation when there is no efficient storage solution for a needing usage scenario. Once we have confirmed our innovation direction, we investigate the causes of inefficiency of the current solutions and avoid them in our design. After examining sufficient number of leading storage solutions, we choose the most suitable system architecture that can be applied in this scenario. We tailor algorithms for this scenario by avoiding the known performance bottlenecks. Finally, we evaluate our design and implementation by comparing it with the several leading storage solutions. We use request latency as the performance measure and also present algorithm overhead, when applicable.

1.2.2 Design of Elasticity Controller

The work on this matter also follows an empirical approach. Our approach is based on first understanding the environmental and system elements/parameters that are influential to the effectiveness and accuracy of an elasticity controller, which directly affects service latency of the controlled systems. Then, we study the technologies that are used in building performance models and the frameworks that are applied in implementing the controller. The result of the studies allows us to discover the unconsidered elements/parameters that influences the effectiveness and accuracy of an elasticity controller. We verify our assumptions on the performance degradation of an elasticity controller if not including the elements/parameters by experiments. Once we have confirmed the space for improving the effectiveness and accuracy of an elasticity controller, we innovate on designing new
CHAPTER 1. INTRODUCTION

performance models that consider those environmental and system elements/parameters. After implementing our controllers using the novel performance model, we evaluate it by comparing it to the original implementation. For the evaluation, we deploy our systems in real platforms and test them with real-world workload, where possible. We use the latency SLA commitment and the resource utilization as the performance measure.

1.2.3 Challenges

We have faced several challenges during this work. First, most of the systems presented in literature are not publicly available to download and experiment with. Thus, we have to implement our own versions of the algorithms/architectures following the description in the literatures for comparisons in most of the cases. Another significant obstacle that we face when doing the experiments are the lack of available real-world workload or data set. For example, in the work of ProRenaTa, we have to create our own synthetic workload by downloading access log traces of Wikipedia from Amazon public dataset. Finally, we have experienced performance interference of virutal machines (VMs) on Cloud platforms since we are often not able to control the allocation of VMs on physical machines. Worse, service latency is a sensitive measure to VM performance interference. We have to deploy and evaluate our systems multiple times to filter out potential performance interference introduced by the activities of other VMs in order to assure the credibility of our results.

1.2.4 Experimental Evaluation

We conduct all of our experiments using virtual machines in a Cloud environment. By using VMs, we are able to use a clean and relatively isolated environment for experiments. However, we have experience performance interference from other VMs on the same platform as explained in the previous section. For the systems that are used for comparisons, since most of them are not open source, we have to implement prototypes by following the algorithms and methodologies described in the literature. We choose to use real-world workloads to increase the credibility of our evaluations. We construct the workload from public data sets in order to facilitate reproducibility.

1.3 Thesis Contributions

The contributions of this thesis are as follows.

1. A performance evaluation of a distributed storage system in a simulated Cloud platform. It identifies the requirements of deploying distributed storage systems in a community Cloud environment.

2. A survey of replication techniques for distributed storage systems, that summarizes the state-of-the-art data replication methods and discusses the consistency issues come with data replication. The survey concludes with open issues, topics for further research.
1.4 Thesis Organization

The rest of this thesis organized as follows. Chapter 2 gives the necessary background and describes the systems used in this research work. Chapter 3 provides an overview of related techniques in achieving high performance distributed storage systems hosted in the Cloud. Thesis contributions are discussed in Chapter 4. Chapter 5 contains conclusions and future work. Part II contains the research papers produced during this work.
Chapter 2

Background

Hosting services in the Cloud are becoming more and more popular because of a set of desired properties provided by the platform, which include low setup cost, unlimited capacity, professional maintenance and elastic provisioning. Services that are elastically provisioned in the Cloud are able to use platform resources on demand, thus saving hosting costs by appropriate provisioning. Specifically, instances are spawned when they are needed for handling an increasing workload and removed when the workload drops. Enabling elastic provisioning saves the cost of hosting services in the Cloud, since users only pay for the resources that are used to serve their workload.

In general, Cloud services can be coarsely characterized in two categories: state-based and stateless. Examples of stateless services include front-end proxies, static web servers, etc. Distributed storage service is a state-based service, where state/data needs to be properly maintained to provide a correct service. Thus, managing a distributed storage system in the Cloud exposes many challenging issues. In this thesis, we focus on the self-management and performance aspect of a distributed storage system deployed in the Cloud. Specifically, we examine techniques in order to design a distributed storage system that can operate efficiently in a Cloud environment [1, 2]. Also, we investigate approaches that support a distributed storage system to perform well in a Cloud environment by achieving a set of desired properties including elasticity, availability, and performance guarantees (Service Level Agreements).

In the rest of this chapter, we present the concepts and techniques used in the papers included in this thesis; Cloud environment, a visualized environment to effectively and economically host services; distributed storage systems, storage systems that are organized in a decentralized fashion; auto-scaling techniques, methods that are used to achieve elasticity in a Cloud environment; OpenStack Swift, a distributed storage system that is used as a study case.

2.1 Cloud Computing

"Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services [5]." A
CHAPTER 2. BACKGROUND

Cloud is the integration of data center hardware and software that provides "X as a service (XaaS)" to clients; value of X can be infrastructure, hardware, platform, and software. Those services in the Cloud are made available in pay-as-you-go manner to public. The advantages of Cloud computing to Cloud providers, consumers, and end users are well understood. Cloud providers make profits in renting the resources, providing services based on their infrastructures to Cloud consumers. Cloud consumers, on the other hand, greatly enjoy the simplified software and hardware maintenance and pay-as-you-go pricing model to start their business. Also, Cloud computing makes a illusion to Cloud consumers that the resources in the Cloud are unlimited and is available whenever requested without building or provisioning their own data centers. End users is able to access the services provided in the Cloud anytime and anywhere with great convenience. Figure 2.1 demonstrates the roles of Cloud provider, Cloud consumer, and end user in Cloud computing.

Based on the insights in [5], there are three innovations in Cloud computing:

1. The illusion of infinite computing resources available on demand, thereby eliminating the need for Cloud consumers to plan far ahead for provisioning;

2. The elimination of an up-front commitment by Cloud consumers, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs;

3. The ability to pay for use of computing resources on a short-term basis as needed (e.g., processors by the hour and storage by the day) and release them when they are no longer needed.
2.2. DISTRIBUTED STORAGE SYSTEM

Elasticity

Elasticity is a property of a system, which allows the system to adjust itself in order to offer satisfactory service with minimum resources (reduced cost) in the presence of changing workloads. Typically, an elastic system is able to add or remove service instances (with proper configurations) according to increasing or decreasing workloads with respect to meet the Service Level Agreements, if any. To support elasticity, a system needs to be scalable, which means that its capability to serve workload is proportional to the number of service instances deployed in the system. Then, the hosting platform needs to be scalable, i.e., having enough resources to allocate whenever requested. Cloud computing is a perfect suit in this aspect. Another requirement to achieve elasticity is to have a controller that can automatically scale the system based on the intensity of the incoming workload. A simple example is the elastic scaling service provided by Amazon using Amazon Cloud Watch [6]. Background of auto-scaling techniques are introduced in Section 2.3.

Service Level Agreement

Service Level Agreements (SLA) define the quality of service that are expected from the service provider. SLAs are usually negotiated and agreed between Cloud service providers and Cloud service consumers. A SLA can define the availability aspect and/or performance aspect of a service, such as service up-time, service percentile latency, etc. A violation of SLA affects both the service provider and consumer. When a service provider is unable to uphold the agreed level of service, they typically pay penalties to the consumers. From the consumers perspective, a SLA violation can result in degraded service to their clients and consequently lead to loss in profits. Hence, SLA commitment is essential to the profit of both Cloud service providers and consumers.

2.2 Distributed Storage System

A distributed storage system provides an unified storage service by aggregating and managing a large number of storage nodes. A scalable distributed storage system can, in theory, aggregate unlimited number of storage nodes, therefore providing unlimited storage capacity. Distributed storage solutions include relational databases, NoSQL databases, distributed file systems, array storages, and key-value stores. The rest of this section provide backgrounds on the three main aspects of a distributed storage system, i.e., organizing structure, data replication and data consistency.

2.2.1 Structures of Distributed Storage Systems

Usually, a distributed storage system is organized either using a hierarchical or symmetric structure.
Hierarchical Structure

A hierarchical distributed storage system is constructed with multiple components responsible for different functionalities. For example, separate components can be designed to maintain storage namespace, request routing or the actual storage. Recent representative systems organized in hierarchical structures are Google File System [7], Hadoop File System [8], Google Spanner [9] and Yahoo! PNUTS [10]. An example is given in Figure 2.2, which shows the storage structure of Yahoo! PNUTS. It uses tablet controller to maintain storage namespace, separate router components to route requests to responsible tablet controllers, message brokers to asynchronously deliver message among different storage regions and storage units to store data.

Symmetric Structure

A symmetrically structured distributed storage system can also be understood as a peer to peer (P2P) storage system. It is a storage system that does not need centralized control and the algorithm running at each node is equivalent in functionality. A distributed storage system organized in this way has robust self-organizing capability since the P2P topology often changes as nodes join or leave the system. It is also scalable since all nodes function the same and organized in a decentralized fashion, i.e., there is no centralized bottleneck. Availability is achieved by having data redundancies in multiple peers in the system.

An efficient resource location algorithm in the P2P overlay is essential to the performance of a distributed storage system built with P2P structure. One core requirement of such algorithm is the capability to adapt to frequent topology changes. Some systems use a centralized namespace service for searching resources, which is proved to be a bottleneck. An elegant solution to this issue is distributed hash table (DHT). It uses the hash of resource names to locate the object. Different routing strategies and heuristics are proposed to improve the routing efficiency.
Distributed Hash Table

Distributed Hash Table (DHT) is widely used in the design of distributed storage systems [11, 12, 2]. DHT is a structured peer-to-peer overlay that can be used for namespace partitioning and request routing. DHT partitions the namespace by assigning each node participating in the system a unique ID. According to the assigned ID, a node is able to find its predecessor (first ID before it) or successor (first ID after it) in the DHT space. Each node maintains the data that fall into the range between its ID and its predecessor’s ID. As an improvement, nodes are allowed to hold multiple IDs, i.e., maintaining data in multiple hashed namespace ranges. These virtual ranges are also called virtual nodes in literature. Applying virtual nodes in a distributed hash table brings a set of advantages including distributing data transfer load evenly among other nodes when a node joins/leaves the overlay and allowing heterogeneous nodes to host different number of virtual nodes, i.e., handling different loads, according to their capacities. Figure 2.3 presents a DHT namespace distributed among four nodes with virtual nodes enabled.

Request routing in a DHT is handled by forwarding request through predecessor links, successor links or finger links. Finger links are established among nodes based on some criteria/heuristics for efficient routing [13, 14]. Algorithms are designed to update those links and stabilize the overlay when nodes join and leave. Load balancing among nodes is also possible by applying techniques, such as virtual nodes, in a DHT.

2.2.2 Data Replication

Data replication is usually employed in a distributed storage system to provide higher data availability and system scalability. In general approaches, data are replicated in different
disks, physical servers, racks, or even data centers. In the presence of data loss or corruption caused by server failures, network failures, or even power outage of the whole data center, the data can be recovered from other correct replicas and continuously serving to the clients. The system scalability is also improved by using the replication techniques. Concurrent clients is able to access the same data at the same time without bottlenecks by having multiple replicas of the data properly managed and distributed. However, data consistency needs to be properly handled as a side effect of data replication and will be briefly introduced in Section 2.2.3.

Replication for Availability

A replicated system is designed to provide services with high availability. Multiple copies of the same data are maintained in the system in order to survive server failures. Through well-designed replication protocol, data loss can be recovered through redundant copies.

Replication for Scalability

Replication is not only used to achieve high availability, but also to make a system more scalable, i.e., to improve ability of the system to meet increasing performance demands in order to provide acceptable level of response time. Imagine a situation, when a system operates under so high workload that goes beyond the system’s capability to handle it. In such situation, either system performance degrades significantly or the system becomes unavailable. There are two general solutions for such scenario: vertical scaling, i.e., scaling up, and horizontal scaling, i.e., scaling out. For vertical scaling, data served on a single server are partitioned and distributed among multiple servers, each responsible for a part of data. In this way, the system is capable to handle larger workloads. However, this solution requires much knowledge on service logic, based on which, data partitioning and distribution need to be performed in order to achieve scalability. Consequently, when scaled up, the system might become more complex to manage. Nevertheless, since only one copy of data is scattered among servers, data availability and robustness are not guaranteed. One the other hand, horizontal scaling replicates data from one server on multiple servers. For simplicity without losing generality, assume all replication servers are identical, and client requests are distributed evenly among these servers. By adding servers and replicating data, system is capable to scale horizontally and handle more requests.

Geo-replication

In general, accessing the data in close proximity means less latencies. This motivates many companies or institutes to have their data/service globally replicated and distributed by using globally distributed storage systems, for example [9, 2]. New challenges appear when designing and operating a globally distributed storage system. One of the most essential issues is the communication overhead among the servers located in different data centers. In this case, the communication latency is usually higher and the link capacity is usually lower.
2.2. DISTRIBUTED STORAGE SYSTEM

2.2.3 Data Consistency

Data replication also brings new challenges for system designers including the challenge of data consistency that requires the system to tackle with the possible divergence of replicated data. Various consistency levels, as shown in Figure 2.4, are proposed based on different usage scenarios and application requirements. There are two general approaches to maintain data consistency among replicas: master-based and quorum-based.

Master-based consistency

A master based consistency protocol defines that, within a replication group, there is a master replica of the object and the other replicas are slaves. Usually, it is designed that the master replica is always up-to-date while the slave replicas can be a bit outdated. The common approach is that the master replica serialize all write operations while the slave replicas are capable of serving parallel read operations.

Quorum-based consistency

A replication group/quorum involves all the nodes that maintain replicas of the same data object. The number of replicas of a data object is the replication degree and the quorum size (N). Assume that read and write operations of a data object are propagated to all the replicas in the quorum. Let us define R and W responses are needed from all the replicas to complete a read or write operation. Various consistency levels can be achieved by configuring the value of R and W. For example, in Cassandra [12], there are multiple consistency choices telling the system how to handle a read or write operation. Specifically, what is the minimum number of R and W to complete a request. Different combinations of R and W values result in different consistency level. For example, sequential consistency can be achieved by having R+W>N.

Figure 2.4: Different Data Consistency Levels
2.2.4 OpenStack Swift

OpenStack Swift is a distributed object storage system, which is part of OpenStack Cloud Software [15]. It consists of several different components, providing functionalities such as highly available and scalable storage, lookup service, and failure recovery. Specifically, the highly available storage service is achieved by data replication in multiple storage servers. Its scalability is provided with the aggregated storage from multiple storage servers. The lookup service is performed through a Swift component called proxy server. Proxy servers are the only access entries for the storage service. The main responsibility of a proxy server is to process the mapping of the names of the requested files to their locations in the storage servers, similar to the functionality of NameNodes in GFS [7] and HDFS [8]. The namespace mapping is provided in a static file called the Ring file. Thus, the proxy server itself is stateless, which ensures the scalability of the entry points in Swift. The Ring file is distributed and stored on all storage and proxy servers. When a client accesses a Swift cluster, the proxy server checks the Ring file, loaded in its memory, and forwards client requests to the responsible storage servers. The high availability and failure recovery are achieved by processes called the replicators, which run on each storage server. Replicators use the Linux rsync utility to push data from a local storage server to other storage servers, which should maintain the same replicated data based on the mapping information provided in the Ring file. By doing so, the under-replicated data are recovered.

2.3 Auto-scaling Techniques

Typical methods used for auto-scaling are threshold-based rules, reinforcement learning or Q-learning (RL), queuing theory, control theory and time series analysis. We have used techniques of reinforcement learning, control theory and time series analysis in the papers presented in the second section of the thesis.

2.3.1 Threshold-based Rules

The representative systems that use threshold-based rules to scale a service are Amazon Cloud Watch [6] and RightScale [16]. Simply speaking, this approach defines a set of thresholds or rules in advance. Violating the thresholds or rules to some extent will trigger the action of scaling. Threshold-based rule is a typical implementation of reactive scaling.

2.3.2 Reinforcement Learning or Q-learning (RL)

Reinforcement learning are usually used to understand the application behaviors by building empirical models. The empirical models are built by learning through direct interaction between monitored metrics and control metrics. After sufficient training, the empirical models are able to be consulted and referred to when making system scaling decisions. The accuracy of the scaling decisions largely depend on the consulted value from the model. And the accuracy of the model depends on the metrics and model selected, as well as the amount of data trained to the model. For example, [17] presents an elasticity controller that
2.3. AUTO-SCALING TECHNIQUES

integrates several empirical models and switches among them to obtain better performance predictions. The elasticity controller built in [18] uses analytical modeling and machine-learning. They argued that by combining both approaches, it results in better controller accuracy.

2.3.3 Queuing Theory

Queuing theory can be also applied to the design of an elasticity controller. It makes reference to the mathematical study of waiting lines, or queues. For example, [19] uses the queueing theory to model a Cloud service and estimates the incoming load. It builds proactive controllers based on the assumption of a queueing model with metrics including the arrival rate, the inter-arrival time, the average number of requests in the queue. It presents an elasticity controller that incorporates a reactive controller for scale up and proactive controllers for scale down.

2.3.4 Control Theory

Elasticity controllers using control theory to scale systems are mainly reactive, but there are also some proactive approximations such as Model Predictive Control (MPC), or even combining a control system with a predictive model. Control systems are be broadly categorized into three types: open-loop, feedback and feed-forward. Open-loop controllers use the current state of the system and its model to estimate the coming input. They do not monitor (use feedback signals) to determine whether the system output has met the desired goal. In contrast, feedback controllers use the output of the system as a signal to correct any errors from the desired value. Feed-forward controllers anticipate errors that might occur in the output before they actually happen. The anticipated behavior of the system is estimated based on a model. Since, there might exist deviations in the anticipated system behavior and the reality, feedback controllers are usually combined to correct prediction errors.

Figure 2.5 illustrates the basic structure of a feedback controller. It usually operates in a MAPE-K (Monitor, Analysis, Plan, Execute, Knowledge) fashion. Briefly, the system monitors the feedback signal of a selected metric as the input. It analyzes the input signal using the method implemented in the controller. The methods can be broadly placed into four categories: fixed gain control, adaptive control, reconfiguring control and model predictive control. After the controller has analyzed the input (feedback) signal, it plans the
scaling actions and sends them to actuators. The actuators are the methods/APIs to resize
the target system. After resizing, another round of feedback signal is input to the controller.

2.3.5 Time Series Analysis

A time-series is a sequence of data points, measured typically at successive time instants
spaced at uniform time intervals. The purpose of applying time series analysis in auto-
scaling problem is to provide predicted value of an interested input metric (CPU load or
input workload) to facilitate the decision making of an elasticity controller. Techniques
used for this purpose in the literature are Moving Average, Auto-regression, ARMA, expo-
nential smoothing and machine learning based approaches.
Chapter 3

Performance Optimization for Distributed Storage Systems Hosted in the Cloud

We optimize the performance of a distributed storage system deployed in the Cloud from two different angles. One optimization is conducted on the design of distributed storage systems. Another approach is to design smart agents/middlewares that helps a distributed storage system to operate efficiently in a Cloud environment.

3.1 Optimizations on the Design of Distributed Storage Systems

We optimize the performance of distributed storage systems with respect to service latency. Service latency of a distributed storage systems is determined by the load of the system and the algorithm designed to process the request. Discussions on proper handling of dynamic load to guarantee the performance of a distributed storage system are presented in the next section. In this section, we investigate designs of storage systems. Data are usually stored in a replicated fashion in a distributed storage system. Data replication guarantees the high availability of the data and increase service concurrency if there is no data consistency constraints. However, some applications may expect the underlying storage system to maintain and return strongly consistent data. The overhead to synchronize the replicated data can significantly increase the service latency. This effect gets more obvious when the communication cost among replicas increases. Geo-replication is such an example. Our focus is to design and implement data consistency algorithms that have low communication overhead with respect to specific workload patterns and system deployment. The reduced data synchronization overhead in a replicated storage system can largely improve its service latency.
3.1.1 Summary of Proposed Optimizations

In Chapter 7, we present a survey of the state of the art replication techniques used in the design of distributed storage systems. Data replication provides the system with a set of desired properties include high performance, tolerance to failures, and elasticity. We have compared and categorized replication techniques used in four systems that define the state of the art in distributed storage systems. We have defined a design space for replication techniques, identified current limitations and challenges. Based on the results of the survey, we have designed our own replication strategy, presented in Chapter 8, that has low data synchronization overhead when achieving strong consistency.

Then, we present GlobLease, an elastic, globally distributed and consistent key-value store. It is organised as multiple distributed hash tables (DHTs) storing replicated data and the namespace. Across DHTs, data lookups and accesses are processed with respect to the locality of DHT deployments. We explore the use of leases in GlobLease to maintain data consistency across DHTs. The leases enable GlobLease to provide fast and consistent read access in a global scale with reduced global communications. The write accesses are optimized by migrating the master copy to the locations, where most of the writes take place. The elasticity of GlobLease is provided in a fine-grained manner in order to effectively and efficiently handle spiky and skewed read workloads. In our evaluation, GlobLease has demonstrated its optimized global performance, in comparison with Cassandra, with read and write latency less than 10 ms in most of the cases.

3.2 Optimizations on the Load of Distributed Storage Systems

In order to benefit from the pay-as-you-go pricing model in the Cloud, a dynamic resource provisioning middleware (an elasticity controller) is needed. A core requirement of elasticity controllers is that it should be able to help saving the provisioning cost of an application without sacrificing its performance. Specifically, the system needs to provision the minimum resources as needed by the application to achieve a desired quality of service, which is usually negotiated between Cloud provider and Cloud consumer and defined as the Service Level Agreement (SLA). In the view of Cloud providers, the dynamic resource provisioning system needs to keep the performance promise in the SLA under any circumstances, e.g. dynamic incoming workloads to the application, otherwise a penalty has to be paid. In the view of Cloud consumer, a violation in the SLA usually causes bad service experiences to their end users and, as a result, influences profit. Thus, it is essential to have a well-designed elasticity controller for both Cloud provider and Cloud consumer that provides the following properties:

1. Accurate resource allocation that satisfy both constraints: minimize provisioning cost and SLA violations.

2. Swift adaptation to workload changes without causing resource oscillation.
3.2. OPTIMIZATIONS ON THE LOAD OF DISTRIBUTED STORAGE SYSTEMS

3. Efficient use of resources under SLA constraints during scaling. Specifically, when scaling up, it is preferable to add instances at the very last possible moment. In contrast, during scaling down, it is better to remove instances as soon as they are not needed anymore. The timings are challenging to control.

The services hosted in the Cloud can be coarsely categorized into two categories: stateless and stateful. Dynamic provisioning of stateless services is relatively easy since less/no overhead is needed to prepare a Cloud VM before it can serve workloads, i.e., adding or removing Cloud VMs affects the performance of the service immediately. On the other hand, scaling a stateful service requires states to be properly transferred/configured from/to VMs. Specifically, when scaling up a stateful system (adding VMs), a VM is not able to function until proper states are transferred to it. When scaling down a stateful system (removing VMs), a VM cannot be safely removed from the system until its states are arranged to be handled/preserved by other VMs. Furthermore, this scaling overhead creates additional workload for the other VMs in the system and can result in the degradation of system performance if not well handled. Thus, it is challenging to scale a stateful system and adds an additional requirement for the design of an elasticity controller;

4. Be aware of scaling overheads, including the consumption of system resources and time, and prevent them from causing SLA violations.

3.2.1 Summary of Proposed Optimizations

In Chapter 9, we present a bandwidth manager (BwMan) for distributed storage systems. BwMan predicts and performs the bandwidth allocation and tradeoffs between multiple service activities in a distributed storage system. The performed bandwidth allocation is able to guarantee the performance of the storage system with respect to meeting Service Level Agreements (SLAs). To make management decisions, BwMan uses statistical machine learning (SML) to build predictive models. This allows BwMan to arbitrate and allocate bandwidth dynamically among different activities to satisfy different SLAs.

Then, in Chapter 10, we present ProRenaTa, an elasticity controller for distributed storage systems. The advantage of ProRenaTa is that it guarantees the specified latency SLAs of the underlying storage system with the provisioning of a minimum number of VM instances. To achieve this, ProRenaTa needs to know an accurate prediction of the incoming workload, which is designed and tested with real-world traces in the chapter. Then, we have built and trained performance models to map the workload intensity and the desired number of VMs. The performance model is used by a proactive controller to prepare VMs in advance and a reactive controller to correct possible prediction errors, which is mapped to the number of VMs. In order to further guarantee the performance of the system and saving the provisioning cost, a data migration model is designed and presented. By using it, ProRenaTa scheduler is able to schedule data migration to/from added/removed VMs without hurting the performance of the system. Furtermore, it also facilitates ProRenaTa to add/remove VMs at the best possible time. With all this, ProRenaTa outperforms the state of the art approaches in guaranteeing a high level of SLA commitments while improving the overall resource utilization.
Chapter 4

Thesis Contribution

In this chapter, we describe the thesis contributions. First, we list the publications produced during this work. Then, we provide details on the contributions of each publication separately and highlight the individual contributions of the thesis author.

4.1 List of Publications


2. Y. Liu, V. Vlassov, *Replication in Distributed Storage Systems: State of the Art, Possible Directions, and Open Issues*, 5th IEEE International Conference on Cyber-enabled distributed computing and knowledge discovery (CyberC), 2013


4. Y. Liu, V. Xhagjika, V. Vlassov, A. Al-Shishtawy, *BwMan: Bandwidth Manager for Elastic Services in the Cloud*, 12th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), 2014

CHAPTER 4. THESIS CONTRIBUTION

4.2 Performance Evaluation of a Distributed Storage System

The evaluation of OpenStack Swift regarding its feasibility to be deployed in a community Cloud environment is published as a research paper [20] and appears as Chapter 6 in this thesis.

Paper Contribution

This paper contributes in two aspects. The first contribution is the identification of the characteristics in a community Cloud environment regarding deploying a distributed storage service. The second contribution is the performance evaluation of Swift, a distributed object storage system, running under different platform factors including hardware, network, and failures. Our results facilitates the understanding of community Cloud characteristics and proves the feasibility of deploying Swift in a community Cloud environment. A community Cloud is established by connecting heterogeneous computing and networking resources in a wide geographic area REF. We have identified three main differences between a community Cloud and data center Cloud:

1. The types of computing resources: powerful dedicated servers vs. heterogeneous computing components;

2. The network features: exclusive high speed networks vs. shared ISP broadband and wireless connections;

3. The maintenance: regular vs. self-organized.

Based on the identification above and real studies on community Clouds REF, we simulate a community Cloud environment in the aspects of hardware resource, network connectivity, and failure rates.

Our evaluation results have established the relationship between the performance of a Swift cluster and the major environment factors in a community Cloud, including limited hardware resources and unstable network features. Furthermore, in order to tackle with relatively frequent server failures in a community Cloud, a self-healing control system is implemented for Swift and proved to be useful. We conclude that it is possible to deploy Swift in a community Cloud environment if those environment factors are in the range of our simulations, which is our future work.

Thesis Author Contribution

The author was the main contributor in this work, including writing most of the article. He came up with the methodology to evaluate the performance aspect of Swift in a simulated community Cloud environment. He also finished all the implementations and evaluations on the platform and presents the results in the paper.
4.3 State-of-the-Art Survey on Distributed Storage Systems

Our study on the replication techniques that are used to design distributed storage systems is published as a research paper [1] and appears as Chapter 7 in this thesis.

Paper Contribution

In this paper, we investigate the following three most essential issues related to data replication, to be considered when developing a replicated distributed storage service: replication for availability, replication for performance, and replica consistency.

To examine these issues, we first present the general techniques that are used to achieve replication in a distributed storage system. Then, we present different levels of data consistency that is usually achieved in production systems according to different usage scenarios and application requirements. Finally, we consider, as case studies, replication and consistency in four production distributed storage systems, namely: Dynamo of Amazon [11], Cassandra of Facebook [12], Spanner of Google [9], and Yahoo! PNUTS [10]. In our view, these four systems and others not considered here (e.g., Voldemort, Swift, and Windows Azure Storage), define state of the art in the field of large-scale distributed storage systems, and furthermore, each of the systems has its own unique properties that distinguish it from the other systems. Our discussion focuses on the replication rationales, usage scenario, and their corresponding consistency models. To be concise, we have investigated flexibility of replica placement and migration mechanisms, management of replication degree, and tradeoff of consistency guarantees for performance in the studied systems. At the end of the paper, we identify open issues and suggest future research directions regarding replication techniques for designing distributed storage systems.

Thesis Author Contribution

The author was the main contributor in this work, including writing most of the article. The survey on the related work and the characterization was studied by the thesis author. He is also the main contributor in summarizing the techniques used for data replication in distributed storage systems. The author is also responsible for discussing the limitations and proposing future research directions based on surveying a large number of research articles.

4.4 A Manageable Distributed Storage System

In [2], we propose a distributed storage system, GlobLease, which is designed for low latency access in a global scale with exposure of rich configuring APIs. The paper appears as Chapter 8 in this thesis.
CHAPTER 4. THESIS CONTRIBUTION

Paper Contribution

In this work, we present GlobLease, an elastic, globally-distributed and consistent key-value store. GlobLease differs from the state of the art systems in three ways. First, it is organised as multiple distributed hash tables (DHTs) storing replicated data and namespace, which allows more flexible management when different DHTs are placed in different locations. Specifically, across DHTs, data lookups and accesses are processed with respect to the locality of DHT deployments. Second, GlobLease uses leases to maintain data consistency among replicas. Using leases allows GlobLease to provide fast and consistent read access in a global scale with reduced global communications. Write accesses are also optimized by migrating the master copy of data to the locations, where most of the writes take place. Third, GlobLease is high adaptable to different workload patterns. Specifically, fine-grained elasticity is achieved in GlobLease using key-based multi-level lease management, which allows GlobLease to precisely and efficiently handle spiky and skewed read workloads.

GlobLease is not only a storage system that achieves optimized global performance, but also a system that can be tuned during runtime. GlobLease exposes many performance tuning APIs, such as, API to adjust a key’s replication degree, API to migrate replicas and API to add lightweight affiliated nodes for handling transient spiky or skewed workloads.

Thesis Author Contribution

The author is a major contributor in this work, including writing all of the paper. He played a major role in designing and implementing the storage system that uses leases. He led the implementation and actively refining the design during the implementation. The author is also responsible in designing the experiments and conducting them in Amazon EC2.

4.5 Arbitrating Resources among Different Workloads in a Distributed Storage System

Our work on bandwidth arbitration among different workloads in a distributed storage system, is published as a conference paper [3]. The complete paper appears as Chapter 9 of this thesis.

Paper Contribution

In this work, we first identify that there are mainly two types of workloads in a distributed storage system: user-centric workload and system-centric workload. User-centric workload is the load that are created by client requests. Workloads that are associated with system maintenance including load rebalancing, data migration, failure recovery, and dynamic reconfiguration are system-centric workload. We demonstrate that without explicitly managing the resources among different workloads leads to unstable performance of the system and results in SLA violations.
4.6 AUTO-SCALING OF A DISTRIBUTED STORAGE SYSTEM

From our experimental observations, in a distributed storage system, both user-centric and system-centric workloads are network bandwidth intensive. Then, we propose BwMan, a network bandwidth manager for distributed storage systems. BwMan arbitrates the bandwidth allocation among individual services and different service activities sharing the same infrastructure. Each service can have a demanded and dedicated amount of bandwidth allocation without interfering among each other. Dynamic and dedicated bandwidth allocation to services supports their elasticity properties with reduced resource consumption and better performance guarantees. In the perspective of user-centric workload, from our evaluation, we show that more than half of the SLA violations is prevented by using BwMan for an elastic distributed storage. In the perspective of system-centric workload, we are able to have a better estimation on the finish time of, for example, failure recovery jobs and data replication jobs. Furthermore, since BwMan controls bandwidth in port granularity, it can be easily extended to adapt to other usage scenarios where network bandwidth is a sharing resource and creates potential bottlenecks.

Thesis Author Contribution

The author is a major contributor in this work, including writing most of the paper. The machine learning techniques used to model the bandwidth arbitration among multiple services is his individual work. The author also motivates the bandwidth management idea by finding concrete usage cases, i.e., SLA commitment while there are data corruptions in a distributed storage system. Also, he is the main contributor in implementing the bandwidth manager. In addition, he proposes and conducts the evaluation plans to validate BwMan.

4.6 Auto-scaling of a Distributed Storage System

Our work on a elasticity controller for a distributed storage system has been accepted to be published as a conference paper and appears as Chapter 10 in this thesis.

Paper Contribution

We provide a design of an elasticity controller for a distributed storage system that combines both proactive and reactive auto-tuning. First, we demonstrate that there are limitations while relying solely on proactive or reactive tuning to auto-scale a distributed storage system. Specifically, reactive controller can scale the system with a good accuracy since scaling is based on observed workload characteristics. However, a major disadvantage of this approach is that the system reacts to workload changes only after it is observed. As a result, SLA violations are observed in the initial phase of scaling because of data/state migration in order to add/remove instances in a distributed storage system and causes a period of disrupted service. Proactive controller, on the other hand, is able to prepare the instances in advance and avoid any disruption in the service. However, the accuracy of workload prediction largely depends on application-specific access patterns. Worse, in some cases workload patterns are not even predictable. Thus, proper methods need to be designed and
applied to deal with the workload prediction inaccuracies, which directly influences the accuracy of scaling that in turn impacts SLA guarantees and the provisioning costs.

Then, we identify that, in essence, proactive and reactive approach complement each other. Proactive approach provides an estimation of future workloads giving a controller enough time to prepare and react to the changes but having the problem of prediction inaccuracy. Reactive approach brings an accurate reaction based on current state of the system but without leaving enough time for the controller to execute scaling decisions. So, we design ProRenaTa, which is an elasticity controller to scale a distributed storage system combining both proactive and reactive approaches.

Lastly, we build a data migration model to quantify the overhead to finish a scale plan in a distributed system, which is not explicitly considered or model in the state of the art works. The data migration model is able to guarantee less SLA violations while scaling the system. Because, by consulting the data migration model, ProRenaTa scheduler is able to smartly arbitrating the resources that are allocated to system resizing under the constraint of guaranteeing the SLA.

In sum, ProRenaTa improves the classic prediction based scaling approach by taking into account the scaling overhead, i.e., data/state migration. Moreover, the reactive controller helps ProRenaTa to achieve better scaling accuracy, i.e., better resource utilization and less SLA violations, without causing interference with the scaling activities scheduled by the proactive controller. Our results indicate that ProRenaTa outperforms the state of the art approaches by guaranteeing a high level of SLA commitments while also improving the overall resource utilization.

**Thesis Author Contribution**

The thesis author was a major contributor in this work, including writing most of the article. He is the main contributor in coming up with the idea and formalizing it. He motivates and approaches the problem in an unique angel. The author also finishes most part of the system implementation, including metric monitoring, proactive controller, reactive controller, ProRenaTa scheduler, and actuators to add and remove storage instances in GlobLease. He is also the main contributor in the evaluation design and implementation. Finally, the author also developed the data migration overhead model and validated its benefits in preventing SLA violations and boosting resource utilization.
Chapter 5

Conclusions and Future Work

In this thesis, we work towards building high performance elastic distributed storage solutions. We approach the problem by investigating the efficiency of the storage solutions themselves and smart agents to guarantee storage system performance under dynamic workload and environment. We give an introduction to distributed storage systems and we provide specific background knowledge on the systems and techniques we have applied in our research. Then, we summarize our contributions by presenting five research papers: a hands-on evaluation of a distributed storage system on a distributed platform, a survey on replication techniques used in distributed storage systems, a geo-distributed storage system that provides low latency services, a bandwidth manager to guarantee bandwidth allocation to elastic storage services, a elasticity controller that achieves better scaling accuracy and lower provision cost.

Providing low latency storage solutions has been a very active research area with a plethora of open issues and challenges to be addressed. Challenges for this matter include: the emerging of novel system usage scenarios, for example, global distribution, the uncertainty and dynamicity of incoming system workload, the performance uncertainty introduced by the virtualized deployment platform. Our work strive to provide low latency storage solutions through investigating the above issues.

5.1 Summary of Results

The flexibility of Cloud computing allows elastic services to adapt to changes in workload patterns in order to achieve desired Service Level Agreements (SLAs) at a reduced cost. Typically, the service adapts to changes in workload by adding or removing service instances (VMs), which for stateful services will require moving data among instances. The SLAs of a distributed Cloud-based service are sensitive to the available network bandwidth, which is usually shared by multiple activities in a single service without being explicitly allocated and managed as a resource. With our work on the bandwidth manager BwMan, bandwidth is allocated smartly among multiple service activities with respect to meet desired SLAs. Experiments show that BwMan is able to reduce SLA violations by a factor of two or more.
More and more IT companies are expanding their businesses and services to a global scale, serving users in several countries. Globally distributed storage systems are needed to reduce data access latency for clients all over the world. Unlike storage solutions that are deployed in one site, global deployments of storage systems introduce new challenges. One essential challenge is that the communication overhead among data replicas when deployed in a global scale has increased dramatically. In the scenario, where applications require strongly consistent data from the underlying storage systems, the increased replica communication overhead will increase service latency dramatically. With our work on the geo-replicated storage GlodLease, we have tailored and invented novel data consistency algorithms to reduce the synchronization overhead among replicas. As a result, GlodLease is able to reduce around 50% of high latency requests while guaranteeing the same data consistency level.

Provisioning stateful services in the Cloud that guarantees high quality of service with reduced hosting cost is challenging to achieve. There are two typical auto-scaling approaches: predictive and reactive. A prediction based elasticity controller leaves the system enough time to react to workload changes while a feedback based controller scales the system with better accuracy. Auto-scaling of the system using either approach alone is not sufficient for achieving satisfactory scaling results in this scenario. It has been shown that there are limitations when using a proactive or reactive approach in isolation to scale a stateful system. To overcome the limitations, we have implemented an elasticity controller, ProRenaTa, which combines both reactive and proactive approaches to leverage on their respective advantages. Furthermore, a data migration model is designed to explicitly handle the scaling overhead. As a result, we show that the combination of reactive and proactive approaches outperforms the state of the art approaches in guaranteeing a higher level of SLA commitments while improving the overall resource utilization.

5.2 Future Work

The research work described here have the opportunities to be improved in many ways. For the work to design low latency storage solutions in a global scale, we are particularly interested in data consistency algorithms that are able to provide the same consistency level while requiring less replica synchronization. We are investigating using metadata and novel message propagation patterns to reduce replica communication overhead.

Despite the contention of bandwidth, we would like to consider more factors that might influence the performance of a distributed storage system deployed in a Cloud environment. Particularly, we are heading towards analyzing the performance interference to the quality of a storage service. Performance interference happens very often when services are hosted in a Cloud environment, where there is usually no guarantee of VM collocations.

Furthermore, we are interested in improving the elasticity controller ProRenaTa. We are willing to consider more complicated workload scenarios to go beyond the assumption of uniformly distributed workload. Also, we would like to explore possibilities to reduce state migration overhead while scaling the system according to intensity of workload.
Bibliography


